# rldesign.P Proportional controller design (P)

Proportional controller design is the task of choosing the gain K with which the closed-loop system performs in a desirable manner. All three performance classes—stability, transient response, and steady-state error—can be affected by changes in the gain. However, with a gain controller there is typically no way to satisfy strict requirements in all categories. Typically, stability can be satisfied and transient response characteristics can be partially satisfied. Varying the gain simply moves the closed-loop poles along the root locus. However, often the root locus does not pass through the closed-loop pole location required for ideal transient response performance. Later, we will learn how to design controllers that do not have this limitation.

Virtually always, we assume it is a requirement for the closed-loop system to be stable, therefore we can immediately restrict our task to selecting from those values of gain K for which the system is stable.

Recall from Chapter trans the relationships between the location of closed-loop poles and the corresponding transient response performance. The parameters rise time $T_r$, peak time $T_p$, settling time $T_s$, and percent overshoot %OS can all be related to the dominant closed-loop pole locations. Criteria will be given in terms of these transient response performance parameters and the design task will be to choose the best gain K such that these requirements are met.

For most problems, we make the first- or second-order assumption for higher-order systems and for first- and second-order systems with zeros (see Lec. trans.approx). Recall that, even if this is an inaccurate assumption, it gives us a starting-point for design. We will always simulate to evaluate the actual performance criteria of a given design.

The following procedure is one way to go about designing a proportional controller. Let us keep in mind the adage that

> plans are useless, but planning is essential.

Here is the procedure.

1. Using the second- or first-order assumption, estimate the ideal location of the closed-loop poles for the desired transient response criteria.
2. Construct a root locus plot and select the location on the root locus that is closest to the desired closed-loop pole location. Using a computer, determine to which value of gain K this location corresponds.
3. Solve for the closed-loop transfer function with this gain.
4. Simulate the response for a unit step command. Evaluate the performance criteria. Iterate if necessary.

**Example rldesign.P–1** **re: proportional controller design for percent overshoot**

For a plant with transfer function

$$\frac{(s+13)(s+15)}{(s+2)(s-2)}$$

design a unity feedback gain controller such that the system has a 20 percent overshoot and minimal settling time.

We will use MATLAB. First, let's define the transfer function.

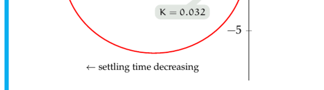
```
G=zpk([-13,-15],[2,-2],1);
```

The desired closed-loop pole location is along the ray corresponding to 20 percent overshoot. Since this is available with the data cursor in the `rlocus` plot, there is no need to compute the damping ratio or the angle of the ray. Let us consider the root locus.

```
figure
rlocus(G) % root locus
grid on
```

This yields the correct root locus, but with insufficient resolution to determine the proper gains. We can do better if we specify a higher resolution for those regions, as follows.

```
figure
Ka=sort([0:1:50,0.22:.001:0.23,
    logspace(-3,3,500),Inf]);
rlocus(G,Ka) % root locus with custom
    gains
grid on
```



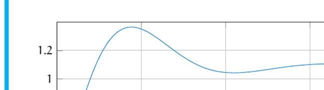

From the figure, we can see that when the gain is either $K = 0.032$ or $K = 0.234$, according to the second-order approximation, the %OS is 20. We prefer the latter because of our requirement to minimize the settling time, which decreases as the closed-loop poles move leftward. Now we must find the closed-loop transfer function, which can be found as follows.

```
Gcl=feedback(K*G,1);
```

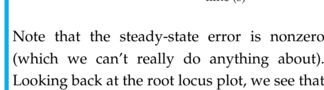
Now we are ready to simulate the step response to evaluate the actual transient response.

```
t1=5; % final time
[y,t]=step(Gcl,t1);
stepinfo(y,t)
```

The command `stepinfo` computes the actual transient response characteristics. The result is %OS = 24.5, greater than our requirement. This discrepancy is not surprising, since we were using the second-order approximation. Let's look at a plot.[a]




Note that the steady-state error is nonzero (which we can't really do anything about). Looking back at the root locus plot, we see that as the gain increases from here, the percent overshoot should decrease. We iterate the gain to obtain $K = 0.35$. The final closed-loop step response is shown, below.




In this last plot we have divided by the steady-state value such that the percent overshoot is clearly visible in the plot. This is a nice idiom, but it is important not to forget that there is still a nonzero steady-state error!

a. It is striking that the initial condition does not appear to be satisfied. This is due to the two zeros, which effectively differentiate the step input, which changes infinitely quickly at the origin.

