Optimal Estimation - Homework 0

Tanner Koza (jtk0018) - January 31, 2022

```
clear
clc
close all
```

Problem #1

Part A)

The differential equation of simple rotation table system was derived from the following sum of moments about the center of the table:

$$\Sigma M = J\ddot{\theta} = \tau - b\dot{\theta}$$

This sum of moments can be rearranged into the following equation of motion:

$$J\ddot{\theta} + b\dot{\theta} = \tau$$

The rotational moment of inertia, J, and rotational damping, b, were given as 10 kg*m^2 and 1 N*m*s/rad, respectively.

```
J = 10; % Rotational Moment of Inertia (kg*m^2)
b = 1; % Rotational Damping (N*m-s/rad)
```

Part B)

The system was then converted into state-space format as follows:

$$\ddot{\theta} = \frac{1}{J}(\tau - b\dot{\theta})$$

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-b}{J} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} \tau$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

The value of the D is zero.

The values of the A, B, C, and D were then used to create a state-space system in MATLAB using the ss() function.

```
A = [0 1; 0 -b/J]; % Dynamic Matrix
B = [0; 1/J]; % Control Matrix
C = [1 0]; % Measurement Matrix
D = 0;

sys_ss = ss(A,B,C,D) % Open Loop System SS
```

```
sys_ss =

  A =
         x1    x2
   x1     0     1
   x2     0  -0.1

  B =
        u1
   x1    0
   x2  0.1

  C =
       x1  x2
   y1   1   0

  D =
       u1
   y1   0

Continuous-time state-space model.
```

```
sys_tf = tf(sys_ss); % Open Loop System TF
```

Part C)

The open loop eigenvalues of this system were determined using the eig() function in MATLAB as follows:

```
s_sys = eig(sys_ss) % Open Loop System Eigenvalues
```

```
s_sys = 2×1
         0
   -0.1000
```

Problem #2

Part A)

The observability matrix for the open loop system was calculated using the obsv() function in MATLAB. Then, the rank of the observability matrix was determined to be full (rank is equal to the number of states) using the rank() function as follows:

```
ob = obsv(A,C); % Observability Matrix
```

```
observability = rank(ob)
```

```
observability = 2
```

Part B)

L was designed by, first, determining the desired eigenvalues of the observer using the desired natural frequency and damping ratio. Then, the place() function was used to determine an L_1 and L_2 that give the desired characteristics associated with the eigenvalues. The place() function is equivalent to finding the desired characteristic equation using $det\{sI - (A - LC)\}$ and then coefficient matching that characteristic equation with the desired coefficients.

```
wn_ob = 100 * pi; % Desired Natural Frequency (rad/s)
zeta_ob = 0.7; % Desired Damping Ratio
s_des_ob = [-wn_ob*zeta_ob - (wn_ob*sqrt(1-zeta_ob^2)*1i), ...
            -wn_ob*zeta_ob + (wn_ob*sqrt(1-zeta_ob^2)*1i)]; % Desired Observer Eigenvalues

L = place(A',C',s_des_ob)' % Observer Gain Matrix
```

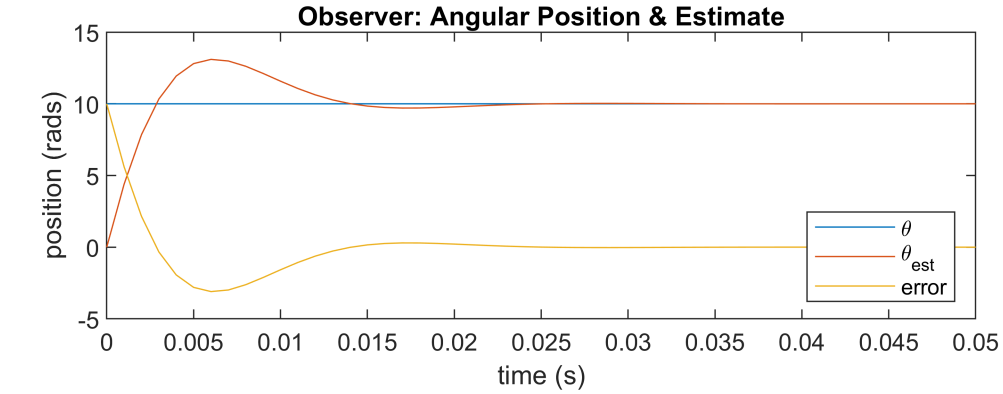
```
L = 2×1
10^4 ×
    0.0440
    9.8652
```

