MECH 6323 - HW 07

Author: Jonas Wagner

Date: 2022-05-08

Problem 1

In this problem, you will design a control law for a nano-positioning stage. These devices can achieve very high precision positioning which is important in applications such as atomic force microscopes (AFMs). The right side of Figure 1 shows a feedback diagram of a nanopositioning device. The system consists of piezo-electric actuation, a flexure stage, and a detection system. As illustrated in the feedback diagram, the flexure stage interacts with the head of an AFM. The left side of Figure 1 shows a diagram of the flexure stage for a nanopositioning device. Typical design requirements for the control law include high bandwidth, high resolution and good robustness.

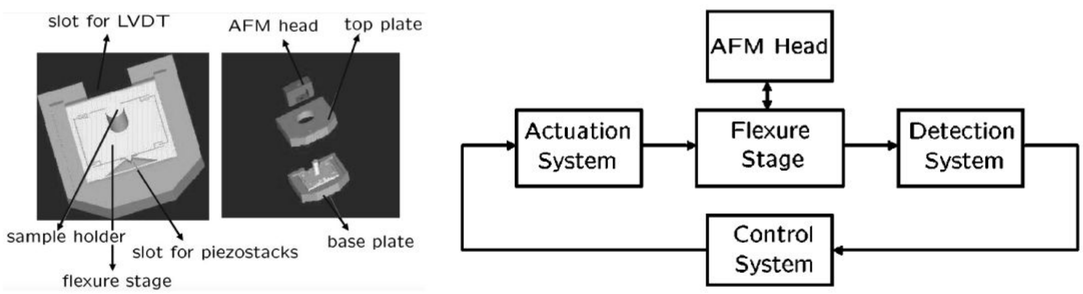


Fig. 1: Nanopositioning flexure stage (left) and feedback diagram (right); figures adapted from Salapaka et. al, *Rev. Sci. Instrum.* 2002.

```
clear
close all
```

Part a

```
nano_rsp = load('npresp.mat')
```

```
nano_rsp = struct with fields:
    Gfr: [1×1 frd]
      w: [1748×1 double]
      G: [1×1×1748 double]
```

```
omega_min = min(nano_rsp.w);
omega_max = max(nano_rsp.w);
```

Estimate system transfer function

```
tf_order = 6;
G_sys = fitfrd(nano_rsp.Gfr, tf_order)
```

```
G_sys =
```

```
  A = 
            x1      x2      x3      x4      x5      x6
   x1    -1468    8540   -8540    8540   -8540    4270
   x2    -4681    6559   -3757    3757   -3757    1878
   x3    -2174    4348   -7150    9952   -9952    4976
   x4    -4695    9390   -9390    6588   -3786    1893
   x5    -1687    3374   -3374    3374   -6176    4489
   x6    -3428    6856   -6856    6856   -6856   625.7
 
  B = 
            u1
   x1    5.156
   x2    11.36
   x3     16.7
   x4    18.15
   x5    11.59
   x6    11.18
 
  C = 
             x1       x2       x3       x4       x5       x6
   y1     114.1   -228.1    228.1   -228.1    228.1   -114.1
 
  D = 
              u1
   y1    0.08876
 
Continuous-time state-space model.
```

```
G_sys_tf = tf(G_sys)
```

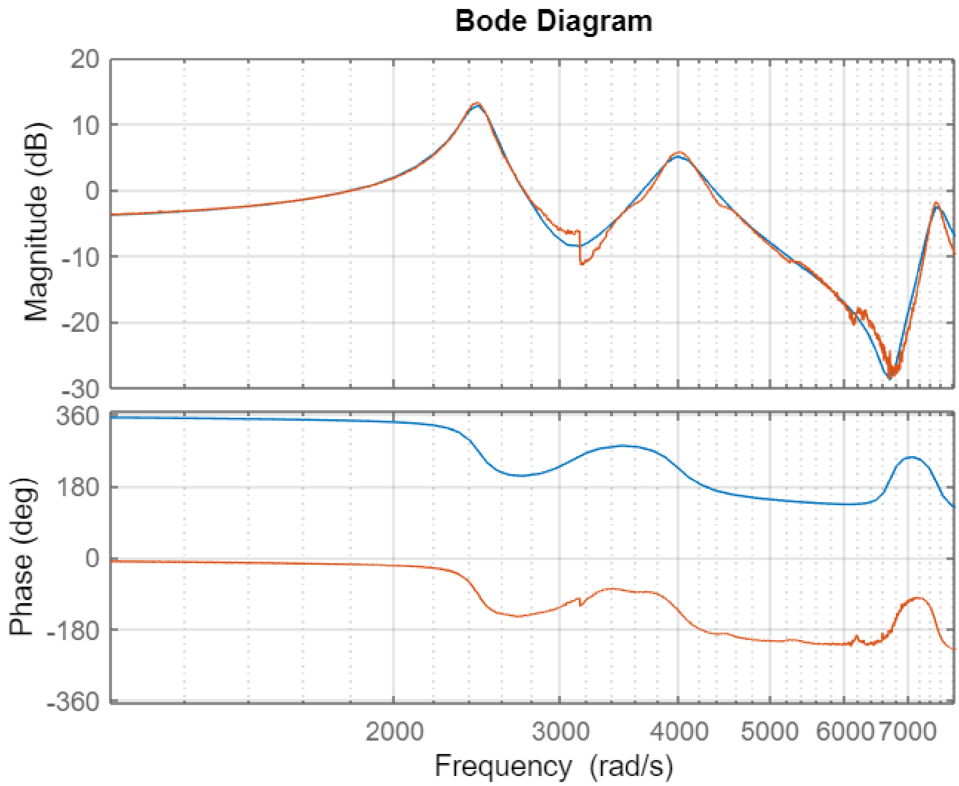
```
G_sys_tf =
 
  0.08876 s^6 - 876.1 s^5 + 1.136e07 s^4 - 4.345e10 s^3 + 4.097e14 s^2 - 2.095e17 s + 3.082e21
  ---------------------------------------------------------------------------------------------
       s^6 + 1021 s^5 + 7.856e07 s^4 + 5.129e10 s^3 + 1.342e15 s^2 + 3.65e17 s + 5.421e21
 
Continuous-time transfer function.
```

```
G_sys_zpk = zpk(G_sys)
```

```
G_sys_zpk =
 
  0.088762 (s^2 + 526.7s + 9.417e06) (s^2 + 276.6s + 4.494e07) (s^2 - 1.067e04s + 8.207e07)
  ------------------------------------------------------------------------------------------
         (s^2 + 186.2s + 6.029e06) (s^2 + 482.7s + 1.6e07) (s^2 + 352.5s + 5.621e07)
 
Continuous-time zero/pole/gain model.
```

Bode Diagram Data

```
figure()
bode(G_sys)
hold on
bode(nano_rsp.Gfr)
grid on
xlim([omega_min, omega_max])
```



Clearly this plot is a good estimation for the frequency response of the system, noting that the phase of the system is offset by a 360 degree phase shift (which implies a need for another set of integrators)

Part b

PI - Implimentation

```
PM_min = 75;
opt = pidtuneOptions( ...
    'PhaseMargin', PM_min, ...
    'DesignFocus', 'disturbance-rejection' ...
    )
```

```
opt =
  pidtune with properties:

         PhaseMargin: 75
    NumUnstablePoles: 0
         DesignFocus: 'disturbance-rejection'
```

```
[C_pi, info] = pidtune(G_sys, 'pi', opt)
```

```
C_pi =

        1
  Ki * ---
        s

  with Ki = 240
```

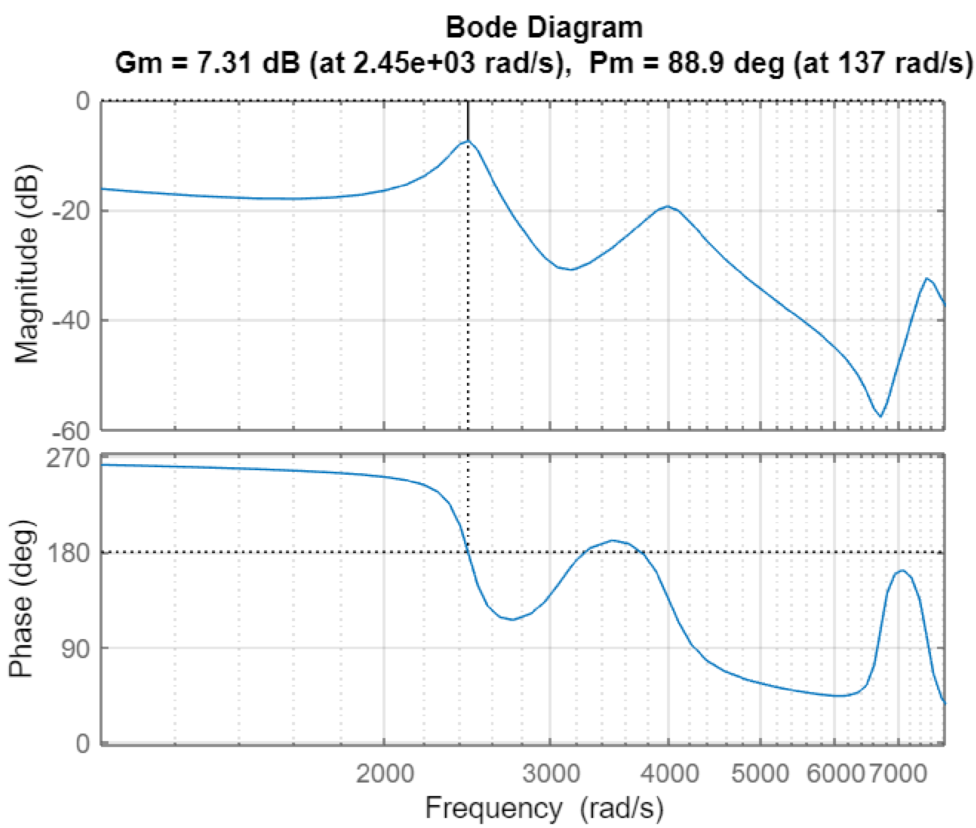
```
Continuous-time I-only controller.
info = struct with fields:
               Stable: 1
    CrossoverFrequency: 136.5946
           PhaseMargin: 88.9408
```

```
allmargin(C_pi * G_sys)
```

```
ans = struct with fields:
     GainMargin: [2.3213 31.8215 13.3190]
    GMFrequency: [2.4472e+03 3.2556e+03 3.7346e+03]
    PhaseMargin: 88.9408
    PMFrequency: 136.5946
    DelayMargin: 0.0114
    DMFrequency: 136.5946
         Stable: 1
```

