rldesign.PLag Proportional-lag controller design

PI control can be approximated by **proportional-lag** control. Instead of adding a true integrator and increasing the system type, which the integral compensator does, yielding zero steady-state error for a system and input combination with finite steady-state error, the **lag compensator** reduces the steady-state error by some finite factor in the same instance. An advantage of using a lag compensator instead of an integrator is that it can be instantiated in a passive circuit.

proportional-lag

lag compensator

Design procedure

The following procedure provides a starting-point for proportional-lag controller design. Let's assume the steady-state error design specification is to improve a finite steady-state error by a factor of α.

1. Design a proportional controller to meet transient response requirements by choosing the gain K_1 for the dominant closed-loop poles to be $p_{1,2}$.
2. Include a cascade lag compensator of the form

$$K_2 \frac{s - z_c}{s - p_c}, \tag{1}$$

   where $p_c < 0$ is a real pole near the origin; z_c is a real zero near αp_c;[2] and, initially, $K_2 = 1$. For minimal effect on the original transient response design, $\mathrm{Re}(p_{1,2}) \ll z_c$, but this is often violated for faster steady-state error compensation.
3. Use a new root locus to tune the gain K_2 such that the closed-loop poles are as

2. There are more precise ways to compute a location of z_c based on a specified factor α of steady-state error improvement that depend on the system type and command. However, given the complex tradeoffs among steady-state error, its speed, and transient response performance, we often will re-adjust the gain in any case, making optimization, here, premature.

desirable as possible. This step can often be omitted.

4. Construct the closed-loop transfer function with the controller

$$K_1 K_2 \frac{s - z_c}{s - p_c}. \tag{2}$$

5. Simulate the time response to see if it meets specifications. Tune. If the steady-state compensation is too slow, try moving z_c and/or p_c leftward. If it is too large, increase the ratio z_c/p_c.

A design example

Let a system have plant transfer function

$$\frac{s + 10}{s^2 + 8s + 25}. \tag{3}$$

Design a proportional-lag controller such that the closed-loop settling time is less than 0.4 seconds and the step response has steady-state error 10 times less than with a proportional controller, alone.

We use Matlab for the design. First, we design a proportional controller to meet the transient response performance criterion that the the settling time T_s is less than 0.4 seconds. The root locus is shown in Figure PLag.1.

```
G = tf([1,10],[1,8,25]);
figure
rlocus(G)
```

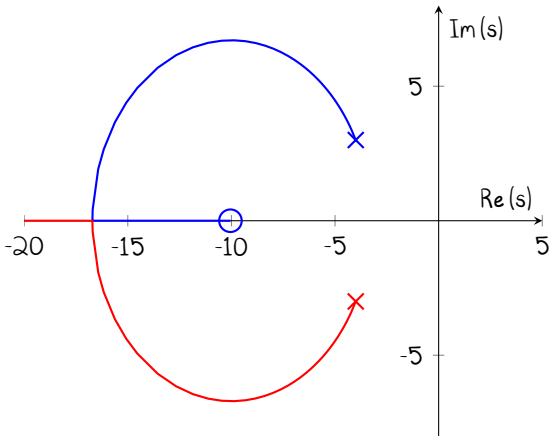

Figure PLag.1: root locus for proportional controller design.

Let's use the second-order approximation that

$$T_s \approx \frac{4}{\zeta\omega_n} = \frac{4}{-\mathrm{Re}(p_{1,2})}, \tag{4}$$

where $p_{1,2}$ are the closed-loop pole locations. For $T_s = 0.4$, $\mathrm{Re}(p_{1,2}) = -10$. This corresponds to a gain of about

$$K_1 = 12. \tag{5}$$

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

```
K1 = 12;
```

```
G_P = feedback(K1*G,1)
```

```
G_P =

     12 s + 120
  ----------------
  s^2 + 20 s + 145

Continuous-time transfer function.
```

Now, we use cascade lag compensation with compensator

$$K_2 \frac{s - z_c}{s - p_c}. \tag{6}$$

For now, we set $K_2 = 1$. Our steady-state error specification is a 10-fold factor of decrease in steady-state error, so we set $\alpha = 10$. If we begin, somewhat arbitrarily, with $p_c = -0.1$, then $z_c = \alpha p_c = -1$, which is still comfortably distant from $p_{1,2}$. Let's construct the compensator and closed-loop transfer function G_{PL}.

```
alpha = 10;
p_c = -0.1;
z_c = alpha*p_c;
C_L = zpk(z_c,p_c,1)
G_PL = feedback(K1*C_L*G,1);
```

```
C_L =

   (s+1)
  -------
  (s+0.1)

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

We could check out the root locus, but as along as we haven't botched something, it should be quite similar to the original. Let's simulate the step responses for the proportional and proportional-lag controllers.

