Satellite Detumbling Using Magnetorquer Control

Sai Chikine
ASEN 6080, Spring 2018

One method of detumbling spacecraft in LEO is to use torque rods (or magnetorquer), which can develop dipole moments and thereby interact with the Earth's magnetic field to provide a control torque. Two different torque control laws for torque rods, different sets of initial conditions, and different orbit inclinations, are tested and analyzed. A Monte Carlo analysis of different initial angular velocities is also performed to compare the two control methods. When considering initial conditions, it is clear that varying initial conditions has an extremely large effect on the performance of both control laws. Varying orbit inclination has less of an effect. It is found that the modulating control law appears to have much lower steady state error.

I. Introduction

DETUMBLING of spacecraft is a well studied and extremely important problem. In this case, we consider a satellite inserted into a low earth orbit that begins with some initial attitude and attitude rate. The goal is then to reduce its attitude rates below some goal threshold. The control torque to do so comes from torque rods, which are rods that can magnetize their coils so as to create a magnetic dipole $\mathbf{m}$. This dipole, through interaction with Earth's magnetic field, creates a torque on the spacecraft.

Two different control laws are analyzed. The first is called Modulating B-dot control. It determines $\mathbf{m}$ based on a variable gain, which is a constant depending on the orbit geometry and spacecraft inertia, and the satellite's current angular velocity vector and the local magnetic field vector $\mathbf{b}$. The second control law considered is bang-bang B-dot control. In this case, the control law simply on or off with the maximum magnitude the actuator is capable off. For each control law, several initial attitudes and attitude rates are considered.

Orbit inclination's effect on control performance is also considered by examining two different orbit inclinations with the same initial attitude and rate conditions. This analysis is used to determine the minimum control authority needed (size of the torque rods) to achieve detumble (to within 1° deg/s) within 3 orbits.

Finally, a Monte Carlo analysis is performed to analyze the impact of the initial tumble of the spacecraft on control performance and results. The simulation is performed for both control laws.

II. Problem Statement

To model the scenario, we use 4 different reference frames:

1. $\mathcal{N}$: Earth fixed inertial (ECI) frame
2. $\mathcal{M}$: Geomagnetic Earth fixed frame
3. $\mathcal{H}$: Hill (rotating) frame
4. $\mathcal{B}$: Satellite fixed body frame

Vectors in these frames can be rotated into any of the others using DCMs (direction cosine matrices), which are simply rotation matrices between the frame bases.

To relate the $\mathcal{N}$ and $\mathcal{H}$ frames, the standard DCM for (3-1-3) Euler angles can be used. The orbit parameters Ω

(RAAN), and ı (inclination) are used along with the spacecraft periapsis ν, respectively, as the (3-1-3) Euler angles. Since the spacecraft is in a circular orbit, ν is given by $\dot{\theta} = \sqrt{\mu/r^3}$. Therefore:

$$[HN] = [M_3(\Omega)][M_1(i)][M_3(\theta_0 + \dot{\theta}t)]$$

Similarly, the rotation matrix relating the $\mathcal{N}$ and $\mathcal{M}$ frames is given by:

$$[HN] = [M_3(0)][M_1(\gamma_m)][M_3(\beta_m)]$$

where the first Euler angle is zero.

The relation between the frames $\mathcal{M}$ and $\mathcal{H}$ is then given by:

$$[HM] = [HN][NM] = [HN][MN]^{-1}$$

III. Numerical Simulations

Python was used for all the simulations presented.

A. Integrator Testing

The first step was to verify that the integrator being used worked as expected. Fourth order Runge-Kutta was used as the integration scheme. Since we used MRPs to represent attitude, MRP switching between the regular and shadow sets was implemented within the integrator.

To test, the spacecraft attitude and attitude rate was integrated over 100s. The angular momentum magnitude and rotational kinetic energy were computed over the time period based on the integration results. These are shown in Figure 1.

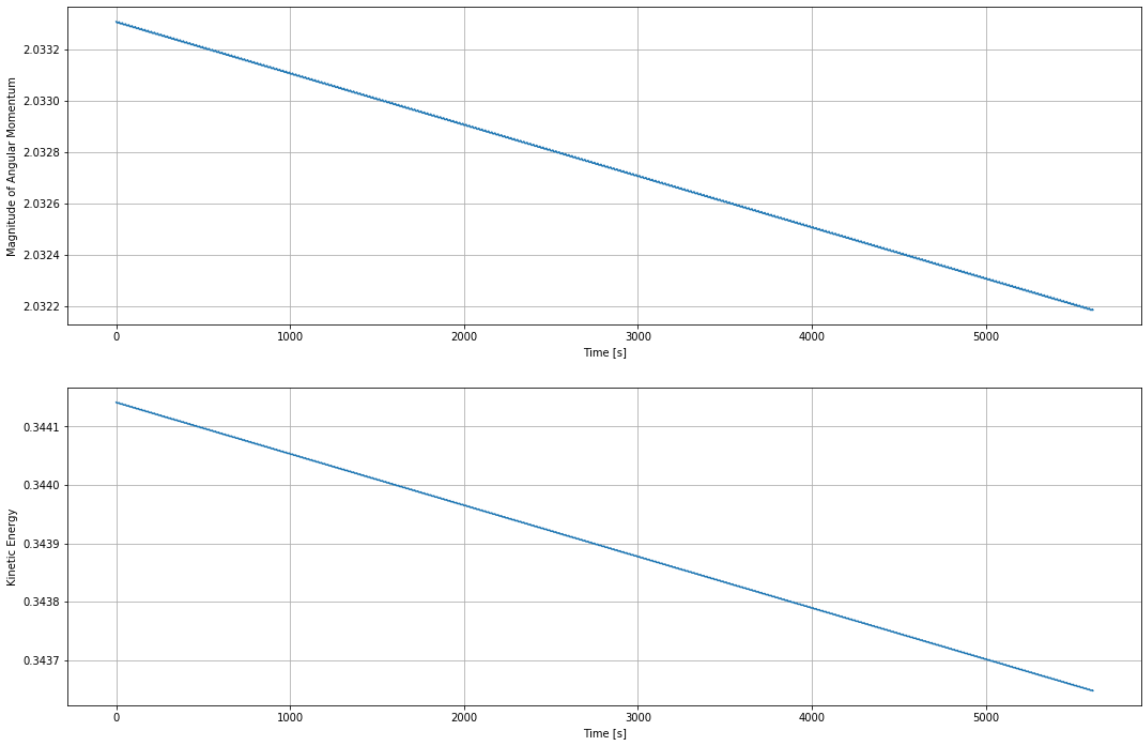

Figure 1: Magnitude of angular momentum vector (top) and rotational kinetic energy (bottom) plotted over time.

We can see that both of these quantities are indeed conserved to high accuracy. Note that conservation will be better enforced through a smaller time step. However, to make computation times reasonable, the time step was kept at 1s.

B. Geomagnetic Modeling

The magnetic field in the Hill frame was calculated using the given formulation. The components (in the Hill frame) over one orbit period are shown in Figure 2.

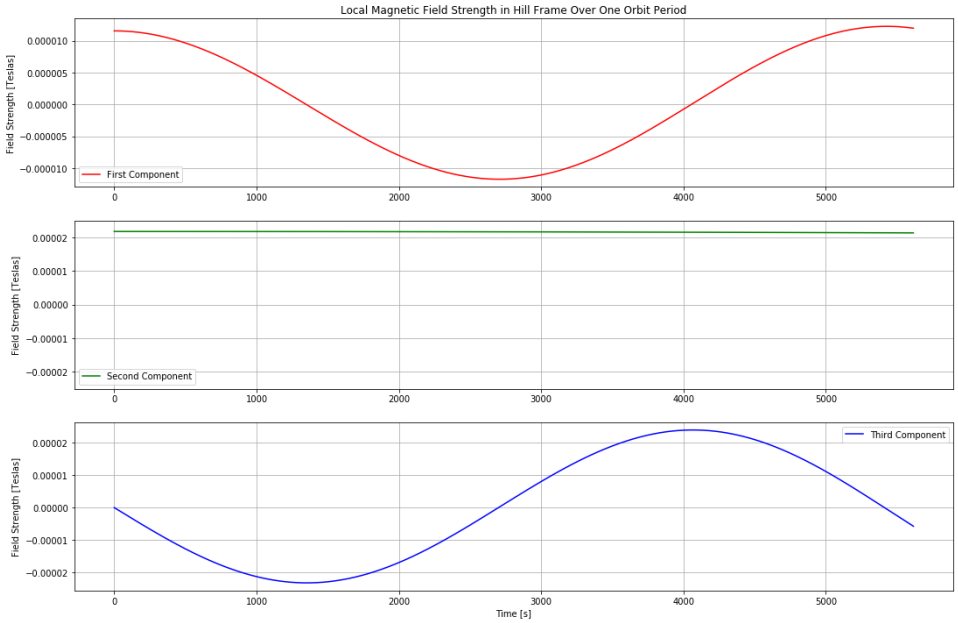

Figure 2: Components of magnetic field in Hill frame (in units of Teslas) shown over one orbit.

The magnetic field in the spacecraft body frame can be obtained by rotating the Hill frame magnetic field appropriately. Specifically,

$$^{B}\mathbf{b} = \left[BN\right]\left[NH\right]^{H}\mathbf{b}$$

Note that [BN] is obtained via the MRP mapping to DCM, and that [NH] is obtained via $[HN]^T$, where [HN] is given through the usual 3-1-3 Euler angle DCM, where the Euler angles relate the geomagnetic frame to the Hill frame. Because the spacecraft is both rotating relative to the inertial frame and the earth is rotating, both of the DCMs need to be updated at each time step. The magnetic field components over 100s in the body frame are shown in Figure 3.

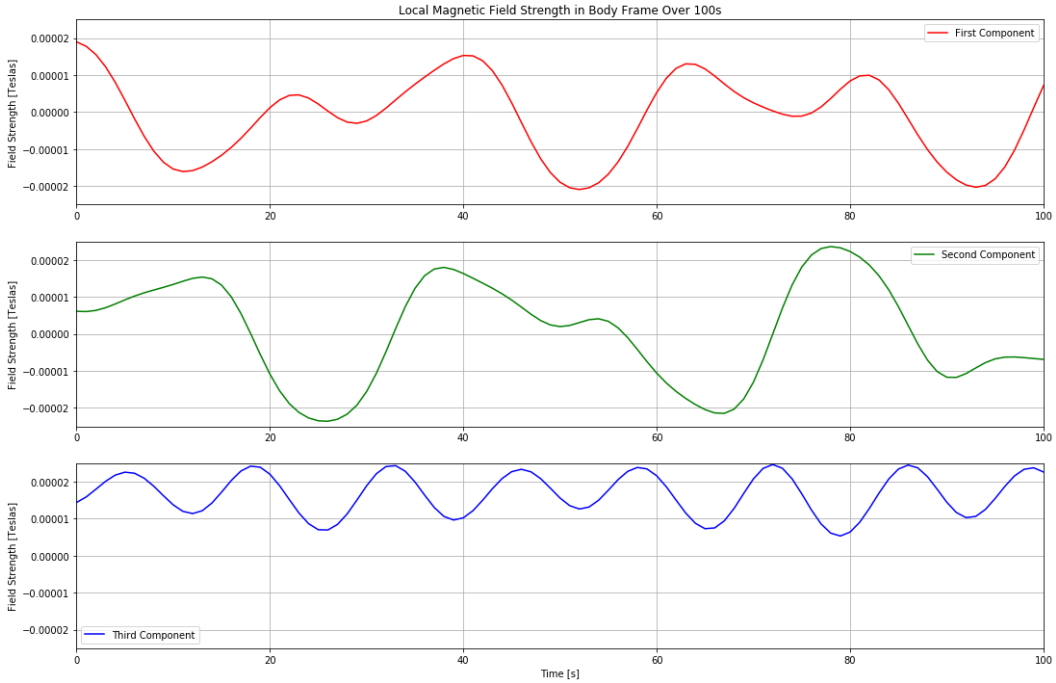

Figure 3: Components of magnetic field in spacecraft body frame (in units of Teslas) shown over 100s

C. Control Law Implementation

It is worth spending a moment here discussing the structure of the code and how the control laws are included in the integration. Both control laws are implemented as separate functions. Their arguments are $(t, \mathbf{X}, \text{parameters})$, where t is the current time, $\mathbf{X}$ is the state, and the parameters are various parameters that the control law needs, such as constants, etc. The control law function calls the magnetic field function and rotates the resulting magnetic field vector into the body frame. It then outputs a control torque $\mathbf{u}$.

The first order DE describing the system (this is what gets passed to the RK4 function) calls a function called `controlLaw(t,X)`. By defining a **`lambda`** function, we can ensure that the control law function only has arguments t and $\mathbf{X}$. This function can be specified when starting the integration. This way, there is no code redundancy, and arbitrary control laws can be used with the same dynamics function.

Note that this method requires four calls to the control law function at each time step due to the nature of RK4 integration. Although this results in loss of speed, the code is kept general and easy to experiment with.

D. Modulating B-Dot Control

The code was implemented straightforwardly as specified. Note that the built-in gain k_w can be multiplied by a manually specified gain K to make the control law more or less aggressive. Code is shown below.

```
def modulatingBDot(t, X, mMax, info):

    # Unpack
    omegaE = info[0] # [deg/s]
    RAAN = info[1] # [deg]
    inc = info[2] # [deg]
    theta0 = info[3] # [deg]
    thetaDot = info[4] # [deg/s]
    orbitRad = info[5] # [km]
    inertiaMat = info[6]
    beta0 = info[7] # [deg]
    gamma = info[8] # [deg]
    M = info[9] # [Tkm^3]

    bHill = magField([M, orbitRad, gamma, omegaE], [RAAN, inc, theta0+thetaDot*t], thetaDot,
    ↪  beta0, t)

    BN = DCMMRP(X[0:3])
    NH = np.transpose(DCM313(RAAN, inc, theta0+thetaDot*t))

    # Transform hill frame vector to body frame
    bBody = BN@NH@bHill

    # Compute gain
    i = np.deg2rad(inc) # [rads]
    gam = np.deg2rad(gamma) # [rads]
    beta = np.deg2rad(beta0 + omegaE*t) # [rads]
    xi_m = np.arccos(np.cos(i)*np.cos(gam) + np.sin(i)*np.sin(gam)*np.cos(np.deg2rad(RAAN) -
    ↪  beta)) # [rads]

    IMin = np.amin(inertiaMat.diagonal())
    kw = 2*np.deg2rad(thetaDot)*(1+np.sin(xi_m))*IMin

    # Compute command dipole
    bBodyHat = bBody/np.linalg.norm(bBody)
    omegaVec = X[3:6]

    mVec = -kw/np.linalg.norm(bBody)*np.cross(bBodyHat, (np.eye(3) - np.outer(bBodyHat,
    ↪  bBodyHat))@omegaVec)

    # Check for saturation (componentwise)
    for i in range(0,3):
        if np.abs(mVec[i]) > mMax:
            mVec[i] = np.sign(mVec[i])*mMax
```

```
    # Compute control torque:
    uVec= np.cross(mVec, bBody)

    # Return
    return([uVec,mVec])
```

E. Bang-Bang B-Dot Control

This control law is on-off control. The actuators are either on with maximum authority (to within a sign) or off. In this case, the command dipole depends on the body relative derivative of the magnetic field vector.

The easiest way to implement this is to use a sufficiently small time step and a Since the structure of the code is encapsulated, the control law function does not have access to the state at the previous time step. To avoid having to mangle the structure of the code, an analytical expression for $\mathbf{b}'$ is used for this control law.