## Example using Python

The following was generated from a Jupyter notebook with the following filename and kernel.

```
notebook filename:
↪ python_root_locus_design_example_01.ipynb
notebook kernel: python3
```

### Problem statement

For a plant with transfer function

$$\frac{15000}{s^4 + 50s^3 + 875s^2 + 6250s + 15000} \quad (1)$$

design a unity feedback proportional controller such that the closed-loop system has 10% overshoot and setting time less than one second. We begin with the usual loading of modules.

```
import numpy as np # for numerics
import control as c # the Control Systems module!
import matplotlib.pyplot as plt # for plots!
```

### Determining ψ

Let's determine a target point ψ for a closed-loop pole.

```
Ts = 1 # sec ... target settling time
OS = 10 # percent ... target overshoot
```

The second-order approximation from Chapter trans tells us that the settling time specification implies a specific Re(ψ) and the

overshoot a specific angle ∠ψ. The real part is found from the expressions

$$T_s = \frac{4}{\zeta\omega_n} \quad \text{and} \quad \mathrm{Re}(\psi) = -\zeta\omega_n \Rightarrow \quad (2)$$

$$\mathrm{Re}(\psi) = -\frac{4}{T_s}. \quad (3)$$

The angle is found via the equations

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}, \quad (4)$$

$$\tan(\angle\psi) = \pi - \frac{\sqrt{1-\zeta^2}}{\zeta}, \quad \text{and} \quad \tan(\angle\psi) = \mathrm{Im}(\psi)/\mathrm{Re}(\psi). \quad (5)$$

Remarkably simple expressions result:

$$\mathrm{Im}(\psi)/\mathrm{Re}(\psi) = \pi - \frac{\sqrt{1-\zeta^2}}{\zeta} \quad (6a)$$

$$\mathrm{Im}(\psi)/\mathrm{Re}(\psi) = \pi + \frac{\pi}{\ln(\%OS/100)}. \quad (6b)$$

So, in the final analysis, the desired pole location ψ (assuming the second-order approximation is valid) is given by the expression

$$\psi = -\frac{4}{T_s}\left(1 - j\frac{\pi}{\ln(100/\%OS)}\right). \quad (7)$$

This formula holds beyond the scope of this problem. We define it as a function.

```
def psi_fun(Ts,pOS):
  return -4/Ts*(1-1j*np.pi/np.log(100/pOS))
psi = psi_fun(Ts,OS)
print("psi = %0.3g + j %0.3g" % (np.real(psi),np.imag(psi)))
```

```
psi = -4 + j 5.46
```

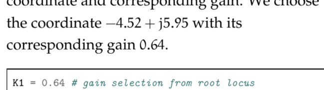
### Design with the root locus

Defining a transfer function in Python is straightforward with the Control Systems module (documentation here).

```
plant_tf = c.TransferFunction(15000,[1,50,875,6250,15000])
```

Now `plant_tf` is a transfer function object. We use the `root_locus` method of the Control Systems module and also place the target point ψ, where we'd like to have a closed-loop pole.

```
p1 = c.rlocus(plant_tf) # compute root locus
plt.plot(np.real(psi),np.imag(psi),'kx')
plt.annotate(
  '$\psi$',
  (np.real(psi),np.imag(psi)),
  textcoords='offset points',
  xytext=(20,-2),
  arrowprops={'arrowstyle':'->'}
)
plt.show() # display the plot
```




The root locus doesn't go through our test point, but it does get close. Our overshoot requirement suggests we should stay along a ray from the origin to the root locus. Double-clicking the locus yields a data cursor that gives the complex coordinate and corresponding gain. We choose the coordinate $-4.52 + j5.95$ with its corresponding gain 0.64.

```
K1 = 0.64 # gain selection from root locus
```

Now we need to evaluate via simulation the transient response performance this yields.

### Check and tune via simulation

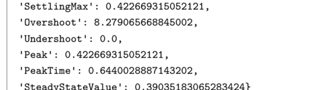
We use the Control Systems module's `feedback` method to find the closed-loop transfer function.

```
controller_tf = K1 # controller transfer function
closed_loop_tf = # closed-loop transfer function
  c.feedback(K1*plant_tf)
```

Now we can simulate the step response using the Control System module method `step_response`.

```
t,y = c.step_response(closed_loop_tf)
```

```
p2 = plt.plot(t,y)
plt.xlabel('time (s)')
plt.ylabel('step response $y(t)$')
plt.grid()
plt.show()
```




It is difficult to evaluate the performance from the graph, so we use the `step_info` method.

```
si = c.step_info(closed_loop_tf)
si
```

```
{'RiseTime': 0.28463337550583534,
 'SettlingTime': 0.9079645665577206,
 'SettlingMin': 0.35175282522378337,
 'SettlingMax': 0.422669315052121,
 'Overshoot': 8.279065668845002,
 'Undershoot': 0.0,
 'Peak': 0.422669315052121,
 'PeakTime': 0.6440028887143202,
 'SteadyStateValue': 0.39035183065283424}
```

Specifically, we want to know the overshoot and settling time.

```
print("percent OS: %3.3g" % si['Overshoot'])
print("settling time: %3.3g" % si['SettlingTime'])
```

```
percent OS: 8.28
settling time: 0.908
```

This is pretty close to the requirements. We could tune the gain to try to get closer.