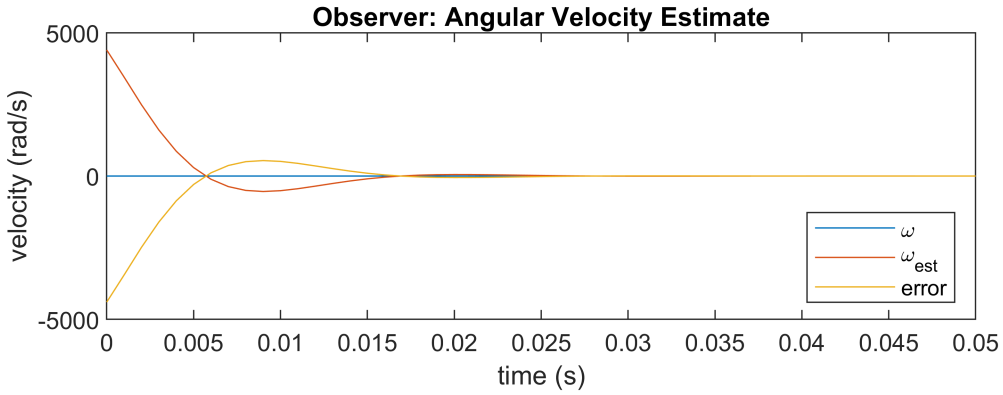
Part C)

The step response of the observer was then simulated using a time step equivalent to the inverse of the 1kHz sampling rate in Problem #5. The intial angular position and velocity were set to 10 rads and 0 rad/s, respectively, while the initial estimate was 0 rads and 0 rad/s. The actual and estimated angular positions and velocities were plotted along with their respective errors.

```
A_ob = A - L*C; % Observer Closed Loop A Matrix
ob_ss = ss(A_ob,L,C,D); % Observer SS

dt = 1/1000; % Simulation Time Step (Sampling Period (s))
t = 0:dt:0.05; % Simulation Time Vector

u = zeros(1,length(t)); % Preallocation
y = zeros(1, length(t));
x = zeros(2,length(t));
dx = zeros(2,length(t));
xhat = zeros(2,length(t));
dxhat = zeros(2,length(t));

x(:,1) = [10; 0]; % Initialized State & State Estimate
xhat(:,1) = [0; 0];

% Simulation
for i = 1:length(t) - 1

    y(i) = C*x(:,i); % Actual State "Measurements"
```

```
    dx(:,i) = A*x(:,i); % Actual State Derivative

    x(:,i+1) = x(:,i) + dx(:,i)*dt; % Integrate Actual States



    dxhat(:,i) = A*xhat(:,i) + B*u(i); % Time Update

    dxhat(:,i) = dxhat(:,i) + L*(y(i) - C*xhat(:,i)); % Measurement Update

    xhat(:,i+1) = xhat(:,i) + dxhat(:,i)*dt; % Integrate Estimated States

end

% Plotting
figure

subplot(2,1,1)
plot(t, x(1,:), t, xhat(1,:)) % Plot Actual & Estimated States
hold on
plot(t, x(1,:) - xhat(1,:)) % Plot Error

title("Observer: Angular Position & Estimate")
xlabel('time (s)')
ylabel('position (rads)')
legend('\theta','\theta_{est}','error','Location','southeast')

subplot(2,1,2)
plot(t, dx(1,:), t, dxhat(1,:))
hold on
plot(t, dx(1,:) - dxhat(1,:))

title("Observer: Angular Velocity Estimate")
xlabel('time (s)')
ylabel('velocity (rad/s)')
legend('\omega','\omega_{est}','error','Location','southeast')
```