By the transport theorem, we have that:

$$\mathbf{b}' =^B \frac{d}{dt}(\mathbf{b}) =^H \frac{d}{dt}(\mathbf{b}) + \omega_{\mathbf{H/B}} \times \mathbf{b}$$
$$\text{where } \omega_{\mathbf{H/B}} = \omega_{\mathbf{H/N}} + \omega_{\mathbf{N/B}} = \omega_{\mathbf{H/N}} - \omega_{\mathbf{B/N}}$$

It is important that all vector quantities are expressed in the same frame for ease of computation. The only term requiring non-trivial effort here is ${}^H\frac{d}{dt}(\mathbf{b})$. Carrying out the computation, we get:

$${}^H\frac{d}{dt}(\mathbf{b}) = \begin{pmatrix} -\sin(\dot{\theta}t - \eta_m)(\dot{\theta}\ddot{\theta} - \eta_m')\sin(\xi_m) + \cos(\dot{\theta}t - \eta_m)\cos(\xi_m)\xi_m' \\ -\sin(\xi_m)\xi_m' \\ -2\cos(\dot{\theta}t - \eta_m)(\dot{\theta}\ddot{\theta} - \eta_m')\sin(\xi_m) - 2\sin(\dot{\theta} - \eta_m)\cos(\xi_m)\xi_m' \end{pmatrix}$$

where

$$\xi_m' = \frac{-\sin(i)\sin(\gamma_m)\sin(\Omega - \beta_m)(-\omega_E)}{\sqrt{1 - (\cos(i)\cos(\gamma_m) + \sin(i)\sin(\gamma_m)\cos(\Omega - \beta_m))^2}}$$
$$\eta_m' = \frac{\sin(\xi_m)\sin(\gamma_m)\cos(\Omega - \beta_m)(-\omega_E) - \sin(\gamma_m)\sin(\Omega - \beta_m)\cos(\xi_m)\xi_m'}{\sqrt{1 - \left(\frac{\sin(\gamma_m)\sin(\Omega - \beta_m)}{\sin(\xi_m)}\right)^2 \sin^2(\xi_m)}}$$

Though cumbersome, this formulation is easy to compute. The implementation is given below.

```
def bangBangBDot(t, X, mMax, info):

    # Unpack
    omegaE = info[0] # [deg/s]
    RAAN = info[1] # [deg]
    inc = info[2] # [deg]
    theta0 = info[3] # [deg]
    thetaDot = info[4] # [deg/s]
    orbitRad = info[5] # [km]
    inertiaMat = info[6]
    beta0 = info[7] # [deg]
    gamma = info[8] # [deg]
    M = info[9] # [Tkm^3]

    # Rotation matrices
    BN = DCMMRP(X[0:3])
    NH = np.transpose(DCM313(RAAN, inc, theta_t0+thetaDot*t))

    # Magnetic field vector calculation
    bHill = magField([M, orbitRad, gamma, omegaE], [RAAN, inc, theta0+thetaDot*t], thetaDot,
    ↪  beta0, t)
    bBody = BN@NH@bHill

    i = np.deg2rad(inc)
```

```
gam = np.deg2rad(gamma)
Omega = np.deg2rad(RAAN)
omega = np.deg2rad(omegaE)
beta = np.deg2rad(beta0 + omegaE*t)

# Compute time derivative of b as seen by H frame
xi_m = np.arccos(np.cos(i)*np.cos(gam) + np.sin(i)*np.sin(gam)*np.cos(Omega-beta))
eta_m = np.arcsin(np.sin(gam)*np.sin(Omega-beta)/np.sin(xi_m))

xi_mPrime =
↪  -(-np.sin(i)*np.sin(gam)*np.sin(Omega-beta)*(-omega))/np.sqrt(1-(np.cos(i)*np.cos(gam) +
↪  np.sin(i)*np.sin(gam)*np.cos(Omega-beta))**2)

eta_mPrime = 1/np.sqrt(1-(np.sin(gam)*np.sin(Omega-beta)/np.sin(xi_m))**2) *
↪  ((np.sin(xi_m)*np.sin(gam)*np.cos(Omega-beta)*(-omega) -
↪  np.sin(gam)*np.sin(Omega-beta)*np.cos(xi_m)*xi_mPrime)/(np.sin(xi_m)**2))

bPrimeRelHBody = (M/orbitRad**3)*BN@NH@np.asarray([
    -np.sin(thetaDot*t-eta_m)*(-eta_mPrime)*np.sin(xi_m) +
    ↪  np.cos(thetaDot*t-eta_m)*np.cos(xi_m)*xi_mPrime,
    -np.sin(xi_m)*xi_mPrime,
    -2*np.cos(thetaDot*t-eta_m)*(-eta_mPrime)*np.sin(xi_m) -
    ↪  2*np.sin(thetaDot*t-eta_m)*np.cos(xi_m)*xi_mPrime
])

# Compute cross product term in transport thm (omega_H/B x b)
omega_HNBody = BN@NH@np.asarray([0, 0, np.deg2rad(thetaDot)])
omega_BNBody = X[3:6]

omega_HBBody = omega_HNBody - omega_BNBody

# Compute b'
bPrimeRelBBody = bPrimeRelHBody + np.cross(omega_HBBody, bBody)

# Compute m
mBody = -1*mMax*np.sign(bPrimeRelBBody)

# Compute control torque u
uBody = np.cross(mBody, bBody)

# Return
return([uBody,mBody])
```

IV. Results

A. Modulating Control Results

The modulating B-Dot control law was used to control the spacecraft over 3 orbit periods, with all three sets of initial conditions given (inclination angle held at $45°$. Note that the 0th IC is the one given in the mission overview. The spacecraft attitude state (MRPs and ω), control torque vector $\mathbf{u}$, and command dipole moment $\mathbf{m}$, over time, are shown in the figures below. Since this is a regulation problem, the control errors are equal to the components of ω. Furthermore, the maximum dipole was kept at the default value of 3 Am^2. Figures 4 and 5 show performance for the zeroth (mission overview) set of ICs.

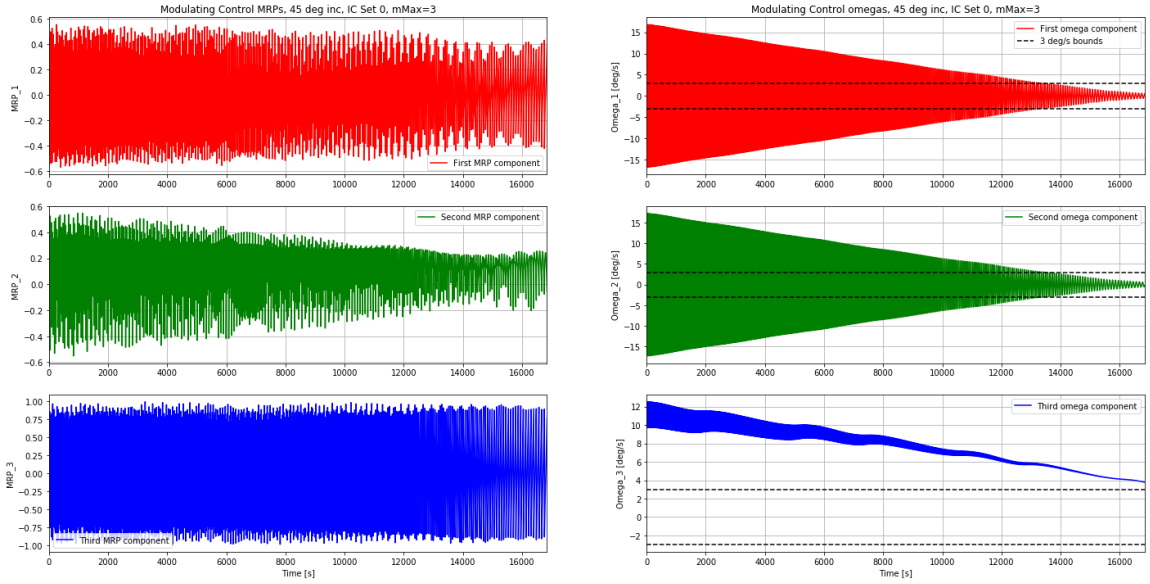

Figure 4: Modulating control: MRP components and ω components using mission overview ICs and $45°$ inclination angle.

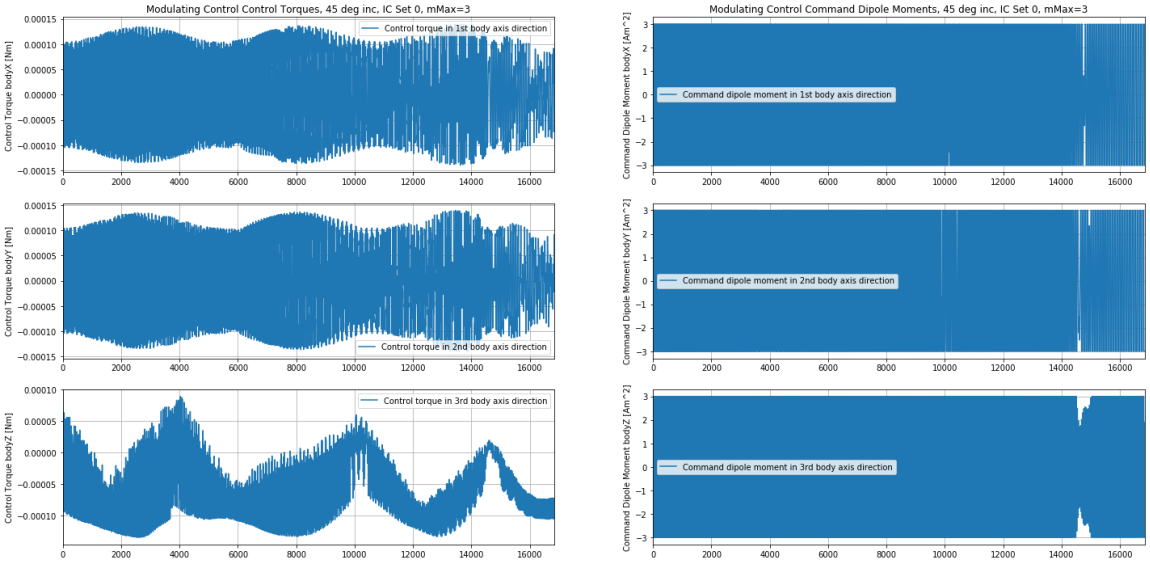

Figure 5: Modulating control: control torque and command dipole components using mission overview ICs and $45°$ inclination angle.

Figures 6 and 7 below show performance for the first set of ICs given in Part 4.

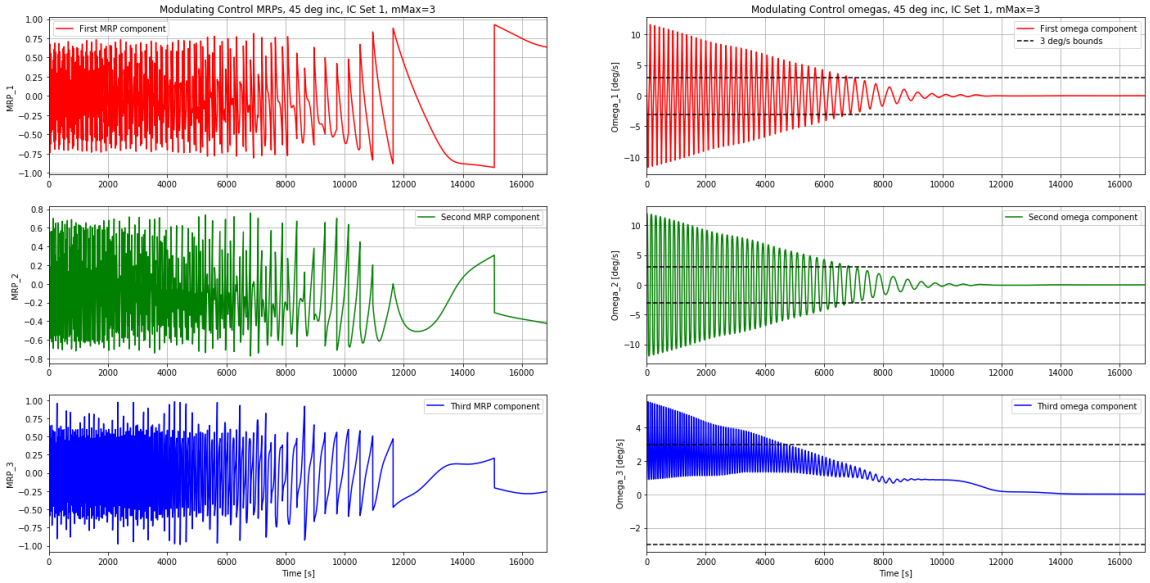

Figure 6: Modulating control: MRP components and ω components using ICs set 1 and 45° inclination angle.

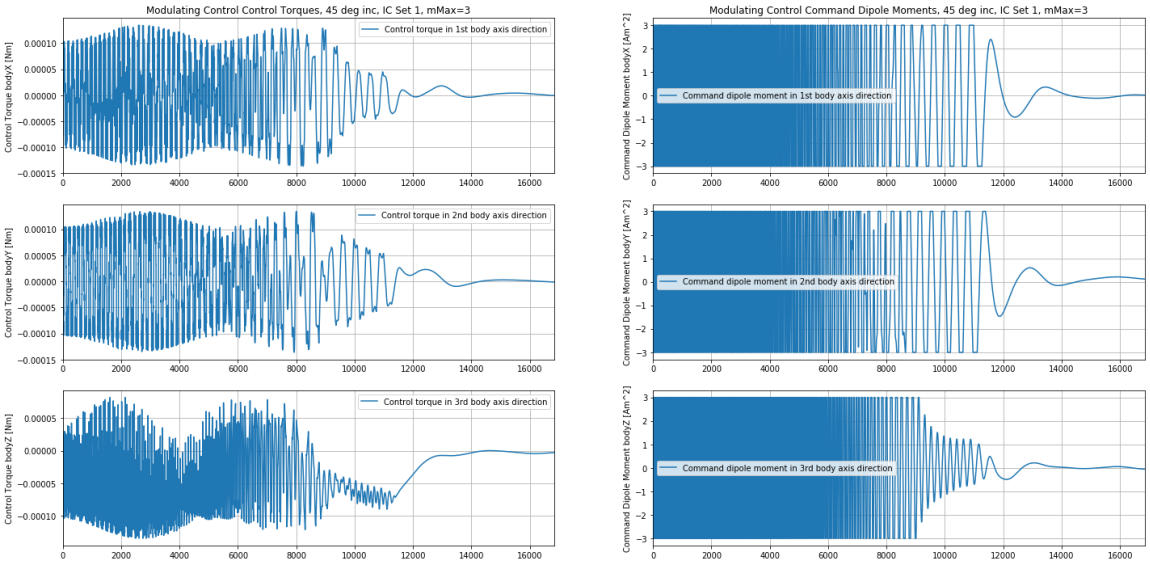

Figure 7: Modulating control: control torque and command dipole components using ICs set 1 and 45° inclination angle.

Figures 8 and 9 below show performance for the second set of ICs given in Part 4.

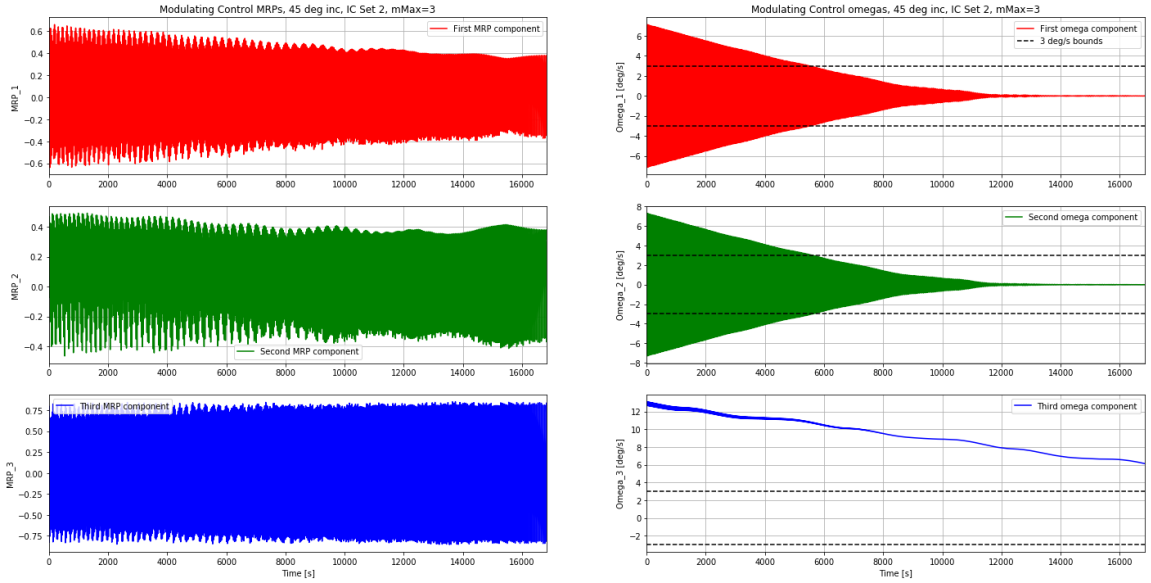

Figure 8: Modulating control: MRP components and ω components using ICs set 2 and 45° inclination angle.

