16-642

PS2

Boxiang Fu

boxiangf

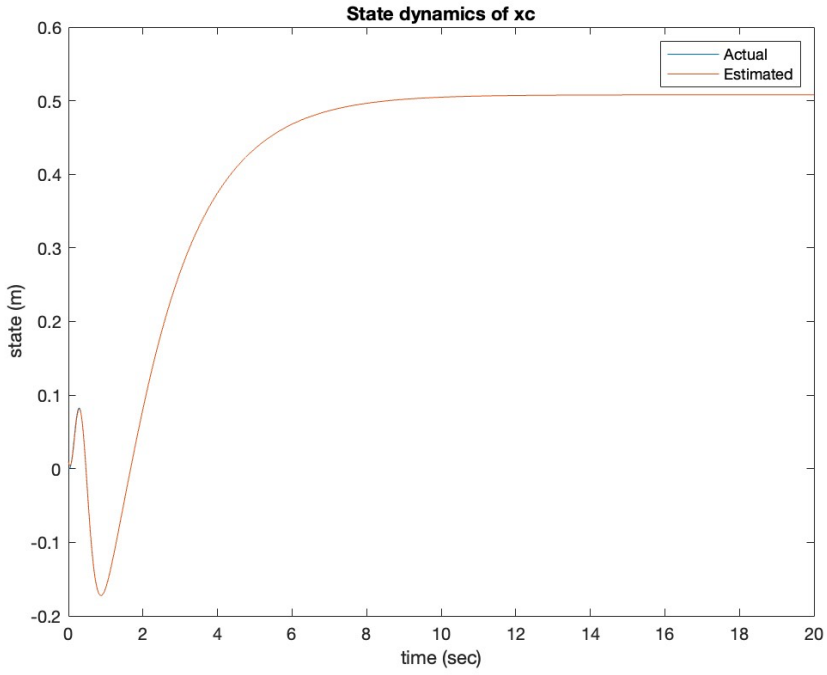
10/05/2024

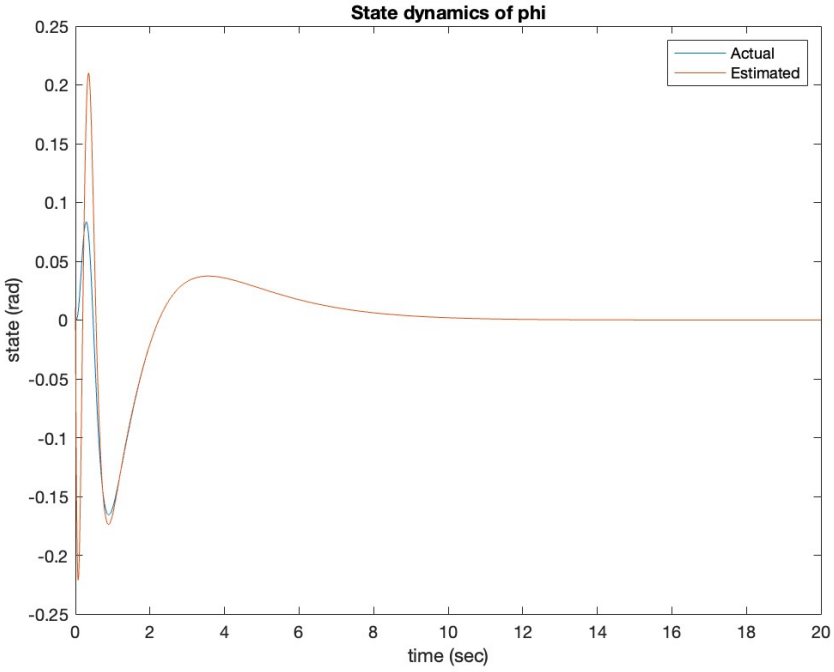
Q1.

