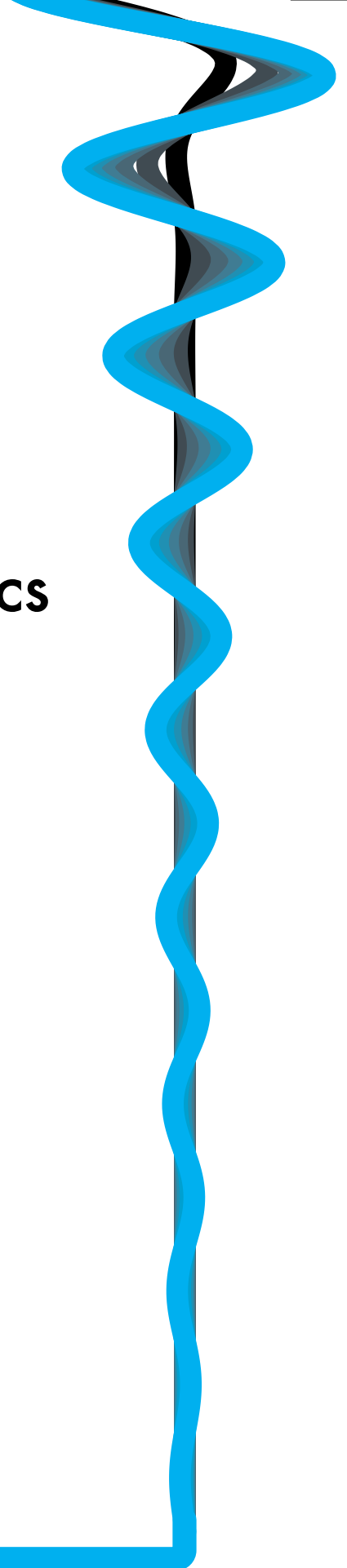




# **Control**

the kernel of cybernetics

Rico AR Picone PhD



# Control

---

the kernel of cybernetics

Rico A. R. Picone  
Department of Mechanical Engineering  
Saint Martin's University

Thursday 24 March, 2022

Copyright © 2022 Rico A. R. Picone All Rights Reserved

---

intro	Introduction	8
intro.perf	Performance . . . . .	10
	Stability . . . . .	10
	Transient Response . . . . .	10
	Steady-State Response . . . . .	10
	Others . . . . .	10
intro.block	Feedback control system block diagrams . . . . .	12
intro.pid	Introducing PID control . . . . .	14
	Ziegler–Nichols tuning method . . . . .	15
intro.pidi	An interactive PID controller design . . . . .	18
	Symbolic transfer functions . . . . .	18
	Symbolic to control transfer functions . . . . .	19
	Defining the closed-loop function . . . . .	20
	Step response . . . . .	20
	Interactive step response . . . . .	21
intro.exe	Exercises for Chapter intro . . . . .	23
	Exe. intro.tabernacle . . . . .	23
	Exe. intro.psalmody . . . . .	23
	Exe. intro.calvous . . . . .	23
	Exe. intro.telesis . . . . .	23
	Exe. intro.postulant . . . . .	23
	Exe. intro.mascaron . . . . .	23
	Exe. intro. . . . .	24
stab	Stability performance	25
stab.intro	Introduction . . . . .	26
	Stability defined by the free response . . . . .	26
	Stability defined by the forced response . . . . .	26
stab.tf	Stability from the transfer function . . . . .	28
	Stability from the poles of a closed-loop transfer function . . . . .	28
	Stability from the form of a closed-loop transfer function . . . . .	29

	stab.routh	Routh-Hurwitz criterion . . . . .	31
		An algorithm for applying the Routh-Hurwitz criterion . . . . .	31
	stab.exe	Exercises for Chapter stab . . . . .	34
		Exe. stab.saginate . . . . .	34
		Exe. stab.spleniculus . . . . .	34
		Exe. stab.break . . . . .	34
		Exe. stab.relax . . . . .	34
trans	Transient response performance		36
	trans.char	Transient response characteristics . . . . .	38
	trans.exact	Exact analytical trans response char of first- and second-order sys .	39
		First-order systems without zeros . . . . .	39
		Second-order systems without zeros . . . . .	40
	trans.approx	Approx analytical transient response characteristics . . . . .	42
	trans.sim	Simulation . . . . .	43
	trans.exe	Exercises for Chapter trans . . . . .	46
		Exe. trans.apiarian . . . . .	46
		Exe. trans.pericentral . . . . .	46
		Exe. trans.rest . . . . .	46
steady	Steady-state response performance		48
	steady.error	Steady-state error for unity feedback systems . . . . .	49
	steady.exe	Exercises for Chapter steady . . . . .	52
		Exe. steady.hypnomancy . . . . .	52
		Exe. steady.nap . . . . .	52
rlocus	Root locus analysis		53
	rlocus.def	Root locus definition . . . . .	54
		Closed-loop poles are hard to find . . . . .	54
		Definition . . . . .	55
		The magnitude and phase criteria . . . . .	55
		What about negative gains and positive feedback? . . . . .	56
	rlocus.sketch	Sketching the root locus . . . . .	57
	rlocus.comp	Generating the root locus via a computer . . . . .	62
		Matlab . . . . .	62
		Python . . . . .	63
	rlocus.exe	Exercises for Chapter rlocus . . . . .	65
		Exe. rlocus.burritosteve . . . . .	65
		Exe. rlocus.dunnage . . . . .	66
		Exe. rlocus.respite . . . . .	67
rldesign	Root-locus design		69