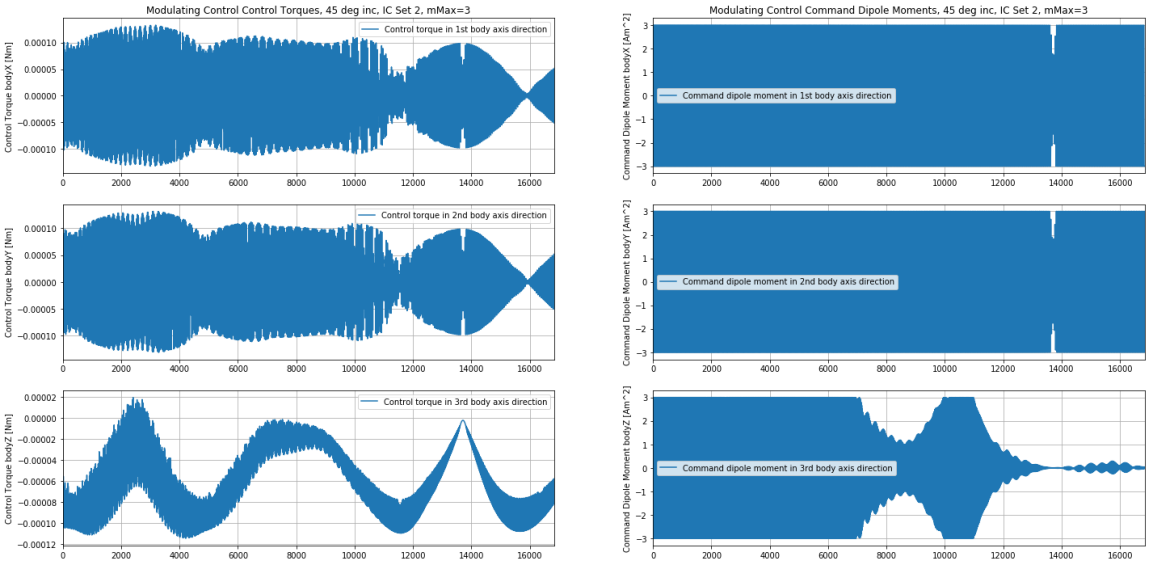

Figure 9: Modulating control: control torque and command dipole components using ICs set 2 and 45° inclination angle.

1. Discussion

We can clearly see that the control performance has a huge dependence on initial conditions. IC set 1 results in rapid convergence, while neither IC set 0 nor IC set 2 achieve our design goals within three orbit periods.

We can also see that the actuators saturate very often. This tells us that bang-bang control should perform largely similarly.

B. Bang-Bang Control Results

The same three scenarios as above were run again, but using the bang-bang control law. Figures 10 and 11 show performance for the zeroth (mission overview) set of ICs.

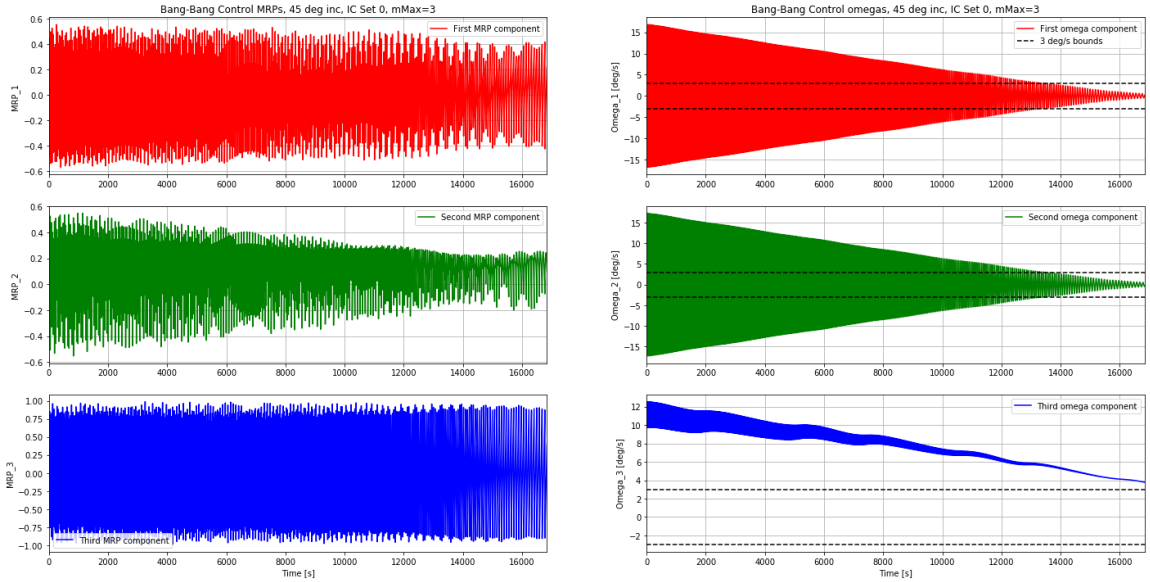

Figure 10: Bang-bang control: MRP components and ω components using mission overview ICs and 45° inclination angle.

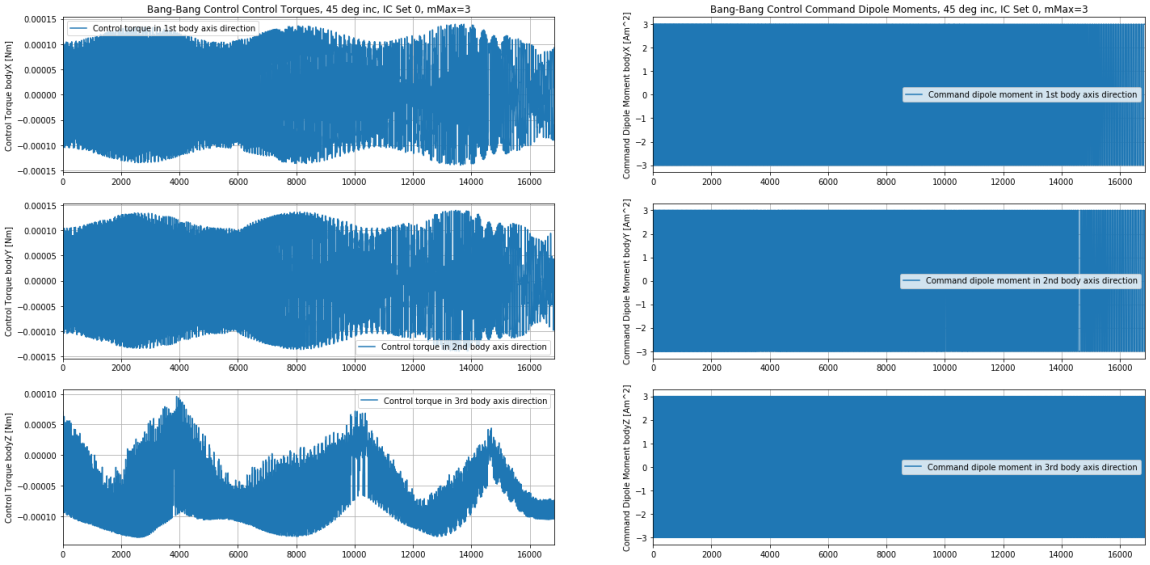

Figure 11: Bang-bang control: control torque and command dipole components using mission overview ICs and 45° inclination angle.

Figures 12 and 13 below show performance for the first set of ICs given in Part 4.

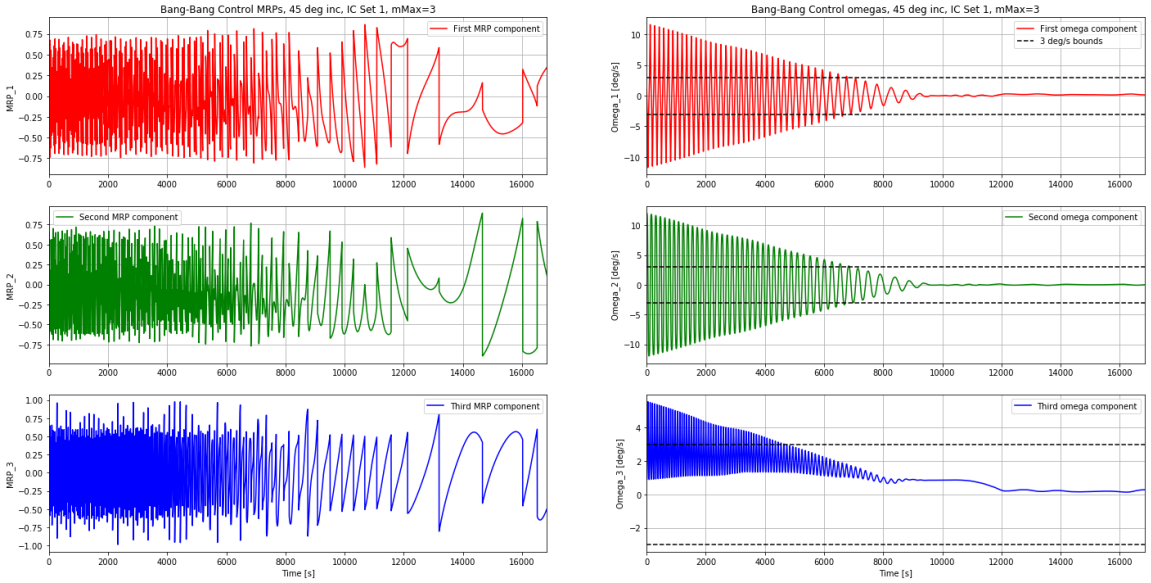

Figure 12: Bang-bang control: MRP components and ω components using ICs set 1 and 45° inclination angle.

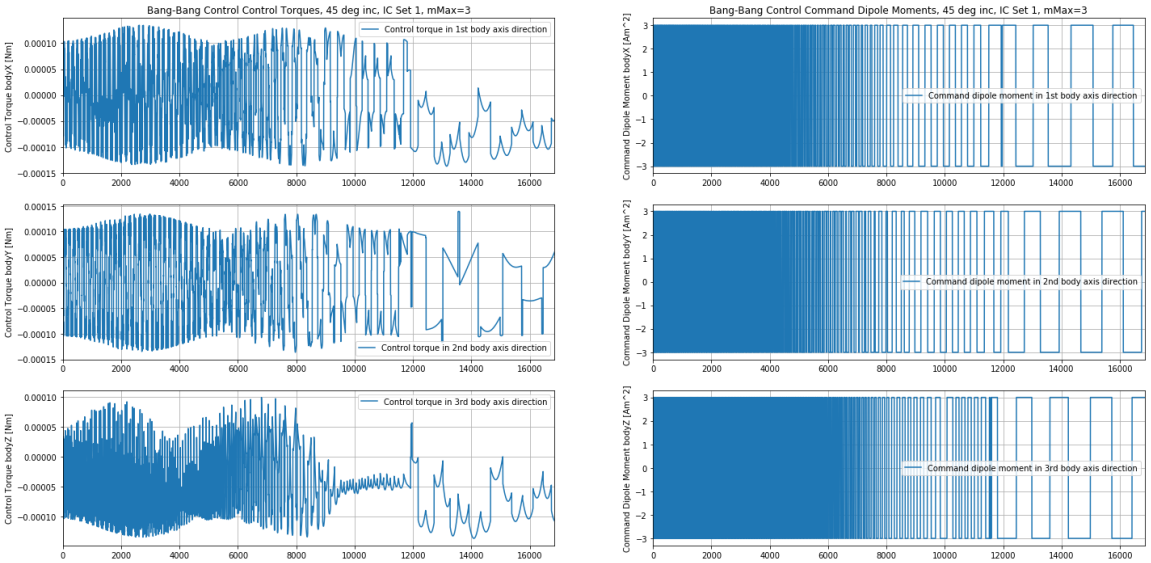

Figure 13: Bang-bang control: control torque and command dipole components using ICs set 1 and 45° inclination angle.

Figures 14 and 15 below show performance for the second set of ICs given in Part 4.

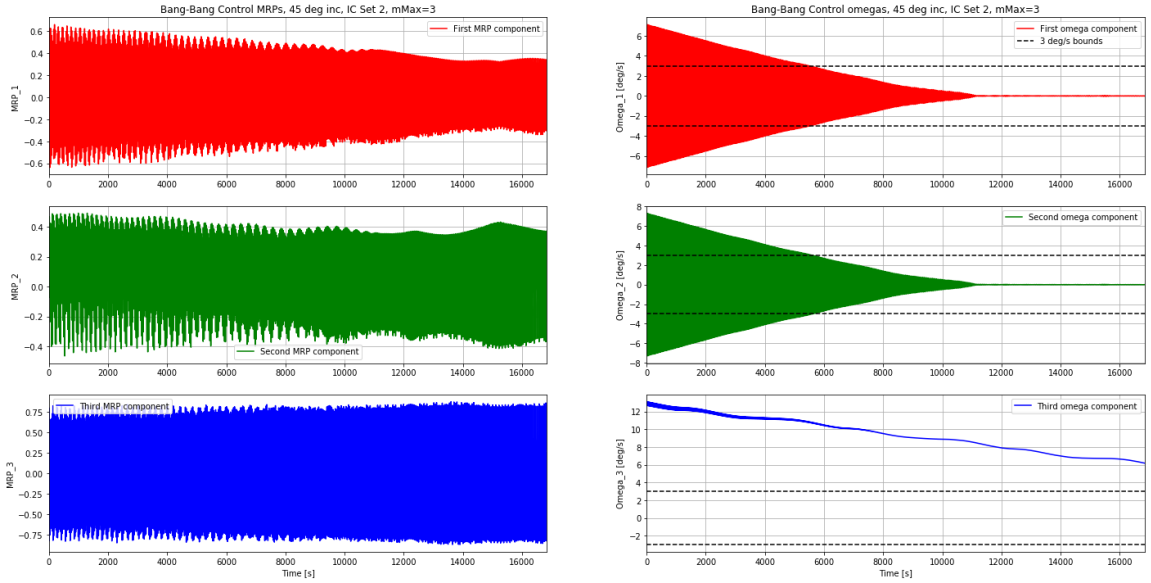

Figure 14: Bang-bang control: MRP components and ω components using ICs set 2 and 45° inclination angle.

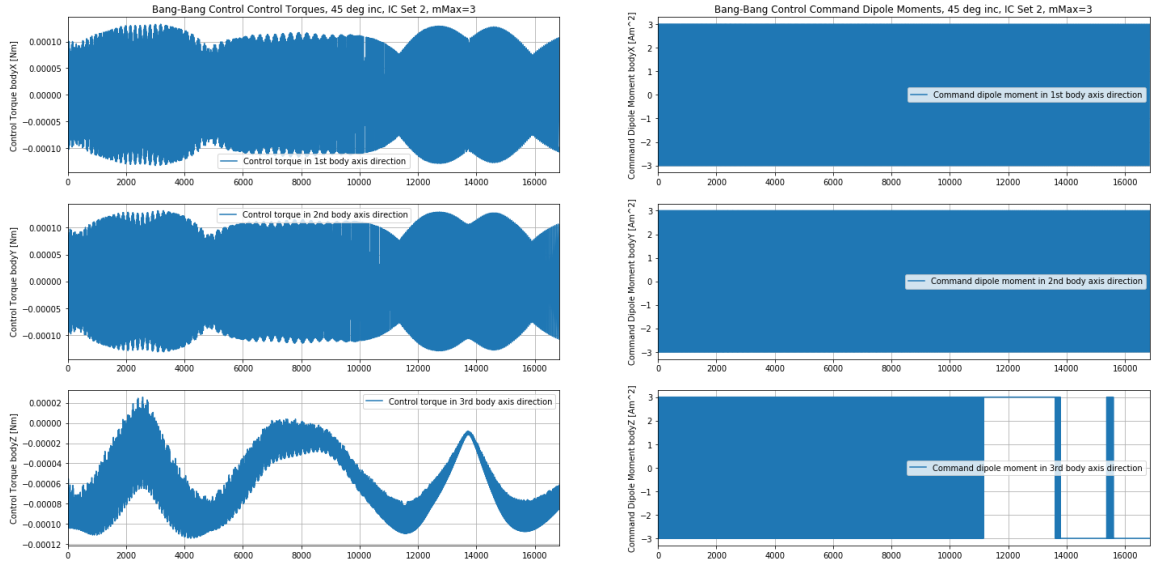

Figure 15: Bang-bang control: control torque and command dipole components using ICs set 2 and 45° inclination angle.

1. Discussion

Judging from a glance, these results appear to be very close to the results obtained using modulating control. One noticeable difference is in IC set 1, where the MRP components are much more erratic than in the modulating control case. Also in this case, we can see a conspicuous lack of smooth convergence to zero in the ω_3 component. It seems to have some steady state error, while the modulating case does not.

We can see that the dipole moments switch between maximally positive and negative, which is what it should be doing.