Bode Diagram of Margin

```
figure
margin(C_pi * G_sys_tf)
xlim([omega_min, omega_max])
grid on
```



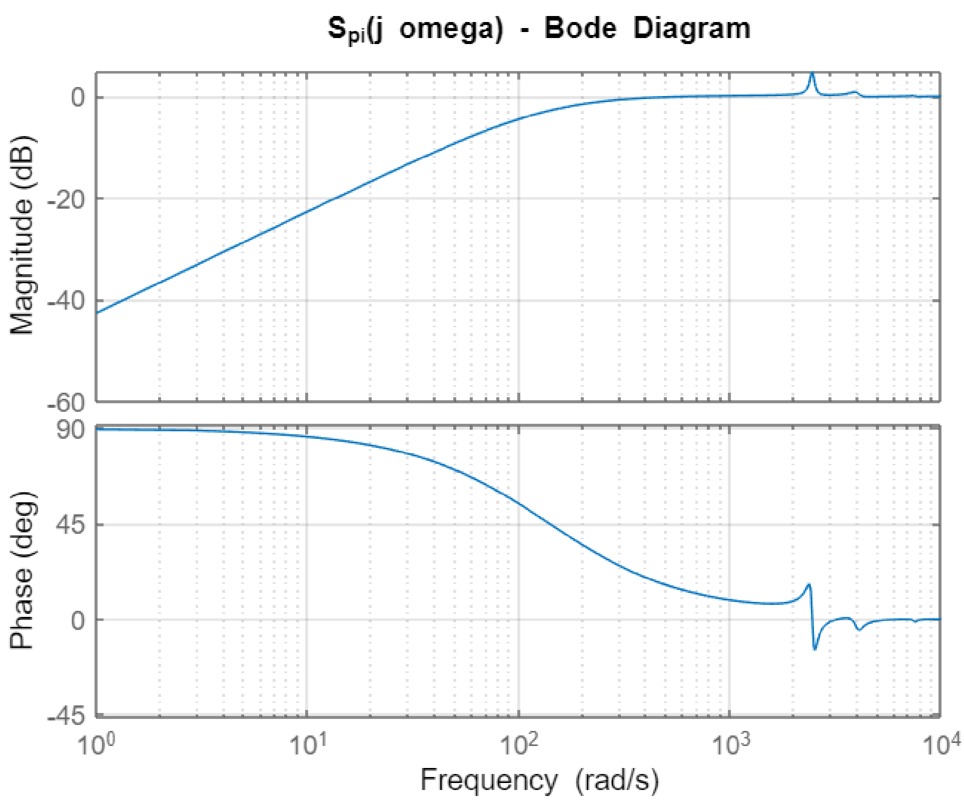
As a result, the single only integral controller is a pretty weird result. However, looking at the results it does make sense.

Sensivity Transfer Function

```
S_pi = zpk(1/(1+C_pi*G_sys))
```

```
S_pi =

       s (s^2 + 186.2s + 6.029e06) (s^2 + 482.7s + 1.6e07) (s^2 + 352.5s + 5.621e07)
  --------------------------------------------------------------------------------------
  (s+138.6) (s^2 + 108.1s + 5.995e06) (s^2 + 446.2s + 1.584e07) (s^2 + 349.8s + 5.615e07)

Continuous-time zero/pole/gain model.
```

```
figure
bode(S_pi)
title('S_{pi}(j omega) - Bode Diagram')
grid on
```

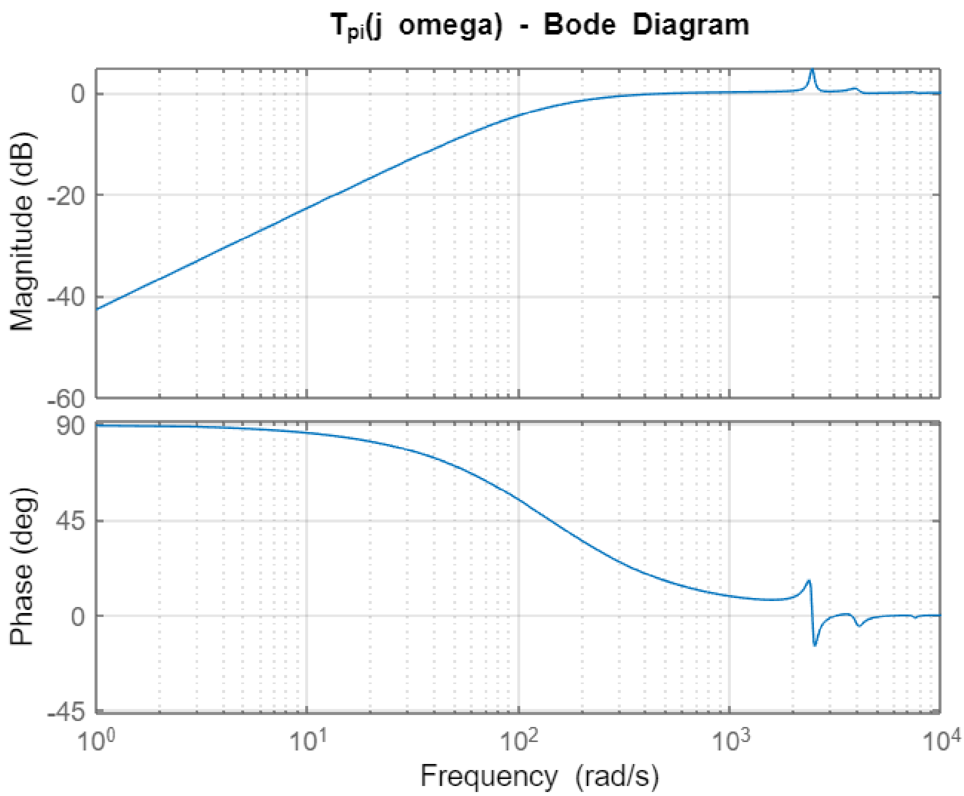


Complimentary Transfer Function

```
T_pi = zpk(1/(1+C_pi*G_sys))
```

```
T_pi =

       s (s^2 + 186.2s + 6.029e06) (s^2 + 482.7s + 1.6e07) (s^2 + 352.5s + 5.621e07)
  --------------------------------------------------------------------------------------
  (s+138.6) (s^2 + 108.1s + 5.995e06) (s^2 + 446.2s + 1.584e07) (s^2 + 349.8s + 5.615e07)

Continuous-time zero/pole/gain model.
```

```
figure
bode(T_pi)
title('T_{pi}(j omega) - Bode Diagram')
grid on
```



Bandwith Calculation

```
bw_threshold = -3; %db
bw = getGainCrossover(S_pi, db2mag(bw_threshold))
```

```
bw = 134.4361
```

Double Integrator implimenation

Instead we can also include an additional integrator into the controller, i.e.

```
G_int = tf(1,[1, 0]);
[C_pi_int, info] = pidtune(G_int * G_sys, 'pi', opt)
```

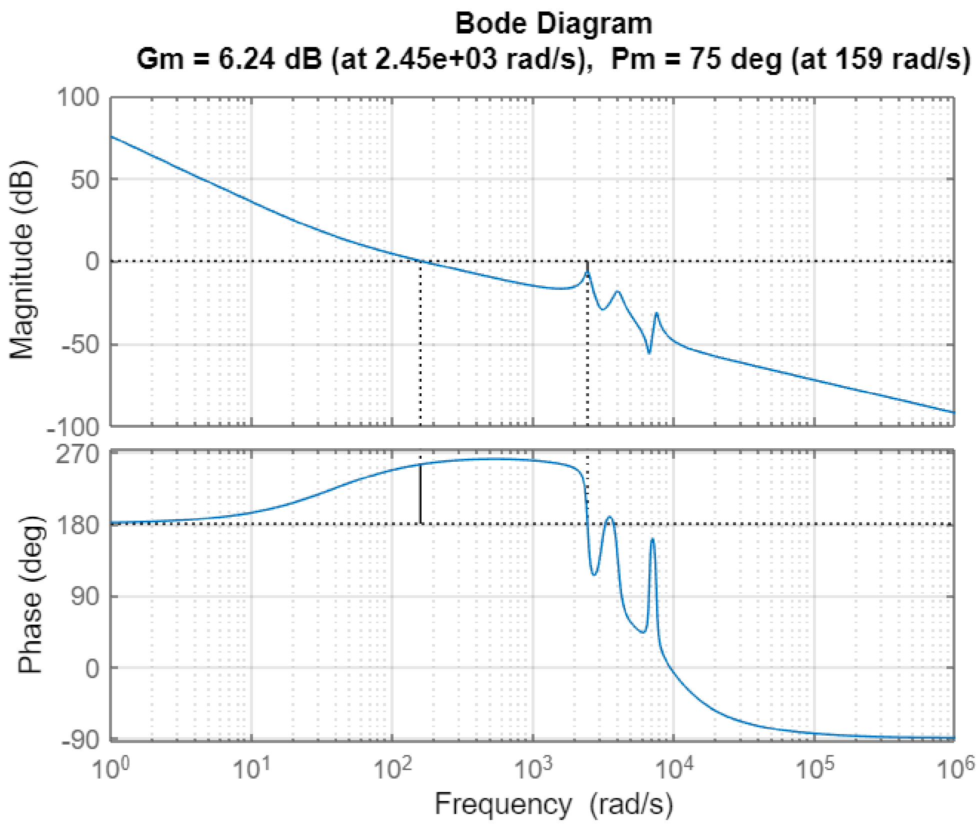
```
C_pi_int =

             1
  Kp + Ki * ---
             s

  with Kp = 271, Ki = 1.06e+04

Continuous-time PI controller in parallel form.
info = struct with fields:
                Stable: 1
    CrossoverFrequency: 159.0729
           PhaseMargin: 75.0000
```

```
allmargin(C_pi_int * G_int * G_sys)
```

