

## rldesign.exe Exercises for Chapter rldesign

### Exercise rldesign.quixotism

Given the design methods we've learned, comment on how the transient response of a system with P-control differs with the inclusion of integral compensation.

### Exercise rldesign.arval

How do P-, PI-, PD-, and PID-control affect a system's performance. (Limit your response to one sentence per controller.)

### Exercise rldesign.22

Let a system have plant transfer function

$$\frac{10(s + 20)}{(s + 10)(s + 4)(s + 1)}. \tag{1}$$

Design a PD controller such that the closed-loop rise time is about 0.2 seconds and the overshoot is just under 25%.

### Exercise rldesign.23

Let a system have plant transfer function

$$\frac{1}{s^3 + 22s^2 + 156s + 232}. \tag{2}$$

Design a PID controller such that the closed-loop settling time is less than 0.5 seconds, the overshoot is less than 10%, and the steady-state error is zero for a step command.

### Exercise rldesign.diurnation

Let a control system have the block diagram in Fig. exe.1, unity feedback  $H(s) = 1$ , and plant transfer function

$$G(s) = \frac{160}{s(s^2 + 16s + 160)}. \tag{3}$$

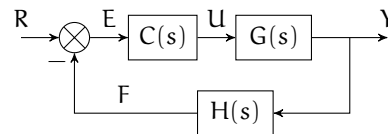


Figure exe.1: a block diagram with a controller  $C(s)$ .

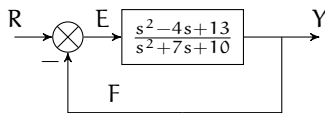
1. Design a PID controller  $C(s)$  such that the closed-loop overshoot is less than 30%, peak time is close to 0.25 seconds, and the steady-state error is zero for a ramp command.
2. Demonstrate the controller performance by simulating and plotting both a step response and a ramp response.
3. Compute the simulated overshoot and peak time (via the step response).

Exercise rldesign.sebatocal

For the system shown below a proportional controller is desired which will provide a 15% overshoot. Using Matlab please,

\_\_\_\_\_/25 p.

1. find the required damping ratio  $\zeta$ ,
2. plot the root locus,
3. design a proportional controller, and
4. simulate the step response to check your work.



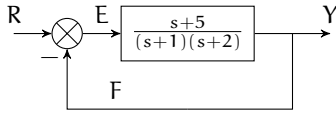
Exercise rldesign.sleep

For the system shown below a proportional-lag controller is desired which will provide a settling time of 0.5 second, and will reduce the steady state error by a factor of 5. Using Matlab please,

\_\_\_\_\_/25 p.

1. determine where on the root locus the closed loop poles should lie,
2. plot the root locus,
3. design a proportional controller,
4. design a cascade lag compensator, and

- simulate the step response to check your design with proportional control alone and with your proportional-lag compensator.



Proportional-lead-lag controller design is much like PID controller design, but the resulting controller does not require active compensation. With our techniques of cascade compensation for lead and lag compensators, one can simply apply both lead and lag compensation in the usual manner. The order of application can be somewhat important because lead compensation can impact steady-state error. A way to proceed is as follows.

- Design a P controller and evaluate its transient response performance.
- Apply lead compensation to improve the transient response. Simulate to verify the transient response performance.
- Apply lag compensation to improve the steady-state error performance.
- Check all performance criteria and adjust gains and zero locations, as-needed.

#### A design example

Let a system have plant transfer function

$$\frac{200}{s^3 + 29s^2 + 170s - 200} \quad (4)$$

Design a P-lead-lag controller such that the closed-loop overshoot is less than 20%, settling time is less than 0.7 seconds, and the steady-state error is less than 3%.