rldesign.gain	Gain from the root locus . . . . .	70
	Gain, analytically and geometrically . . . . .	70
	Gain, the easy way . . . . .	71
rldesign.P	Proportional controller design (P) . . . . .	72
	Example using Python . . . . .	76
rldesign.beyondP	Beyond proportional design . . . . .	81
rldesign.PI	Proportional-integral (PI) controller design . . . . .	82
	Design procedure . . . . .	82
rldesign.PLag	Proportional-lag controller design . . . . .	87
	Design procedure . . . . .	87
	A design example . . . . .	88
rldesign.PD	Proportional-derivative (PD) controller design . . . . .	93
	Design procedure . . . . .	94
	A design example . . . . .	95
rldesign.PLead	Proportional-lead design . . . . .	101
	Design procedure . . . . .	102
	A design example . . . . .	103
rldesign.PID	Prop-integral-derivative controller design . . . . .	109
	A design example . . . . .	110
rldesign.PLeLa	Proportional-lead-lag controller design . . . . .	119
	A design example . . . . .	119
rldesign.multd	Multiple derivative compensators . . . . .	126
	Causality . . . . .	127
rldesign.exe	Exercises for Chapter rldesign . . . . .	128
	Exe. rldesign.quixotism . . . . .	128
	Exe. rldesign.arval . . . . .	128
	Exe. rldesign.22 . . . . .	128
	Exe. rldesign.23 . . . . .	128
	Exe. rldesign.diurnation . . . . .	128
	Exe. rldesign.sebatical . . . . .	129
	Exe. rldesign.sleep . . . . .	129
	A design example . . . . .	130
rldesign.exe	Exercises for Chapter rldesign . . . . .	137
	Exe. rldesign.quixotism . . . . .	137
	Exe. rldesign.arval . . . . .	137
	Exe. rldesign.29 . . . . .	137
	Exe. rldesign.30 . . . . .	137
	Exe. rldesign.diurnation . . . . .	137
	Exe. rldesign.sebatical . . . . .	138
	Exe. rldesign.sleep . . . . .	138

freq	Frequency response analysis	140
freq.intro	Introduction . . . . .	141
freq.bode	Bode plots . . . . .	143
freq.bodesimp	Bode plots for simple transfer functions . . . . .	145
freq.bodesketch	Sketching Bode plots . . . . .	149
freq.nyquist	Nyquist criterion . . . . .	152
	Introduction . . . . .	152
	A description of the Nyquist criterion . . . . .	152
	Sketching Nyquist plots . . . . .	155
freq.nystab	Stability from the Nyquist plot . . . . .	159
	Stability from the positive $j\omega$ -axis image, alone . . . . .	159
	Gain margin and phase margin . . . . .	160
freq.nybode	Stability, GM, and PM from Bode plots . . . . .	162
freq.freqtime	Relations among time and frequency domain reps . . . . .	164
	The second-order assumption . . . . .	164
	Closed-loop percent overshoot from the closed-loop bandwidth . . . . .	164
	Closed-loop %OS and $\zeta$ from $\Phi_M$ . . . . .	165
	Closed-loop $T_s$ and $T_p$ from the open-loop system . . . . .	166
freq.exe	Exercises for Chapter freq . . . . .	168
freqd	Frequency response design	169
freqd.gain	Transient response design by adjusting the gain . . . . .	170
freqd.exe	Exercises for Chapter freqd . . . . .	171
	Exe. freqd.libricide . . . . .	171
ss	State-space design	172
ss.sfdbck	Controller design method . . . . .	173
	Solving for the gain via the phase-variable canonical form . . . . .	174
	Steady-state error . . . . .	176
ss.exe	Exercises for Chapter ss . . . . .	180
A	Mathematical topics	181
A.01	Complex functions . . . . .	182
	A geometric interpretation of complex functions . . . . .	183
B	Linear systems theory topics	185
B.01	Controllability, observability, and stabilizability . . . . .	186
	Controllability . . . . .	186
B.02	Canonical forms of the state model . . . . .	187
	Phase-variable canonical form . . . . .	187
C	Physical topics	189

C.01	Decibels . . . . .	190
Bibliography		191

## Introduction

1 Control theory is the study of control systems: systems that control the behavior of other systems. In control theory, a system—usually called the plant—is analyzed, often with system dynamics, then a control system is designed to control the plant.

2 A control system usually controls the plant via a controller that changes one or more of the plant's inputs. The plant's outputs are variables we would like to know and/or control. A graphical representation of this called a block diagram is shown in [Fig. intro.1](#).

