```
%% William Valentine
% ECE885 Project 2
% H-infinity design for UAV pitch rate controller
clear; clc; close all;
s = tf('s');
```

Identified plant G_qde(s)

```

%               Mde(s + Zw)
% G(s) = ------------------------- * e^(-tau*s)
%         s^2 + 2*zeta*wn*s + wn^2

% Nominal parameters
Mde_nom  = -120.1;
Zw_nom   =  9.24;
wn_nom   = 10.9;
zeta_nom =  0.92;
tau_nom  =  0.0575;

% Min/Max Parameter Values From Sys ID
Mde_min  = -137.43;  Mde_max  = -105.65;
Zw_min   =  8.28;    Zw_max   = 10.33;
wn_min   = 10.11;    wn_max   = 11.32;
zeta_min =  0.85;    zeta_max = 0.99;
tau_min  =  0.05;    tau_max  = 0.06;

% Nominal plant transfer function
G_nom = (Mde_nom*s + Mde_nom*Zw_nom) / ...
    (s^2 + 2*zeta_nom*wn_nom*s + wn_nom^2);

% Approximate time delay with 1st-order pade
[numD,denD] = pade(tau_nom,1);    % 1st-order Pade
Gdelay = tf(numD, denD);
G      = Gdelay * G_nom; % Construct nominal plant with time delay

disp('Pitch-rate plant G(s) excluding Tau= q/δe:');
```

```
Pitch-rate plant G(s) excluding Tau= q/δe:
```

```
G_nom
```

```
G_nom =

     -120.1 s - 1110
  ---------------------
  s^2 + 20.06 s + 118.8

Continuous-time transfer function.
Model Properties
```

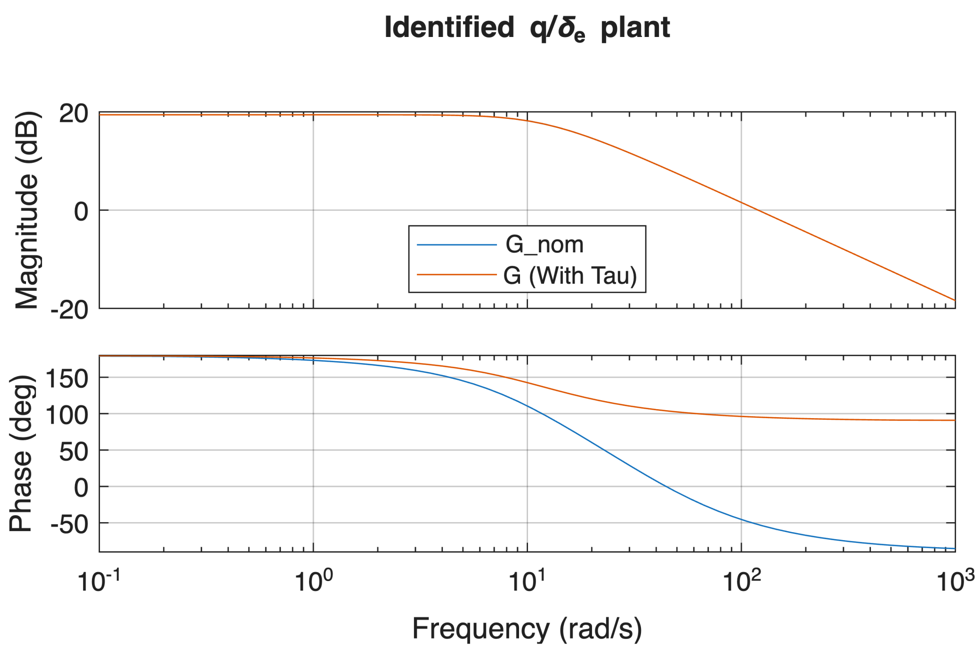
```
disp('Full Pitch-rate plant G(s) = q/δe:');
```

```
Full Pitch-rate plant G(s) = q/δe:
```

```
G
```

```
G =
 
    120.1 s^2 - 3068 s - 3.86e04
  --------------------------------
  s^3 + 54.84 s^2 + 816.4 s + 4133
 
Continuous-time transfer function.
Model Properties
```

```
figure; bode(G); grid on;
hold on;
bode(G_nom);
legend('G_nom','G (With Tau)','Location','Best');
title('Identified q/\delta_e plant');
```



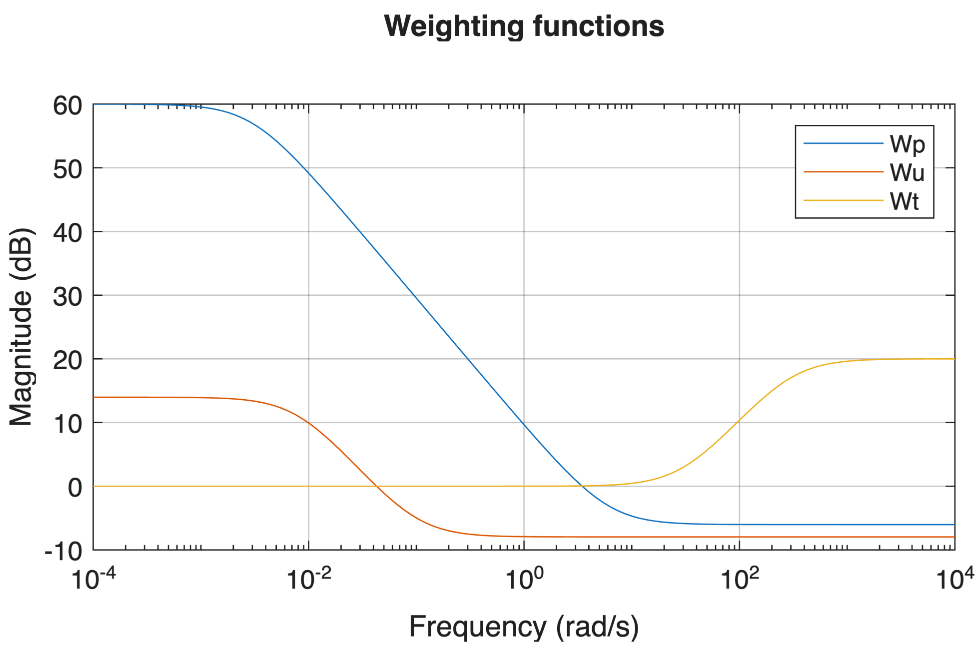
Performance specs

```
wb   = 3;       % 3 rad/s desired closed loop bandwidth
Ms    = 2;      % Desired peak sensitivity
A    = 0.001;  % Steady state tracking error
```