Drawing attention to the last case (IC set 2), notice that the control torques using modulation control become smooth towards the end of the run. However, with bang-bang control, there are large jumps in the control torque components. This is because the modulating control law is able to apply small torques by producing small command dipoles. Bang-bang control is unable to do this, and so we see spikes in the torques produced. This explains the lack of smooth

convergence in the ω components: even a small error will produce large torque spikes that will produce overshoot/undershoot.

C. Effect of Varying Gain

We can easily add a custom gain to the modulating control law by multiplying the computed $\mathbf{m}$ by a constant scalar K before we check for actuator saturation. However, since the actuators seem to be saturated most of the time anyway, this should not make a huge difference. Similarly, in the case of bang-bang control, we cannot specify a larger gain. We can, however, use a gain smaller than 1. This should result in slower performance across the board, but also less oscillation and less steady state error.

To test this, both control laws were used on IC set 1 again, but with a 0.5 gain multiplier. The results are shown in Figures 16 through 19.

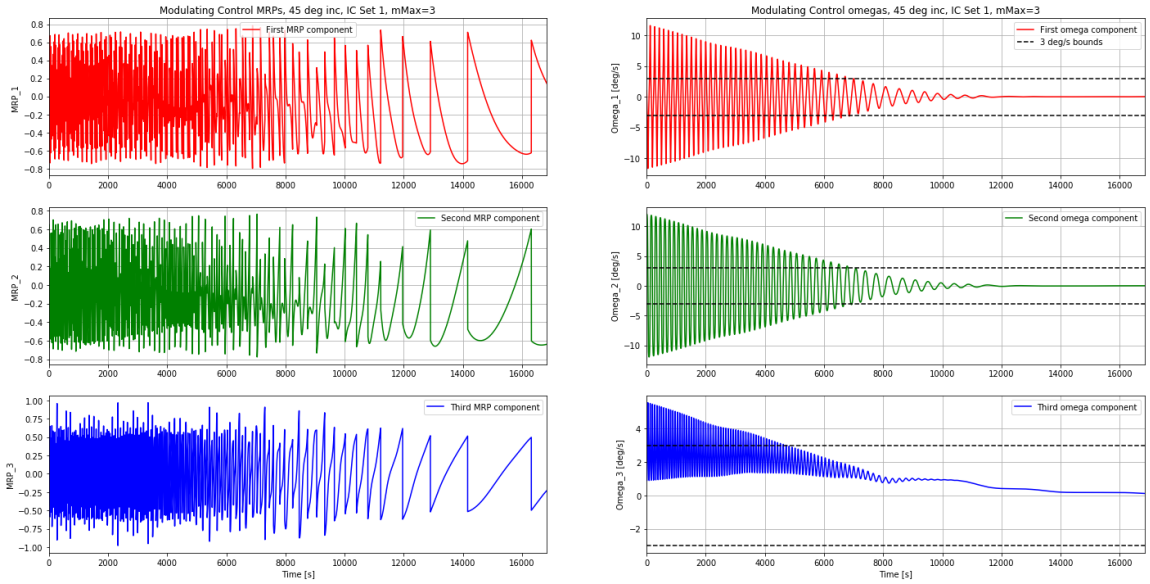

Figure 16: Modulating control: MRP components and ω components using ICs set 1 and 45° inclination angle, with 0.5 gain.

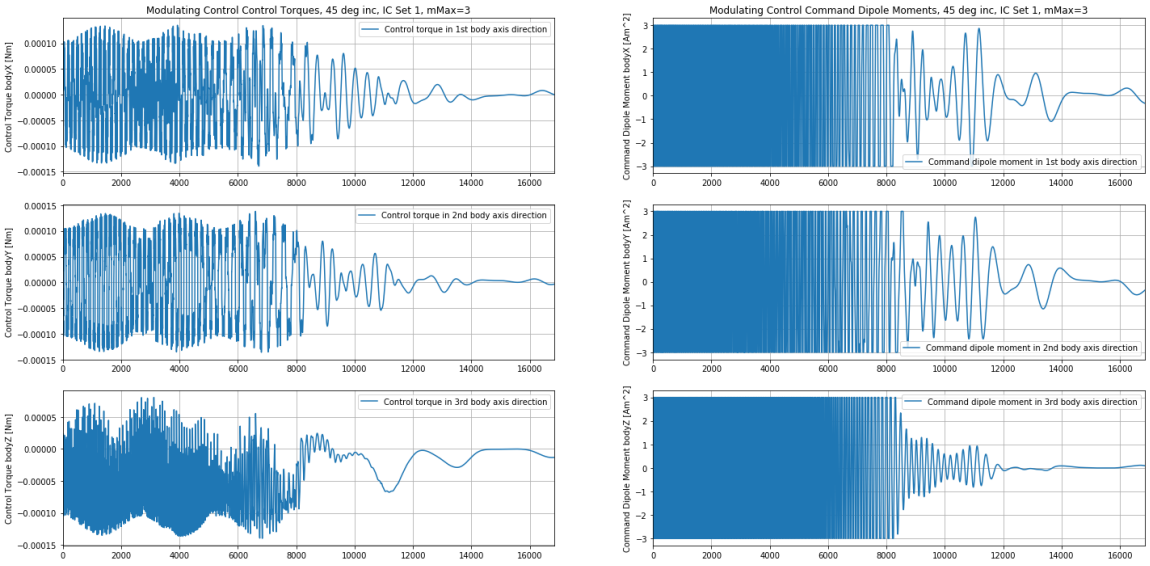

Figure 17: Modulating control: control torque and command dipole components using ICs set 1 and 45° inclination angle, with 0.5 gain.

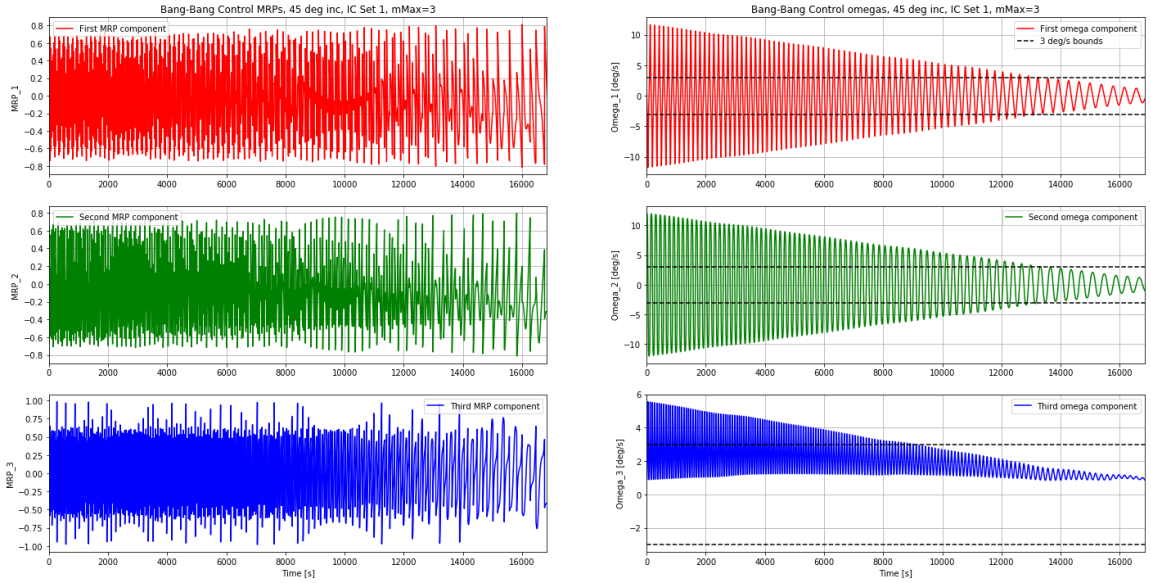

Figure 18: Bang-bang control: MRP components and ω components using ICs set 1 and 45° inclination angle, with 0.5 gain.

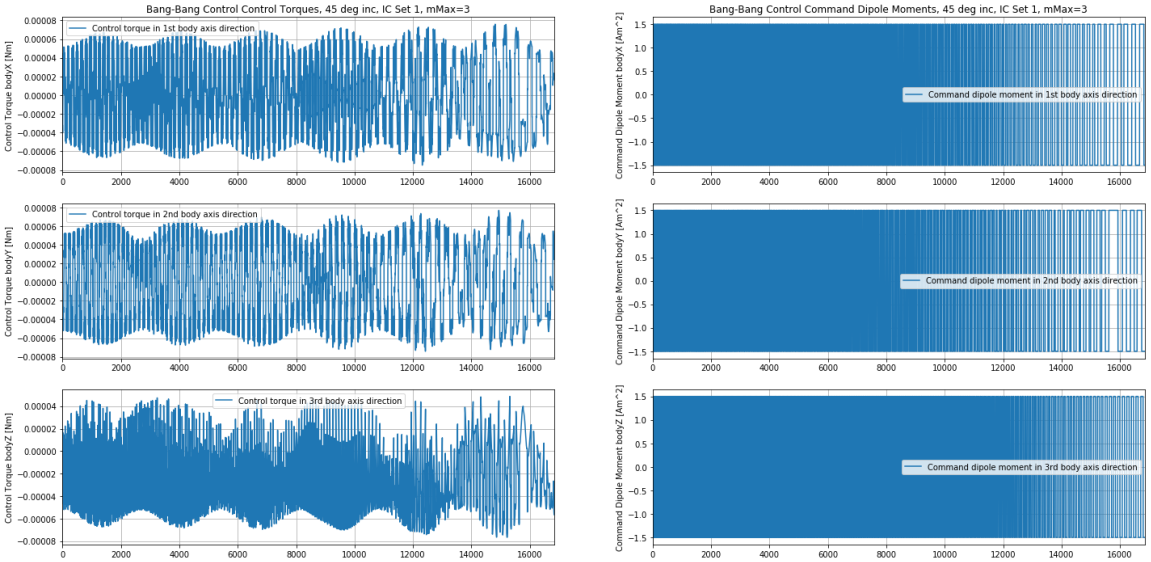

Figure 19: Bang-bang control: control torque and command dipole components using ICs set 1 and 45° inclination angle, with 0.5 gain.

As expected, both control laws perform noticeably slower since they are less aggressive. Examining the bang-bang control law performance more closely, we can see that the erraticness around zero error seems to be reduced. Furthermore, the bang-bang control torques seem to be smoother.

D. Effect of Orbit Inclinations: $i = 15^\circ$

Here, we will look at two different orbit inclinations (with both modulating and bang-bang control laws). In all the following cases, IC set 0 (mission overview ICs) are used. No extra gain multiplier is added.

1. Modulating Control

The larger inclination angle case results in better performance for ω_1 and ω_2 and slightly better performance in ω_3. When compared with the $i = 45^\circ$ case above with the corresponding ICs, we can see that the behavior is qualitatively

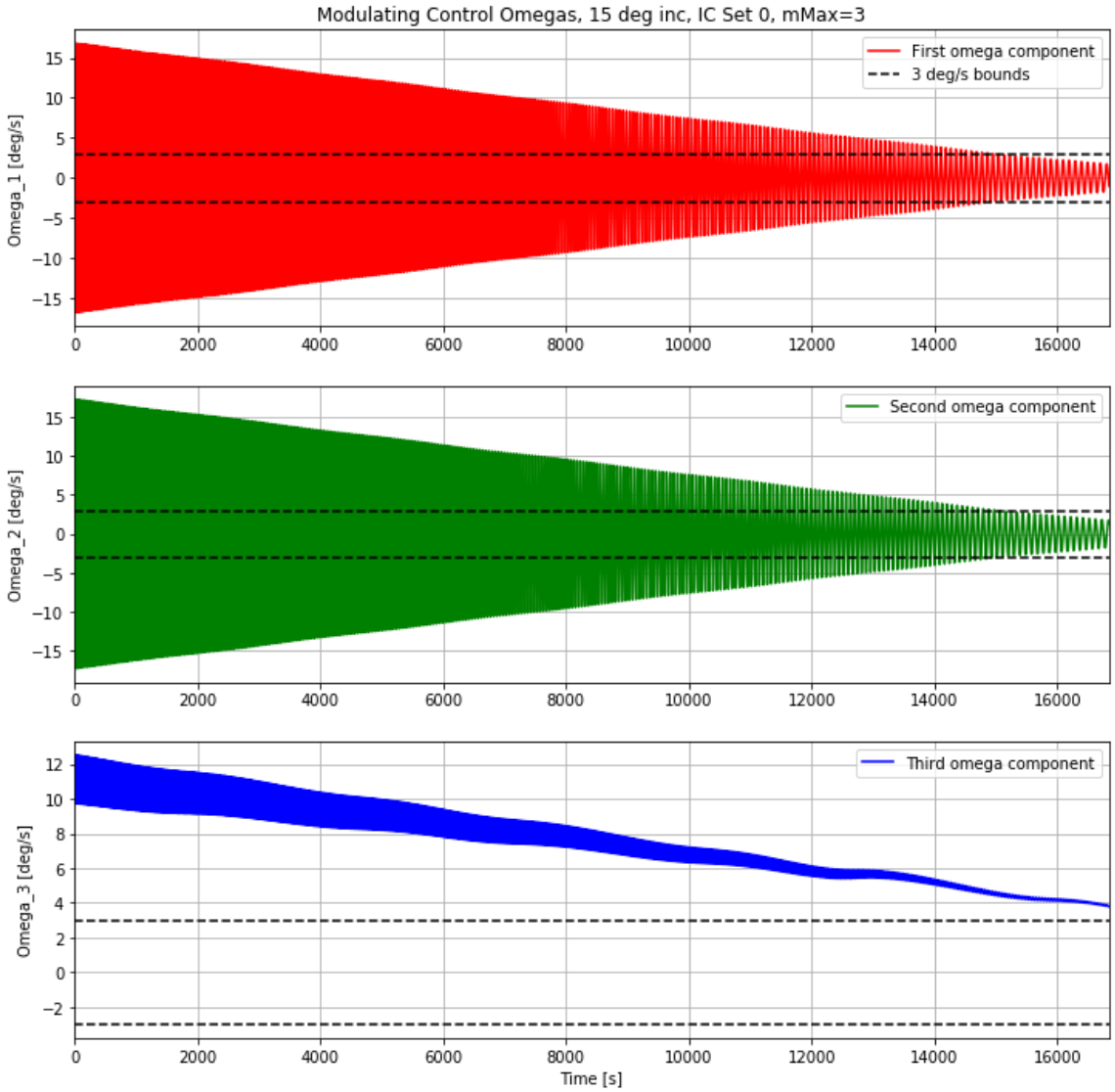

Figure 20: Modulating control: ω components using mission overview ICs and 15° inclination angle.

similar, although it is slower or faster depending on the inclination. We can infer that given the same set of ICs, changes in orbit inclination can result in significantly different behavior.

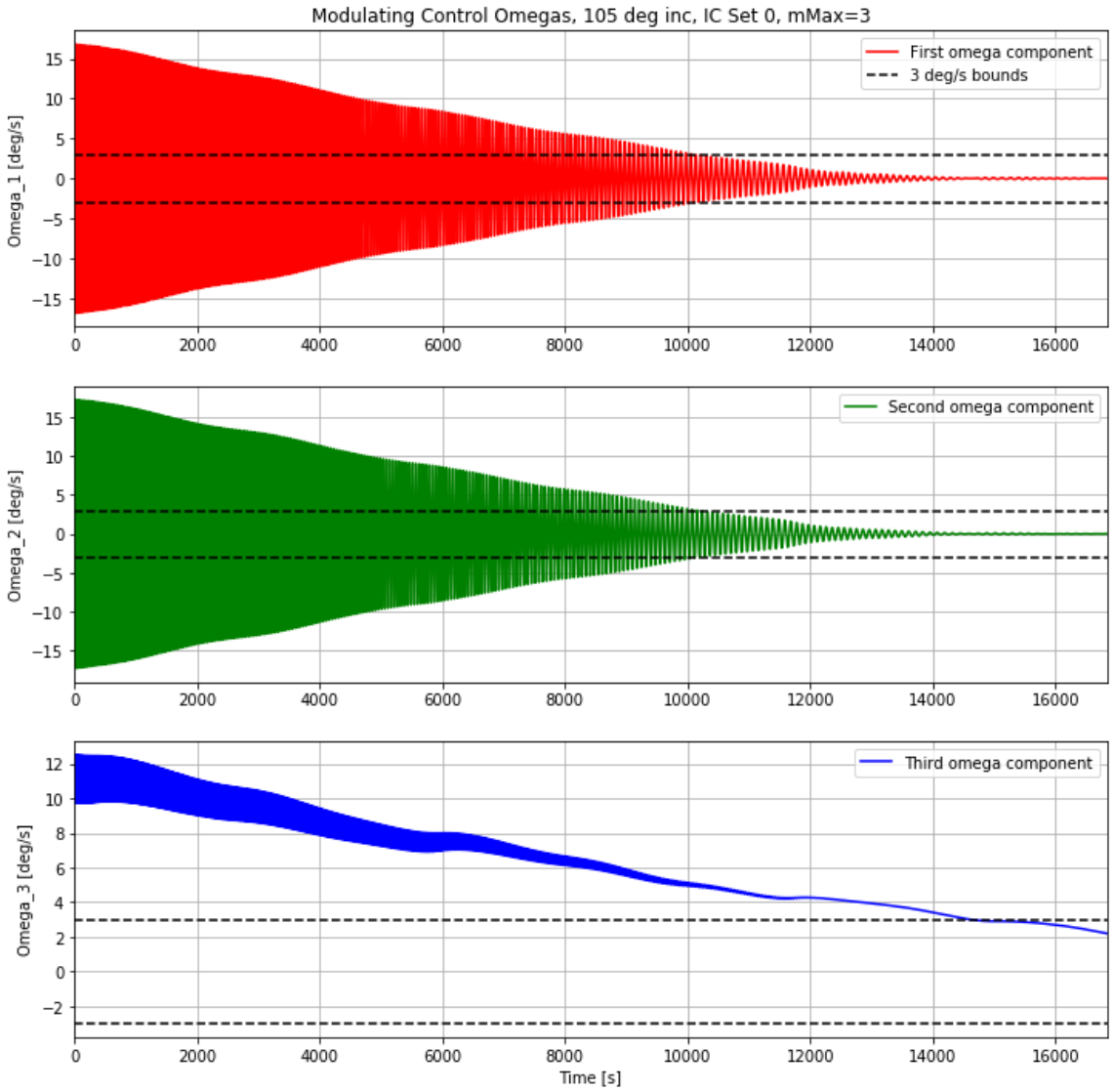

Figure 21: Modulating control: ω components using mission overview ICs and 105° inclination angle.