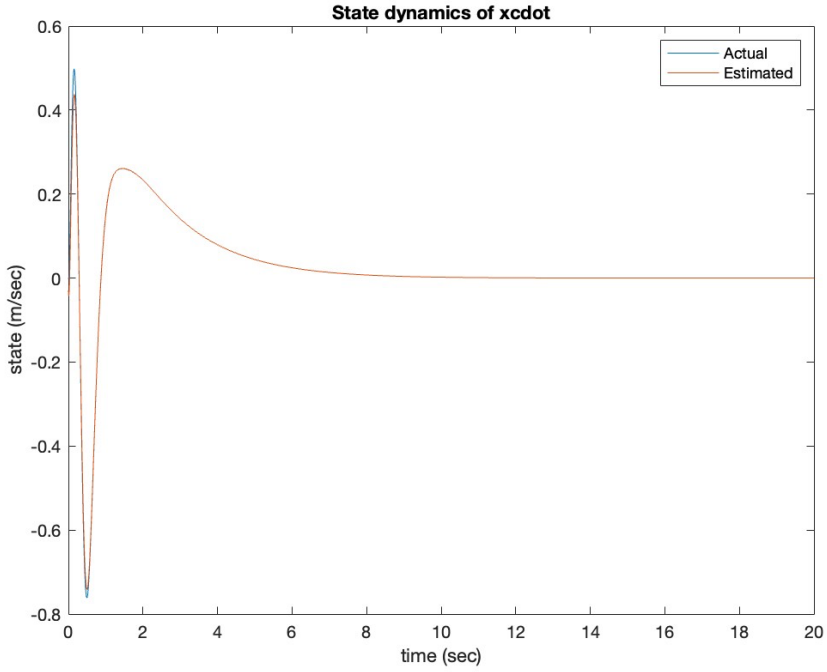
Using the parameters as Problem 2H in Problem Set 1 and the estimated starting state of $\hat{x}_0 = [0.01, 0.01, -0.03, 0.01]^T$, we design the error dynamics eigenvalues to be

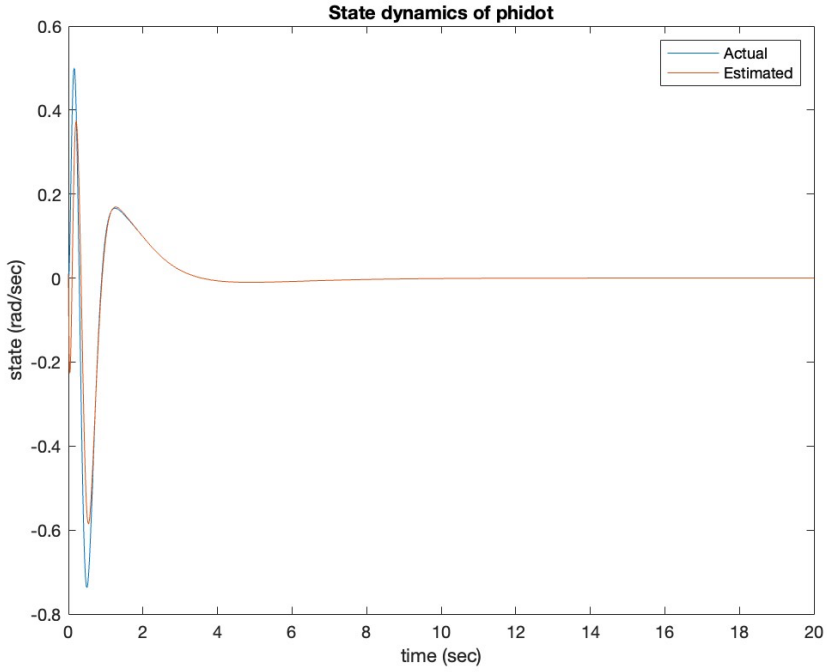
$$\Lambda_0 = [-5 + i, -5 - i, -6 + i, -6 - i]^T$$

Note that we could not set the error dynamics eigenvalues to be too far to the left since this will cause the system to be unstable (refer to Problem 2F in Problem Set 1). The state vs. time plots are as follows:




State dynamics of phi
Actual
Estimated
state (rad)
time (sec)
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
2
4
6
8
10
12
14
16
18
20


State dynamics of xcdot
Actual
Estimated
state (m/sec)
time (sec)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2
4
6
8
10
12
14
16
18
20



Note that all the estimated states converge to the actual states within a relatively short time (i.e. less than 2 seconds).

