```
% Reinforcement Learning Final Term Paper
% [Applications of Nash Differential Games to Aerospace]

% Clear All Variables in our Workspace
clear all;

% Inertia Matrix
J_ast = [90 10 5;
         10 50 7;
         5 7 50];

% Eigendecomposition of Inertia Matrix
[V, D] = eig(J_ast);
eigenvalues = diag(D);

% Get the orthogonal rotation matrix
rotation_matrix = V;

% We are given C1, C2, and C3
C1 = eye(3);
C2 = [0.8829 0 0.4695;
      0.4695 0 -0.8829;
      0 1 0];

C3 = [0.7986 -0.6018 0;
      -0.6018 -0.7986 0;
      0 0 -1];

% Solve for G1, G2, and G3
G1 = rotation_matrix' * C1;
G2 = rotation_matrix' * C2;
G3 = rotation_matrix' * C3;

% Solve for the inertia matrix of the combined spacecraft
J = rotation_matrix' * J_ast * rotation_matrix;

% Get Jx, Jy, and Jz
Jx = J(1, 1);
Jy = J(2, 2);
Jz = J(3, 3);

% w0 Value as given to us
w0 = 7.292 .* 10^(-5);

% Storage values
v1 = (-1 .* ((Jy - Jz)      ./ Jx) .* w0^2);
v2 = (-1 .* ((Jy - Jz - Jx) ./ Jx) .* w0);
v3 = (-1 .* ((Jy - Jx)      ./ Jz) .* w0^2);
v4 = ( 1 .* ((Jy - Jz - Jx) ./ Jz) .* w0);

% Set up A and B matrices
A = [0  0 0  1  0 0 ;
```

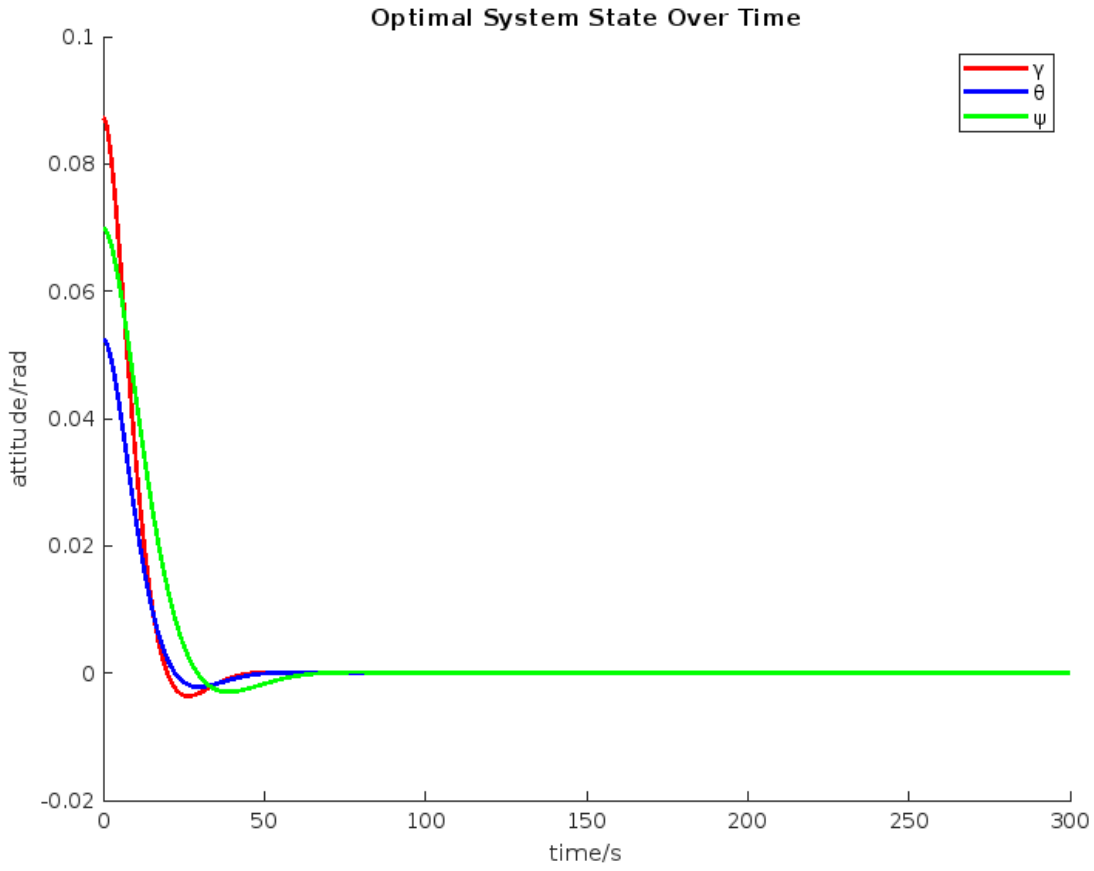
```
     0  0 0  0  1 0 ;
     0  0 0  0  0 1 ;
     v1 0 0  0  0 v2;
     0  0 0  0  0 0 ;
     0  0 v3 v4 0 0 ];