2. Bang-Bang Control

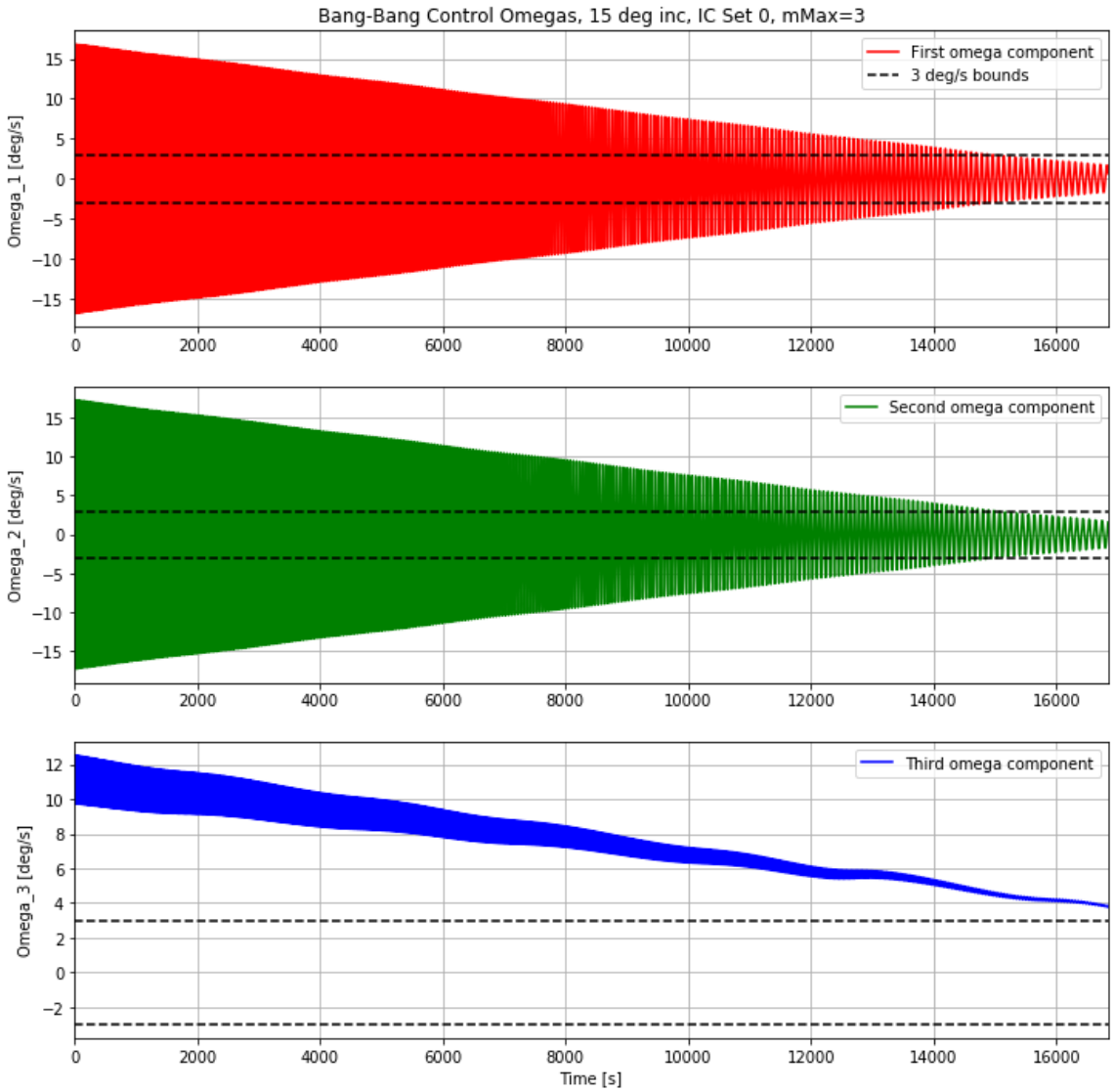

Figure 22: Bang-bang control: ω components using mission overview ICs and 15° inclination angle.

The bang-bang control law works almost identically to the modulating control law here.

Based on these results, it would be reasonable to say that these control laws do not vary significantly in performance due to inclination changes. At the very least, it can be said that initial conditions have much more of an impact than orbit inclination.

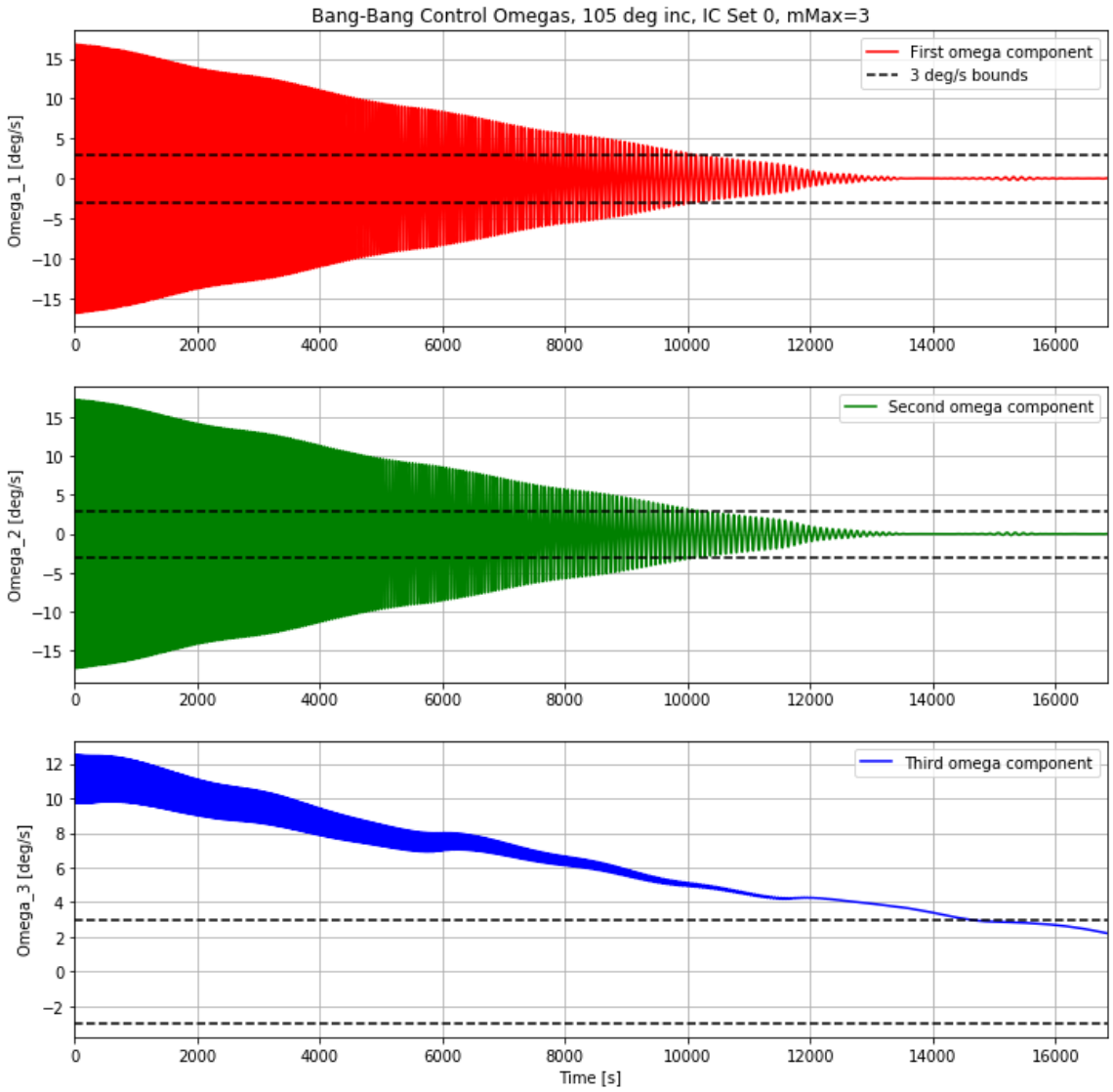

Figure 23: Bang-bang control: ω components using mission overview ICs and 105° inclination angle.

E. Minimum Torque Rod Sizing

Here, we try to find the minimum torque rod actuation capability (dipole moment magnitude) such that (using IC set 0) all ω components are driven to less than $1^\circ/s$ within three orbit periods. We do this for both modulating and bang-bang control. Since both control laws are fairly close to the new goal already, we can do a quick trial and error (effectively performing a manual bisection method for rootfinding) to find the required values.

1. Modulating Control Minimum Size

The minimum size for the modulating control law was found to be $\mathbf{m}_{\text{Max,Min}} = 4.205$. With this value, the results are shown in Figure 24.

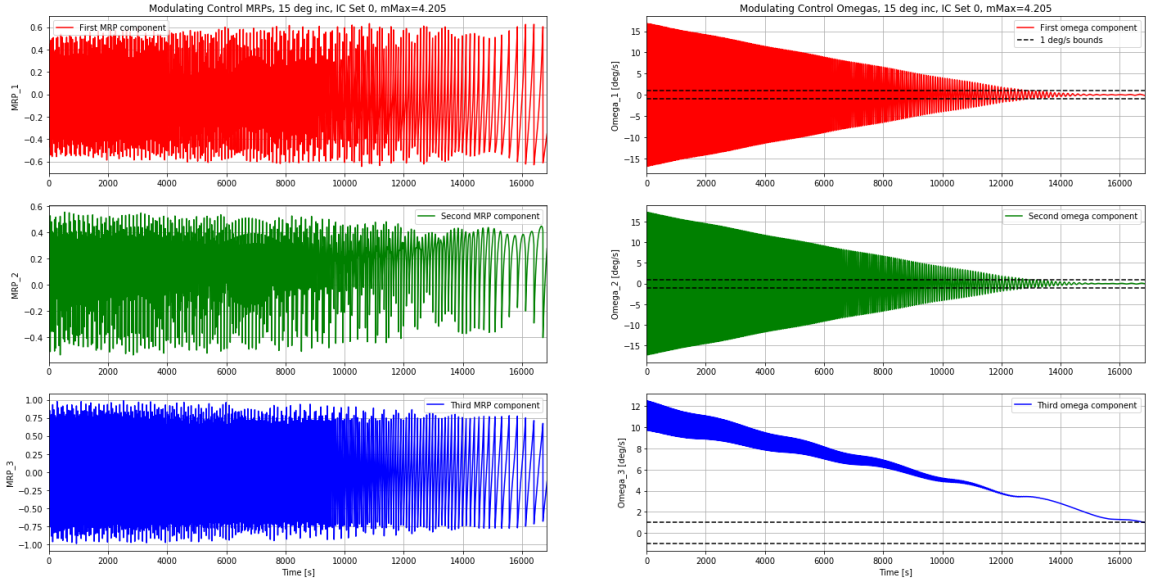

Figure 24: Modulating control: MRP and ω components using mission overview ICs and 15° inclination angle, with $\mathbf{m}_{\text{Max,Min}} = 4.205$.

2. Bang-Bang Control Minimum Size

The minimum size of the torque rods for the bang-bang case was found to be $\mathbf{m}_{\text{Max,Min}} = 4.05$. Using this value, the results obtained are shown in Figure 25.

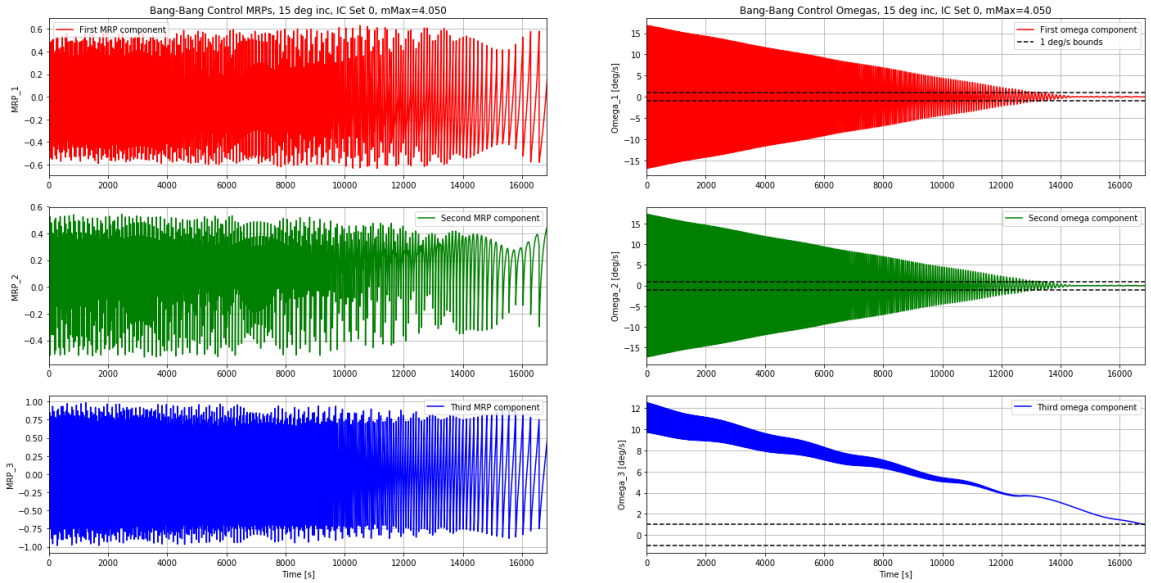

Figure 25: Bang-bang control: MRP and ω components using mission overview ICs and 15° inclination angle, with $\mathbf{m_{Max,Min}} = 4.050$.

F. Monte-Carlo Analysis

Here, we test 25 randomly selected ICs with $|\omega_i|$ between 10 and 16 $^\circ/s$ using modulating control. The orbit inclination is held at 45°, and the torque rid size is increased to $4\ Am^2$.

1. Modulating Control

Results for the above scenario utilizing modulating control are shown in Figure 26.