Q2.

Given the plant model

$$\ddot{y}(t) + 13\dot{y}(t) + 78y(t) = \ddot{u}(t) + 4\dot{u}(t) + 80u(t)$$

A.

Applying the Laplace transformation gives:

$$\mathcal{L}[\ddot{y}(t) + 13\dot{y}(t) + 78y(t)] = \mathcal{L}[\ddot{u}(t) + 4\dot{u}(t) + 80u(t)]$$

$$s^2Y(s) + 13sY(s) + 78Y(s) = s^2U(s) + 4sU(s) + 80U(s)$$

$$Y(s)(s^2 + 13s + 78) = U(s)(s^2 + 4s + 80)$$

$$Y(s) = \frac{s^2 + 4s + 80}{s^2 + 13s + 78}U(s)$$

$$\therefore G(s) = \frac{s^2 + 4s + 80}{s^2 + 13s + 78}$$

Where $G(s)$ is the open-loop transfer function for the system.

B.

Closing the loop with unity negative feedback and defining $T(s) = \frac{Y(s)}{R(S)}$, we get

$$Y(s) = G(s)[R(s) - Y(s)]$$

$$Y(s) = G(s)R(s) - G(s)Y(s)$$

$$Y(s)\big(1 + G(s)\big) = G(s)R(s)$$

$$\frac{Y(s)}{R(s)} = \frac{G(s)}{1 + G(s)}$$

$$T(s) = \frac{\frac{s^2 + 4s + 80}{s^2 + 13s + 78}}{1 + \frac{s^2 + 4s + 80}{s^2 + 13s + 78}}$$

$$T(s) = \frac{\frac{s^2 + 4s + 80}{s^2 + 13s + 78}}{\frac{2s^2 + 17s + 158}{s^2 + 13s + 78}}$$

$$\therefore T(s) = \frac{s^2 + 4s + 80}{2s^2 + 17s + 158}$$

C.

Using MATLAB, we find the poles and zeros of $T(s)$ to be:

$$poles = \{-4.25 + 7.8062i, -4.25 - 7.8062i\}$$

$$zeros = \{-2 + 8.7178i, -2 - 8.7178i\}$$

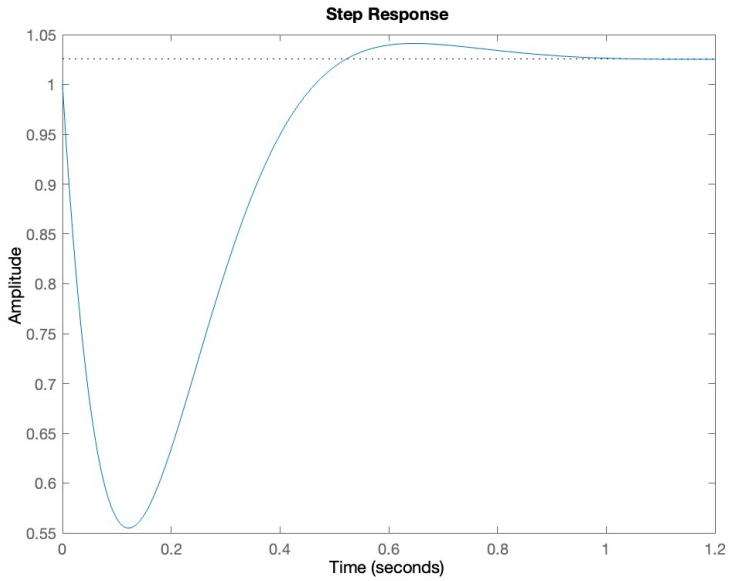
D.