Weighting functions for mixed-sensitivity H-infinity design

```
% Wp: shape sensitivity S for tracking / disturbance rejection
Wp = (s/Ms + wb) / (s + wb*A);

% Wu: penalize control effort
Wu = 0.4 * (s + 0.1) / (s + 0.008);
```

```
% Wt: penalize high-frequency T for noise & robustness
w_noise = 30; % start penalizing around 30 rad/s
Wt = (s/w_noise + 1) / (s/(10*w_noise) + 1);

figure; bodemag(Wp, Wu, Wt); grid on;
legend('Wp','Wu','Wt');
title('Weighting functions');
```



H-infinity mixed-sensitivity synthesis

```
[Kh, CL, gamma_opt] = mixsyn(G, Wp, Wu, Wt);

disp('Optimal H-infinity gamma:');
```

```
Optimal H-infinity gamma:
```

```
gamma_opt
```

```
gamma_opt =
1.1419
```

```
disp('H-infinity controller K(s):');
```

```
H-infinity controller K(s):
```

```
Kh
```

```
Kh =

  A = 
              x1          x2          x3          x4          x5          x6
   x1     -0.003   2.633e-16  -6.166e-16   4.183e-15  -8.601e-15  -5.821e-15
```

```
    x2      -4.621      0.3645        17.7      -43.14       5.018       10.11
    x3  -2.267e-17  1.906e-18         -300       480.4      -383.5      -301.6
    x4      -295.8       23.84        1133       -2816       295.6       638.8
    x5   1.186e-17   -9.97e-19   2.335e-18          32   3.592e-17   2.324e-17
    x6   2.837e-17  -2.386e-18   5.586e-18  -3.591e-17          16   5.561e-17

  B =
                u1
    x1           2
    x2   9.742e-18
    x3   3.215e-18
    x4   3.316e-19
    x5  -1.682e-18
    x6  -4.023e-18

  C =
            x1      x2      x3      x4      x5      x6
    y1  -18.48    1.49   70.78  -172.6   20.07   40.43

  D =
        u1
    y1   0

Continuous-time state-space model.
Model Properties
```

Closed-Loop Sensitivity Functions

```
L  = minreal(G*Kh);        % loop transfer
```

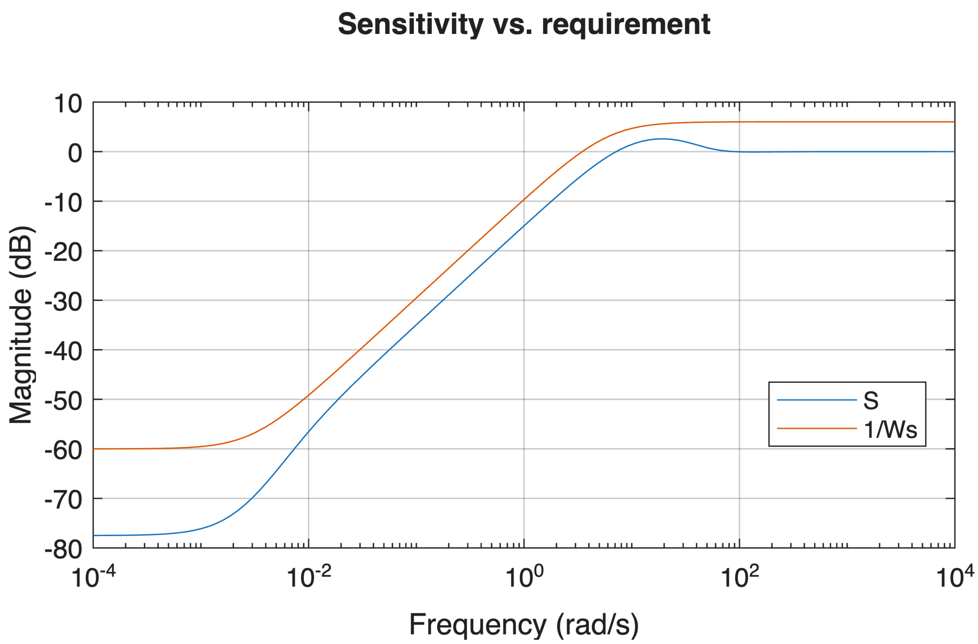
```
3 states removed.
```

```
S  = feedback(1, L);       % sensitivity
T  = feedback(L, 1);       % complementary sensitivity
KS = minreal(Kh*S);        % weighted control effort
```