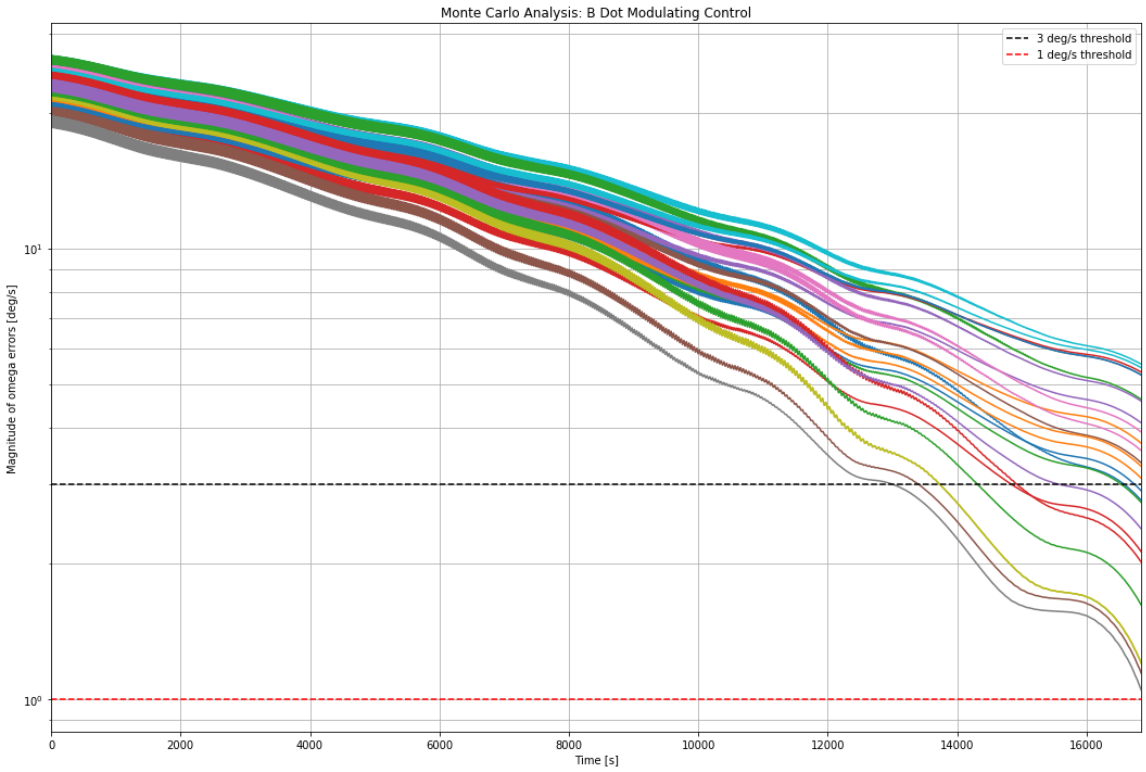

Figure 26: Modulating control: norm of control error over 3 orbits of all 25 runs plotted on log scale.

The average settling time over the 25 runs was found to be $14599\ s$. This was found by finding the time to cross the $3^\circ/s$ threshold for each run and averaging those times.

2. Bang-Bang Control

The same was repeated using bang-bang control instead. These results are shown in Figure 27.
With bang-bang control, the average settling time over the runs was found to be $15834\ s$.

3. Discussion

We can see from the results (both graphically and using the average threshold settling time) that the modulating control law is slightly faster than the bang-bang method. We also see that, in both cases, the settling time varies significantly depending on the initial conditions.

Within the time frame of 3 orbit periods, the modulating control law clearly shows lower error. However, if restricted to this time frame, it is unclear what the long term behavior of each control law looks like. This makes the steady state error hard to judge at this point. Based on what we know about these control laws, the modulating control law should be able to achieve zero steady state error (asymptotic convergence), while the bang-bang control law may not be able to.

To evaluate these predictions, the Monte-Carlo analysis was repeated, but over 5 orbit periods instead of 3. The results are shown in Figures 28 and 29.
Now, we can clearly see that the modulating control law does indeed seem to show asymptotic convergence to zero steady state error. The bang-bang control law appears to show errors that approach zero, but oscillate within some finite region around it (appears to show Lagrange stability).

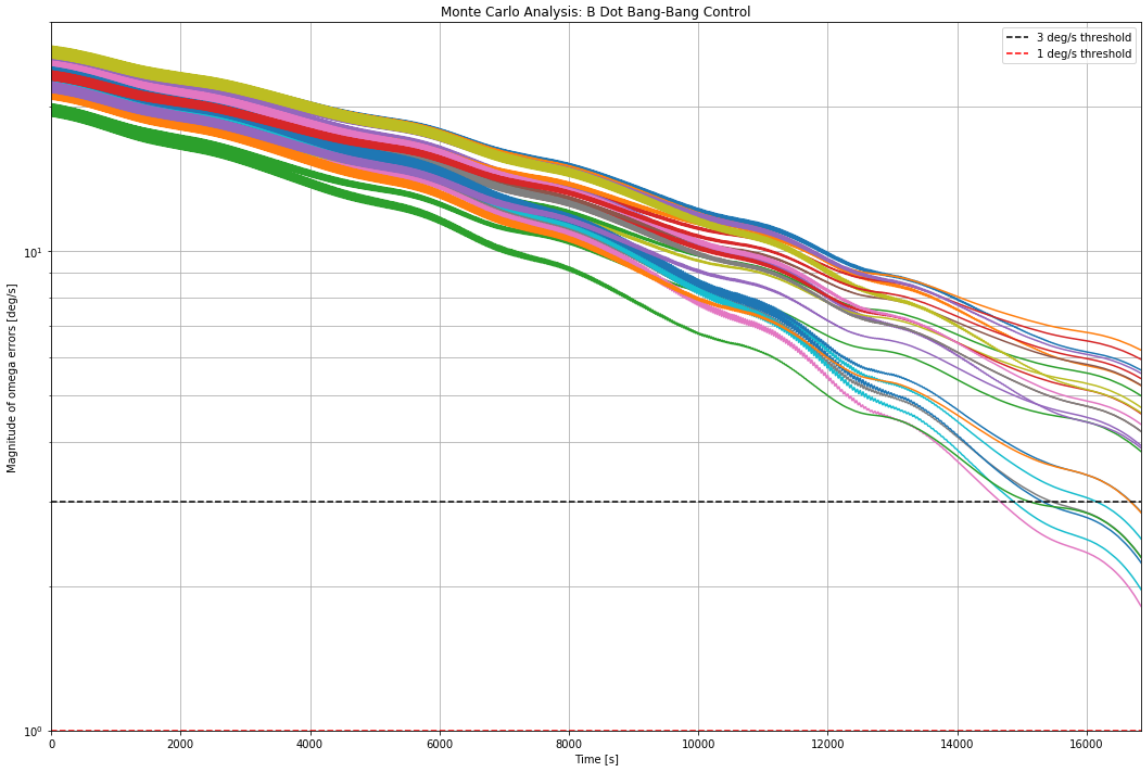

Figure 27: Bang-bang control: norm of control error over 3 orbits of all 25 runs plotted on log scale.

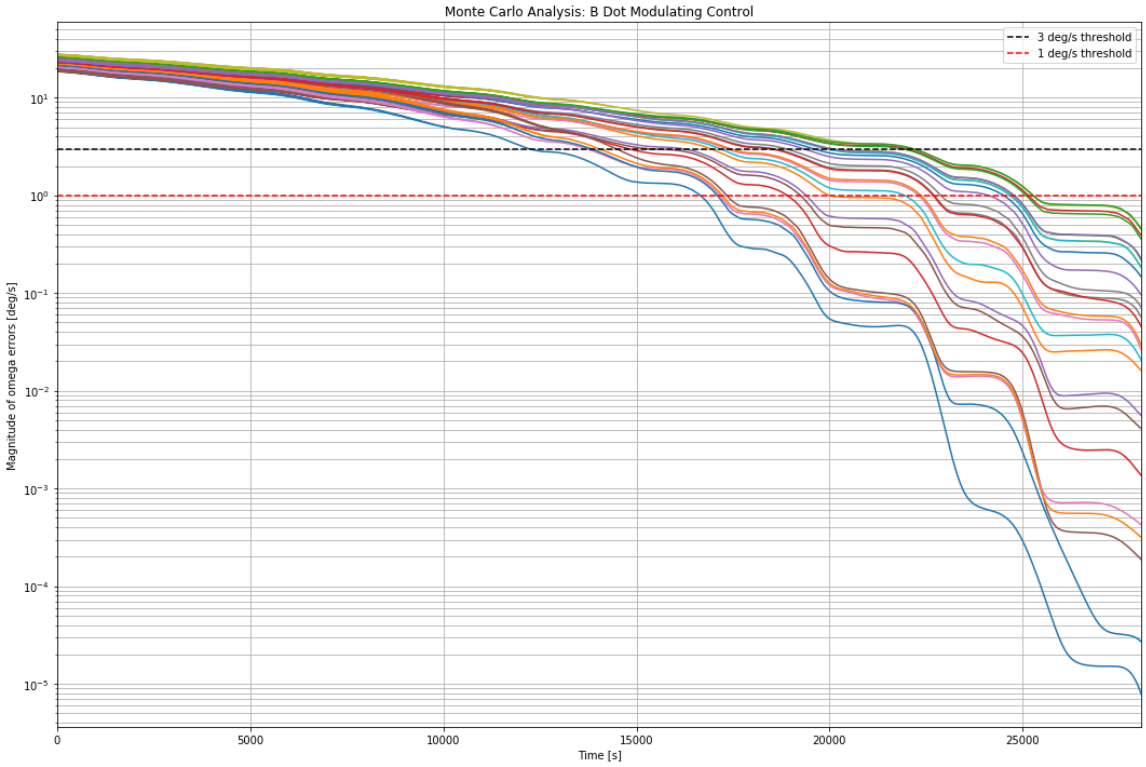

Figure 28: Modulating control: norm of control error over 5 orbits of all 25 runs plotted on log scale.

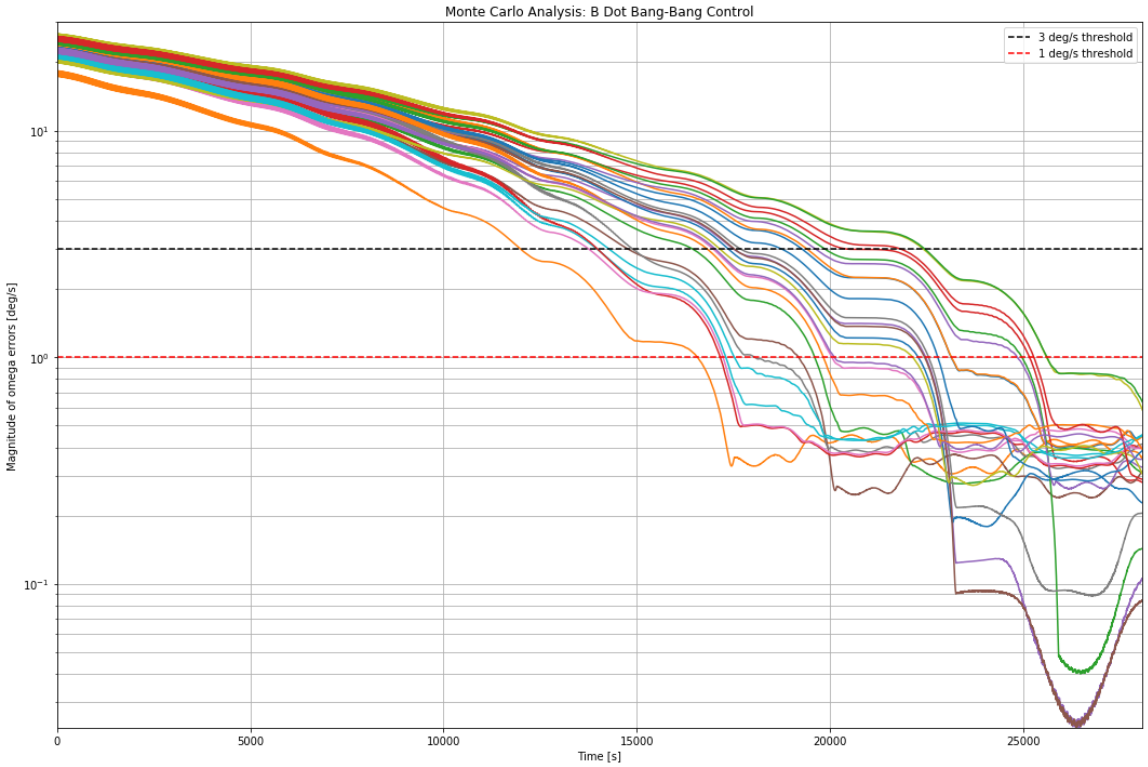

Figure 29: Bang-bang control: norm of control error over 5 orbits of all 25 runs plotted on log scale.

V. Conclusion

One key takeaway from this is the difference in asymptotic behavior of the two control laws. Based on the Monte-Carlo analysis, we saw that the modulating control law could have Lyapunov or asymptotic stability, while the bang-bang method appears to be only Lagrange stable. We also saw that inclination effects are small compared to the effect of varying initial conditions.

A future direction might be to explore the total power consumed by each control law. Although settling time and error are important considerations, total control effort expended (total power) would be a significant factor to take into account when choosing a control law for a real spacecraft.