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

```
      GainMargin: [0 2.0518 27.8933 11.9425]
     GMFrequency: [0 2.4457e+03 3.2626e+03 3.7282e+03]
     PhaseMargin: 75.0002
     PMFrequency: 159.0759
     DelayMargin: 0.0082
     DMFrequency: 159.0759
          Stable: 1
```

```
figure
margin(C_pi_int * G_int * G_sys)
grid on
```

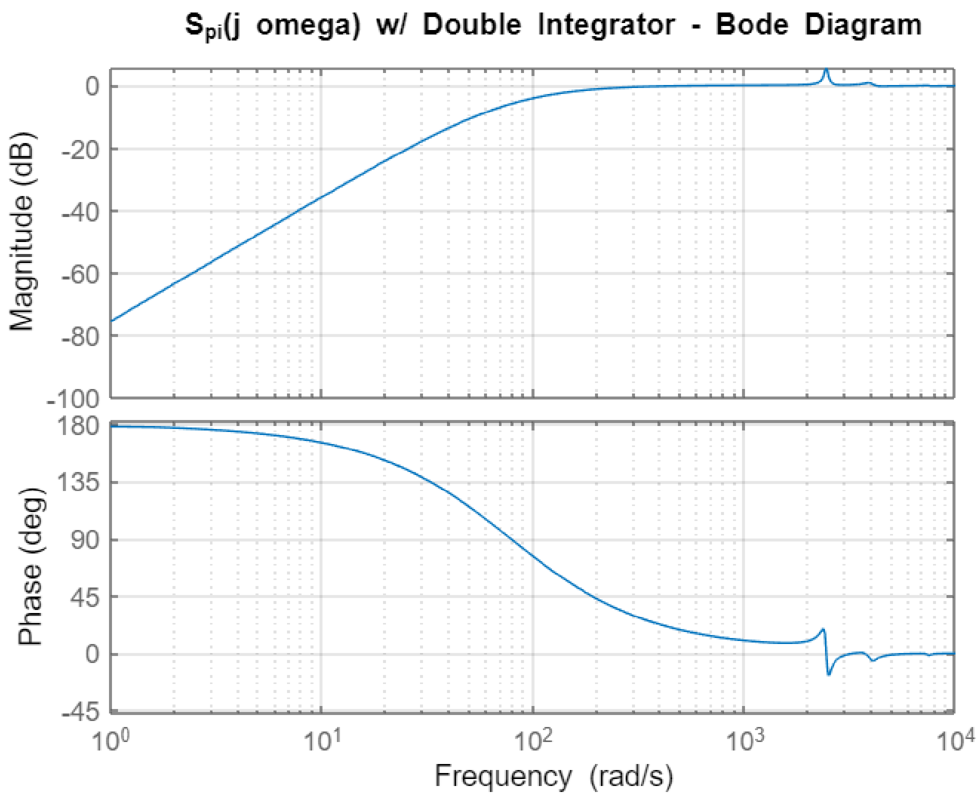


Sensivity Transfer Function

```
S_pi_int = zpk(1/(1+C_pi_int*G_int*G_sys))
```

```
S_pi_int =
 
            s^2 (s^2 + 186.2s + 6.029e06) (s^2 + 482.7s + 1.6e07) (s^2 + 352.5s + 5.621e07)
  ----------------------------------------------------------------------------------------------------
  (s^2 + 156.3s + 6120) (s^2 + 97.81s + 5.989e06) (s^2 + 441.9s + 1.582e07) (s^2 + 349.5s + 5.614e07)
 
Continuous-time zero/pole/gain model.
```

```
figure
bode(S_pi_int)
title('S_{pi}(j omega) w/ Double Integrator - Bode Diagram')
grid on
```

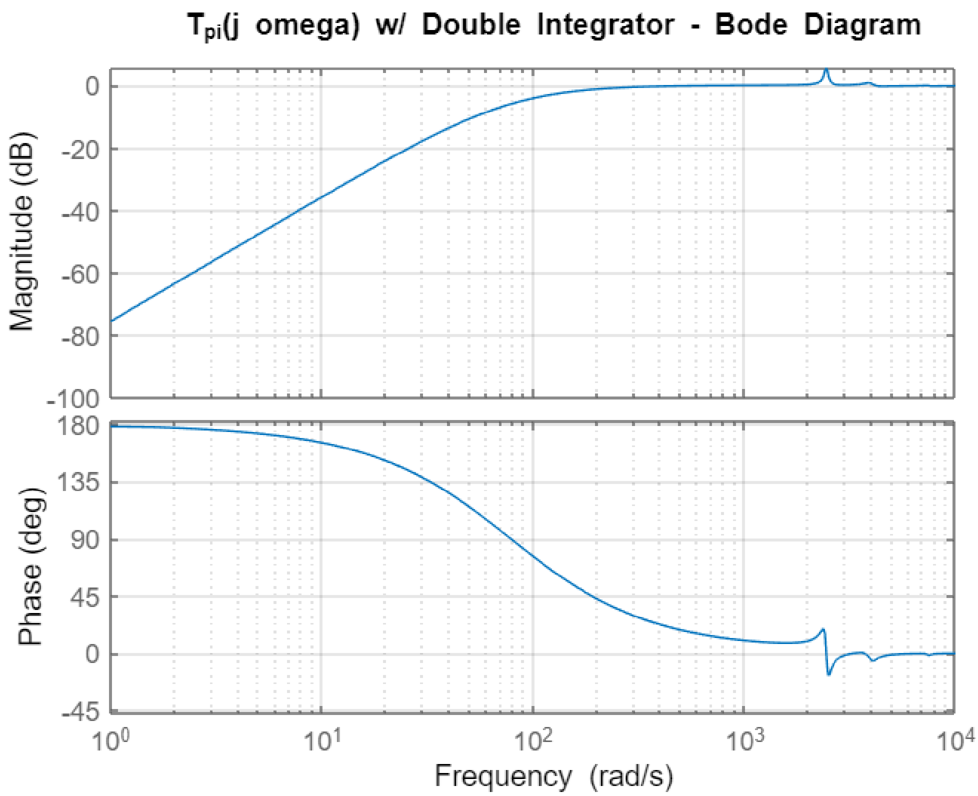


Complimentary Transfer Function

```
T_pi_int = zpk(1/(1+C_pi_int*G_int*G_sys))
```

```
T_pi_int =

           s^2 (s^2 + 186.2s + 6.029e06) (s^2 + 482.7s + 1.6e07) (s^2 + 352.5s + 5.621e07)
  ---------------------------------------------------------------------------------------------------
  (s^2 + 156.3s + 6120) (s^2 + 97.81s + 5.989e06) (s^2 + 441.9s + 1.582e07) (s^2 + 349.5s + 5.614e07)

Continuous-time zero/pole/gain model.
```

```
figure
bode(T_pi_int)
title('T_{pi}(j omega) w/ Double Integrator - Bode Diagram')
grid on
```



Bandwith Calculation

```
bw_threshold = -3; %db
bw = getGainCrossover(S_pi_int, db2mag(bw_threshold))
```

```
bw = 117.7945
```

The bandwidth of this method is not as great as the original PI implimenation. I think the performance is better in certain situation though and may be worth implimenting (even if it isn't really a PI controller)

Part c

Specs:

1. Bandwidth ($|S(j\omega)| = -3$ dB) is around 250 Hz
2. $|S(j\omega)| \leq 1.5 \ \ \forall_\omega$
3. Slope below bandwidth = 20 dB/decade
4. DC gain of $S \leq -80$dB
5. $|T(j\omega)| < -3$dB @ 500 Hz
6. $|T(j\omega)| \leq 1.5 \ \ \forall_\omega$
7. $|T(j\omega)| < -40$dB as $\omega \to \infty$
8. $|C_\infty S(j\omega)| \leq 10 \ \ \forall_\omega$

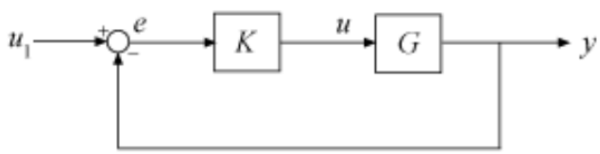
Design Weights

In order design using the mixed sensitivity design approach, we shape $S_{pi}(s)$ and $T_{pi}(s)$ to achieve the desired performance and robustness specs using weighting functions that are inverse of thoose desired shapes.

`[K,CL,gamma,info] = mixsyn(G,W1,W2,W3)` computes a controller that minimizes the H_∞ norm of the weighted closed-loop transfer function

$$M(s) = \begin{bmatrix} W_1 S \\ W_2 KS \\ W_3 T \end{bmatrix},$$

where $S = (I + GK)^{-1}$ and $T = (I - S)$ is the complementary sensitivity of the following control system.



`mixsyn` computes the controller `K` that yields the minimum $||M(s)||_\infty$, which is returned as `gamma`. For the returned controller K,

$$\begin{aligned} ||S||_\infty &\le \gamma|W_1^{-1}| \\ ||KS||_\infty &\le \gamma|W_2^{-1}| \\ ||T||_\infty &\le \gamma|W_3^{-1}|. \end{aligned}$$

Description

`makeweight` is a convenient way to specify loop shapes, target gain profiles, or weighting functions for applications such as controller synthesis and control system tuning.

`W = makeweight(dcgain,[freq,mag],hfgain)` creates a first-order, continuous-time weight $W(s)$ satisfying these constraints: example

$$\begin{aligned} W(0) &= \text{dcgain} \\ W(\text{Inf}) &= \text{hfgain} \\ |W(j \cdot \text{freq})| &= \text{mag}. \end{aligned}$$

In other words, the gain of `W` passes through `mag` at the finite frequency `freq`.