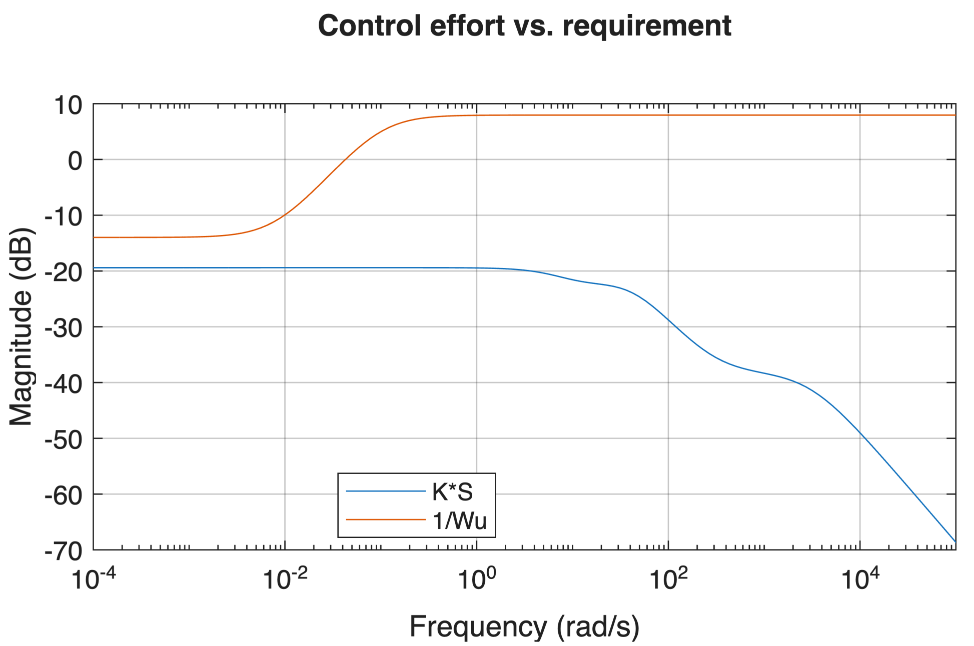
```
6 states removed.
```

Check Frequency-Domain Specs

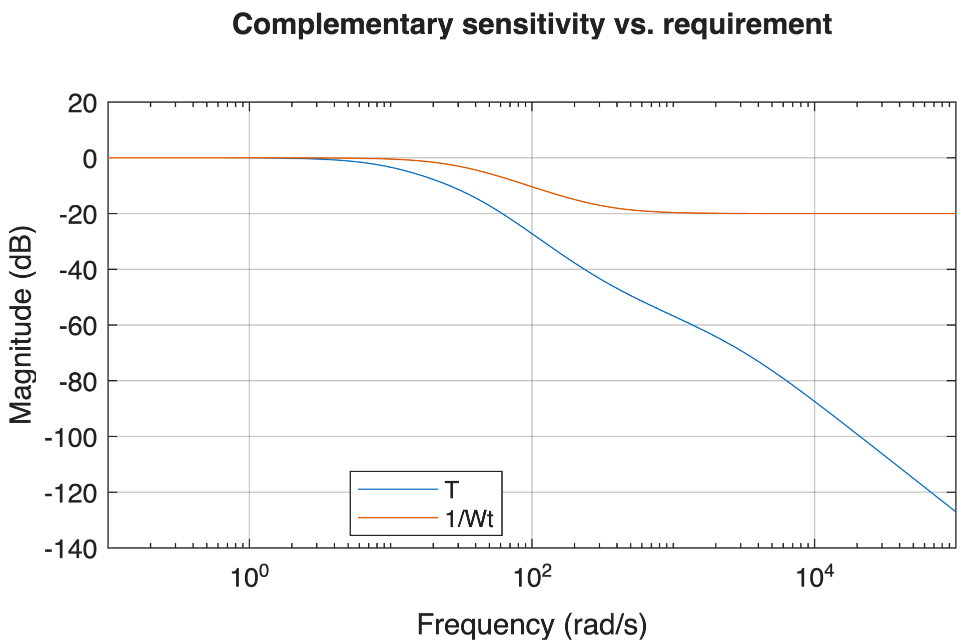
```
% Sensitivity
figure;
bodemag(S, 1/Wp); grid on;
legend('S','1/Ws','Location','Best');
title('Sensitivity vs. requirement');
```




```
% Control Effort
figure;
bodemag(KS, 1/Wu); grid on;
legend('K*S','1/Wu','Location','Best');
title('Control effort vs. requirement');
```

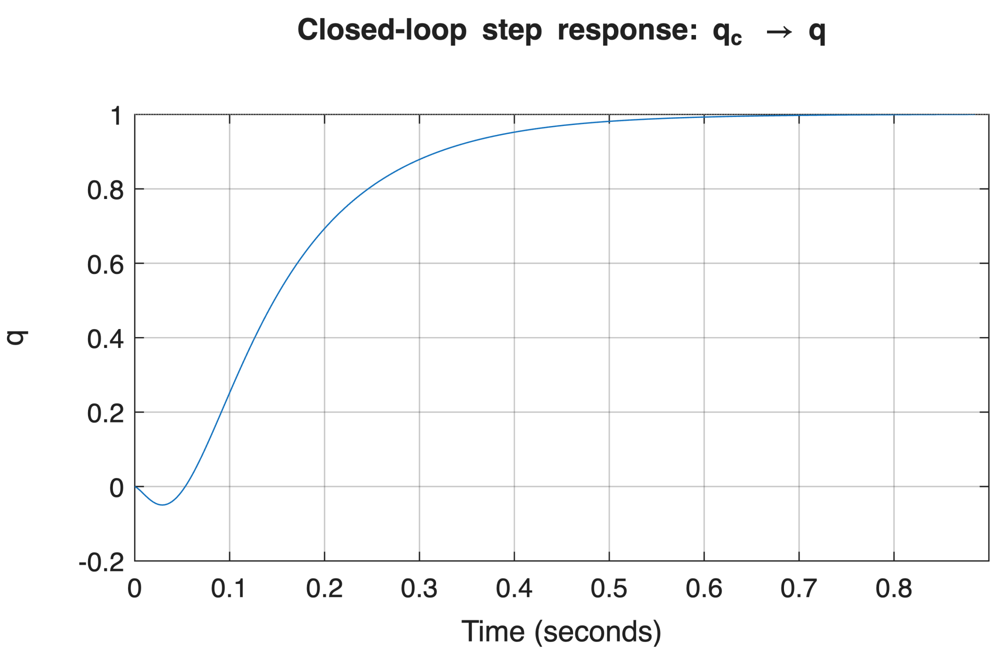



```
% Complementary Sensitivity
figure;
bodemag(T, 1/Wt); grid on;
legend('T','1/Wt','Location','Best');
title('Complementary sensitivity vs. requirement');
```