I = eye(size(J));  % Create an identity matrix of the same size as J
B1 = vertcat(zeros(3, 3), J \ I * G1);
B2 = vertcat(zeros(3, 3), J \ I * G2);
B3 = vertcat(zeros(3, 3), J \ I * G3);

% All of these equal 6 which proves that (A, Bi) is always controllable for
% all Bi
rank1 = rank([B1, A * B1, A^2 * B1, A^3 * B1, A^4 * B1, A^5 * B1]);
rank2 = rank([B2, A * B2, A^2 * B2, A^3 * B2, A^4 * B2, A^5 * B2]);
rank3 = rank([B3, A * B3, A^2 * B3, A^3 * B3, A^4 * B3, A^5 * B3]);

% Initialize the state value
x_initial = [0.0873, 0.0524, 0.0698, 0,0,0]';

% Get values of Q matrices
Q1=0.005*eye(6);
Q2=0.005*eye(6);
Q3=0.005*eye(6);

% Get values of R matrices
R11=0.01*eye(3);
R12=0.01*eye(3);
R13=0.01*eye(3);
R21=0.01*eye(3);
R22=0.01*eye(3);
R23=0.01*eye(3);
R31=0.01*eye(3);
R32=0.01*eye(3);
R33=0.01*eye(3);

% Set values of S based on B and R matrices
S1  = B1 / R11 * B1';
S12 = B1 / R11 * R21 / R11 * B1';
S13 = B1 / R11 * R31 / R11 * B1';
S2  = B2 / R22 * B2';
S21 = B2 / R22 * R12 / R22 * B2';
S23 = B2 / R22 * R32 / R22 * B2';
S3  = B3 / R33 * B3';
S31 = B3 / R33 * R13 / R33 * B3';
S32 = B3 / R33 * R23 / R33 * B3';

% Solve for Initial Iterative Matrices P1, P2, and P3
P1 = are(A, S1, Q1);
P2 = are(A-S1*P1, S2, Q2 + P1*S12*P1);