3 If a plant is mathematically modeled with sufficient accuracy, a controller can specify the plant's input (which is the controller output) to produce the desired plant output. This is called open-loop control. However, most plants are not modeled well-enough to do this. It is especially difficult to model outside disturbances, like sudden jolts from being bumped and environmental interference. Moreover, small errors in modeling can accumulate over time.

4 For these reasons, most control systems include feedback: measurements of the plant's outputs, as shown in [Fig. intro.2](#). Frequently, the feedback is compared to a command: the desired output. The difference is the error, from which the controller determines the control effort (the plant's input). Determining how the controller should respond to effectively control

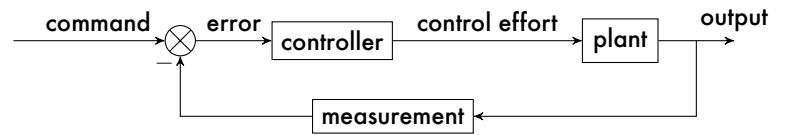


**Figure intro.1:** an open-loop control system block diagram.



the plant's output (i.e. minimize error) is the subject of much of control theory.

5 There are several types of controllers used in control theory, and many adjustable parameters for each type. Determining which type of controller and parameter values are best for a given system are some of the primary tasks of the control engineer. There are design trade-offs to be made. Some controllers will be more expensive to implement than others because they will require more-expensive hardware. And some controllers will perform better than others in ways we will explore in [Lec. intro.perf](#).



**Figure intro.2:** a feedback control system block diagram.

## intro.perf Performance

1 Control system performance is measured in the following ways.

### Stability

2 A control system must be designed such that the plant's output response is stable: its free response must not diverge from equilibrium.

### Transient Response

3 Transient response of the plant's output is often important for a control system. A designer may have identified such requirements as "the velocity free response must be  $0 \pm 1$  m/s in 5 seconds and thereafter." Or "the pressure step response must not overshoot its final value." These sorts of requirements are common. As with many design techniques, some iteration is usually needed in order to meet all requirements.

### Steady-State Response

4 Steady-state response of the plant's output is another important consideration for a control system. After the transient response has decayed, the steady-state response must meet certain criteria, such as "the position steady-state response to a unit ramp function must be within 5 mm of the desired position."

### Others

5 Cost, weight, complexity, and many other factors must be considered in control system design. One of the most important of these is robustness: the control system's ability to perform as desired when system parameters change from their nominal values. This is important because the parameters of any

implementation of a control system will differ from their nominal values at least slightly.

## intro.block Feedback control system block diagrams

1 As we have already seen, a useful tool for designing control systems is the block diagram. The plant and the controller are represented as blocks. Usually a transfer function (or transfer function matrix) can describe the function of each block. A typical block diagram is shown in Fig. def.1.

2 In this configuration, a command function  $R(s)$  is provided to the control system. The feedback  $H(s)Y(s)$  is subtracted from  $R(s)$  to give the error  $E(s)$ . This is fed to the controller  $C(s)$ . The output of the controller is the control effort  $U(s)$ , which is the input of the plant  $G(s)$ . The output  $Y(s)$ , after being fed back as  $H(s)Y(s)$ , is what the control system is attempting to make equal to the command  $R(s)$ , therefore, ideally  $E(s) = 0$ .

3 Block diagrams express algebraic relationships. (The blocks do not dynamically “load” each other.) In the case of Fig. def.2, the relationships are

$$E(s) = R(s) - F(s) \tag{1a}$$

$$U(s) = C(s)E(s) \tag{1b}$$

$$Y(s) = G(s)U(s) \tag{1c}$$

$$F(s) = H(s)Y(s). \tag{1d}$$

The closed-loop transfer function is defined as  $Y(s)/R(s)$ . This important transfer function shows how the system should respond to commands, of key importance for most performance criteria.

### Example intro.block-1

Given the feedback block diagram of Fig. def.1 (left), solve for the closed loop transfer function  $Y(s)/R(s)$ .

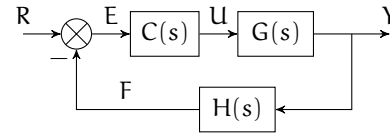


Figure block.1: a block diagram for a controller  $C(s)$ .

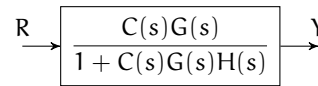


Figure block.2: a block diagram with the corresponding closed-loop transfer function block, derived in Example intro.block-1.