Since the poles of $T(s)$ are in the negative half-plane (i.e. its real values are negative), it means the system is asymptotically stable. The system will also oscillate since the imaginary value of the poles is non-zero (and relatively large). This is because when applying the inverse Laplace transform back into the time domain, the real part of a pole forms the exponentiation term, while the imaginary part forms the sinusoidal term (Lecture 7 Slide 24). So, a large imaginary value will

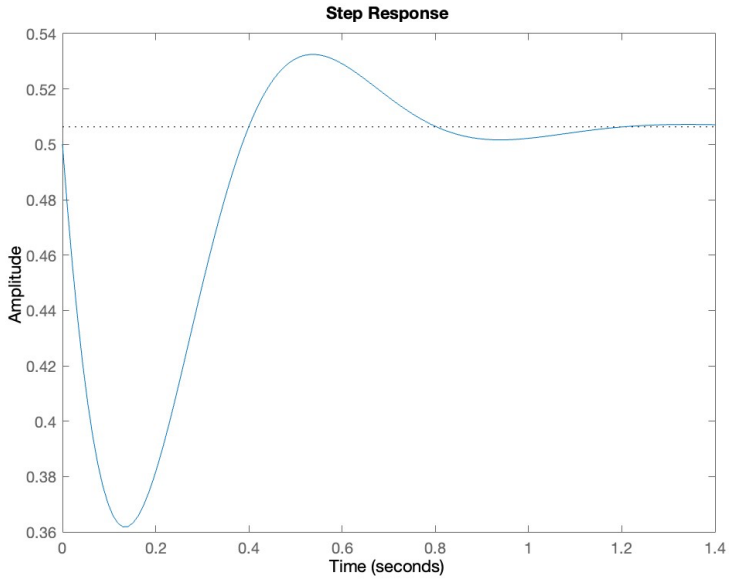
cause a highly oscillatory term. Finally, since the zeros are relatively far from the poles, it does not affect much of the dynamics of the system. There might be a slight decrease in the total response and could potentially increase maximum percent overshoot (Lecture 8 Slide 27), but any effects should not be too evident.

E.

The step-response of the open-loop system is:



The step-response of the closed-loop system is:



F.

For the open-loop system, the Final Value Theorem gives the steady state value as:

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\big(s \times Y(s)\big)$$

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\left(s \times \frac{1}{s} \times G(s)\right)$$

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\left(\frac{s^2 + 4s + 80}{s^2 + 13s + 78}\right)$$

$$\lim_{t\to\infty} y(t) = \frac{80}{78}$$

$$\lim_{t\to\infty} y(t) \approx 1.0256$$

For the closed-loop system, the Final Value Theorem gives the steady state value as:

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\big(s \times Y(s)\big)$$

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\left(s \times \frac{1}{s} \times T(s)\right)$$

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0}\left(\frac{s^2 + 4s + 80}{2s^2 + 17s + 158}\right)$$

$$\lim_{t\to\infty} y(t) = \frac{80}{158}$$

$$\lim_{t\to\infty} y(t) \approx 0.5063$$

Q3.

For the transfer function

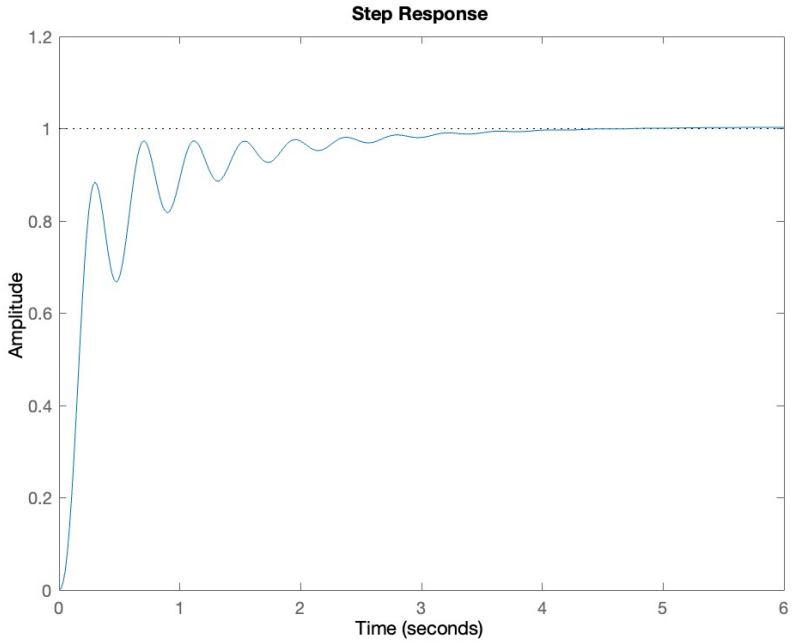
$$G(s) = \frac{20s + 17}{s^4 + 9s^3 + 231s^2 + 400s + 60}$$