Problem #3

Part A)

The controllability matrix for the open loop system was calculated using the ctrb() function in MATLAB. Then, the rank of the controllability matrix was determined to be full (rank is equal to the number of states) using the rank() function as follows:

```
co = ctrb(A,B); % Observability Matrix
controllability = rank(co)
```

```
controllability = 2
```

Part B)

K was designed by, first, determining the desired eigenvalues of the controller using the desired natural frequency and damping ratio. Then, the place() function was used to determine an K_1 and K_2 that give the desired characteristics associated with the eigenvalues. The place() function is equivalent to finding the desired characteristic equation using $det\{sI-(A-BK)\}$ and then coefficient matching that characteristic equation with the desired coefficients.

```
wn_co = 20 * pi; % Desired Natural Frequency (rad/s)
zeta_co = 0.7; % Desired Damping Ratio
s_des_co = [-wn_co*zeta_co - (wn_co*sqrt(1-zeta_co^2)*1i), ...
            -wn_co*zeta_co + (wn_co*sqrt(1-zeta_co^2)*1i)]; % Desired Controller Eigenvalues
```

```
K = place(A,B,s_des_co) % Controller Gain Matrix
```

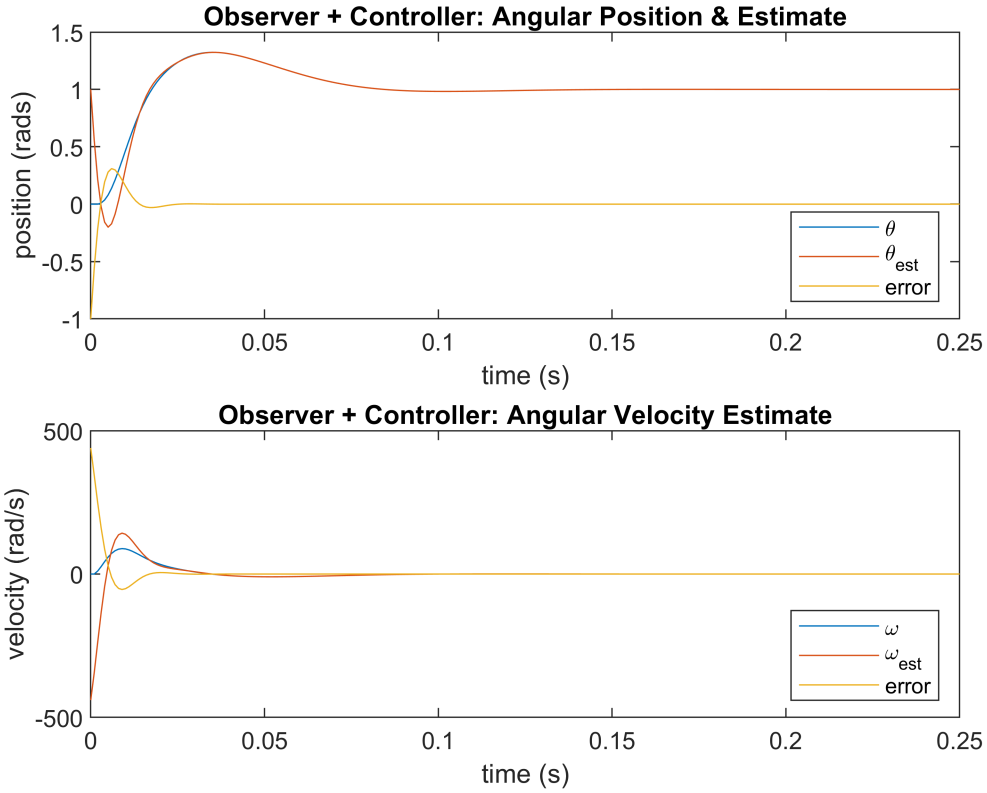
```
K = 1×2
10^4 ×
    3.9478    0.0879
```

Part C)