### re: Closed-loop transfer function



# intro.pid Introducing PID control

1 One of the most ubiquitous types is the proportional-integral-derivative (PID) controller. It has a transfer function with real constants  $K_P$ ,  $K_I$ , and  $K_D$ :

$$C(s) = \underbrace{K_P}_{\text{proportional}} + \underbrace{K_I/s}_{\text{integral}} + \underbrace{K_D s}_{\text{derivative}} . \quad (1)$$

Remember: the controller operates on the error  $E(s)$ , so the PID controller effectively sums terms proportional to the error, its integral, and its derivative. Inspecting this in the time domain with error  $e(t)$  by taking the inverse Laplace transform of the output  $U(s) = C(s)E(s)$ ,

$$u(t) = \underbrace{K_P e(t)}_{\text{proportional}} + K_I \underbrace{\int_0^t e(\theta) d\theta}_{\text{integral}} + \underbrace{K_D \dot{e}(t)}_{\text{derivative}} . \quad (2)$$

2 So the control effort  $u$  is responsive to:

- P** the amount and direction of error (reactive, spring-like),
- I** the accumulation of error over time (memoried, mass-like), and
- D** the time rate of change of the error (anticipatory, damper-like).

Although the mechanical spring-mass-damper analog above has its limitations, it is helpful for our intuition. More generally, we can consider the three constants  $K_P$ ,  $K_I$ , and  $K_D$  to be “knobs” with which we can include more or less of each term.

3 Just how a controller will affect the closed-loop response is significantly dependent on the plant dynamics. Therefore, there is no way to make fully general statements about the impact of each of the PID terms. This is why we need the detailed analytic design tools of [Chapter rldesign](#) and the intervening chapters hence. However, for some simple systems, we can make the assertions of [Table pid.1](#).

4 There are many methods of tuning a PID controller: selecting  $K_P$ ,  $K_I$ , and  $K_D$  to meet certain performance criteria. The root locus design method of [Chapter rldesign](#) and the

**Table pid.1:** occasionally true generalities about PID controller terms.

Proportional	Integral	Derivative
<ul style="list-style-type: none"> <li>• is the workhorse</li> <li>• speeds up responses</li> <li>• can lead to instability when too large</li> </ul>	<ul style="list-style-type: none"> <li>• improves or eliminates steady-state error</li> <li>• slows down the response</li> <li>• becomes a liability when it can't forget (integral windup)</li> </ul>	<ul style="list-style-type: none"> <li>• speeds up the response</li> <li>• can yield jitter when measurement noise is large</li> <li>• can lead to instability when measurement noise is large</li> </ul>

frequency response design method of Chapter freqd allow us to precisely design for specific performance criteria. However, there are times when specific performance criteria and involved analysis are not available or convenient. In these cases, hand-tuning is possible via several algorithms. One such algorithm is presented in the following section.

Ziegler–Nichols tuning method

5 The Ziegler–Nichols method of tuning a PID controller is presented in the following algorithm.

1. Set  $K_P, K_I, K_D = 0$ .
2. Increase  $K_P$  until a marginally stable response<sup>1</sup> is observed.
3. Record this ultimate gain  $K_u$  and the oscillation period  $T_u$ .
4. Set the controller gains:

1. This can be the impulse, step, or free response. Furthermore, it can be oscillatory.

$$K_P = 0.6K_u \quad K_I = 1.2K_u/T_u \quad K_D = 3K_u T_u/40. \tag{3}$$

Example intro.pid–1

For the block diagram of Fig. pid.1, with the plant

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

use the Ziegler–Nichols method to design a PID controller  $C(s)$ .

We proceed with Matlab, symbolically at first. Let's define the transfer functions.

```
syms S kp ki kd % S is the laplace transform s
G_sym = 15000/(S^4+50*S^3+875*S^2+6250*S+15000); % plant
C_sym = kp + ki/S + kd*S; % PID controller transfer fun
```

From the preceding lecture's ??, the closed-loop transfer function is as follows.

```
CL_sym = simplify( ...
    C_sym*G_sym/(1+C_sym*G_sym) ...
```

re: hand–tuning a PID controller

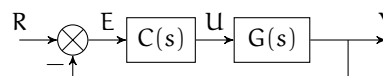


Figure pid.1: block diagram for Example intro.pid-1.

```
)

CL_sym =

(15000*kd*S^2 + 15000*kp*S + 15000*ki)/(15000*S +
↪ 15000*ki + 15000*S*kp + 15000*S^2*kd + 6250*S^2
↪ + 875*S^3 + 50*S^4 + S^5)
```

I have created a function `sym_to_tf` that creates a `tf` object, which we'll need for simulation.<sup>a</sup>

```
type sym_to_tf.m
```

```
function tf_obj = sym_to_tf(sym_tf,s_var)
% TODO test to make sure s_var is in
↪ symvar(sym_tf) ...
syms(symvar(sym_tf))
syms s
sym_tf = subs(sym_tf,s_var,s);
tf_str = char(sym_tf);
s = tf([1,0],[1]);
eval(['tf_obj = ',tf_str,'];)
```

Let's wrap it in a function of our own `K_sub`, which will create a closed-loop `tf` object from our `CL_sym` with the PID gains included.

```
K_sub = @(Kp,Ki,Kd) sym_to_tf( ...
    subs( ...
        CL_sym, ...
        {kp,ki,kd}, ...
        {Kp,Ki,Kd} ...
    ), ...
    S ...
);
K_sub(1,0,0) % e.g.
```

```
ans =

          15000 s
-----
s^5 + 50 s^4 + 875 s^3 + 6250 s^2 + 30000 s

Continuous-time transfer function.
```