We tune the PID controller as follows:

Iteration	K_p	K_i	K_d	**Rise Time (sec)**	**Max % Overshoot**	**Steady State Error**

1	1	1	1	8.1582	13.7161	0
2	10	1	1	8.7982	0	0
3[1]	50	1	1	0.5669	0	0
4	50	10	1	0.5492	0.2958	0

We obtain the required specifications on the 4th iteration. The step-response plot with these parameters is displayed below:



[1] While this meets the specifications, the convergence to the steady state is very slow. We increase K_i to decrease convergence time.

Appendices:

A.1. Code for Q1

```
% MEC
% Q1
clear;

% Parameters
gamma = 2;
alpha = 1;
beta = 1;
D = 1;
mu = 3;

% Populate A, B, and C matrix
A = zeros(4,4);
A(1,3) = 1;
A(2,4) = 1;
A(3,2) = 1;
A(3,3) = -3;
A(4,2) = 2;
A(4,3) = -3;

B = zeros(4,1);
B(3,1) = 1;
B(4,1) = 1;

C = [39.37 0 0 0];

% Check controllability
Q = [B A*B A*A*B A*A*A*B];
cont = rank(Q);

% Check observability
Q0 = [C; C*A; C*A*A; C*A*A*A];
obs = rank(Q0);

% Q matix for LQR
Qu = 10;

Qx = zeros(4,4);
Qx(1,1) = 30;
Qx(2,2) = 5;
Qx(3,3) = 5;
Qx(4,4) = 5;

% LQR
[Kc,S,P] = lqr(A,B,Qx,Qu);

% Eigenvalues of closed-loop system
eig_cl = eig(A-B*Kc);

% Design eigenvalues of error dynamics system
p0 = [complex(-5,1);
      complex(-5,-1);
      complex(-6,-1);
      complex(-6,1)];

K0 = place(A', C', p0)';

% Eigenvalues of error dynamics system
eig_ed = eig(A-K0*C);

% Tracking controller
Kf = -inv((C * inv(A - B * Kc) * B));

% Timespan
T = 0.01;
tspan = [0 20];
t_vector = 0:T:20;
```

```
 68.
 69. % Initial conditions (first 4 values are actual state, and last 4 values
 70. % are estimated state)
 71. x0 = transpose([0, 0, 0, 0, 0.01, 0.01, -0.03, 0.01]);
 72.
 73. % Run ode45 for original non-linear system
 74. [t, x] = ode45(@(t, x) odefunnl(t, x, gamma, alpha, beta, D, mu, Kc, K0, Kf, A, B, C), t_vector,
x0);
 75.
 76. % Plotting
 77. figure();
 78. plot(t_vector,x(:,1));
 79. hold on
 80. plot(t_vector,x(:,5));
 81. title("State dynamics of xc");
 82. legend("Actual", "Estimated");
 83. xlabel("time (sec)");
 84. ylabel("state (m)");
 85. hold off
 86.
 87. figure();
 88. plot(t_vector,x(:,2));
 89. hold on
 90. plot(t_vector,x(:,6));
 91. title("State dynamics of phi");
 92. legend("Actual", "Estimated");
 93. xlabel("time (sec)");
 94. ylabel("state (rad)");
 95. hold off
 96.
 97. figure();
 98. plot(t_vector,x(:,3));
 99. hold on
100. plot(t_vector,x(:,7));
101. title("State dynamics of xcdot");
102. legend("Actual", "Estimated");