The step response of the observer and controller combined (equivalent compensator) was then simulated using a time step equivalent to the inverse of the 1kHz sampling rate in Problem #5. The intial angular position and velocity were set to 0 rads and 0 rad/s, respectively, while the initial estimate was 1 rads and 0 rad/s. This allows the plots to easily display the observer dynamics alongside the controller dynamics. A reference of 1 rad was used. The actual and estimated angular positions and velocities were plotted along with their respective errors.

```
A_co = A - B*K;
co_ss = ss(A_co,B,C,D); % Controller SS

t = 0:dt:0.25;

u = zeros(1,length(t)); % Clearing Variables & Preallocation
y = zeros(1, length(t));
x = zeros(2,length(t));
dx = zeros(2,length(t));
xhat = zeros(2,length(t));
dxhat = zeros(2,length(t));

x(:,1) = [0; 0]; % Initialized State & State Estimate
xhat(:,1) = [1; 0];
r = [1; 0]; % Reference

for i = 1:length(t) - 1

    u(i) = K*(r -xhat(:,i)); % Control Input

    y(i) = C*x(:,i); % Actual State "Measurements"

    dx(:,i) = A*x(:,i) + B*u(i); % Actual State Derivative

    x(:,i+1) = x(:,i) + dx(:,i)*dt; % Integrate Actual States



    dxhat(:,i) = A*xhat(:,i) + B*u(i); % Time Update

    dxhat(:,i) = dxhat(:,i) + L*(y(i) - C*xhat(:,i)); % Measurement Update

    xhat(:,i+1) = xhat(:,i) + dxhat(:,i)*dt; % Integrate Estimated States

end

% Plotting
figure
```

```
subplot(2,1,1)
plot(t, x(1,:), t, xhat(1,:)) % Plot Actual & Estimated States
hold on
plot(t, x(1,:) - xhat(1,:)) % Plot Error

title("Observer + Controller: Angular Position & Estimate")
xlabel('time (s)')
ylabel('position (rads)')
legend('\theta','\theta_{est}','error','Location','southeast')

subplot(2,1,2)
plot(t, dx(1,:), t, dxhat(1,:))
hold on
plot(t, dx(1,:) - dxhat(1,:))

title("Observer + Controller: Angular Velocity Estimate")
xlabel('time (s)')
ylabel('velocity (rad/s)')
legend('\omega','\omega_{est}','error','Location','southeast')
```