W_1 - Shaping S: $\quad ||S||_\infty \le \gamma|W_1^{-1}|$

```
dcgain_1 = db2mag(-80);% Spec 4
hfgain_1 = 1.5; % Spec 2
bw_1 = 250; % Spec 1
W_1 = makeweight(dcgain_1, [2*pi*bw_1, db2mag(-3)], hfgain_1)
```

```
W_1 =

  A = 
          x1
   x1  -2934
```

```
  B =
        u1
   x1   64

  C =
            x1
   y1   -68.77

  D =
         u1
   y1   1.5

Continuous-time state-space model.
```

W_2 - Shaping KS: $\quad ||KS||_\infty \leq \gamma |W_2^{-1}|$

```
u_max = 10;
W_2 = tf(1/u_max)
```

```
W_2 =

  0.1

Static gain.
```

W_3 - Shaping T: $\quad ||T||_\infty \leq \gamma |W_3^{-1}|$

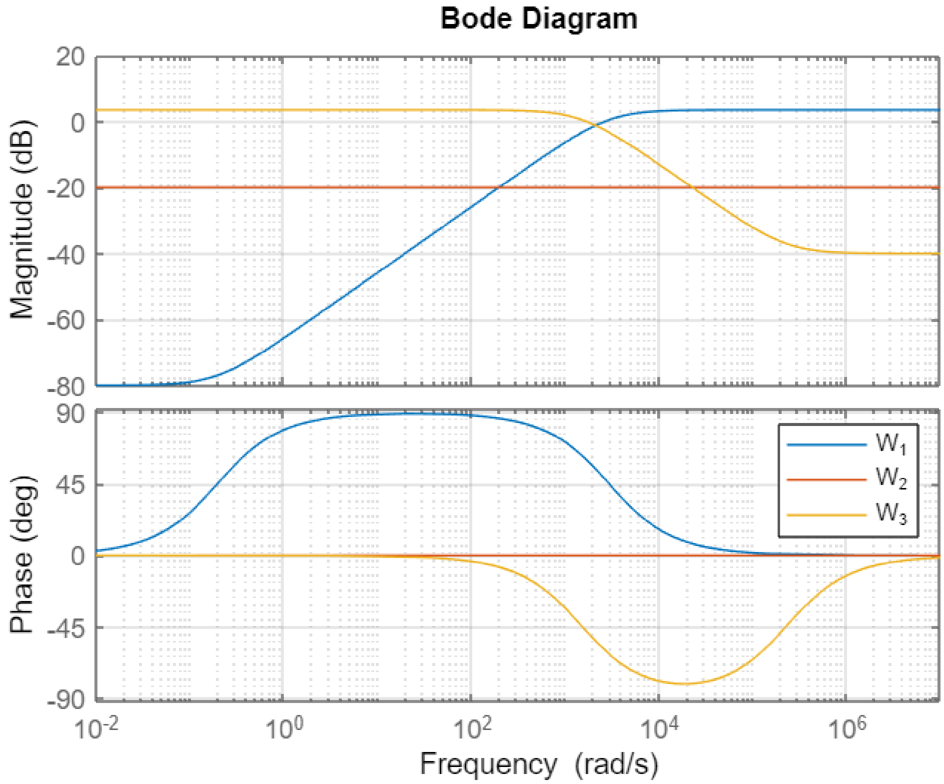
```
dcgain_3 = 1.5; % Spec 6 (max KS should be less then 1.5)
hfgain_3 = db2mag(-40); % Spec 7
bw_3 = 450; %should be less then 500 to ensure below -3dB @ 500 Hz
W_3 = makeweight(dcgain_3, [2*pi*bw_3, db2mag(-3)], hfgain_3)
```

```
W_3 =

  A =
           x1
   x1   -1513

  B =
        u1
   x1   64

  C =
           x1
   y1   35.24

  D =
          u1
   y1   0.01

Continuous-time state-space model.
```

Plotting Weighting functions

```
figure
hold on
bode(W_1)
bode(W_2)
```

```
bode(W_3)
legend('W_1','W_2','W_3')
grid on
```



H_∞ controller Calculation

```
[C_Hinf,CL,gamma,info] = mixsyn(G_sys,W_1,W_2,W_3)
```

```
C_Hinf =

  A = 
                x1          x2          x3          x4          x5          x6          x7          x8
   x1        -2934   -1.51e-10   2.547e-11   4.657e-10  -9.313e-10   6.985e-10  -1.164e-10   8.731e-11
   x2    9.353e+04  -1.527e+04  -5.738e+04  -3.866e+06   4.894e+06  -4.182e+06   2.764e+06  -6.636e+05
   x3    8.488e+04  -1.249e+04  -6.017e+04  -3.487e+06    4.42e+06  -3.774e+06   2.486e+06  -5.914e+05
   x4    1.871e+05  -2.752e+04    -1.34e+05  -7.697e+06   9.756e+06  -8.331e+06   5.494e+06  -1.311e+06
   x5     2.75e+05  -4.046e+04  -1.924e+05  -1.132e+07   1.434e+07  -1.224e+07   8.074e+06  -1.925e+06
   x6    2.989e+05  -4.397e+04  -2.114e+05   -1.23e+07   1.558e+07  -1.331e+07   8.781e+06  -2.095e+06
   x7    1.908e+05  -2.807e+04  -1.336e+05  -7.853e+06    9.95e+06  -8.497e+06   5.601e+06  -1.334e+06
   x8    1.841e+05  -2.709e+04  -1.308e+05  -7.576e+06     9.6e+06  -8.198e+06   5.405e+06  -1.291e+06

  B = 
           u1
   x1      64
   x2   113.4
   x3     103
   x4   226.9
   x5   333.6
   x6   362.5
   x7   231.4
   x8   223.3
```

```
  C = 
              x1          x2          x3          x4          x5          x6          x7          x8
   y1   1.646e+04       -2422  -1.139e+04    -6.78e+05     8.59e+05  -7.336e+05   4.839e+05  -1.155e+05
 
  D = 
        u1
   y1  19.97
 
Continuous-time state-space model.


CL =
 
  A = 
               x1          x2          x3          x4          x5          x6          x7          x8          x9
   x1       -2934           0       -2633        5266       -5266        5266       -5266        2633  -3.373e+04
   x2           0       -1513        2633       -5266        5266       -5266        5266       -2633   3.373e+04
   x3           0           0       -5704   1.701e+04  -1.701e+04   1.701e+04  -1.701e+04        8506   3.062e+04
   x4           0           0  -1.402e+04   2.523e+04  -2.243e+04   2.243e+04  -2.243e+04   1.121e+04   6.747e+04
   x5           0           0   -1.59e+04    3.18e+04   -3.46e+04    3.74e+04   -3.74e+04    1.87e+04    9.92e+04
   x6           0           0  -1.961e+04   3.922e+04  -3.922e+04   3.642e+04  -3.362e+04   1.681e+04   1.078e+05
   x7           0           0  -1.121e+04   2.241e+04  -2.241e+04   2.241e+04  -2.522e+04   1.401e+04    6.88e+04
   x8           0           0  -1.262e+04   2.523e+04  -2.523e+04   2.523e+04  -2.523e+04        9815   6.641e+04
   x9           0           0       -2633        5266       -5266        5266       -5266        2633  -3.667e+04
   x10          0           0       -4667        9335       -9335        9335       -9335        4667   3.373e+04
   x11          0           0       -4236        8472       -8472        8472       -8472        4236   3.062e+04
   x12          0           0       -9335   1.867e+04  -1.867e+04   1.867e+04  -1.867e+04        9335   6.747e+04
   x13          0           0  -1.372e+04   2.745e+04  -2.745e+04   2.745e+04  -2.745e+04   1.372e+04    9.92e+04
   x14          0           0  -1.492e+04   2.983e+04  -2.983e+04   2.983e+04  -2.983e+04   1.492e+04   1.078e+05
   x15          0           0       -9520   1.904e+04  -1.904e+04   1.904e+04  -1.904e+04        9520    6.88e+04
   x16          0           0       -9189   1.838e+04  -1.838e+04   1.838e+04  -1.838e+04        9189   6.641e+04
 
              x15         x16
   x1   -9.915e+05   2.367e+05
   x2    9.915e+05  -2.367e+05
   x3    8.999e+05  -2.148e+05
   x4    1.983e+06  -4.734e+05
   x5    2.916e+06   -6.96e+05
   x6    3.168e+06  -7.564e+05
   x7    2.022e+06  -4.828e+05
   x8    1.952e+06   -4.66e+05
   x9   -9.915e+05   2.367e+05
   x10   1.006e+06   -2.44e+05
   x11   8.913e+05  -2.106e+05
   x12   1.979e+06  -4.715e+05
   x13   2.906e+06  -6.911e+05
   x14   3.165e+06  -7.545e+05
   x15   2.016e+06  -4.783e+05
   x16   1.945e+06  -4.654e+05
 
  B = 
          u1
   x1   23.08
   x2   40.92
   x3   37.13
   x4   81.83
   x5   120.3
   x6   130.8
   x7   83.45
   x8   80.55
   x9   23.08
   x10  40.92
   x11  37.13
   x12  81.83
```

```
    x13  120.3
    x14  130.8
    x15  83.45
    x16  80.55

  C =
                x1          x2          x3          x4          x5          x6          x7          x8          x9
    y1      -68.77           0      -61.72       123.4      -123.4       123.4      -123.4       61.72      -790.6
    y2           0           0      -82.16       164.3      -164.3       164.3      -164.3       82.16       593.8
    y3           0       35.24      0.4114     -0.8229      0.8229     -0.8229      0.8229     -0.4114       5.271

               x15         x16
    y1  -2.324e+04        5548
    y2   1.745e+04       -4167
    y3       154.9      -36.99

  D =
              u1
    y1     0.541
    y2    0.7203
    y3  0.006393

Input groups:
    Name    Channels
     U1        1

Output groups:
    Name    Channels
     Y1      1,2,3

Continuous-time state-space model.
gamma = 1.4549
info =
  hinfINFO with properties:

    gamma: 1.4549
        X: [8×8 double]
        Y: [8×8 double]
       Ku: [-325.4596 42.8192 1.8687e+03 9.4406e+03 -1.4372e+04 1.3922e+04 -1.0845e+04 3.5761e+03]
       Kw: [-869.6446 127.2423 943.6919 3.5037e+04 -4.4783e+04 3.8441e+04 -2.5510e+04 6.1675e+03]
       Lx: [8×1 double]
       Lu: 7.2026
     Preg: [5×2 ss]
       AS: [2×2 ss]
```