Time-domain simulations

```
% Step in pitch-rate command
figure; step(T); grid on;
title('Closed-loop step response: q_c \rightarrow q');
ylabel('q');
```

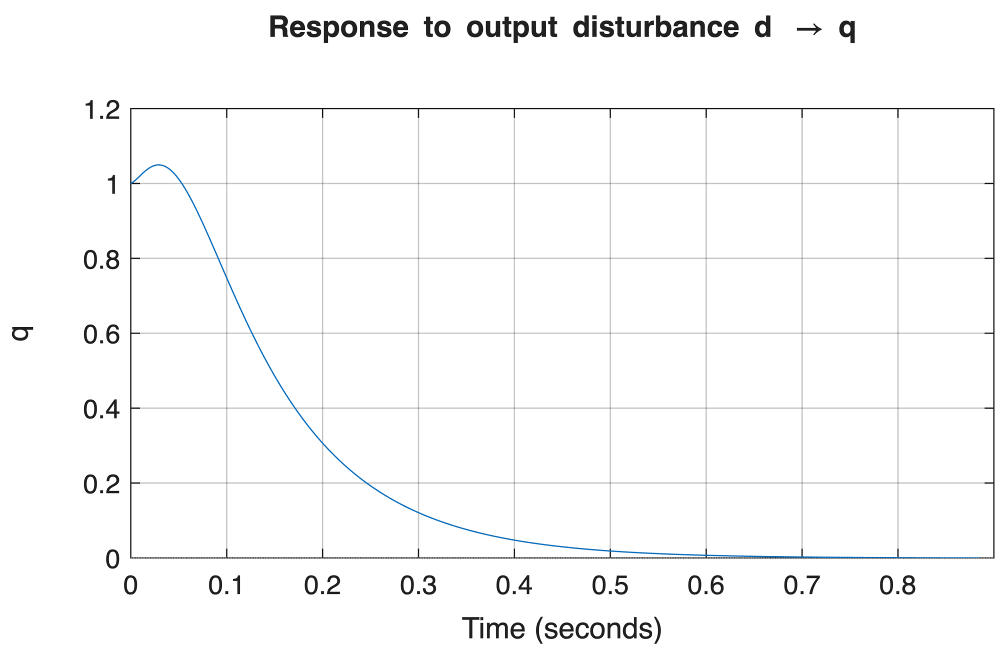



```
stepinfo(T)
```

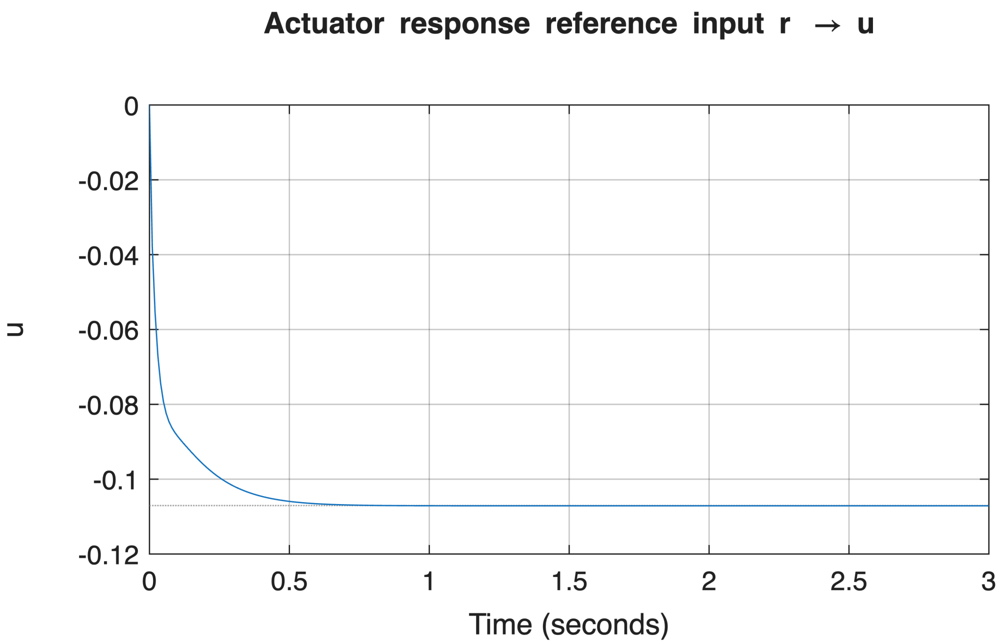
```
ans = struct with fields:
```

```
        RiseTime: 0.2464
   TransientTime: 0.4862
    SettlingTime: 0.4913
     SettlingMin: 0.8999
     SettlingMax: 1.0002
       Overshoot: 0.0379
      Undershoot: 4.9693
            Peak: 1.0002
        PeakTime: 0.9512
```

```
% Step in Disturbance at plant output
figure; step(S); grid on;
title('Response to output disturbance d \rightarrow q');
ylabel('q');
```