Now let's use `impulse` to simulate the response starting with a small proportional gain.

```
[y,t] = impulse(K_sub(1,0,0));
```



Now, we should plot the result – see Fig. pid.2.

```
figure
plot(t,y)
grid on
xlabel('time (s)')
ylabel('impulse response')
```

If we iteratively increase  $K_P = 1 \rightarrow 3 \rightarrow 5.25$  (the response for each of these values is plotted in Fig. pid.3), we find that around the last value, the system becomes marginally stable and therefore

$$K_u = 5.25. \tag{4}$$

The oscillation period appears to be around  $T_u = 0.56$  seconds. Defining these quantities, we can now compute  $K_I$  and  $K_D$  from Eq. 3.

```
Ku = 5.25;
Tu = 0.56;
KP = 0.6*Ku;
KI = 1.2*Ku/Tu;
KD = 3*Ku*Tu/40;
disp(sprintf( ...
    'KP = %0.2f, KI = %0.2f, KD = %0.2f', ...
    KP,KI,KD ...
))
```

KP = 3.15, KI = 11.25, KD = 0.22

Let's try out this controller for step response and see how it looks.

```
[y,t] = step(K_sub(KP,KI,KD));
figure
plot(t,y)
xlabel('time (s)')
ylabel('step response')
```

The resulting step response is plotted in Fig. pid.4. We didn't have specific expectations for performance, here, but this result is a nice, average-looking step response with some overshoot and a decent settling time.

a. The function is available in the repo: [github.com/ricopicone/matlab-rico](https://github.com/ricopicone/matlab-rico).

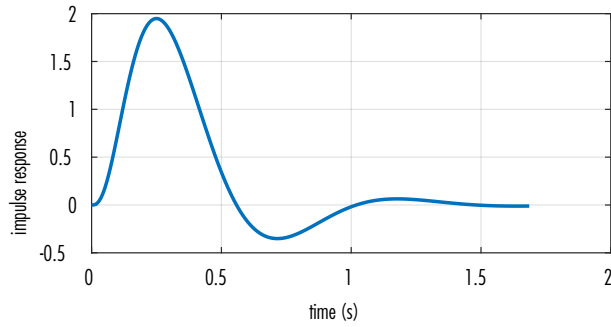


Figure pid.2: impulse response with (small)  $K_P = 1$ .

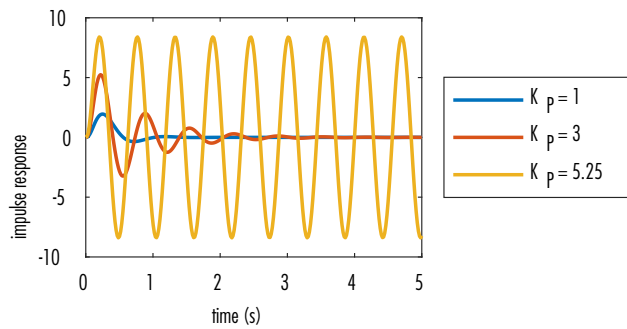
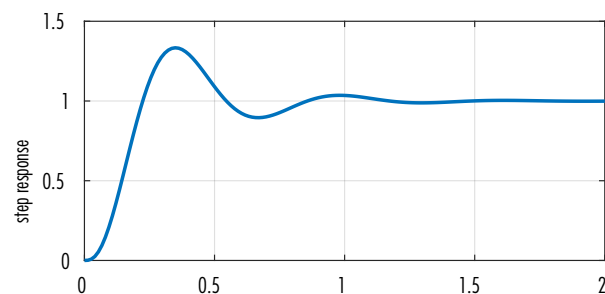




Figure pid.3: impulse responses with  $K_I = K_D = 0$  and  $K_P$  as shown.



## intro.pidi An interactive PID controller design

1 In this lecture, we will build an interactive PID control design tool in Python. However, you need not install Python<sup>2</sup> to try the design tool: it is available at the following web page.

[click to launch interactive page in browser](#)

It may take a few minutes to load the Jupyter notebook.<sup>3</sup> Once it does, click  . This will run the Python code that comprises the remainder of this lecture. Scroll to the bottom of the webpage to interact with the PID gains that update the closed-loop step response plot!

2 For the unity feedback block diagram of Fig. pidi.1, we will design a PID controller  $C(s)$ . Design requirements are (a) less than 20 percent overshoot, (b) an initial peak in less than 0.2 seconds, and (c) zero steady-state error for a step response.