VI. Code

```
#Imports

%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import scipy.linalg as scilin
from pylab import rcParams, savefig
rcParams['figure.figsize'] = 24, 12

import warnings; warnings.simplefilter('ignore')

# Orbit Params
orbitAlt = 450 # Orbit altitude [km]
rE = 6378 # Earth radius [km]
mu = 3.986e5 # Earth gravitational paramter [km^3/s^2]
omegaE = np.rad2deg(7.2921159e-5) # Earth rotational rate [deg/s]

orbitRad = rE + orbitAlt # [km]
thetaDot = np.rad2deg(np.sqrt(mu/orbitRad**3)) # Orbit angular rate (circular orbit) [deg/s]
orbitPeriod = 2*np.pi*np.sqrt(orbitRad**3/mu) # [seconds]

# Geomagnetic Params
M = 7.838e6 # dipole moment [Tkm^3]
gammaM = 17 # dipole tilt angle [degrees]

# Initial LEO Angles
orbitRAAN = 0 # [degrees]
orbitInc = 45 # [degrees]
theta_t0 = 0 # [degrees]

# Spacecraft Params
m = 30 # Spacecraft mass [kg]

# Spacecraft Initial Rotational Info (B rel. to N)
MRP0 = np.array([0.3, 0.2, 0.4]) # Initial attitude, MRPs
omegaVec0 = np.array([15, 8, 12]) # Initial omega IN B FRAME, [deg/s]
scInertia_B = np.array([
    [3.5, 0, 0],
    [0, 5, 0],
    [0, 0, 8]
]) # Inertia matrix IN B FRAME

# Tilde Matrix from Vector

def tildeMat(vector):

    tildeMat = np.array([
        [0, -vector[2], vector[1]],
        [vector[2], 0, -vector[0]],
        [-vector[1], vector[0], 0]
    ])

    return tildeMat

# 313 DCM Function

def DCM313(RAAN, inc, theta):

    R = np.deg2rad(RAAN)
    i = np.deg2rad(inc)
    th = np.deg2rad(theta)

    DCM313 = np.empty([3,3])

    DCM313[0,0] = np.cos(th)*np.cos(R) - np.sin(th)*np.cos(i)*np.sin(R)
    DCM313[0,1] = np.cos(th)*np.sin(R) + np.sin(th)*np.cos(i)*np.cos(R)
```

```
    DCM313[0,2] = np.sin(th)*np.sin(i)

    DCM313[1,0] = -np.sin(th)*np.cos(R) - np.cos(th)*np.cos(i)*np.sin(R)
    DCM313[1,1] = -np.sin(th)*np.sin(R) + np.cos(th)*np.cos(i)*np.cos(R)
    DCM313[1,2] = np.cos(th)*np.sin(i)

    DCM313[2,0] = np.sin(i)*np.sin(R)
    DCM313[2,1] = -np.sin(i)*np.cos(R)
    DCM313[2,2] = np.cos(i)

    return(DCM313)

# MRP DCM Function

def DCMMRP(sigmaVec):
    sigmaSquared = np.transpose(sigmaVec)@sigmaVec
    sigmaTilde = tildeMat(sigmaVec)
    #
    DCMMRP = np.eye(3) + (8*sigmaTilde@sigmaTilde -
    ↪  4*(1-sigmaSquared)*sigmaTilde)/((1+sigmaSquared)**2)

    return(DCMMRP)

# RK4 Integrator

# RK4 integrator for attitude and attitude rate
# Uses MRPs for attitude description
# State vector is [sigma_BN
#                  omega_BN]

def RK4Attitude(f, ICs, timeEndPoints, h):
    t0 = timeEndPoints[0]
    tf = timeEndPoints[-1] + h
    numSteps = int(np.ceil((tf-t0)/h))

    dim = len(ICs)
    stateHistory = np.empty((dim, numSteps))

    t = t0
    state = ICs

    for i in range(0, numSteps):

        stateHistory[:,i] = state

        k1 = f(t, state)
        k2 = f(t+h*0.5, state+h*k1*0.5)
        k3 = f(t+h*0.5, state+h*k2*0.5)
        k4 = f(t+h, state+h*k3)

        state = state + h/6*(k1 + 2*k2 + 2*k3 + k4)

        # Switch to shadow set for MRPs
        sigma = state[0:3] # First 3 elements of state are elements of MRP vector
        if abs(np.linalg.norm(sigma)-1) <= 1e-16:
            sigma = -sigma/(np.linalg.norm(sigma)**2)
            state[0:3] = sigma # Put switched MRP back into state vector
        elif np.linalg.norm(sigma) >= 1:
            sigma = -sigma/(np.linalg.norm(sigma)**2)
            state[0:3] = sigma # Put switched MRP back into state vector

        t = t + h

    return stateHistory

# Attitude (and omega) First Order DE

def fullAttitudeDE(t, X, info):
```

```
    # Unpack info
    I = info[0]
    T = info[1]

    # Unpack state vector
    MRPs = X[0:3]
    omegaVec = X[3:6]

    # MRP Kinematic DE
    sigma = np.linalg.norm(MRPs)
    AMat = np.array([
        [1 - sigma**2 + 2*MRPs[0]**2, 2*(MRPs[0]*MRPs[1] - MRPs[2]), 2*(MRPs[0]*MRPs[2] +
        ↪  MRPs[1])],
        [2*(MRPs[1]*MRPs[0] + MRPs[2]), 1 - sigma**2 + 2*MRPs[1]**2, 2*(MRPs[1]*MRPs[2] -
        ↪  MRPs[0])],
        [2*(MRPs[2]*MRPs[0] - MRPs[1]), 2*(MRPs[2]*MRPs[1] + MRPs[0]), 1 - sigma**2 +
        ↪  2*MRPs[2]**2]
    ])

    MRPsDot = 0.25*AMat@omegaVec

    # Omega DE
    omegaTilde = tildeMat(omegaVec)
    omegaVecDot = scilin.inv(I)@(np.cross(-omegaVec, I@omegaVec) + T)

    # Repack into 6x1
    XDot = np.concatenate((MRPsDot, omegaVecDot), axis=0)

    return XDot

# Integrator setup for testing energy/momentum

tSpan = [0, orbitPeriod]
h = 1 # time step, [s]

# Params for scenario
torqueVec = np.array([0, 0, 0]) # This should be in B frame
info = [scInertia_B, torqueVec]
ICs_Overview = np.concatenate((MRP0, np.deg2rad(omegaVec0)), axis=0)

# Integrate
odefunc = lambda t, X: fullAttitudeDE(t, X, info)
results = RK4Attitude(odefunc, ICs_Overview, tSpan, h)

tVec = np.arange(tSpan[0], tSpan[-1]+h, h)

# Look at angular momentum and rotational kinetic energy

angMom_t = np.empty([len(results[0,:]),])
T_t = np.empty([len(results[0,:]),])

for i in range(0, len(results[0,:])):
    angMom_t[i] = np.linalg.norm(scInertia_B@results[3:6,i])
    T_t[i] = 0.5*np.linalg.norm(np.transpose(results[3:6,i])@scInertia_B@results[3:6,i])

angMom_0 = scInertia_B@np.deg2rad(omegaVec0)
angMom_f = scInertia_B@results[3:6,-1]

print(np.linalg.norm(angMom_0))
print(np.linalg.norm(angMom_f))

T_0 = 0.5*np.linalg.norm(np.transpose(np.deg2rad(omegaVec0))@scInertia_B@np.deg2rad(omegaVec0))
T_f = 0.5*np.linalg.norm(np.transpose(results[3:6,-1])@scInertia_B@results[3:6,-1])

print(T_0)
print(T_f)

# Plot ang mom and energy over time
```

```
tSpan = [0, orbitPeriod]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)

plt.figure(figsize=(18,12))
plt.subplot(211)
plt.plot(tVec, angMom_t)
plt.ylabel('Magnitude of Angular Momentum')
plt.xlabel('Time [s]')
plt.grid()

plt.subplot(212)
plt.plot(tVec, T_t)
plt.ylabel('Kinetic Energy')
plt.xlabel('Time [s]')
plt.xlabel('Time [s]')
plt.grid()

plt.savefig('angMomEnergyCons.png', bbox_inches='tight')

# Magnetic field function
#
# Returns magnetic field vector (in Hill frame) as a function of geometry and time

def magField(info, eulerAngs, thetaDot, beta0, t):

    # Unpack info
    M = info[0] # Magnetic dipole moment
    r = info[1] # Orbit radius
    gamma = np.deg2rad(info[2]) # Tilt angle between inertial and geomagnetic frames
    omegaE = info[3] # STILL IN DEGREES/s HERE

    # Convert angles to radians
    RAAN = np.deg2rad(eulerAngs[0])
    inc = np.deg2rad(eulerAngs[1])
    theta = np.deg2rad(eulerAngs[2])
    thetaDot = np.deg2rad(thetaDot)

    # Compute beta
    beta = np.deg2rad(beta0 + omegaE*t)

    # Compute xi_m, eta_m
    xi_m = np.arccos(np.cos(inc)*np.cos(gamma) + np.sin(inc)*np.sin(gamma)*np.cos(RAAN - beta))
    eta_m = np.arcsin(np.sin(gamma)*np.sin(RAAN - beta)/np.sin(xi_m))

    # Compute magnetic field vector (units of Teslas)
    bVec_HFrame = M/r**3*np.asarray([
        np.cos(thetaDot*t - eta_m)*np.sin(xi_m),
        np.cos(xi_m),
        -2*np.sin(thetaDot*t - eta_m)*np.sin(xi_m)
    ])

    # Return
    return(bVec_HFrame)

# Compute one orbit period

tStep = 1 # Timestep for computing magnetic field
timeOrbitPeriod = np.linspace(0, orbitPeriod, int(np.ceil(orbitPeriod/tStep)))

# Array to store magnetic field over time
magVec_HFrame_orbitPeriod = np.empty((3, int(np.ceil(orbitPeriod/tStep))))

print(magVec_HFrame_orbitPeriod.shape)

# Info for magnetic field function
info = [M, orbitRad, gammaM, omegaE]
```

```
for i in range(0, len(timeOrbitPeriod)):

    t = timeOrbitPeriod[i] # Time

    # Euler angles (time varying)
    eulerAngs = [orbitRAAN, orbitInc, theta_t0+thetaDot*t]

    magVec_HFrame_orbitPeriod[:,i] = magField(info, eulerAngs, thetaDot, 0, t)

# Plot local magnetic field in Hill frame (over one orbit period)

plt.figure(figsize=(18,12))
plt.subplot(311)
plt.plot(timeOrbitPeriod, magVec_HFrame_orbitPeriod[0,:], 'r-', label='First Component')
plt.title('Local Magnetic Field Strength in Hill Frame Over One Orbit Period')
plt.legend()
plt.ylabel('Field Strength [Teslas]')
plt.grid()
plt.subplot(312)
plt.plot(timeOrbitPeriod, magVec_HFrame_orbitPeriod[1,:], 'g-', label='Second Component')
plt.ylim([-2.5e-5, 2.5e-5])
plt.legend()
plt.ylabel('Field Strength [Teslas]')
plt.grid()
plt.subplot(313)
plt.plot(timeOrbitPeriod, magVec_HFrame_orbitPeriod[2,:], 'b-', label='Third Component')
plt.xlabel('Time [s]')
plt.ylabel('Field Strength [Teslas]')
plt.legend()
plt.grid()

# Save figure
plt.savefig('BFieldHill.png', bbox_inches='tight')

plt.show()

beta0 = 0 # [degrees]

# Array to store magnetic field vector in body frame over time
magVec_BFrame_100s = np.empty((3, len(tVec)))

info = [M, orbitRad, gammaM, omegaE]

beta0 = 0

for i in range(0, len(tVec)):

    t = tVec[i]

    BN = DCMMRP(results[0:3,i])
    NH = np.transpose(DCM313(orbitRAAN, orbitInc, theta_t0+thetaDot*t))

    # Euler angles (time varying)
    eulerAngs = [orbitRAAN, orbitInc, theta_t0+thetaDot*t]

    # Magnetic field vector in Hill frame
    magVec_HFrame = magField(info, eulerAngs, thetaDot, beta0, t)

    # Transform hill frame vector to body frame
    magVec_BFrame_100s[:,i] = BN@NH@magVec_HFrame

# Plot local magnetic field in body frame (over 100s)

plt.figure(figsize=(18,12))
plt.subplot(311)
plt.plot(tVec, magVec_BFrame_100s[0,:], 'r-', label='First Component')
plt.title('Local Magnetic Field Strength in Body Frame Over 100s')
plt.legend()
```

```
plt.grid()
plt.ylabel('Field Strength [Teslas]')
plt.ylim([-2.5e-5, 2.5e-5])
plt.xlim([0,100])
plt.subplot(312)
plt.plot(tVec, magVec_BFrame_100s[1,:], 'g-', label='Second Component')
plt.legend()
plt.grid()
plt.ylabel('Field Strength [Teslas]')
plt.ylim([-2.5e-5, 2.5e-5])
plt.xlim([0,100])
plt.subplot(313)
plt.plot(tVec, magVec_BFrame_100s[2,:], 'b-', label='Third Component')
plt.legend()
plt.xlabel('Time [s]')
plt.ylabel('Field Strength [Teslas]')
plt.legend()
plt.ylim([-2.5e-5, 2.5e-5])
plt.xlim([0,100])
plt.grid()

# Save figure
plt.savefig('BFieldBody.png', bbox_inches='tight')

plt.show()

# Attitude (and omega) First Order DE WITH CONTROL LAW
# 'controlLaw' is a function with arguments (t,X) that outputs a torque vector and command
↪  magnetic dipole

def fullAttitudeDEControl(t, X, info, controlLaw):
    # Unpack info
    I = info[0]
    T = info[1]

    # Unpack state vector
    MRPs = X[0:3]
    omegaVec = X[3:6]

    # MRP Kinematic DE
    sigma = np.linalg.norm(MRPs)

    AMat = ((1-sigma**2)*np.eye(3) + 2*tildeMat(MRPs) + 2*np.outer(MRPs, MRPs))

    MRPsDot = 0.25*AMat@omegaVec

    # Compute control torque
    u = controlLaw(t, X)[0]

    # Omega DE
    omegaTilde = tildeMat(omegaVec)
    omegaVecDot = scilin.inv(I)@(np.cross(-omegaVec, I@omegaVec) + u)

    # Repack into 6x1
    XDot = np.concatenate((MRPsDot, omegaVecDot), axis=0)

    return XDot

# Modulating B-dot control law

def modulatingBDot(t, X, mMax, info):

    # Unpack
    omegaE = info[0] # [deg/s]
    RAAN = info[1] # [deg]
    inc = info[2] # [deg]
    theta0 = info[3] # [deg]
    thetaDot = info[4] # [deg/s]
```

```
    orbitRad = info[5] # [km]
    inertiaMat = info[6]
    beta0 = info[7] # [deg]
    gamma = info[8] # [deg]
    M = info[9] # [Tkm^3]

    bHill = magField([M, orbitRad, gamma, omegaE], [RAAN, inc, theta0+thetaDot*t], thetaDot,
    ↪  beta0, t)

    BN = DCMMRP(X[0:3])
    NH = np.transpose(DCM313(RAAN, inc, theta0+thetaDot*t))

    # Transform hill frame vector to body frame
    bBody = BN@NH@bHill

    # Compute gain
    i = np.deg2rad(inc) # [rads]
    gam = np.deg2rad(gamma) # [rads]
    beta = np.deg2rad(beta0 + omegaE*t) # [rads]
    xi_m = np.arccos(np.cos(i)*np.cos(gam) + np.sin(i)*np.sin(gam)*np.cos(np.deg2rad(RAAN) -
    ↪  beta)) # [rads]

    IMin = np.amin(inertiaMat.diagonal())
    kw = 2*np.deg2rad(thetaDot)*(1+np.sin(xi_m))*IMin

    # Compute command dipole
    bBodyHat = bBody/np.linalg.norm(bBody)
    omegaVec = X[3:6]

    mVec = -1*kw/np.linalg.norm(bBody)*np.cross(bBodyHat, (np.eye(3) - np.outer(bBodyHat,
    ↪  bBodyHat))@omegaVec)

    # Check for saturation (componentwise)
    for i in range(0,3):
        if np.abs(mVec[i]) > mMax:
            mVec[i] = np.sign(mVec[i])*mMax

    # Compute control torque:
    uVec= np.cross(mVec, bBody)

    # Debugging
#     print('t is: ', t)
#     print('State is: ', X)
#     print('theta dot is: ', thetaDot)
#     print('Gain kw is: ', kw)
#     print('xi is [degrees]: ', np.rad2deg(xi_m))
#     print('m is: ', mVec)
#     print('b (body frame) is: ', bBody)
#     print('control torque vector u is: ', uVec, '\n')

    # Return
    return([uVec,mVec])

mMax = 3 # [Am^2]

# Initial Conditions
MRP_i1 = np.asarray([0.1, 0.1, 0.4])
omegaVec_i1 = np.asarray([1, 12, 1])
ICs_1 = np.concatenate((MRP_i1, np.deg2rad(omegaVec_i1)), axis=0)

MRP_i2 = np.asarray([0.35, 0.2, 0.15])
omegaVec_i2 = np.asarray([6, 4, 13])
ICs_2 = np.concatenate((MRP_i2, np.deg2rad(omegaVec_i2)), axis=0)

controlParams = [omegaE, orbitRAAN, orbitInc, theta_t0, thetaDot, orbitRad, scInertia_B, beta0,
↪  gammaM, M]

externalTorque = np.asarray([0, 0, 0])
```

```
# Bang-Bang B-dot control law

def bangBangBDot(t, X, mMax, info):

    # Unpack
    omegaE = info[0] # [deg/s]
    RAAN = info[1] # [deg]
    inc = info[2] # [deg]
    theta0 = info[3] # [deg]
    thetaDot = info[4] # [deg/s]
    orbitRad = info[5] # [km]
    inertiaMat = info[6]
    beta0 = info[7] # [deg]
    gamma = info[8] # [deg]
    M = info[9] # [Tkm^3]

    # Rotation matrices
    BN = DCMMRP(X[0:3])
    NH = np.transpose(DCM313(RAAN, inc, theta_t0+thetaDot*t))

    # Magnetic field vector calculation
    bHill = magField([M, orbitRad, gamma, omegaE], [RAAN, inc, theta0+thetaDot*t], thetaDot,
    ↪  beta0, t)
    bBody = BN@NH@bHill

    i = np.deg2rad(inc)
    gam = np.deg2rad(gamma)
    Omega = np.deg2rad(RAAN)
    omega = np.deg2rad(omegaE)
    beta = np.deg2rad(beta0 + omegaE*t)

    # Compute time derivative of b as seen by H frame
    xi_m = np.arccos(np.cos(i)*np.cos(gam) + np.sin(i)*np.sin(gam)*np.cos(Omega-beta))
    eta_m = np.arcsin(np.sin(gam)*np.sin(Omega-beta)/np.sin(xi_m))

    xi_mPrime =
    ↪  -(-np.sin(i)*np.sin(gam)*np.sin(Omega-beta)*(-omega))/np.sqrt(1-(np.cos(i)*np.cos(gam)+np.sin(i)*np.sin(g

    eta_mPrime =
    ↪  1/np.sqrt(1-(np.sin(gam)*np.sin(Omega-beta)/np.sin(xi_m))**2)*((np.sin(xi_m)*np.sin(gam)*np.cos(Omega-bet
    ↪  - np.sin(gam)*np.sin(Omega-beta)*np.cos(xi_m)*xi_mPrime)/(np.sin(xi_m)**2))

    bPrimeRelHBody = (M/orbitRad**3)*BN@NH@np.asarray([
        -np.sin(thetaDot*t-eta_m)*(-eta_mPrime)*np.sin(xi_m) +
        ↪  np.cos(thetaDot*t-eta_m)*np.cos(xi_m)*xi_mPrime,
        -np.sin(xi_m)*xi_mPrime,
        -2*np.cos(thetaDot*t-eta_m)*(-eta_mPrime)*np.sin(xi_m) -
        ↪  2*np.sin(thetaDot*t-eta_m)*np.cos(xi_m)*xi_mPrime
    ])