P3 = are(A-S1*P1-S2*P2, S3, Q3 + P1*S13*P1 + P2*S23*P2);

iterations = 10;
```

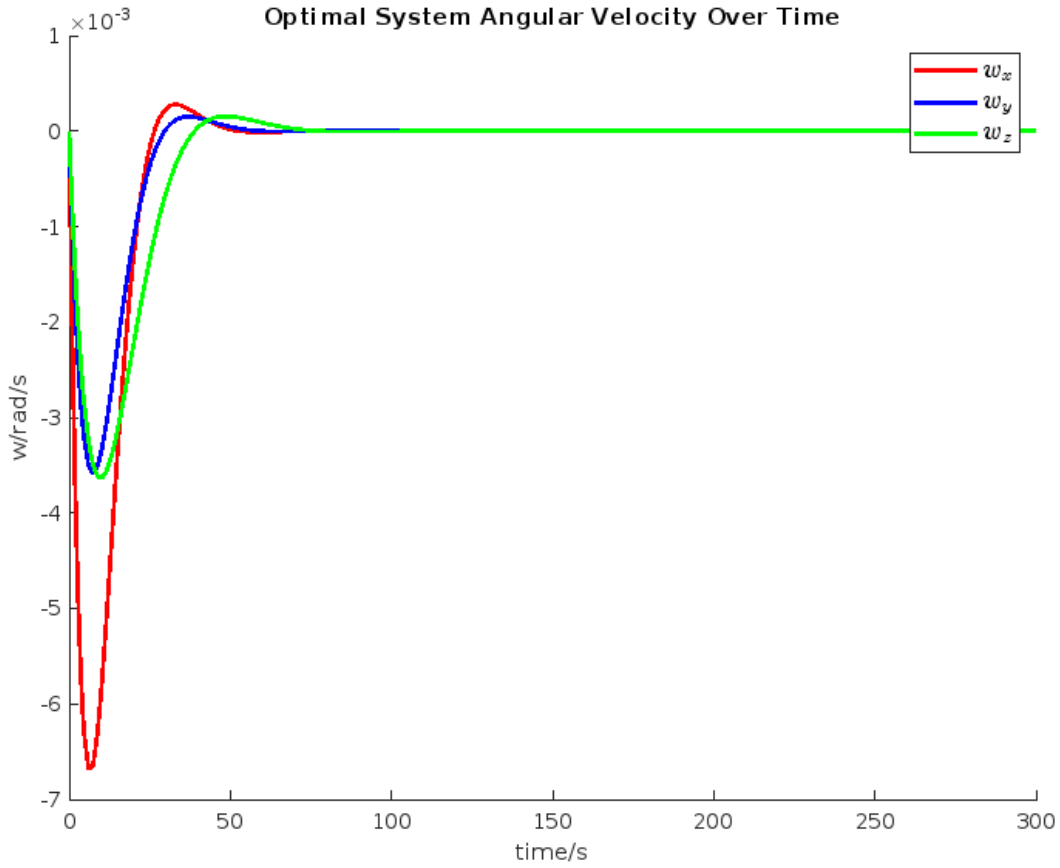
```
% Store P1, P2, P3 Values to see if there is convergence
P1_values = cell(1, iterations + 1);
P2_values = cell(1, iterations + 1);
P3_values = cell(1, iterations + 1);

P1_values{1} = P1;
P2_values{1} = P2;
P3_values{1} = P3;

% Store Performance Criterion Values for each Player
J1_values = cell(1, iterations + 1);
J1_values{1} = x_initial' * P1 * x_initial;

J2_values = cell(1, iterations + 1);
J2_values{1} = x_initial' * P2 * x_initial;

J3_values = cell(1, iterations + 1);
J3_values{1} = x_initial' * P3 * x_initial;

% Conduct Lyapunov Iterations
for i = 1:iterations
    P1_updated = lyap2((A-S1*P1-S2*P2-
S3*P3)',Q1+P1*S1*P1+P2*S21*P2+P3*S31*P3);
    P2_updated = lyap2((A-S1*P1-S2*P2-
S3*P3)',Q2+P1*S12*P1+P2*S2*P2+P3*S32*P3);
    P3_updated = lyap2((A-S1*P1-S2*P2-
S3*P3)',Q3+P1*S13*P1+P2*S23*P2+P3*S3*P3);

    P1 = P1_updated;
    P2 = P2_updated;
    P3 = P3_updated;

    P1_values{i + 1} = P1_updated;
    P2_values{i + 1} = P2_updated;
    P3_values{i + 1} = P3_updated;

    J1_values{i + 1} = 0.5 * x_initial' * P1 * x_initial;
    J2_values{i + 1} = 0.5 * x_initial' * P2 * x_initial;
    J3_values{i + 1} = 0.5 * x_initial' * P3 * x_initial;