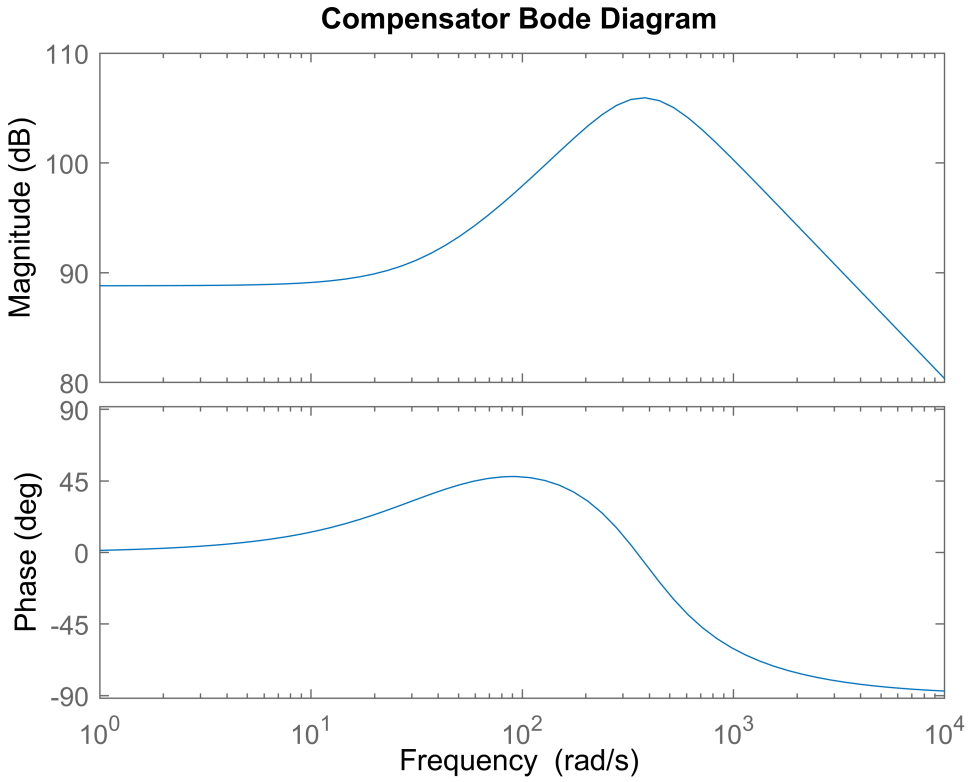
Problem #4

Part A)

The compensator is described by an $A_{CL} = A - BK - LC$, $B = L$, and $C = K$. The state-space compensator was then created using ss() and converted to a transfer function using tf() for use in Part B). A Bode Plot for

the compensator was produced using bode() and shows the equivalent compensator has an initial zero that provides phase lead, making it most similar to a lead compensator.

```
A_comp = A - B*K - L*C;
B_comp = L;
C_comp = K;
D_comp = D;

comp_ss = ss(A_comp,B_comp,C_comp,D_comp);
comp_tf = tf(comp_ss);

figure;
bode(comp_ss);
title('Compensator Bode Diagram')
```



Part B)

The closed loop system was created using the feedback() function with the compensator and open loop system transfer functions. The closed loop transfer function is as follows:

$$\frac{\theta[s]}{\tau[s]} = \frac{1.04e7s + 3.986e8}{s^4 + 527.8s^3 + 1.413e5s^2 + 1.042e7s + 3.896e8}$$

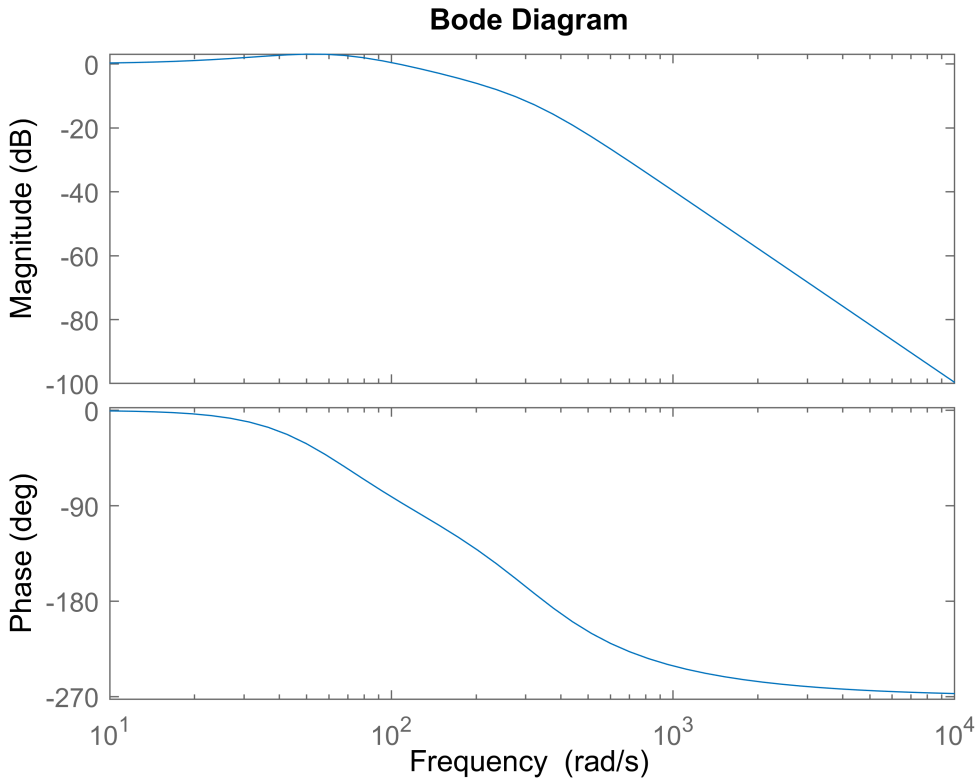
```
comp_cl = feedback(comp_tf*sys_tf,1)
```

```
comp_cl =

                  1.04e07 s + 3.896e08
  ------------------------------------------------------
  s^4 + 527.8 s^3 + 1.413e05 s^2 + 1.042e07 s + 3.896e08

Continuous-time transfer function.
```

Part C)

The Bode Plot of the closed loop system was created using bode().