First, load some general-purpose Python packages.

```
import numpy as np # for numerics
import sympy as sp # for symbolics
import control as c # the Control Systems module
import matplotlib as mpl # for plots
import matplotlib.pyplot as plt # also for plots
from IPython.display import display, Markdown, Latex
```

The following Python packages are specific for the interactive widget.

```
from ipywidgets import *
%matplotlib widget
```

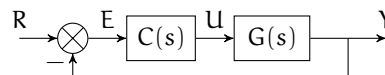
Symbolic transfer functions

Let's investigate the transfer functions symbolically. We begin by defining the Laplace  $s$  and gain symbolic variables.

```
s, K_p, K_i, K_d = sp.symbols('s K_p K_i K_d')
```

2. For more on Python, see [python.org](http://python.org).

3. For more on Jupyter, see [jupyter.org](http://jupyter.org).



**Figure pidi.1:** a unity feedback control loop.

3. Python code in this section was generated from a Jupyter notebook named `pid_interactive_design_python.ipynb` with a `python3` kernel.

We will design a PID controller for a plant with the following transfer function.

```
G_sym = 15000/(s**4+50*s**3+875*s**2+6250*s+15000)
display(G_sym)
```

$$\frac{15000}{s^4 + 50s^3 + 875s^2 + 6250s + 15000}$$

The controller has the following symbolic transfer function.

```
C_sym = K_p + K_i/s + K_d*s
display(C_sym)
```

$$K_d s + \frac{K_i}{s} + K_p$$

The closed-loop transfer function for the unity feedback system is as follows.

```
T_sym = sp.simplify(
    C_sym*G_sym/(1+C_sym*G_sym)
)
T_num, T_den = list( # for simplifying
    map(
        lambda x: sp.collect(x,s),
        sp.fraction(T_sym)
    )
)
T_sym = T_num/T_den
display(T_sym)
```

$$\frac{15000K_i + s(15000K_d s + 15000K_p)}{15000K_i + s(15000K_d s + 15000K_p + s^4 + 50s^3 + 875s^2 + 6250s + 15000)}$$

### Symbolic to control transfer functions

The control package has objects of type `TransferFunction` that will be useful for simulation in the next section. We begin by defining a function to convert a symbolic transfer function to a `control.TransferFunction` object.

```
def sym_to_tf(tf_sym,s_var):
    global s # changes s globally!
    S = s_var
    s = sp.symbols('s')
    tf_sym = tf_sym.subs(S,s)
    tf_str = str(tf_sym)
```

```
s = c.TransferFunction.s
ldict = {}
exec('tf_out = '+tf_str,globals(),ldict)
tf_out = ldict['tf_out']
return tf_out
```

This isn't smooth, but it works. Note that `tf_sym` must have no symbolic variables besides `s_var`, the Laplace  $s$ . We can apply this to `G_sym`, then, but not yet `C_sym`.

```
type(sym_to_tf(G_sym,s))
```

```
control.xferfcn.TransferFunction
```

### Defining the closed-loop function

We need to create a function that specifies the gains, substitutes them into the symbolic closed-loop transfer function, then converts it to a control package `TransferFunction` object via `sym_to_tf`.

```
def pid_CL_tf(CL_sym,Kp=0,Ki=0,Kd=0):
    sp.symbols('K_p K_i K_d')
    s = c.TransferFunction.s
    CL_subs = CL_sym.subs({K_p: Kp, K_i: Ki, K_d: Kd})
    return sym_to_tf(CL_subs,s)
```

For instance, we can let  $K_p = 1$  and  $K_i = K_d = 0$ .

```
display(
    pid_CL_tf(T_sym,Kp=1)
)
```

$$\frac{1.5 \times 10^4}{s^4 + 50s^3 + 875s^2 + 6250s + 3 \times 10^4}$$

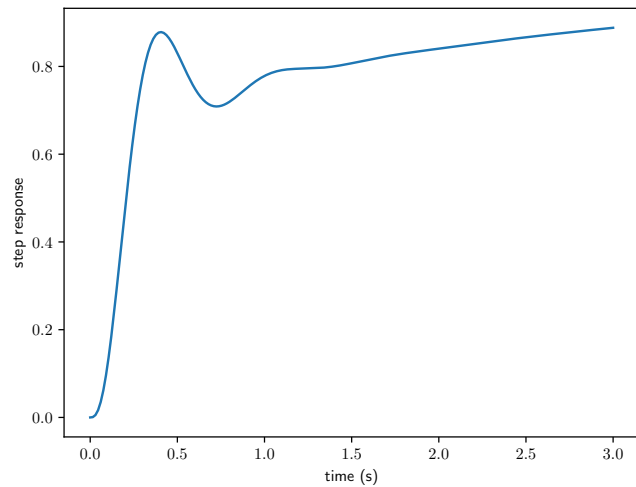
### Step response

It is straightforward to use the control package's `step_response` function to get a step response for a single set of gains.

```
gains = {'Kp':2, 'Ki':1, 'Kd':0.1}
sys_CL = pid_CL_tf(T_sym,**gains)
t_step = np.linspace(0,3,200)
t_step,y_step = c.step_response(sys_CL, t_step)
```

Now let's plot it. The result is shown in [Fig. pidi.2](#).