end

% Define each policy accordingly
F1 = R11 \ (B1' * P1);
F2 = R22 \ (B2' * P2);
F3 = R33 \ (B3' * P3);

% Define Microsatellite Control Strategies
u1 = @(x) (-F1 * x);
u2 = @(x) (-F2 * x);
u3 = @(x) (-F3 * x);

% Define time span
time = [0, 300];
```

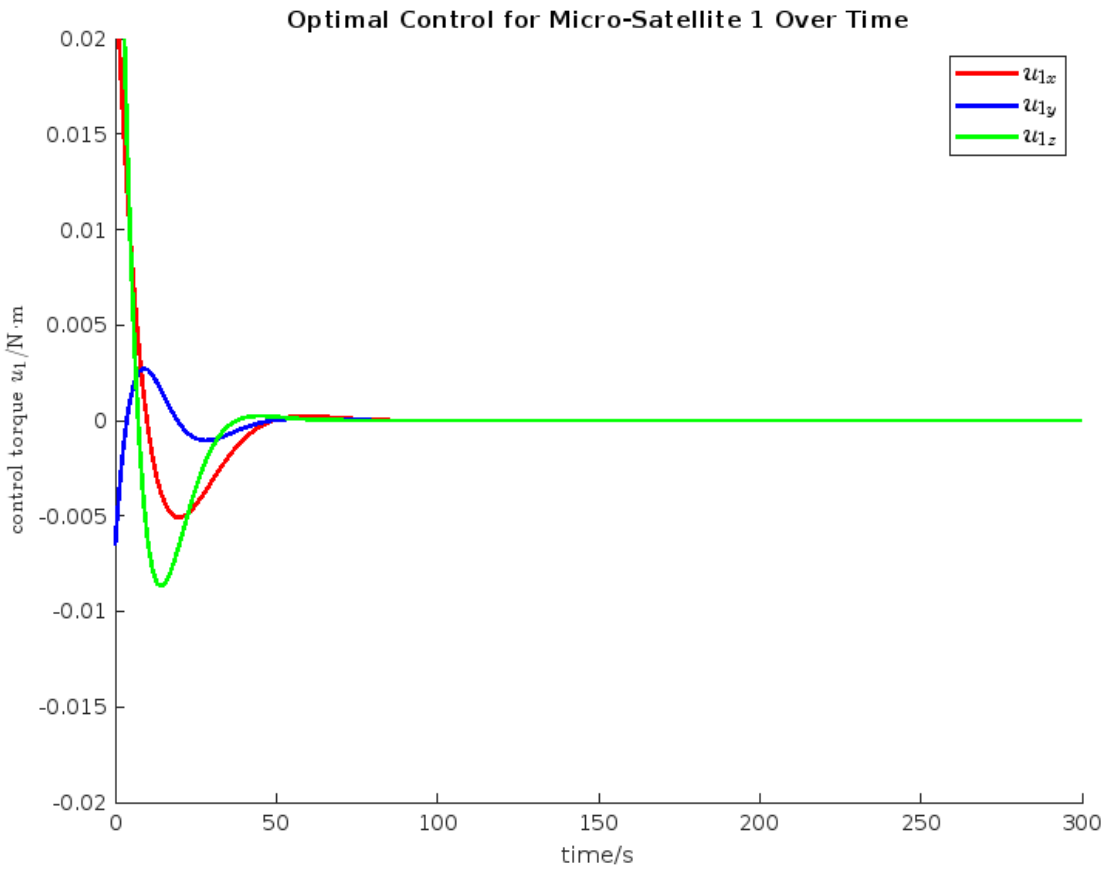
```
% Define the system dynamics function
sys_dynamics = @(t, x) (A * x + B1 * u1(x) + B2 * u2(x) + B3 * u3(x));
[t, x] = ode89(sys_dynamics, time, x_initial);

figure;
hold on;
plot(t, x(:, 1), 'r-', 'LineWidth', 2);
plot(t, x(:, 2), 'b-', 'LineWidth', 2);
plot(t, x(:, 3), 'g-', 'LineWidth', 2);
hold off;

% Add labels and legend
title('Optimal System State Over Time');
xlabel('time/s');
ylabel('attitude/rad');
legend('', '', '');

% Set x-axis limits
xlim([0, 300]);

% Set y-axis limits
ylim([-0.02, 0.1]);


figure;
hold on;
plot(t, x(:, 4), 'r-', 'LineWidth', 2);
plot(t, x(:, 5), 'b-', 'LineWidth', 2);
plot(t, x(:, 6), 'g-', 'LineWidth', 2);
hold off;

% Add labels and legend
title('Optimal System Angular Velocity Over Time');
xlabel('time/s');
ylabel('w/rad/s');
legend({'$w_x$', '$w_y$', '$w_z$'}, 'Interpreter', 'latex', 'FontSize', 12);

% Set x-axis limits
xlim([0, 300]);

% Set y-axis limits
ylim([-7 .* 10^(-3), 1 .* 10^(-3)]);