    # Compute cross product term in transport thm (omega_H/B x b)
    omega_HNBody = BN@NH@np.asarray([0, 0, np.deg2rad(thetaDot)])
    omega_BNBody = X[3:6]

    omega_HBBody = omega_HNBody - omega_BNBody

    # Compute b'
    bPrimeRelBBody = bPrimeRelHBody + np.cross(omega_HBBody, bBody)

    # Compute m
    mBody = -1*mMax*np.sign(bPrimeRelBBody)

    # Compute control torque u
    uBody = np.cross(mBody, bBody)

    # Debug
#     print('u is: ', uBody)
```

```
#     print('b is: ', bBody)
#     print('m is: ', mBody, '\n')

    # Return
    return([uBody,mBody])

%%time

mMax = 3 # [Am^2]

# Initial Conditions
MRP_i1 = np.asarray([0.1, 0.1, 0.4])
omegaVec_i1 = np.asarray([1, 12, 1])
ICs_1 = np.concatenate((MRP_i1, np.deg2rad(omegaVec_i1)), axis=0)

MRP_i2 = np.asarray([0.35, 0.2, 0.15])
omegaVec_i2 = np.asarray([6, 4, 13])
ICs_2 = np.concatenate((MRP_i2, np.deg2rad(omegaVec_i2)), axis=0)

externalTorque = np.asarray([0, 0, 0])

controlParams = [omegaE, orbitRAAN, orbitInc, theta_t0, thetaDot, orbitRad, scInertia_B, beta0,
↪  gammaM, M]

controlFuncMod = lambda t, X: modulatingBDot(t, X, mMax, controlParams)
controlFuncBang = lambda t, X: bangBangBDot(t, X, mMax, controlParams)

odefuncMod = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncMod)
odefuncBang = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncBang)

tSpan = [0, 3*orbitPeriod]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)
h = 1 # time step, [s]

stepN = int(np.ceil((tSpan[-1]+h)/h))

ICsVec = [ICs_Overview, ICs_1, ICs_2]

resultsPart4 = np.empty((2,3,6,stepN))

for i in range(0,3):
    resultsPart4[0,i,:,:] = RK4Attitude(odefuncMod, ICsVec[i], tSpan, h) # modulating control
    resultsPart4[1,i,:,:] = RK4Attitude(odefuncBang, ICsVec[i], tSpan, h) # bang bang control


%%time

# Compute control error and control torque
# (control error is just omega)

controlTorquesPart4 = np.empty((2,3,3,stepN))
controlDipolePart4 = np.empty((2,3,3,stepN))

# Compute torques
for i in range(0,3):
    for j in range(0,stepN):

        mod = controlFuncMod(tVec[j], resultsPart4[0,i,:,j])
        bang = controlFuncBang(tVec[j], resultsPart4[1,i,:,j])

        controlTorquesPart4[0,i,:,j] = mod[0] # modulating control torque
        controlTorquesPart4[1,i,:,j] = bang[0] # bang bang control torque
        controlDipolePart4[0,i,:,j] = mod[1] # modulating control moment dipole
        controlDipolePart4[1,i,:,j] = bang[1] # bang bang control moment dipole

# All plots for part 4
```

```
controlTypes = ['Modulating Control', 'Bang-Bang Control']
for i in range(0,2): # loop over mod vs bang
    for j in range(0,3): # loop over ICs

        plt.figure()

        #Plot Omegas
        plt.subplot(322)
        plt.plot(tVec, np.rad2deg(resultsPart4[i,j,3,:]), 'r-', label='First omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--', label='3 deg/s bounds')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_1 [deg/s]')
        plt.legend()
        plt.grid()
        plt.title(controlTypes[i]+' omegas, 45 deg inc, IC Set %i, mMax=3'%(j))
        plt.subplot(324)
        plt.plot(tVec, np.rad2deg(resultsPart4[i,j,4,:]), 'g-', label='Second omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_2 [deg/s]')
        plt.legend()
        plt.grid()
        plt.subplot(326)
        plt.plot(tVec, np.rad2deg(resultsPart4[i,j,5,:]), 'b-', label='Third omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_3 [deg/s]')
        plt.xlabel('Time [s]')
        plt.legend()
        plt.grid()

        #Plot MRPs
        plt.subplot(321)
        plt.plot(tVec, (resultsPart4[i,j,0,:]), 'r-', label='First MRP component')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('MRP_1')
        plt.legend()
        plt.grid()
        plt.title(controlTypes[i]+' MRPs, 45 deg inc, IC Set %i, mMax=3'%(j))
        plt.subplot(323)
        plt.plot(tVec, (resultsPart4[i,j,1,:]), 'g-', label='Second MRP component')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('MRP_2')
        plt.legend()
        plt.grid()
        plt.subplot(325)
        plt.plot(tVec, (resultsPart4[i,j,2,:]), 'b-', label='Third MRP component')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('MRP_3')
        plt.xlabel('Time [s]')
        plt.legend()

        # Save figures
        #plt.savefig(controlTypes[i]+'States45_IC%i_mMax3.png'%(j), bbox_inches='tight')

        # Plot control torque and magnetic dipole moment
        plt.figure()

        # Plot control torques first
        plt.subplot(321)
        plt.plot(tVec, controlTorquesPart4[i,j,0,:], label='Control torque in 1st body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Control Torque bodyX [Nm]')
```

```
        plt.legend()
        plt.grid()
        plt.title(controlTypes[i]+' Control Torques, 45 deg inc, IC Set %i, mMax=3'%(j))
        plt.subplot(323)
        plt.plot(tVec, controlTorquesPart4[i,j,1,:], label='Control torque in 2nd body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Control Torque bodyY [Nm]')
        plt.legend()
        plt.grid()
        plt.subplot(325)
        plt.plot(tVec, controlTorquesPart4[i,j,2,:], label='Control torque in 3rd body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Control Torque bodyZ [Nm]')
        plt.legend()
        plt.grid()

        # Plot command moment dipoles
        plt.subplot(322)
        plt.plot(tVec, controlDipolePart4[i,j,0,:], label='Command dipole moment in 1st body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Command Dipole Moment bodyX [Am^2]')
        plt.legend()
        plt.grid()
        plt.title(controlTypes[i]+' Command Dipole Moments, 45 deg inc, IC Set %i, mMax=3'%(j))
        plt.subplot(324)
        plt.plot(tVec, controlDipolePart4[i,j,1,:], label='Command dipole moment in 2nd body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Command Dipole Moment bodyY [Am^2]')
        plt.legend()
        plt.grid()
        plt.subplot(326)
        plt.plot(tVec, controlDipolePart4[i,j,2,:], label='Command dipole moment in 3rd body axis
        ↪  direction')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Command Dipole Moment bodyZ [Am^2]')
        plt.legend()
        plt.grid()

        # Save figures
        #plt.savefig(controlTypes[i]+'Controls45_IC%i_mMax3.png'%(j), bbox_inches='tight')

%%time

incVec = [15, 105] # inclination angles in degrees

mMax = 3
externalTorque = np.asarray([0, 0, 0])

tSpan = [0, 3*orbitPeriod]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)
h = 1 # time step, [s]

stepN = int(np.ceil((tSpan[-1]+h)/h))

ICsVec = [ICs_Overview, ICs_1, ICs_2]

resultsPart5a = np.empty((2,2,6,stepN))

for i in range(0,len(incVec)):

    controlParams = [omegaE, orbitRAAN, incVec[i], theta_t0, thetaDot, orbitRad, scInertia_B,
    ↪  beta0, gammaM, M]

    controlFuncMod = lambda t, X: modulatingBDot(t, X, mMax, controlParams)
```

```
    controlFuncBang = lambda t, X: bangBangBDot(t, X, mMax, controlParams)

    odefuncMod = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
    ↪  controlFuncMod)
    odefuncBang = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
    ↪  controlFuncBang)

    resultsPart5a[0,i,:,:] = RK4Attitude(odefuncMod, ICsVec[0], tSpan, h) # modulating
    resultsPart5a[1,i,:,:] = RK4Attitude(odefuncBang, ICsVec[0], tSpan, h) # bang-bang

# Part a plots

for i in range(0,2):
    for j in range(0,2):

        plt.figure(figsize=(12,12))

        #Plot Omegas
        plt.subplot(311)
        plt.plot(tVec, np.rad2deg(resultsPart5a[i,j,3,:]), 'r-', label='First omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--', label='3 deg/s bounds')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_1 [deg/s]')
        plt.legend()
        plt.grid()
        plt.title(controlTypes[i]+' Omegas, %i deg inc, IC Set 0, mMax=3'%(incVec[j]))
        plt.subplot(312)
        plt.plot(tVec, np.rad2deg(resultsPart5a[i,j,4,:]), 'g-', label='Second omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_2 [deg/s]')
        plt.legend()
        plt.grid()
        plt.subplot(313)
        plt.plot(tVec, np.rad2deg(resultsPart5a[i,j,5,:]), 'b-', label='Third omega component')
        plt.plot([0,tSpan[-1]], [3,3], 'k--')
        plt.plot([0,tSpan[-1]], [-3,-3], 'k--')
        plt.xlim([0,tSpan[-1]])
        plt.ylabel('Omega_3 [deg/s]')
        plt.xlabel('Time [s]')
        plt.legend()
        plt.grid()

        plt.savefig(controlTypes[i]+'Omegas%i_IC0_mMax3.png'%(incVec[j]), bbox_inches='tight')

%%time

# 5b Stuff

mMaxMinMod = 4.205
mMaxMinBang = 4.05

controlParams = [omegaE, orbitRAAN, incVec[0], theta_t0, thetaDot, orbitRad, scInertia_B, beta0,
↪  gammaM, M]

controlFuncMod = lambda t, X: modulatingBDot(t, X, mMaxMinMod, controlParams)
controlFuncBang = lambda t, X: bangBangBDot(t, X, mMaxMinBang, controlParams)

odefuncMod = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncMod)
odefuncBang = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncBang)

resultsPart5b = np.empty((2,6,stepN))

resultsPart5b[0,:,:] = RK4Attitude(odefuncMod, ICsVec[0], tSpan, h)
```

```
resultsPart5b[1,:,:] = RK4Attitude(odefuncBang, ICsVec[0], tSpan, h)

# 5b plots

tSpan = [0, 3*orbitPeriod]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)

mVec = [mMaxMinMod, mMaxMinBang]

for i in range(0,2):

    plt.figure()

    #Plot Omegas
    plt.subplot(322)
    plt.plot(tVec, np.rad2deg(resultsPart5b[i,3,:]), 'r-', label='First omega component')
    plt.plot([0,tSpan[-1]], [1,1], 'k--', label='1 deg/s bounds')
    plt.plot([0,tSpan[-1]], [-1,-1], 'k--')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('Omega_1 [deg/s]')
    plt.legend()
    plt.grid()
    plt.title(controlTypes[i]+' Omegas, 15 deg inc, IC Set 0, mMax=%1.3f'%(mVec[i]))
    plt.subplot(324)
    plt.plot(tVec, np.rad2deg(resultsPart5b[i,4,:]), 'g-', label='Second omega component')
    plt.plot([0,tSpan[-1]], [1,1], 'k--')
    plt.plot([0,tSpan[-1]], [-1,-1], 'k--')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('Omega_2 [deg/s]')
    plt.legend()
    plt.grid()
    plt.subplot(326)
    plt.plot(tVec, np.rad2deg(resultsPart5b[i,5,:]), 'b-', label='Third omega component')
    plt.plot([0,tSpan[-1]], [1,1], 'k--')
    plt.plot([0,tSpan[-1]], [-1,-1], 'k--')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('Omega_3 [deg/s]')
    plt.xlabel('Time [s]')
    plt.legend()
    plt.grid()

    #Plot MRPs
    plt.subplot(321)
    plt.plot(tVec, (resultsPart5b[i,0,:]), 'r-', label='First MRP component')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('MRP_1')
    plt.legend()
    plt.grid()
    plt.title(controlTypes[i]+' MRPs, 15 deg inc, IC Set 0, mMax=%1.3f'%(mVec[i]))
    plt.subplot(323)
    plt.plot(tVec, (resultsPart5b[i,1,:]), 'g-', label='Second MRP component')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('MRP_2')
    plt.legend()
    plt.grid()
    plt.subplot(325)
    plt.plot(tVec, (resultsPart5b[i,2,:]), 'b-', label='Third MRP component')
    plt.xlim([0,tSpan[-1]])
    plt.ylabel('MRP_3')
    plt.xlabel('Time [s]')
    plt.legend()
    plt.grid()

    plt.savefig(controlTypes[i]+'States15_IC0_mMax%1.3f.png'%(mVec[i]), bbox_inches='tight')

# Create Random ICs, Setup Stuff, and Arrays to Store Results

# Number of Monte Carlo runs
```

```
nRuns = 25

ICsMod = np.empty((6,nRuns))
ICsBang = np.empty((6,nRuns))

# Generate random initial omegas
# Note: uses overview ICs for MRPS
for i in range(0,nRuns):
    ICsMod[:,i] = np.concatenate((MRP0, np.deg2rad(np.random.randint(10, high=17, size=3))),
    ↪  axis=0)
    ICsBang[:,i] = np.concatenate((MRP0, np.deg2rad(np.random.randint(10, high=17, size=3))),
    ↪  axis=0)

tSpan = [0, 5*orbitPeriod]
h = 1 # time step, [s]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)

monteCarloResultsMod = np.empty((6,len(tVec),nRuns))
monteCarloResultsBang = np.empty((6,len(tVec),nRuns))

mMax = 4

controlParams = [omegaE, orbitRAAN, orbitInc, theta_t0, thetaDot, orbitRad, scInertia_B, beta0,
↪  gammaM, M]

controlFuncMod = lambda t, X: modulatingBDot(t, X, mMax, controlParams)
controlFuncBang = lambda t, X: bangBangBDot(t, X, mMax, controlParams)

odefuncMod = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncMod)
odefuncBang = lambda t, X: fullAttitudeDEControl(t, X, [scInertia_B, externalTorque],
↪  controlFuncBang)

%%time

# Monte Carlo Runs

for i in range(0,nRuns):

    monteCarloResultsMod[:,:,i] = RK4Attitude(odefuncMod, ICsMod[:,i], tSpan, h)
    monteCarloResultsBang[:,:,i] = RK4Attitude(odefuncBang, ICsBang[:,i], tSpan, h)

%%time

tSpan = [0, 5*orbitPeriod]
h = 1 # time step, [s]
tVec = np.arange(tSpan[0], tSpan[-1]+h, h)

# Compute norms

# Arrays to save norms
omegaNormsMod = np.empty((len(tVec),nRuns))
omegaNormsBang = np.empty((len(tVec),nRuns))

# Settling times
setTimesMod = np.empty((nRuns))
setTimesBang = np.empty((nRuns))

for i in range(0,nRuns):
    setTimeModFound = False
    setTimeBangFound = False
    for j in range(0,len(tVec)):
        omegaNormsMod[j,i] = np.linalg.norm(np.rad2deg(monteCarloResultsMod[3:6,j,i]))
        omegaNormsBang[j,i] = np.linalg.norm(np.rad2deg(monteCarloResultsBang[3:6,j,i]))

        # Check for 3 deg/s threshold crossing
        if (np.rad2deg(omegaNormsMod[j,i])<=3):
            setTimeModFound = True
```

```
            setTimesMod[i] = tVec[j]
        if (np.rad2deg(omegaNormsBang[j,i])<=3):
            setTimeBangFound = True
            setTimesBang[i] = tVec[j]

# Average over runs to find avg settling time
setTimeAvgMod = np.nanmean(setTimesMod)
setTimeAvgBang = np.nanmean(setTimesBang)

print(setTimeAvgMod)
print(setTimeAvgBang)

# Monte Carlo Plots

# Plot Omega Norms - Mod Control
plt.figure(figsize=(18,12))
for i in range(0,nRuns):
    plt.semilogy(tVec, omegaNormsMod[:,i])

plt.semilogy([0,tVec[-1]],[3,3], 'k--',label='3 deg/s threshold')
plt.semilogy([0,tVec[-1]],[1,1], 'r--',label='1 deg/s threshold')
plt.xlim([0,tVec[-1]])
plt.xlabel('Time [s]')
plt.ylabel('Magnitude of omega errors [deg/s]')
plt.title('Monte Carlo Analysis: B Dot Modulating Control')
plt.legend()
plt.grid(True, which='both')

# Save fig
plt.savefig('MonteCarloMod5.png', bbox_inches='tight')

plt.show()

# Plot Omega Norms - Bang-Bang Control
plt.figure(figsize=(18,12))
for i in range(0,nRuns):
    plt.semilogy(tVec, omegaNormsBang[:,i])

plt.semilogy([0,tVec[-1]],[3,3], 'k--',label='3 deg/s threshold')
plt.semilogy([0,tVec[-1]],[1,1], 'r--',label='1 deg/s threshold')
plt.xlim([0,tVec[-1]])
plt.ylim([0,30])
plt.xlabel('Time [s]')
plt.ylabel('Magnitude of omega errors [deg/s]')
plt.title('Monte Carlo Analysis: B Dot Bang-Bang Control')
plt.legend()
plt.grid(True, which='both')

# Save fig
plt.savefig('MonteCarloBang5.png', bbox_inches='tight')

plt.show()
```