```
t_a = linspace(0,2,100); % simulation time
y_P = step(G_P,t_a); % p control step response
y_PL = step(G_PL,t_a); % p-lag control step response
```

Let's look at the simulation results, shown in Figure PLag.2. The settling time for the proportional controller looks about right, but the steady-state error is about 18%. We'd like it to be about 1.8%. The lag compensator has a similar transient and a slow steady-state error decrease. It's so slow that we can't really evaluate its size after two seconds. Rather than extend the simulation, we choose to speed up the steady-state error compensation by moving the compensator pole and zero leftward.

```
C_L2 = zpk(2*z_c,2*p_c,1)
G_PL2 = feedback(K1*C_L2*G,1);
y_PL2 = step(G_PL2,t_a);
```

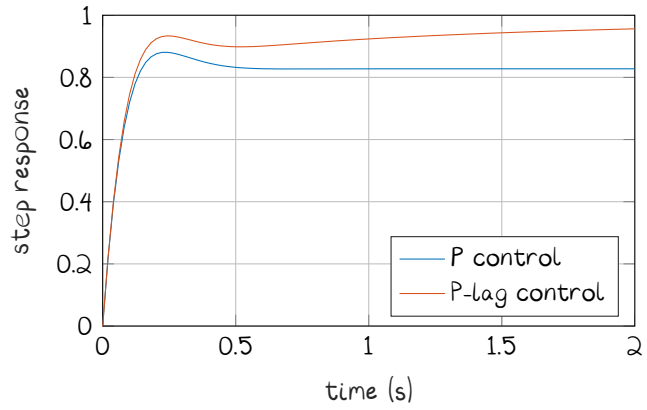

Figure PLag.2: step responses for proportional and proportional-lag controllers (initial design).

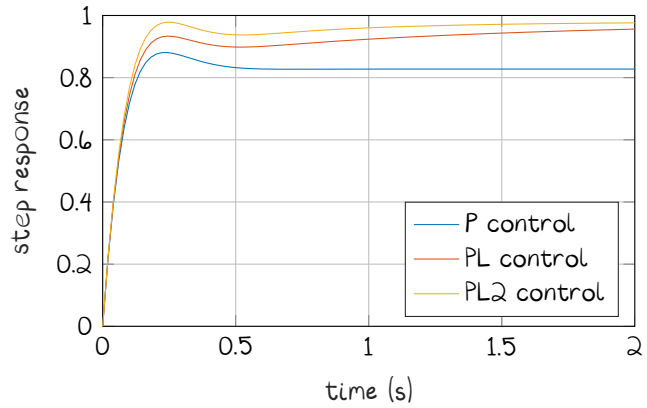

Figure PLag.3: step responses for proportional and proportional-lag controllers (secondary design).

```
C_L2 =

   (s+2)
  -------
  (s+0.2)

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

From Figure PLag.3, we see that there's improvement. Let's try increasing the gain K_2 **and** moving the compensator pole and zero leftward more aggressively to see if we can

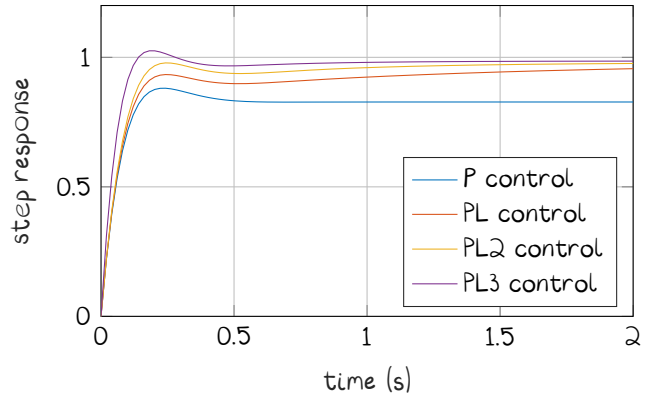

Figure PLag.4: step responses for proportional and proportional-lag controllers (tertiary design).

speed things up a bit.

```
K2 = 1.45; % compensator gain
C_L3 = K2*zpk(2.8*z_c,2.8*p_c,1)
G_PL3 = feedback(K1*C_L3*G,1);
y_PL3 = step(G_PL3,t_a);
```

```
C_L3 =

  1.45 (s+2.8)
  ------------
    (s+0.28)

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

From Figure PLag.4, it appears to meet both specifications. Let's use `stepinfo` to investigate the transient performance.

```
si_PL3 = stepinfo(y_PL3,t_a);
si_PL3.SettlingTime
```

```
ans =

    0.2660
```

This more than meets our settling time requirement of 0.4 seconds. The steady-state error can be approximated as follows.

```
disp(...
  sprintf(...
    'steady-state error: %0.3g%%',...
    100*(1-y_PL3(end))...
  )...
)
```

```
steady-state error: 1.46%
```

This meets our goal of 1.8%. Further iteration could be tighten-up the design.