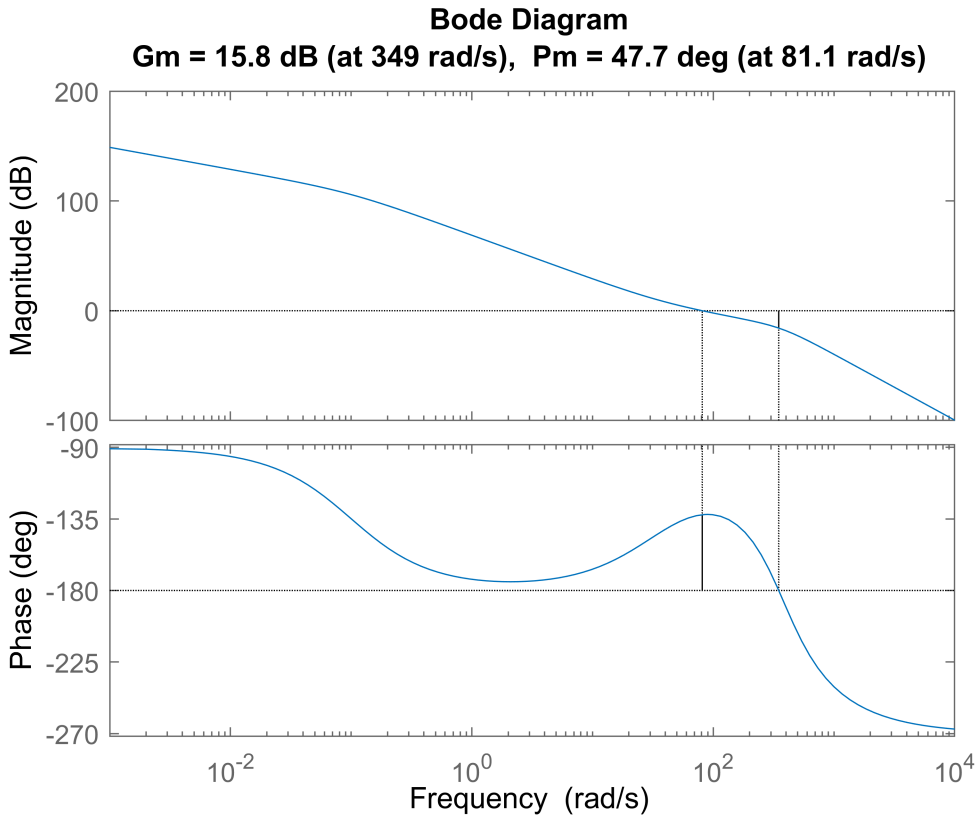
```
figure;
bode(comp_cl)
```



Part D)

The gain and phase margin of the open loop system were calculated using margin().

```
margin(comp_tf*sys_tf)
```



Problem #5

Part A)

The open loop system model was discretized using the c2dm() function and then turned into a state-space system using ss(). The eigenvalues of the system were then found with eig()

```
[A_z, B_z, C_z, D_z] = c2dm(A, B, C, D, dt); % Continuous to Discrete

sysz_ss = ss(A_z,B_z,C_z,D_z) % Discretized Open Loop SS
```

```
sysz_ss =

  A =
          x1      x2
   x1      1   0.001
   x2      0  0.9999

  B =
          u1
   x1   5e-08
   x2  0.0001

  C =
       x1  x2
   y1   1   0

  D =
       u1
   y1   0
```