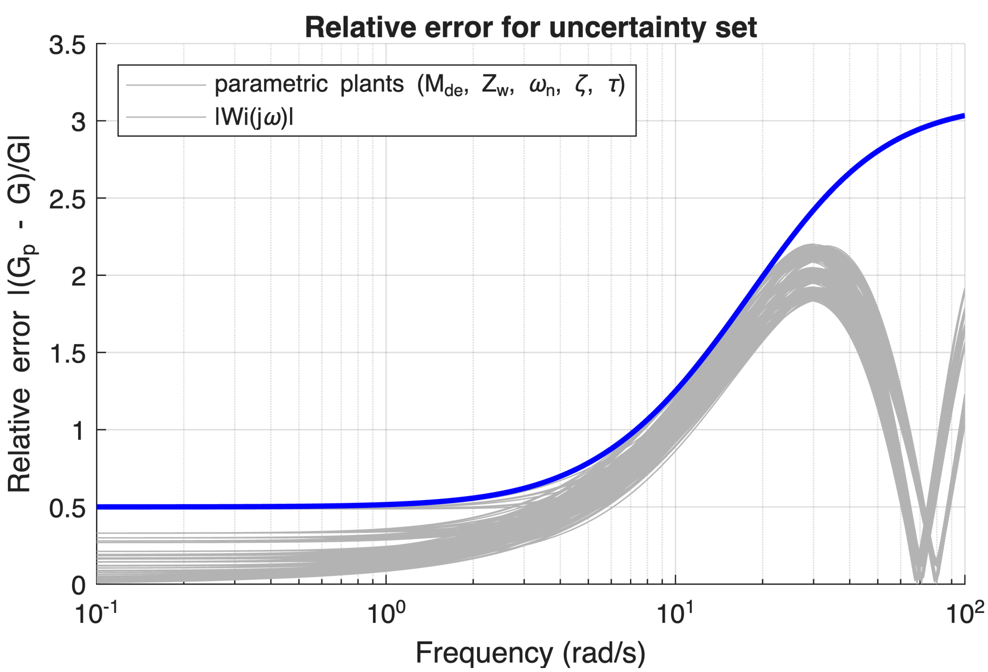

```
%% Control input
figure; step(Kh*S, 3); grid on;
title('Actuator response reference input r \rightarrow u');
ylabel('u');
```



Robust Control Analysis

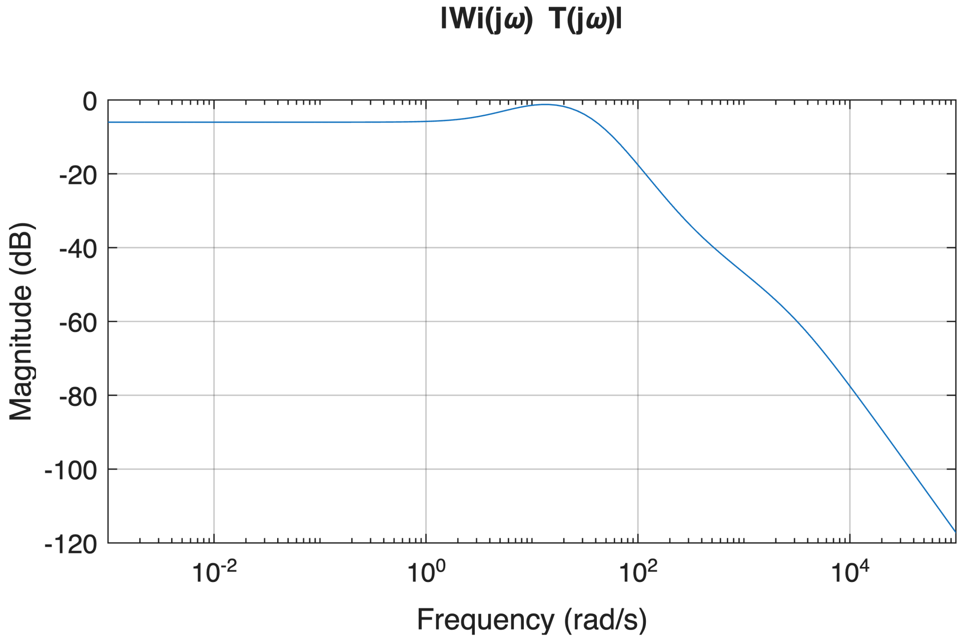
```
%% Add multiplicative uncertainty Wi
Wi = 0.5 * (s/4 + 1) / (s/25 + 1); %Shape Wi to encompass all uncertain
plants

% Variations for each parameters
Mde_grid   = [Mde_min,  Mde_nom,  Mde_max];
Zw_grid    = [Zw_min,   Zw_nom,   Zw_max];
wn_grid    = [wn_min,   wn_nom,   wn_max];
zeta_grid  = [zeta_min, zeta_nom, zeta_max];
tau_grid   = [tau_min,  tau_nom,  tau_max];

% Frequency grid
w = logspace(-1, 2, 400);   % 0.1–100 rad/s

% Nominal plant frequency response
G_nom_w = squeeze(freqresp(G, w));

figure; hold on; grid on;
title('Relative error for uncertainty set');
xlabel('Frequency (rad/s)');
ylabel('Relative error |(G_p - G)/G|');

for Mde_i   = Mde_grid
    for Zw_i    = Zw_grid
        for wn_i    = wn_grid
            for zeta_i = zeta_grid
```

```
                % Plant for this parameter combo
                Gp0 = (Mde_i*s + Mde_i*Zw_i) / ...
                    (s^2 + 2*zeta_i*wn_i*s + wn_i^2);
                Gp0_w = squeeze(freqresp(Gp0, w));  % freq response of
delay-free part

                % Sweep tau values
                for tau_i = tau_grid
                    delay_factor = exp(-1j * w * tau_i); % Pure delay
factor
                    Gp_w = Gp0_w .* delay_factor'; % Full plant with delay
                    E = abs((Gp_w - G_nom_w) ./ G_nom_w); % Relative error
                    loglog(w, E, 'Color', [0.7 0.7 0.7]);
                end
            end
        end
    end
end

% Overlay |Wi(jω)|
Wi_w = squeeze(freqresp(Wi, w));
loglog(w, abs(Wi_w), 'b', 'LineWidth', 2);
set(gca,'XScale','log');
legend('parametric plants (M_{de}, Z_w, \omega_n, \zeta, \tau)', ...
    '|Wi(j\omega)|', 'Location','NorthWest');
```




```
%% Check Wi * T < 1
WT = minreal(Wi * T);

figure; bodemag(WT); grid on;
title('|Wi(j\omega) T(j\omega)|');
```