103. xlabel("time (sec)");
104. ylabel("state (m/sec)");
105. hold off
106.
107. figure();
108. plot(t_vector,x(:,4));
109. hold on
110. plot(t_vector,x(:,8));
111. title("State dynamics of phidot");
112. legend("Actual", "Estimated");
113. xlabel("time (sec)");
114. ylabel("state (rad/sec)");
115. hold off
116.
117. % Create function for original non-linear ODE
118. function dxdt = odefunnl(t, x, gamma, alpha, beta, D, mu, Kc, K0, Kf, A, B, C)
119.     % Find feedback control from linearized ODE
120.     yd = 20 * square(0.02 * pi * t);
121.
122.     x1 = x(1);
123.     x2 = x(2);
124.     x3 = x(3);
125.     x4 = x(4);
126.     x1hat = x(5);
127.     x2hat = x(6);
128.     x3hat = x(7);
129.     x4hat = x(8);
130.     x = [x1; x2; x3; x4];
131.     xhat = [x1hat; x2hat; x3hat; x4hat];
132.
133.     u = -(Kc * xhat) + (Kf * yd);
134.     y = C*x;
135.
136.     % Correction term for dxdt.hat
137.     corr_term = K0 * (y - C * xhat);
138.
139.     dxdt = zeros(8,1);
```

```
    % Dynamics of actual system
    dxdt(1) = x3;
    dxdt(2) = x4;
    dxdt(3) = (-alpha*sin(x2)*beta*x4^2 + alpha*u - alpha*mu*x3 +
cos(x2)*sin(x2)*D*beta)/(alpha*gamma - beta^2*cos(x2)^2);
    dxdt(4) = (-cos(x2)*sin(x2)*beta^2*x4^2 + u*cos(x2)*beta + sin(x2)*D*gamma -
mu*x3*cos(x2)*beta)/(alpha*gamma - beta^2*cos(x2)^2);

    % Dynamics of estimated system
    dxdt(5) = x3hat + corr_term(1);
    dxdt(6) = x4hat + corr_term(2);
    dxdt(7) = (-alpha*sin(x2hat)*beta*x4hat^2 + alpha*u - alpha*mu*x3hat +
cos(x2hat)*sin(x2hat)*D*beta)/(alpha*gamma - beta^2*cos(x2hat)^2) + corr_term(3);
    dxdt(8) = (-cos(x2hat)*sin(x2hat)*beta^2*x4hat^2 + u*cos(x2hat)*beta + sin(x2hat)*D*gamma -
mu*x3hat*cos(x2hat)*beta)/(alpha*gamma - beta^2*cos(x2hat)^2) + corr_term(4);
end
```

A.2. Code for Q2

```
% MEC
% Q2
clear;

% Part c
num = [1 4 80];
denom =[2 17 158];

Ts = tf(num, denom);

zeros = zero(Ts);
poles = pole(Ts);

% Part e

step(Ts);

num_ol = [1 4 80];
denom_ol = [1 13 78];
Gs = tf(num_ol, denom_ol);
step(Gs);
```

A.3. Code for Q3

```
% MEC
% Q3
clear;

% Transfer function
num = [20 17];
denom = [1 9 231 400 60];
Gs = tf(num, denom);

% Create PID controller
Kp = 50;
Ki = 10;
Kd = 1;
Tf = 0;
C = pid(Kp,Ki,Kd,Tf);

% Closed-loop transfer function
Tcl = feedback(Gs*C, 1);

% Step Response
step_Tcl = step(Tcl, 50);

% Plot
```

```
step(Tcl);

S = stepinfo(Tcl);
disp(S);

% Steady-state error
sse = step_Tcl(end)-1;
disp(sse);
```