### Determining $\psi$

We use Matlab for the design.<sup>16</sup> First, we must determine what the specified transient response criteria imply for the locations of our closed-loop poles. Let one of these desired pole locations be called  $\psi$ . The transient response performance criteria are as follows.

```
Ts = .7; % sec ... spec settling time
OS = 20; % percent ... spec overshoot
sse = .03; % fraction of 1
```

The second-order approximation from [Chapter trans](#) tells us that the overshoot requirement implies a specific damping ratio  $\zeta$ , or, equivalently,  $\angle\psi$ :

$$\angle\psi = \pi - \arccos \zeta. \quad (5)$$

Additionally, the settling time requirement implies a specific  $\text{Re}(\psi)$  via

$$T_s = -4/\text{Re}(\psi). \quad (6)$$

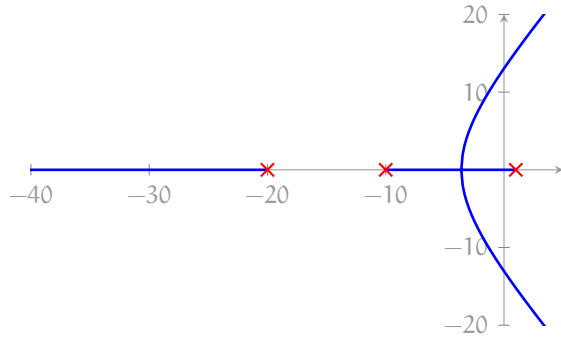
```
zeta = -log(OS/100)/sqrt(pi^2+(log(OS/100))^2);
psi_angle = pi - acos(zeta);
psi_re = -4/Ts;
psi_im = psi_re*tan(psi_angle);
psi = psi_re + j*psi_im;
disp(sprintf('psi = %0.3g + j %0.3g',real(psi),imag(psi)))
```

```
psi = -5.71 + j 11.2
```

### P control

We design a proportional controller that gets us as close as possible to  $\psi$ . The root locus is shown in [Figure multd.2](#).

```
G = tf([200],[1,29,170,-200]);
figure
rlocus(G)
```



**Figure multd.2:** root locus without compensation.

Although we cannot get close to  $\psi$  on the root locus, we can at least meet our %OS specification by choosing a gain of about

$$K_1 = 5. \tag{7}$$

Let's construct the compensator and corresponding closed-loop transfer function  $G_P$  for gain control.

```
K_1 = 5;
G_P = feedback(K_1*G,1);
```

### Lead compensation

Now, we use cascade lead compensation with compensator

$$K_2 \frac{s - z_{ld}}{s - p_{ld}}. \tag{8}$$

For now, we set  $K_2 = 1$ . Let's also set, arbitrarily,  $p_{ld} = -30$ . From Eq. 5b, we compute the compensator zero

$$\theta_c = \pi - \angle G(\psi) \quad \text{and} \quad z_c = \text{Re}(\psi) - |\text{Im}(\psi)| / \tan(\theta_c + \angle(\psi - p_c)).$$

```
p_ld = -30;
theta_ld = pi - angle(evalfr(G,psi));
theta_p_ld = angle(psi-p_ld);
z_ld = real(psi) - abs(imag(psi))/tan(theta_ld + theta_p_ld);
disp(sprintf('theta_ld = %0.3g deg',rad2deg(theta_c)))
disp(sprintf(...
'pole phase contribution = %0.3g deg',...
rad2deg(theta_p_c)...
```

```

))
disp(sprintf('z_ld = %0.3g',z_ld))

```

```

theta_ld = 48 deg
pole phase contribution = 24.7 deg
z_ld = -9.19

```

By construction,  $\psi$  is on the root locus, so we can find  $K_2$  directly from Eq. 2.

```

C_sans = zpke(z_ld,p_ld,1); % without gain
K_2 = 1/abs(evalfr(K_1*C_sans*G,psi));
C_ld = K_1*K_2*C_sans;
disp(sprintf('K_2 = %0.3g',K_2))

```

```

K_2 = 6.45

```

Let's compute the closed-loop controller  $C_{lead}$ , and the closed-loop transfer function  $G_{lead}$ .

```

G_Plead = feedback(C_ld*G,1);

```

### Lag compensation

Now, we use cascade lag compensation with compensator

$$K_3 \frac{s - z_{lg}}{s - p_{lg}} \quad (9)$$

For now, we set  $K_3 = 1$ .