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
line, = ax.plot(t_step, y_step)
plt.xlabel('time (s)')
plt.ylabel('step response')
plt.show()
```



**Figure pidi.2:** step response with  $K_p, K_i, K_d = 2, 1, 0.1$ .

### Interactive step response

The following essentially repeats the same process of

1. setting the PID gains with `pid_CL_tf`,
2. simulating with `step_response`, and
3. plotting the response.

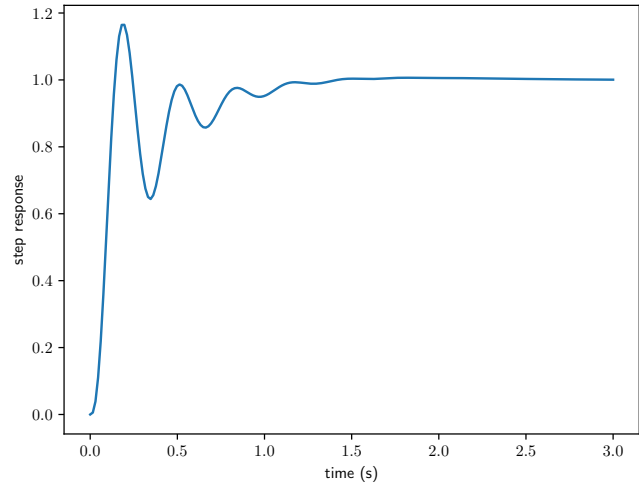
The caveat is that this happens with a GUI interaction callback function update that sets new gains (based on the GUI sliders), simulates, and replaces the old line on the plot. The final plot is shown in ???. It appears to meet our performance requirements.

```
%matplotlib widget
# simulate
t_step = np.linspace(0,3,200)
sys_CL = pid_CL_tf(T_sym,Kp=1)
t_step,y_step = c.step_response(sys_CL, t_step)

# initial plot
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
line, = ax.plot(t_step, y_step)
plt.xlabel('time (s)')
plt.ylabel('step response')
plt.show()
```

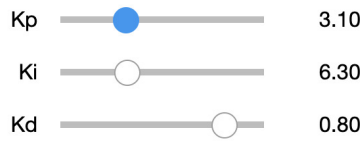
```
# GUI callback function
def update(Kp = 1.0, Ki = 0.0, Kd = 0.0):
    global t_step, kp, ki, kd
    kp,ki,kd = Kp,Ki,Kd
    sys_CL = pid_CL_tf(T_sym,Kp=Kp,Ki=Ki,Kd=Kd)
    t_step,y_step = c.step_response(sys_CL, t_step)
    line.set_ydata(y_step)
    ax.relim()
    ax.autoscale_view()
    fig.canvas.draw_idle()
    plt.show()