Part d

H_∞-controller Margin Calculations

```
allmargin(C_Hinf*G_sys)
```

```
ans = struct with fields:
     GainMargin: 6.5542
    GMFrequency: 4.2065e+03
    PhaseMargin: [53.6057 -125.7859]
    PMFrequency: [1.2243e+03 1.6588e+06]
    DelayMargin: [7.6422e-04 2.4644e-06 0]
    DMFrequency: [1.2243e+03 1.6588e+06 Inf]
         Stable: 1
```

```
figure
```

```
margin(C_Hinf * G_sys)
grid on
```

PI - vs H_∞ - controller Bode Comparrision

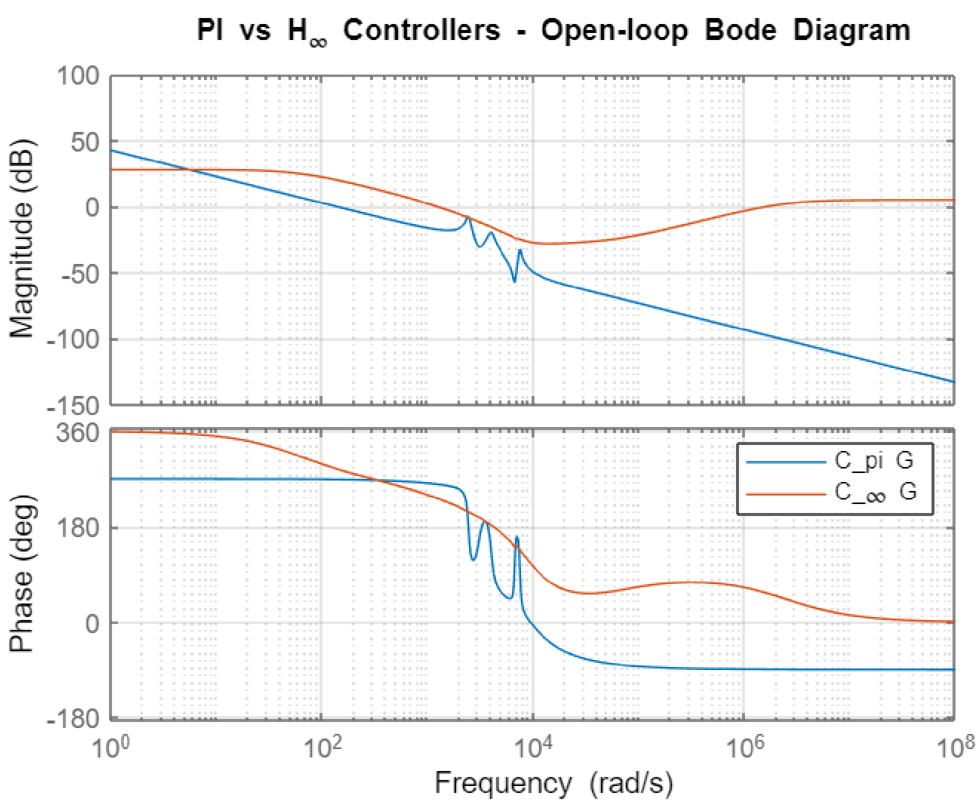
```
figure
hold on
bode(C_pi)
bode(C_Hinf)
legend('C_{pi}', 'C_{H_\infty}')
grid on
```

Loop Transfer Functions

```
sys_pi = feedback(C_pi * G_sys, 1);
sys_Hinf = feedback(C_Hinf * G_sys, 1);
```

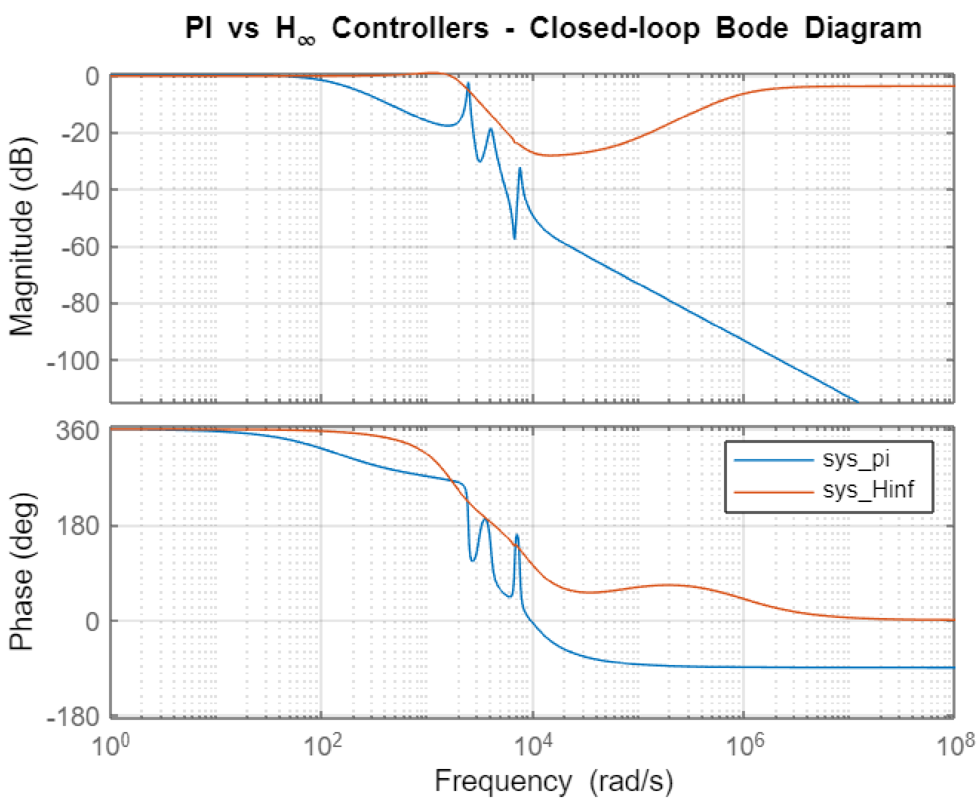
Bode Open Loop

```
figure
hold on
bode(C_pi * G_sys, C_Hinf * G_sys)
legend('C_{pi} G', 'C_\infty G')
title('PI vs H_\infty Controllers - Open-loop Bode Diagram')
grid on
```



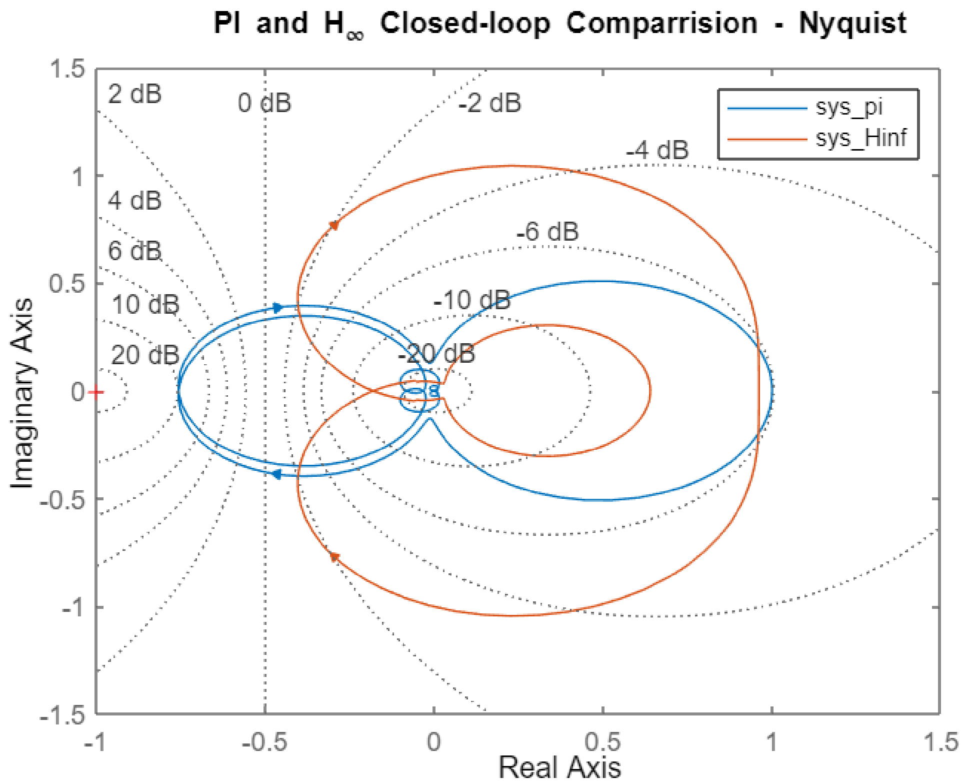
Bode Closed Loop

```
figure
hold on
bode(sys_pi,sys_Hinf)
legend
title('PI vs H_\infty Controllers - Closed-loop Bode Diagram')
grid on
```



Nyquist Loops

```
figure
nyquist(sys_pi, sys_Hinf)
legend
title('PI and H_\infty Closed-loop Comparrision - Nyquist')
grid on
```



Problem 2

The Generic Transport Model (GTM) is a turbine powered subscale model of a civilian transport aircraft, which was developed by NASA Langley as a platform to validate control laws. The model has a wing span of 7 ft, and weighs around 55 lbs. Under normal operation, the aircraft flies at an altitude of 700 to 1100 ft, and with an airspeed of 70 and 85 knots. In this problem you will use the signal-weighted H_∞ method to design a control law for the GTM.