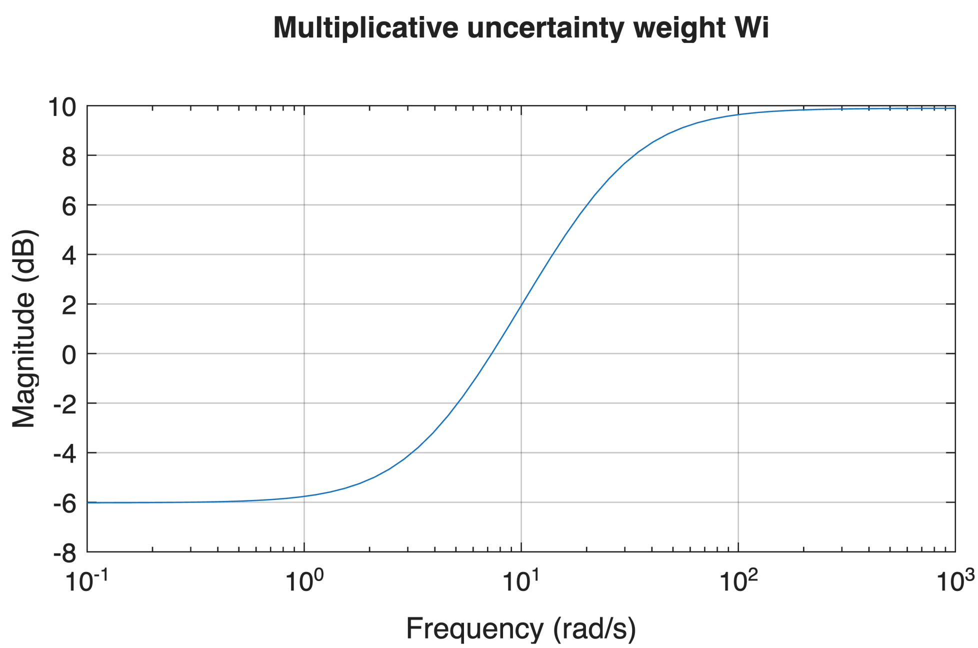
```
yline(0, '--');   % 0 dB line (|WT| = 1);
```




```
gamma_WT = norm(WT, inf); % Check infinity norm of |Wi*T|
fprintf('||Wi * T||_inf = %.3f\n', gamma_WT);
```

```
||Wi * T||_inf = 0.870
```

```
% Wi Frequency Response
figure; bodemag(Wi); grid on;
title('Multiplicative uncertainty weight Wi');
```




```
% Normalized, unknown stable LTI block with ||Delta||_inf <= 1
Delta = ultidyn('Delta',[1 1]);
```

```
% Multiplicative output uncertainty:
G_unc = G * (1 + Wi * Delta);

% Robust stability analysis with uncertainty

% Closed-loop from reference to output with uncertain plant
CL_unc = feedback(G_unc*Kh, 1);

[stabmarg, destabunc, report_stab] = robuststab(CL_unc);

disp('Robust stability report:');
```

```
Robust stability report:
```

```
disp(report_stab);
```

```
System is robustly stable for the modeled uncertainty.
 -- It can tolerate up to 115% of the modeled uncertainty.
 -- There is a destabilizing perturbation amounting to 115% of the modeled uncertainty.
 -- This perturbation causes an instability at the frequency 12.4 rad/seconds.
 -- Sensitivity with respect to each uncertain element is:
      100% for Delta. Increasing Delta by 25% decreases the margin by 25%.
```

```
disp('Robust stability margin (lower bound):');
```