The steady-state error for the lead compensated system is given by the following.

```

Kp_ld = evalfr(C_ld*G,0);
ess_ld = 1/(1+Kp_ld);
disp(sprintf('steady-state error = %0.3g',ess_ld))

```

```

steady-state error = -0.113

```

The negative value implies the output is larger than the input. Reducing this to the given requirement implies an approximate ratio of compensator zero to pole  $\alpha$ , as follows.

```

alpha = abs(ess_ld)/sse

```

```
alpha =
    3.7533
```

If we begin, somewhat arbitrarily, with  $p_{lg}$  and  $z_{lg} = \alpha p_{lg}$ . Let's construct the compensator and closed-loop transfer function  $G_{PLL}$ .

```
p_lg = -.1;
z_lg = alpha*p_lg;
C_sans = zpk(z_lg,p_lg,1);
G_PLL = feedback(C_sans*C_ld*G,1);
```

### Simulate

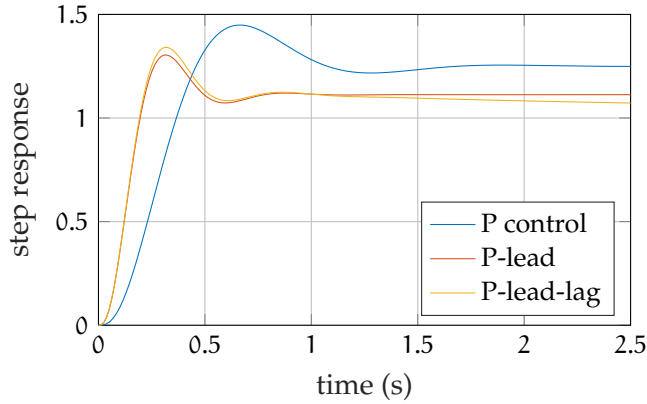
Our placement of the  $\psi$  depended on the second-order approximation's accuracy. In any case, we simulate the step response to test the efficacy of the P-lead and P-lead-lag controller designs and compare them with the P controller.

```
t_a = linspace(0,2.5,200); % s ... sim time
y_P = step(G_P,t_a); % P controlled step response
y_Plead = step(G_Plead,t_a); % P-lead step resp.
y_PLL = step(G_PLL,t_a); % P-lead-lag step resp.
```

```
figure
plot(t_a,y_P);hold on;
plot(t_a,y_Plead);
plot(t_a,y_PLL);
xlabel('time (s)');
ylabel('step response');
grid on
legend(...
    'P control','P-lead','P-lead-lag',...
    'location','southeast'...
);
```

The responses, shown in [Figure multd.3](#), suggest the lead and lead-lag compensated controllers nearly meet the transient requirements. Let's use `stepinfo` to compute more accurate transient response characteristics for the different controllers.

```
disp('P control')
si_P = stepinfo(y_P,t_a);
```



**Figure multd.3:** step responses for proportional, proportionalLead, and proportionalLead-lag controllers.

```

disp(sprintf('settling time: %0.3g',si_P.SettlingTime))
disp(sprintf('percent overshoot: %0.3g\n',si_P.Overshoot))
si_Plead = stepinfo(y_Plead,t_a);
disp('P-lead control')
disp(sprintf(...
    'settling time: %0.3g',si_Plead.SettlingTime ...
))
disp(sprintf(...
    'percent overshoot: %0.3g\n',si_Plead.Overshoot...
))
si_PLL = stepinfo(y_PLL,t_a);
disp('P-lead-lag control')
disp(sprintf(...
    'settling time: %0.3g',si_PLL.SettlingTime ...
))
disp(sprintf(...
    'percent overshoot: %0.3g\n',si_PLL.Overshoot...
))
    
```

```

P control
settling time: 1.41
percent overshoot: 16

P-lead control
settling time: 0.689
percent overshoot: 17.2

P-lead-lag control
settling time: 1.57
percent overshoot: 25.1
    
```

The stepinfo results are not very precise for the P-lead-lag controller due to the slow steady-state compensation, which isn't



completely finished by the end of the simulation. Adjusting compensator zeros and poles may improve things, but a trade-off emerges between overshoot and steady-state compensation: speeding up the latter increases the overshoot rather sharply.

The steady-state requirement can be checked analytically.

```
Kp_PLL = evalfr(C_sans*C_ld*G,0);  
ess_PLL = 1/(1+Kp_PLL);  
disp(sprintf('steady-state error = %0.3g',ess_PLL))
```

```
steady-state error = -0.0277
```

This is less than 3%, per the requirement; however, the compensation does take a relatively long time to approach this small error.