A nonlinear simulation model of the GTM has been developed from extensive wind tunnel and flight tests. The model can be linearized at a particular flight condition, yielding a linear model of the aircraft dynamics. The nominal fight condition for control design is level flight at 800 ft and 80 knots. The longitudinal short-period dynamics, denoted G, are described by the following state equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\alpha \\ q\end{bmatrix} = \begin{bmatrix}-2.4714 & 0.9514 \\ -43.9070 & -3.4738\end{bmatrix}\begin{bmatrix}\alpha \\ q\end{bmatrix} + \begin{bmatrix}-0.2501 \\ -44.9478\end{bmatrix}\delta_{elev}$$

where the state vector corresponds to angle-of-attack α [rad] and pitch rate q [rad/s]. The control surface input is elevator deflection δ_{elev} [rad]. The elevator actuator is modeled as a 5 Hz (= 31.42 rad/sec) first-order filter with DC gain equal to 1. This actuator saturates at 0.349 rads, i.e., it is physically constrained to $\delta_{elev} \leq 0.349$ rads. A rate gyro sensor measures pitch rate with noise that has standard deviation of 0.0067 rad/sec.

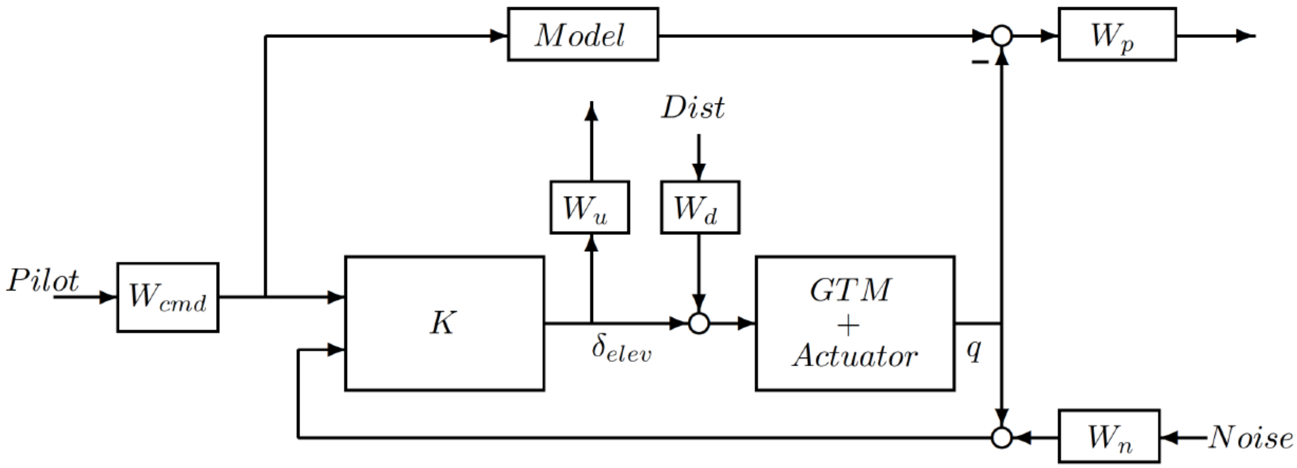


Fig. 2: Longitudinal system interconnection.

The main control objective is to design a Stability Augmentation System (SAS) to increase damping in the aircraft's oscillatory modes. A model-matching H_∞ control problem is formulated to achieve the desired robustness and performance characteristics. The longitudinal control system interconnection is shown in Figure 2. The inputs to the controller are pilot longitudinal stick command and pitch rate feedback. The actual input commands are assumed to have magnitude ≤ 0.2618 rads. The ideal model for matching is denoted as "Model" in the diagram. It will be chosen to mimic the open-loop aircraft behavior at low and high frequency but with improved damping in the oscillatory modes.

```
clear
close all
```

Model Definition

Plant Definition

```
A = [
    - 2.4714     0.9514
    -43.9070    -3.4738
];
B = [
    - 0.2501
    -44.9478
];
C = [0 1];
D = 0;
G = ss(A,B,C,D);
G.InputName = '\delta_elev_act';
G.StateName = {'alpha', 'q'};
G.OutputName = {'q'}
```

```
G =
```

```
  A =
             alpha       q
   alpha    -2.471  0.9514
   q        -43.91  -3.474

  B =
            \delta_elev_
   alpha         -0.2501
   q              -44.95

  C =
         alpha      q
   q         0      1

  D =
         \delta_elev_
   q                0

Continuous-time state-space model.
```

Actuator Defintion

First order Actuator filter

```
p_act = 5 * 2 * pi;
k_act = 1;
G_act = ss(tf(k_act * p_act, [1 p_act]));
G_act.InputName = '\delta_elev_sum';
G_act.OutputName = '\delta_elev_act';
G_act.StateName = '\delta_elev_filter'
```

```
G_act =

  A =
                 \delta_elev_
   \delta_elev_        -31.42

  B =
                 \delta_elev_
   \delta_elev_             4

  C =
                 \delta_elev_
   \delta_elev_         7.854

  D =
                 \delta_elev_
   \delta_elev_             0

Continuous-time state-space model.
```

Saturation?

Note: idk how to actually impliment a nonlinearirty within this type of a system... so idk, do we just want another crazy filter going to zero at 0.349 somehow? idk, why aren't we using Simulink since it would be way easier (even if we did it via code...)

Rate Gyro Sensor Error

Just a change in standard diveation from 1 to 0.0067...

```
std_gyro = 0.0067;
W_n = tf(std_gyro);
W_n.InputName = 'Noise';
W_n.OutputName = 'q_noise'
```

```
W_n =

  From input "Noise" to output "q_noise":
  0.0067

Static gain.
```

```
[wn_G, zeta_G, p_G] = damp(G)
```

```
wn_G = 2×1
    7.0964
    7.0964
zeta_G = 2×1
    0.4189
    0.4189
p_G = 2×1 complex
  -2.9726 + 6.4438i
  -2.9726 - 6.4438i
```

```
[num_G, den_G] = tfdata(G);
zeta_damp = 0.8;
num_damp = num_G;
den_damp = den_G;
den_damp{1}(2) = den_G{1}(2) * (zeta_damp / zeta_G(1))
```

```
den_damp = 1×1 cell array
    {[1 11.3542 50.3583]}
```

```
G_damp = tf(num_damp, den_damp);
G_damp.InputName = 'W_{cmd}';
G_damp.OutputName = 'q_{ideal}'
```

```
G_damp =

  From input "W_{cmd}" to output "q_{ideal}":
    -44.95 s - 100.1
  ---------------------
  s^2 + 11.35 s + 50.36

Continuous-time transfer function.
```

```
[wn_G_damp, zeta_G_damp, p_G_damp] = damp(G_damp)
```

```
wn_G_damp = 2×1
    7.0964
    7.0964
zeta_G_damp = 2×1
    0.8000
    0.8000
p_G_damp = 2×1 complex
  -5.6771 + 4.2578i
  -5.6771 - 4.2578i
```

Part b

W_n - Gyro measurment error std

```
W_n % Gyro std from above
```

```
W_n =

  From input "Noise" to output "q_noise":
  0.0067

Static gain.
```

W_u - Output to a normalized

```
W_u = tf(1/0.349); % \delta_elev normalized from \delta_elev = [-0.349,0.349] to W_cmd = [-1,1]
W_u.InputName = '\delta_{elev}';
W_u.OutputName = 'W_{u}'
```

```
W_u =

  From input "\delta_{elev}" to output "W_{u}":
  2.865

Static gain.
```

W_{cmd} - Pilot comand input to setpoint

```
W_cmd = tf(0.2618); % Input normalized from Pilot = [-1,1] to W_cmd = [-0.2618,0.2618]
W_cmd.InputName = 'Pilot';
W_cmd.OutputName = 'W_{cmd}'
```

```
W_cmd =

  From input "Pilot" to output "W_{cmd}":
  0.2618

Static gain.
```

W_p - Plant error output

```
error_max = 0.01;
W_p = tf(1/error_max); % Normalize from q = [-0.01,0.01] to W_p = [-1,1]
W_p.InputName = 'e';
W_p.OutputName = 'W_{p}'
```

```
W_p =

  From input "e" to output "W_{p}":
  100

Static gain.
```

W_d - Disturbance Input

```
delta_limit = 0.349;
W_d_max = 0.15;
W_d = tf(delta_limit * W_d_max); % Gussing the Input normalized from u = [-1,1] to W_d = 0.15 *
```

```
W_d.InputName = 'Dist';
W_d.OutputName = 'delta_{elev}_d'
```

```
W_d =

  From input "Dist" to output "delta_{elev}_d":
  0.05235

Static gain.
```

Part c

Although... I'm not sure why we are using connect (or even more outdated syssic) and all of this outdated structuring method instead of just setting this up and implimenting it within Simulink

Model Definition

```
Model = G_act * G_damp;
Model.InputName = 'W_{cmd}';
Model.OutputName = 'q_{ideal}';
```

Connect Definition

```
inputvar = {'Pilot', 'Dist', 'Noise', '\delta_{elev}'};
outputvar = {'W_{p}', 'W_{u}', 'W_{cmd}', 'q_{sensor}'};
APs = {'q'};

e_sum = sumblk('e = q_{ideal} - q');
q_sum = sumblk('q_{sensor} = q + q_{noise}');
delta_sum = sumblk('\delta_{elev}_sum = \delta_{elev} + \delta_{elev}_d');
cnct_P = connect( ...
    G, G_act, Model, W_n, W_d, W_p, W_u, W_cmd,...
    e_sum, q_sum, delta_sum,...
    inputvar, outputvar,...
    APs)
```

```
cnct_P =

  Generalized continuous-time state-space model with 4 outputs, 4 inputs, 3 states, and no blocks.

Type "ss(cnct_P)" to see the current value, "get(cnct_P)" to see all properties, and "cnct_P.Blocks" to interact wit
```

```
P = ss(cnct_P)
```

```
P =

  A =
                \delta_elev_             ?             ?
   \delta_elev_        -31.42        -22.47        -6.256
   ?                        0        -11.35        -6.295
   ?                        0             8             0

  B =
                       Pilot          Dist         Noise  \delta_{elev
   \delta_elev_            0             0             0             0
   ?                   2.094             0             0             0
   ?                       0             0             0             0
```

```
  C =
             \delta_elev_            ?            ?
   W_{p}            785.4            0            0
   W_{u}                0            0            0
   W_{cmd}              0            0            0
   q_{sensor}           0            0            0

  D =
                    Pilot         Dist        Noise  \delta_{elev
   W_{p}                0            0            0            0
   W_{u}                0            0            0        2.865
   W_{cmd}         0.2618            0            0            0
   q_{sensor}           0            0            0            0

Continuous-time state-space model.
```

```
Gs = tf(P)
```

```
Gs =

  From input "Pilot" to output...
              -3.697e04 s - 8.233e04
   W_{p}:  --------------------------------
           s^3 + 42.77 s^2 + 407.1 s + 1582

   W_{u}:  0

   W_{cmd}:  0.2618

   q_{sensor}:  0

  From input "Dist" to output...
   W_{p}:  0

   W_{u}:  0

   W_{cmd}:  0

   q_{sensor}:  0

  From input "Noise" to output...
   W_{p}:  0

   W_{u}:  0

   W_{cmd}:  0

   q_{sensor}:  0

  From input "\delta_{elev}" to output...
   W_{p}:  0

   W_{u}:  2.865

   W_{cmd}:  0

   q_{sensor}:  0

Continuous-time transfer function.
```

Part d

Design H_∞ - controller

`[K,CL,gamma] = hinfsyn(P,nmeas,ncont)` computes a stabilizing H_∞-optimal controller `K` for the plant `P`. The plant has a partitioned form

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix},$$

where:

- w represents the disturbance inputs.
- u represents the control inputs.
- z represents the error outputs to be kept small.
- y represents the measurement outputs provided to the controller.