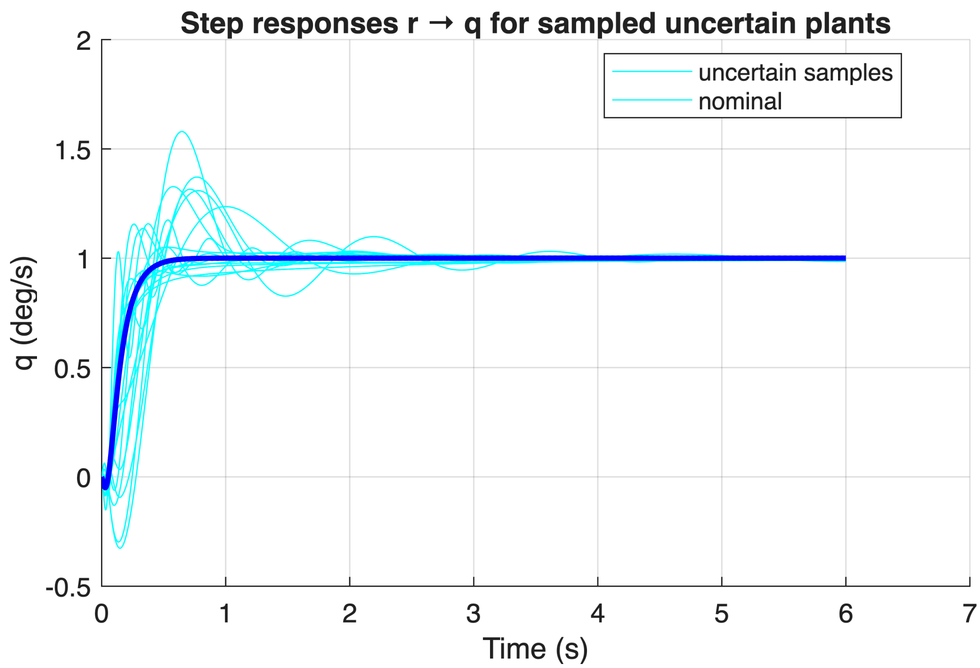
```
Robust stability margin (lower bound):
```

```
disp(stabmarg.LowerBound);
```

```
    1.1512
```

Monte-Carlo sampling of uncertain plants (step responses)

```
nsamples = 20;
tfinal   = 6;

figure; hold on; grid on;
title('Step responses r → q for sampled uncertain plants');
xlabel('Time (s)'); ylabel('q (deg/s)');

for k = 1:nsamples
    Gk = usample(G_unc); %Sample some random plants within the
multiplicative uncertainty set
    CLk = feedback(Gk*Kh, 1);
    [y,t] = step(CLk, tfinal);
    plot(t, y, 'c','LineWidth', 0.5);
end

% Plot nominal response on top (thicker line)
[y_nom, t_nom] = step(T, tfinal);
plot(t_nom, y_nom, 'b', 'LineWidth', 2);
legend('uncertain samples','nominal','Location','Best');
```



Frequency-domain view of uncertain loop

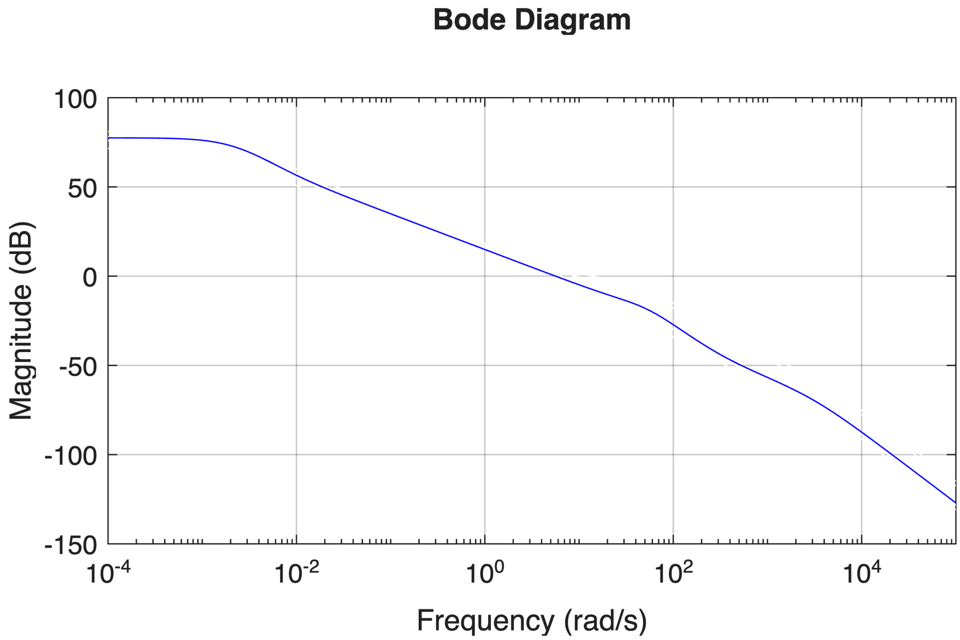
```
% Sample a few plants and plot L bode
figure; hold on;
title('Sensitivity function for sampled uncertain plants');
for k = 1:5
    Gk = usample(G_unc); %Sample some random plants within the
multiplicative uncertainty set
    Lk = minreal(Gk*Kh);
    bodemag(Lk,'w');
end
```

```
3 states removed.
3 states removed.
3 states removed.
3 states removed.
3 states removed.
```

```
L_nom = minreal(G*Kh);
```

```
3 states removed.
```

```
bodemag(L_nom,'b');   % nominal plant
grid on;
```



Test controller on UAV

```
%% Discretize controller for 100 hz
Ts = 0.01;  % 100 Hz
Kh_d = c2d(Kh, Ts, 'tustin');
disp('Discretized H_infinity controller');
```

```
Discretized H_infinity controller
```