% Now we will be plotting control torques of the microsatellites

% Define time span
time = [0, 300];

% Define the system dynamics function
sys_dynamics = @(t, x) (A * x + B1 * u1(x) + B2 * u2(x) + B3 * u3(x));
[t, x] = ode89(sys_dynamics, time, x_initial);

figure;
hold on;
```

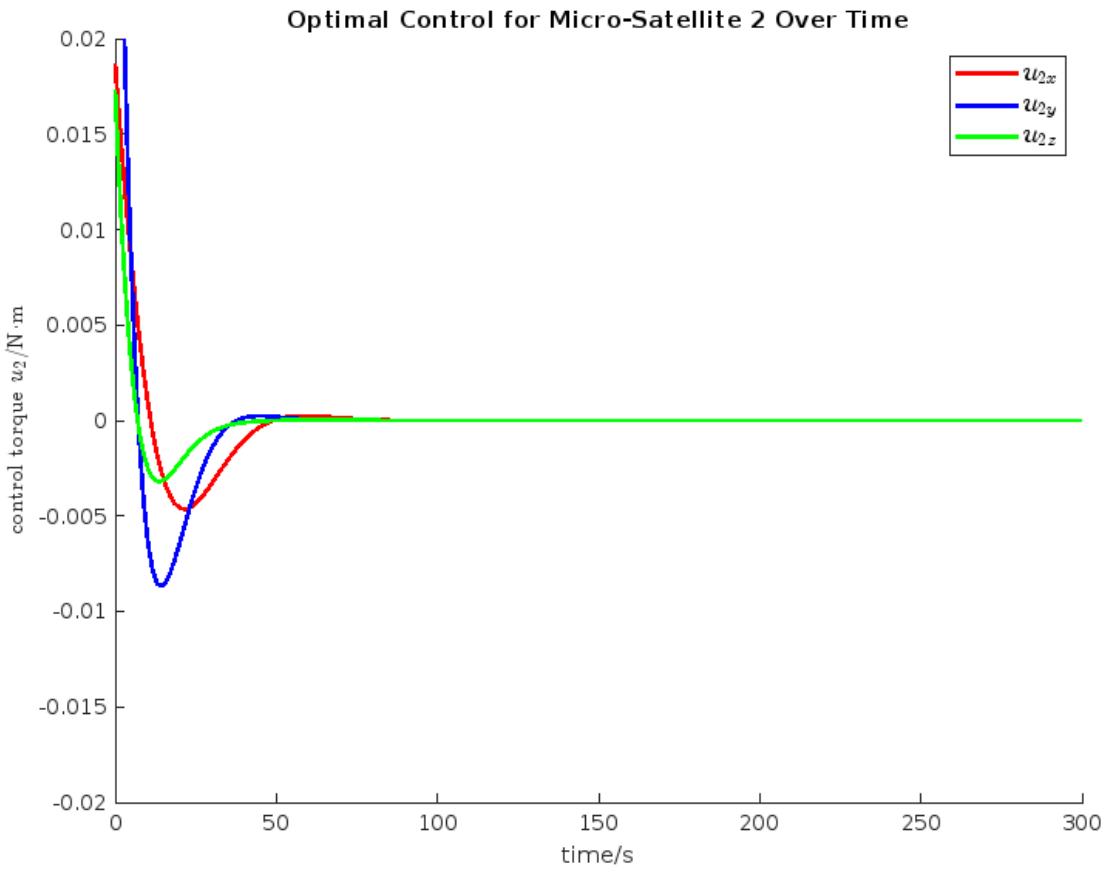
```
u1_values = u1(x');
u1_values = u1_values';
plot(t, u1_values(:, 1), 'r-', 'LineWidth', 2);
plot(t, u1_values(:, 2), 'b-', 'LineWidth', 2);
plot(t, u1_values(:, 3), 'g-', 'LineWidth', 2);
hold off;

% Add labels and legend
title('Optimal Control for Micro-Satellite 1 Over Time');
xlabel('time/s');
ylabel('control torque $u_1$/N$\cdot$m', 'Interpreter', 'latex');
legend({'$u_{1x}$', '$u_{1y}$', '$u_{1z}$'}, 'Interpreter', 'latex',
'FontSize', 12);

% Set x-axis limits
xlim([0, 300]);

% Set y-axis limits
ylim([-0.02, 0.02]);

figure;
hold on;
u2_values = u2(x');
u2_values = u2_values';
plot(t, u2_values(:, 1), 'r-', 'LineWidth', 2);
plot(t, u2_values(:, 2), 'b-', 'LineWidth', 2);
plot(t, u2_values(:, 3), 'g-', 'LineWidth', 2);
hold off;

% Add labels and legend
title('Optimal Control for Micro-Satellite 2 Over Time');
xlabel('time/s');
ylabel('control torque $u_2$/N$\cdot$m', 'Interpreter', 'latex');
legend({'$u_{2x}$', '$u_{2y}$', '$u_{2z}$'}, 'Interpreter', 'latex',