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

```
sysz_tf = tf(sysz_ss); % Discretized Open Loop TF

z_sys = eig(A_z) % Discretized Open Loop Eigenvalues
```

```
z_sys = 2×1
    1.0000
    0.9999
```

Part B)

The desired eigenvalues for the observer were discretized and L was calculated in the same fashion as Problem #2, Part B).

```
z_des_ob = exp(s_des_ob*dt); % Discretized Observer Eigenvalues

L_z = place(A_z', C_z', z_des_ob)'; % Discretized Observer Gain Matrix
```

Part C)

The desired eigenvalues for the controller were discretized and K was calculated in the same fashion as Problem #3, Part B).

```
z_des_co = exp(s_des_co*dt); % Discretized Controller Eigenvalues

K_z = place(A_z, B_z, z_des_co); % Discretized Controller Gain Matrix
```

Part E)

The closed loop observer and controller (equivalent compensator) poles were found with eig() after taking the discrete compensator and discrete system transfer functions creating the discrete closed loop system with feedback().

```
A_compz = A_z - B_z*K_z - L_z*C_z;
B_compz = L_z;
C_compz = K_z;
D_compz = 0;

compz_ss = ss(A_compz,B_compz,C_compz,D_compz);
compz_tf = tf(compz_ss);

compz_cl = feedback(compz_tf*sysz_tf, 1);

z_compz_cl = eig(compz_cl)
```

```
z_compz_cl = 4×1 complex
   0.9560 + 0.0429i
   0.9560 - 0.0429i
   0.7825 + 0.1786i
   0.7825 - 0.1786i
```

Part F)

The discretized closed loop transfer function was also found with feedback() in Part E).

$$\frac{\theta[s]}{\tau[s]} = \frac{0.004225s^2 + 0.0001495s - 0.004076}{s^4 + 3.477s^3 + 4.552s^2 - 2.665s + 0.5899}$$

```
compz_cl
```

```
compz_cl =

      0.004225 s^2 + 0.0001495 s - 0.004076
  ------------------------------------------
  s^4 - 3.477 s^3 + 4.552 s^2 - 2.665 s + 0.5899

Continuous-time transfer function.
```

Problem #6

The actual and estimated angular position step responses of the continuous and discrete equivalent compensator were plotted using the same simulation method and initial conditions in Problem #3, Part C). The responses show that the discrete system slightly lags, but its response is essentially indistinguishable from the continuous compensator's response.

```
% Plotting
figure

subplot(2,1,1)
plot(t, x(1,:)) % Plot Actual & Estimated States
hold on

subplot(2,1,2)
plot(t, xhat(1,:))
hold on

u = zeros(1,length(t)); % Clearing Variables & Preallocation
y = zeros(1, length(t));
x = zeros(2,length(t));
dx = zeros(2,length(t));
xhat = zeros(2,length(t));
dxhat = zeros(2,length(t));

x(:,1) = [0; 0]; % Initialized State & State Estimate
xhat(:,1) = [1; 0];

% Simulation
for i = 1:length(t) - 1

    u(i) = K_z*(r -xhat(:,i)); % Control Input

    y(i) = C_z*x(:,i); % Actual State "Measurements"

    x(:,i+1) = A_z*x(:,i) + B_z*u(i); % Actual State Derivative
```

```
    xhat(:,i+1) = A_z*xhat(:,i) + B_z*u(i) + L_z*(y(i) - C_z*xhat(:,i)); % Measurement Update
end

% Plotting
subplot(2,1,1)
plot(t, x(1,:)) % Plot Actual & Estimated States
hold on

title("Continuous vs. Discrete Compensator: Actual Step Response")
xlabel('time (s)')
ylabel('position (rads)')
legend('\theta_{cont}','\theta_{disc}','Location','southeast')

subplot(2,1,2)
plot(t, xhat(1,:))
hold on

title("Continuous vs. Discrete Compensator: Estimated Step Response")
xlabel('time (s)')
ylabel('position (rads)')
legend('\theta_{cont}','\theta_{disc}','Location','southeast')
```