`nmeas` and `ncont` are the number of signals in y and u, respectively. y and u are the last outputs and inputs of `P`, respectively. `hinfsyn` returns a controller `K` that stabilizes `P` and has the same number of states. The closed-loop system `CL = lft(P,K)` achieves the performance level `gamma`, which is the H_∞ norm of `CL` (see `hinfnorm`).

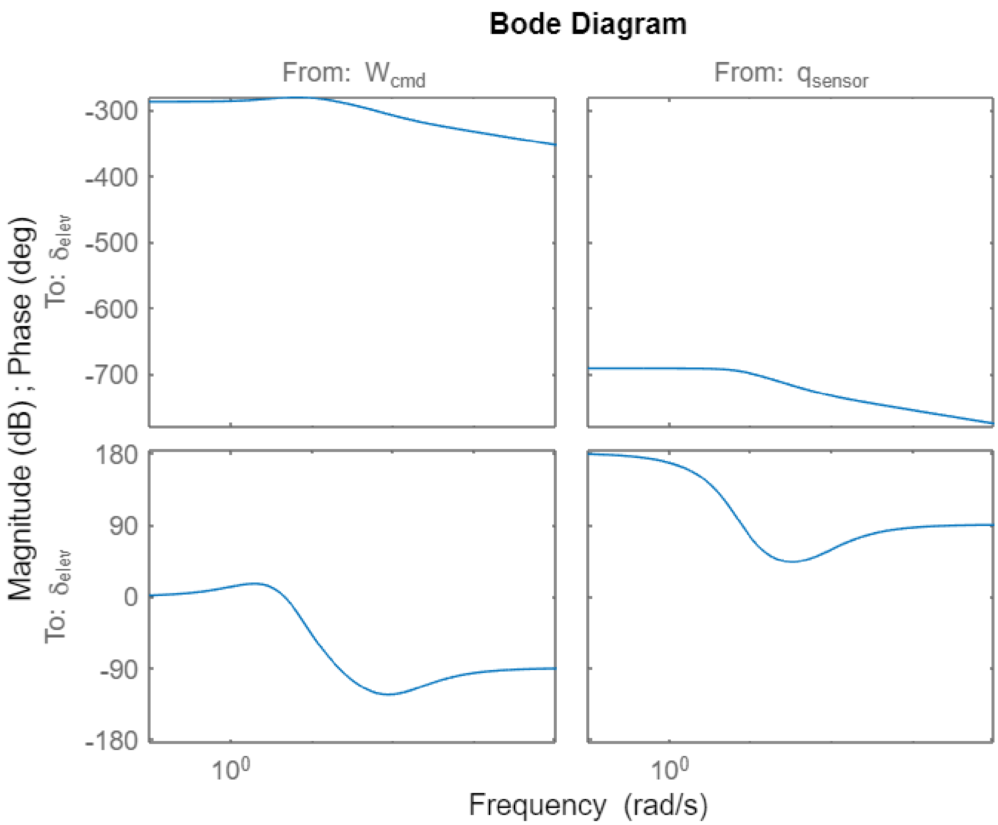
```
nmeas = 2;
ncont = 1;
[K,CL,gamma,info] = hinfsyn(P,nmeas,ncont);
K.InputName = {'W_{cmd}', 'q_{sensor}'};
K.OutputName = {'\delta_{elev}'}
```

```
K =

  A = 
               x1          x2          x3
   x1      -31.42      -22.47      -6.256
   x2           0      -11.35      -6.295
   x3   7.117e-17           8  -7.795e-18

  B = 
            W_{cmd}  q_{sensor}
   x1           0   9.481e-36
   x2           8  -9.146e-21
   x3   2.243e-16   2.247e-20

  C = 
                          x1          x2          x3
   \delta_{elev  -1.415e-14   2.694e-15   3.746e-16

  D = 
                     W_{cmd}  q_{sensor}
   \delta_{elev           0           0

Continuous-time state-space model.
```

```
% CL.InputName = inputvar(1:3);
% CL.OutputName = outputvar(1:2)
```

Bode of Control Law

```
figure
bode(K)
```



Nominal Feedback System

The question is worded terribly... assumin negative feedback control w a normalized pilot signal still being weighted

```
w_cmd = W_cmd;
w_cmd.InputName = 'Pilot';
w_cmd.OutputName = 'wcmd'
```

```
w_cmd =

  From input "Pilot" to output "wcmd":
  0.2618

Static gain.
```

```
k = K;
k.InputName = {'wcmd','q'};
k.OutputName = '\delta_{elev}'
```

```
k =

  A = 
              x1         x2         x3
   x1     -31.42     -22.47     -6.256
   x2          0     -11.35     -6.295
   x3  7.117e-17          8  -7.795e-18

  B = 
            wcmd          q
   x1          0  9.481e-36
```

```
   x2           8  -9.146e-21
   x3   2.243e-16   2.247e-20

  C =
                        x1          x2          x3
   \delta_{elev  -1.415e-14   2.694e-15   3.746e-16

  D =
                wcmd     q
   \delta_{elev    0     0

Continuous-time state-space model.
```

```
G_act_plant = G_act * G;
G_act_plant.InputName = '\delta_{elev}';
G_act_plant.OutputName = 'q'
```

```
G_act_plant =

  A =
                \delta_elev_         alpha             q
   \delta_elev_        -31.42             0             4
   alpha                    0        -2.471        0.9514
   q                        0        -43.91        -3.474

  B =
                \delta_{elev
   \delta_elev_            0
   alpha             -0.2501
   q                  -44.95

  C =
       \delta_elev_         alpha             q
   q          7.854             0             0

  D =
       \delta_{elev
   q             0

Continuous-time state-space model.
```

```
G_act_plant.StateName = {};
cnct_nom = connect( ...
    G_act_plant, w_cmd, k, ...
    {'Pilot'}, ... Inputs
    {'q'}, ... Outputs
    {'wcmd','\delta_{elev}'} ... APs
    )
```

```
cnct_nom =

  Generalized continuous-time state-space model with 1 outputs, 1 inputs, 6 states, and the following blocks:
    AnalysisPoints_: Analysis point, 2 channels, 1 occurrences.

Type "ss(cnct_nom)" to see the current value, "get(cnct_nom)" to see all properties, and "cnct_nom.Blocks" to intera
```

```
sys_cls = ss(getIOTransfer(cnct_nom, 'wcmd', 'q')) %ss(cnct_nom)
```

```
sys_cls =
```

```
  A = 
             x1          x2          x3          x4          x5          x6
   x1      -31.42           0           4           0           0           0
   x2           0      -2.471      0.9514   3.538e-15  -6.737e-16  -9.369e-17
   x3           0      -43.91      -3.474   6.359e-13  -1.211e-13  -1.684e-14
   x4   7.447e-35           0           0      -31.42      -22.47      -6.256
   x5  -7.184e-20           0           0           0      -11.35      -6.295
   x6   1.765e-19           0           0   7.117e-17           8  -7.795e-18

  B = 
            wcmd
   x1          0
   x2          0
   x3          0
   x4          0
   x5          8
   x6  2.243e-16

  C = 
          x1     x2     x3     x4     x5     x6
   q   7.854      0      0      0      0      0

  D = 
       wcmd
   q      0

Continuous-time state-space model.
```

```
% sys_nom = ss(cnct_nom)
```

Actually... that was over complicated... and mabye wrong... testing this....

```
inputvar = {'Pilot'};
outputvar = {'q'};
APs = {'\delta_{elev}'};

G_act.InputName = {'\delta_{elev}'};
G.OutputName = {'q'}
```

```
G =
 
  A = 
          alpha       q
   alpha  -2.471  0.9514
   q      -43.91  -3.474
 
  B = 
         \delta_elev_
   alpha      -0.2501
   q           -44.95
 
  C = 
       alpha      q
   q       0      1
 
  D = 
       \delta_elev_
   q              0
 
Continuous-time state-space model.
```

```
cnct_cls = connect( ...
    G, G_act, W_cmd, K, ...
    inputvar, outputvar,...
    APs)
```