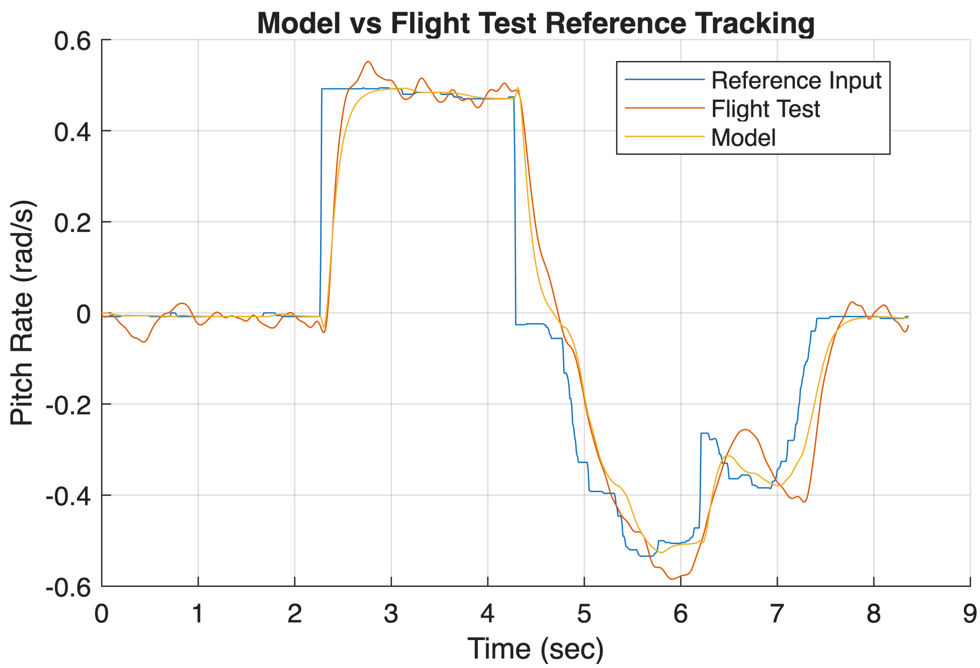
```
Kh_d %Load the discretized controller onto flight controller for testing
```

```
Kh_d =

  A = 
               x1          x2          x3          x4          x5          x6
   x1           1   2.896e-18   2.527e-18   2.159e-18  -9.199e-17  -5.498e-17
   x2   -0.005927           1    0.009079    -0.02725   -0.007466   0.0003741
   x3     -0.2432      0.0196      0.1726      0.1642      -2.104      -1.243
   x4     -0.2929     0.02361      0.4487     -0.8023     -0.5709    -0.04383
   x5    -0.04687    0.003777     0.07179     0.03163      0.9087   -0.007014
   x6   -0.003749   0.0003022    0.005743    0.002531      0.1527      0.9994

  B = 
               u1
   x1        0.02
   x2  -5.927e-05
   x3   -0.002432
   x4   -0.002929
   x5  -0.0004687
   x6  -3.749e-05

  C = 
           x1      x2      x3      x4      x5      x6
   y1  -2.371  0.1911   3.632   -10.9  -2.986  0.1496

  D = 
```

```
            u1
   y1  -0.02371

Sample time: 0.01 seconds
Discrete-time state-space model.
Model Properties
```

Flight test step input time histories

```
flightData = load('P_CL_Step_M50D02_run2.mat');
startIndex = 964;
stopIndex = 1800;

figure; hold on; grid on;

time = flightData.time(startIndex:stopIndex);
time = time - time(1);
q_raw   = flightData.q(startIndex:stopIndex);
input = flightData.lnStkStm(startIndex:stopIndex);
Fc = 5; %5 hz filter cutoff
q_filt  = lowpass(q_raw, Fc, 100);%Low pass filter measured pitch rate data

% Simulated response
q_sim = lsim(T,input,time);

plot(time,input);
plot(time,q_filt);
plot(time,q_sim);
legend('Reference Input', 'Flight Test', 'Model','Location','Best');
xlabel('Time (sec)');
ylabel('Pitch Rate (rad/s)');
title('Model vs Flight Test Reference Tracking')
```



Flight test frequency responses

```
load('PFWUAVCL_COM_ABCDE_ele_blo.mat'); %Import frequency response data
obtained from CIFER

w_cifer = freq(96:923); % Import frequency range (rad/s)

%mag_cifer_dB  = mag2db(db2mag(mag(96:923))/2); %%% I DON'T KNOW WHERE THIS
FACTOR OF 2 CAME FROM!!! (BUT IT WORKS)
mag_cifer_dB=mag(96:923);
phase_cifer_deg = phase(96:923);

% Evaluate loop transfer function on the CIFER frequencies
[magL, phaseL] = bode(L, w_cifer);

magL = squeeze(magL);
phaseL = squeeze(phaseL);

magL_dB = 20*log10(magL);
phaseL_deg = phaseL-360; %Phase shift to align with CIFER

% Plot overlay
figure;

% Magnitude plot
subplot(2,1,1);
semilogx(w_cifer, magL_dB, 'b-', 'LineWidth', 1.5); hold on;
semilogx(w_cifer, mag_cifer_dB, 'ro', 'LineWidth', 1.5, 'MarkerSize', 3);
```

```
grid on;
ylabel('Magnitude (dB)');
legend('Model L(j\omega)', 'Flight Test', 'Location', 'Best');

% Phase plot
subplot(2,1,2);
semilogx(w_cifer, phaseL_deg, 'b-', 'LineWidth', 1.5); hold on;
semilogx(w_cifer, phase_cifer_deg, 'ro', 'LineWidth', 1.5, 'MarkerSize', 3);
grid on;
xlabel('\omega (rad/s)');
ylabel('Phase (deg)');
legend('Model L(j\omega)', 'Flight Test', 'Location', 'Best');
sgtitle('Nominal model vs. flight test identified loop transfer function');
```