'FontSize', 12);

% Set x-axis limits
xlim([0, 300]);

% Set y-axis limits
ylim([-0.02, 0.02]);

figure;
hold on;
u3_values = u3(x');
u3_values = u3_values';
plot(t, u3_values(:, 1), 'r-', 'LineWidth', 2);
plot(t, u3_values(:, 2), 'b-', 'LineWidth', 2);
plot(t, u3_values(:, 3), 'g-', 'LineWidth', 2);
hold off;

% Add labels and legend
title('Optimal Control for Micro-Satellite 3 Over Time');
xlabel('time/s');
```

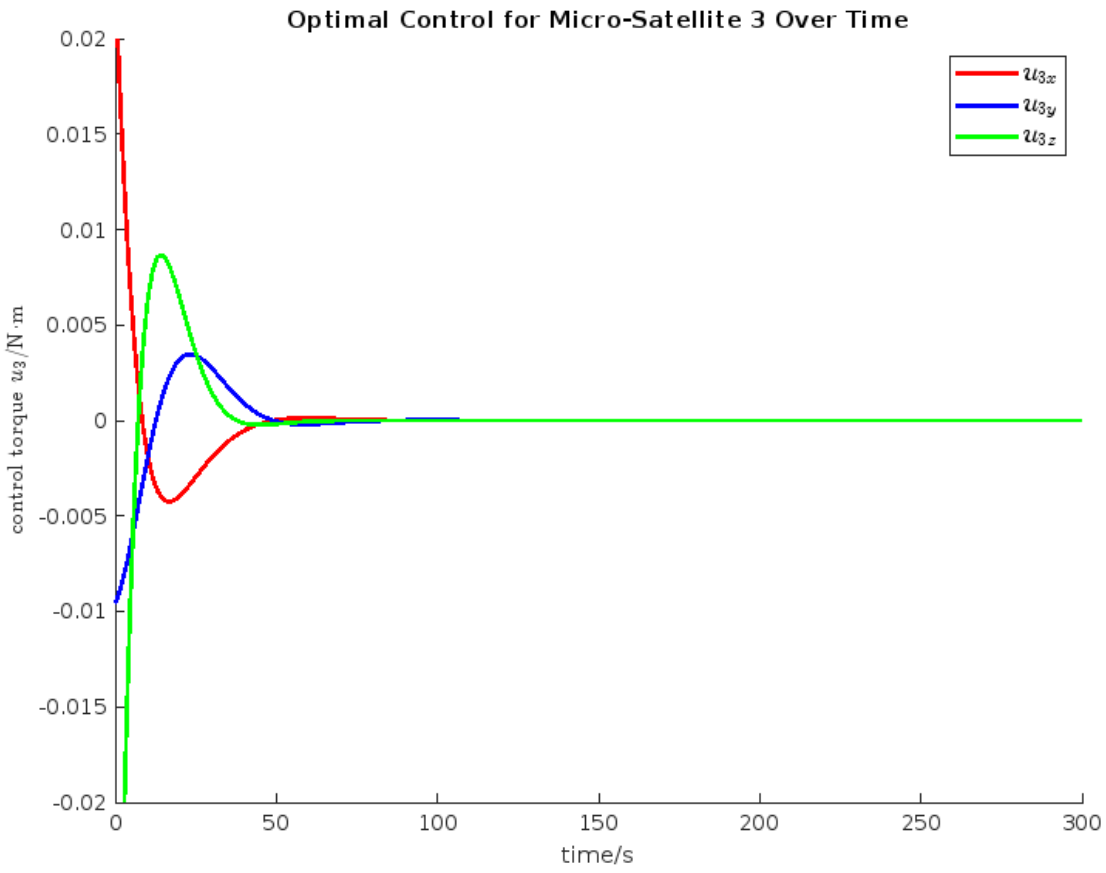
```
ylabel('control torque $u_3$/N$\cdot$m', 'Interpreter', 'latex');
legend({'$u_{3x}$', '$u_{3y}$', '$u_{3z}$'}, 'Interpreter', 'latex',
'FontSize', 12);

% Set x-axis limits
xlim([0, 300]);

% Set y-axis limits
ylim([-0.02, 0.02]);
```




Optimal System Angular Velocity Over Time
×10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
w/rad/s
0
50
100
150
200
250
300
time/s
w_x
w_y
w_z


Optimal Control for Micro-Satellite 1 Over Time
0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
control torque u_1/N·m
0
50
100
150
200
250
300
time/s
u_{1x}
u_{1y}
u_{1z}


Optimal Control for Micro-Satellite 2 Over Time
0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
control torque u_2/N·m
0
50
100
150
200
250
300
time/s
u_{2x}
u_{2y}
u_{2z}



Published with MATLAB® R2023b