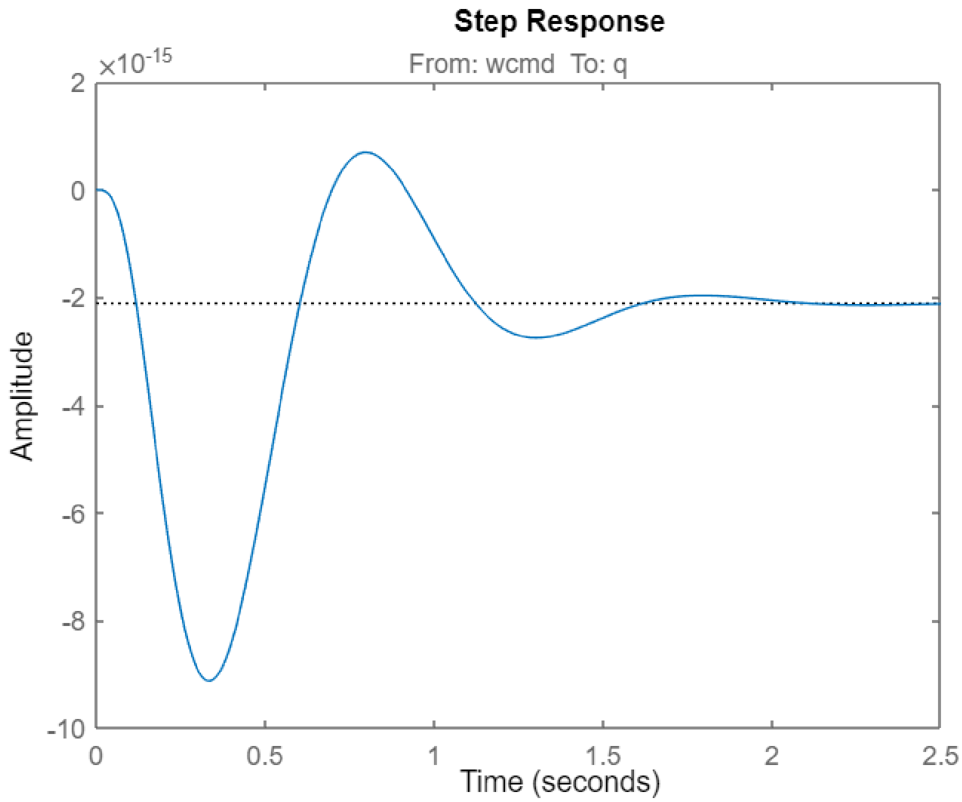
```
cnct_cls =

  Generalized continuous-time state-space model with 1 outputs, 1 inputs, 6 states, and the following blocks:
    AnalysisPoints_: Analysis point, 1 channels, 1 occurrences.

Type "ss(cnct_cls)" to see the current value, "get(cnct_cls)" to see all properties, and "cnct_cls.Blocks" to intera
```

```
% sys_cls = ss(cnct_cls)
sys_cls = ss(getIOTransfer(cnct_nom, 'wcmd', 'q'))
```

```
sys_cls =

  A = 
              x1         x2         x3         x4         x5         x6
   x1     -31.42          0          4          0          0          0
   x2          0     -2.471     0.9514  3.538e-15 -6.737e-16 -9.369e-17
   x3          0     -43.91     -3.474  6.359e-13 -1.211e-13 -1.684e-14
   x4  7.447e-35          0          0     -31.42     -22.47     -6.256
   x5 -7.184e-20          0          0          0     -11.35     -6.295
   x6  1.765e-19          0          0  7.117e-17          8 -7.795e-18

  B = 
            wcmd
   x1          0
   x2          0
   x3          0
   x4          0
   x5          8
   x6  2.243e-16

  C = 
          x1     x2     x3     x4     x5     x6
   q   7.854      0      0      0      0      0

  D = 
       wcmd
   q      0

Continuous-time state-space model.
```

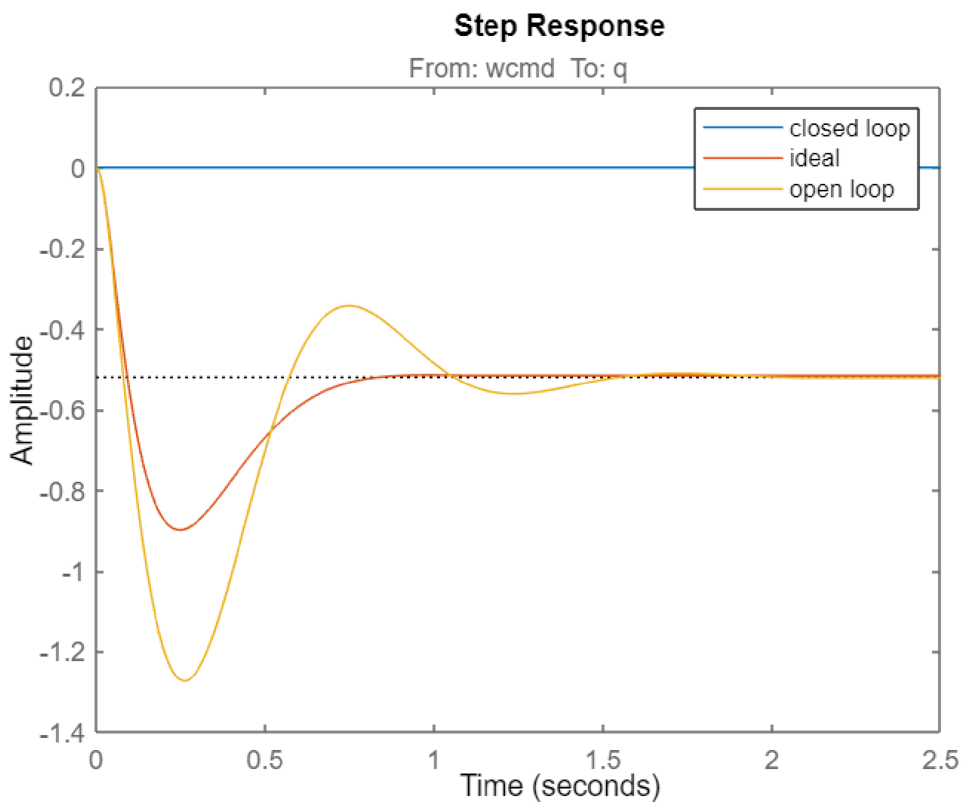
Feedback System Response

```
figure
opt = stepDataOptions;
opt.StepAmplitude = 0.2618;
step(sys_cls, opt)
```



Comparrision Responses

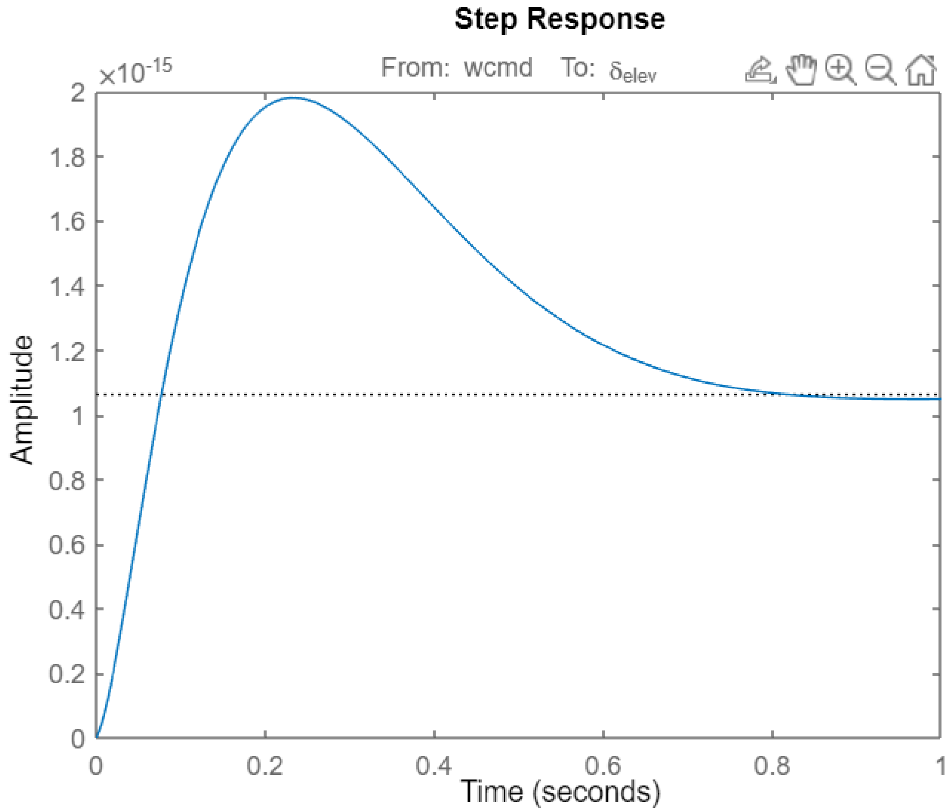
```
figure
hold on
step(sys_cls, opt)
step(Model, opt)
step(G_act_plant, opt)
legend('closed loop', 'ideal', 'open loop')
```




```
figure
T_cmd_delta = ss(getIOTransfer(cnct_nom, 'wcmd', '\delta_{elev}'))
```

```
T_cmd_delta =

  A = 
              x1         x2         x3         x4         x5         x6
   x1     -31.42          0          4          0          0          0
   x2          0     -2.471     0.9514  3.538e-15 -6.737e-16 -9.369e-17
   x3          0     -43.91     -3.474  6.359e-13 -1.211e-13 -1.684e-14
   x4  7.447e-35          0          0     -31.42     -22.47     -6.256
   x5 -7.184e-20          0          0          0     -11.35     -6.295
   x6  1.765e-19          0          0  7.117e-17          8 -7.795e-18

  B = 
            wcmd
   x1          0
   x2          0
   x3          0
   x4          0
   x5          8
   x6  2.243e-16

  C = 
                        x1         x2         x3         x4         x5         x6
   \delta_{elev          0          0          0 -1.415e-14  2.694e-15  3.746e-16

  D = 
                  wcmd
   \delta_{elev      0

Continuous-time state-space model.
```

```
step(T_cmd_delta, opt)
```



Ultimently there is an issue here that I've been unable to solve before submitting this assignment. The following two problems are actually really simple to impliment if I had the time to do so... (i.e. just do the conect and hinfsyn code again)

I will say again though ***WHY ARE WE NOT USING SIMULINK?????*** this is so much simpler to do (even if you don't use the GUI it's not as tricky as this)

Final Parts

```
fname = matlab.desktop.editor.getActiveFilename;
export(fname, [fname(1:length(fname)-4),'.pdf']);
%saves the .mlx file to a .pdf in the same directory with the same name
```