# interaction definition
interact(
    update,
    Kp=(0.0,10.0),
    Ki=(0.0,20.0),
    Kd=(0.0,1.0)
);
```



The sliders appear as shown in Fig. pidi.4.

**Figure pidi.3:** step response from interaction with  $K_p, K_i, K_d = 3.1, 6.2, 0.8$ .



**Figure pidi.4:** this is how the sliders should look.

## intro.exe Exercises for Chapter intro

### Exercise intro.tabernacle

If a control system has a constant controller  $C(s) = K$ , unity feedback ( $H(s) = 1$ ), and plant

$$G(s) = \frac{10}{(s+2)(s+5)},$$

what is the closed-loop transfer function?

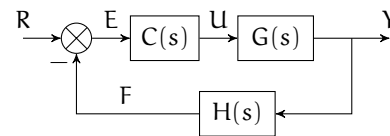
Express the result as a single fraction of polynomials in  $s$ .

### Exercise intro.psalmody

What are the three primary performance criteria for most feedback control systems?

### Exercise intro.calvous

Consider the block diagram of Fig. exe.1. Derive the transfer function from the command  $R(s)$  to the error  $E(s)$ ; that is,  $E(s)/R(s)$ . This is sometimes called the error transfer function.



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

### Exercise intro.telesis

If a PID control system suffers from poor steady-state performance, which term of a PID controller—that is, P, I, or D—is most likely to help and why?

### Exercise intro.postulant

If a PID control system suffers from slow transient response performance, increasing which PID terms—that is, of P, I, and D—are most likely to help and why?

### Exercise intro.mascaron

A given feedback control system meets its transient performance requirements, but has a finite steady-state error for a unit step command. How might you recommend

augmenting the controller to achieve zero steady-state error?

Exercise intro.



**stab**

---

# **Stability performance**

## stab.intro Introduction

Of the three most significant control system specifications—stability, transient response, and steady-state error—stability is the most important. We will now turn to stability considerations, limiting ourselves to linear, time-invariant (LTI) systems.

Recall that a system response can be considered to be composed of two parts: (1) the free response<sup>1</sup> and (2) the forced<sup>2</sup> response. This terminology will be used throughout the following.

### Stability defined by the free response

Using the concept of the free response, we define the following types of stability for LTI systems<sup>3</sup>.

1. An LTI system is asymptotically stable if the free response approaches zero as time approaches infinity.
2. An LTI system is unstable if the free response grows without bound as time approaches infinity.
3. An LTI system is marginally stable if the free response neither decays nor grows but remains constant or oscillates as time approaches infinity.

### Stability defined by the forced response

An alternate formulation of the stability definitions above is called the bounded-input bounded-output (BIBO) definition of stability, and states the following<sup>4</sup>.

1. A system is BIBO stable if every bounded input yields a bounded output.
2. A system is BIBO unstable if any bounded input yields an unbounded output.

1. This is sometimes called the “natural” or “initial condition” or “unforced” response.

2. The forced response is sometimes called the “input response.”

3. N. Nise, 2015.

4. *ibidem*.

In terms of BIBO stability, marginal stability, then, means that a system has a bounded response to some inputs and an unbounded response to others. For instance, a second-order undamped system response to a sinusoidal input at the natural frequency is unbounded, whereas every other input yields a bounded output.

Although we focus on the definitions of stability in terms of the free response, it is good to understand BIBO stability, as well.

## stab.tf Stability from the transfer function

Stability from the poles of a closed-loop transfer function

From our definitions in terms of the free response ([Lecture stab.intro](#)), we see that a closed-loop LTI system is asymptotically stable if all its poles<sup>5</sup> have negative real parts (i.e. are in the left half-plane).

Conversely, a closed-loop LTI system is unstable if it has at least one pole with a positive real part (i.e. in the right half-plane) and/or has poles of multiplicity greater than one on the imaginary axis.

Finally, a closed-loop LTI system is marginally stable if it is not unstable but has at least one pole with zero real part (i.e. on the imaginary axis) and if none of these has multiplicity greater than one.

5. Recall that poles, eigenvalues, and roots of the characteristic equation are all equivalent.

### Example stab.tf–1

Given the plant transfer function

$$G(s) = \frac{1}{(s^2 + 3)}$$

find the unity (negative) feedback closed-loop transfer function and comment on its stability. Let the command be  $R(s)$  and the output  $Y(s)$ .

**re: Stability of a closed-loop transfer function from its poles**



Stability from the form of a closed-loop transfer function

Let  $a_i, b_i, c \in \mathbb{R}$  be constant coefficients and the denominator of a closed-loop transfer function be the polynomial

$$b_n s^n + b_{n-1} s^{n-1} + \dots + b_0 = c(s - a_1)(s - a_2) \dots (s - a_n). \tag{1}$$

If a system is stable, it must have all left half-plane poles, so

1. all  $a_i$  must have negative real parts, which (non-obviously) implies that
2. all  $b_i$  must be positive and, additionally,
3. all  $b_i$  must be nonzero for  $0 \leq i \leq n$  (i.e. no “missing” powers of  $s$ ).

However, these  $b_i$  conditions are merely necessary conditions for stability, meaning that they are necessary for stability, but not sufficient (something more is needed to ensure stability).<sup>6</sup> However, if they are not met, this is a sufficient condition to draw the conclusion that the control system is unstable (i.e. nothing more needed).

6. The logical statement  $P \Rightarrow Q$  means  $P$  is sufficient for  $Q$  and  $Q$  is necessary for  $P$ . That is, if  $P$  then  $Q$  (sufficiency) and if not  $Q$  then not  $P$  (necessity). Necessity and sufficiency are duals. Let  $P$  be “the system is stable” and  $Q$  be “all  $b_i$  are positive.” Then if any  $b_i$  is negative ( $\neg Q$ ), then the system is unstable ( $\neg P$ ). But if  $Q$ , it does not necessarily follow that  $P$ —more information is required.

**Example stab.tf-2**

Given the closed-loop transfer functions

$$G_1(s) = \frac{s + 4}{(s + 3)(s + 10)(s + 22)}, \quad G_2(s) = \frac{s^2 + 2s + 5}{s^2 - 5s + 8},$$

$$G_3(s) = \frac{1}{s^3 + s + 4}, \quad \text{and} \quad G_4(s) = \frac{s^2 + 5}{s^4 + 3s^3 + s^2 + s + 3}.$$

**re: Stability of a closed-loop transfer function by inspection**

- comment on the stability of each without
- solving for poles.



## stab.routh Routh–Hurwitz criterion

There is no practical way to find the roots of a polynomial greater than degree four.<sup>7</sup> An implication of this is that we cannot practically solve (analytically) for the poles of a closed-loop transfer function with degree greater than four. Fortunately, numerical root finders can handle these higher-order systems with ease. However, there is a drawback to using numerical root finders to determine stability: design parameters, which show up in the coefficients of the denominator polynomial of a transfer function, must be assigned a specific value. A couple of mathematicians<sup>8</sup> in the late 19<sup>th</sup> century came up with a clever test—called the Routh-Hurwitz stability criterion<sup>9</sup>—for learning much about the stability of a system without computing its poles; moreover, the test yields an analytically tractable way to determine ranges over which design parameters yield stable closed-loop systems.

### An algorithm for applying the Routh-Hurwitz criterion

We consider an algorithm for this test. First, we address the “basic” algorithm and refer the reader to N. Nise (2015) for the two exceptions that arise when Column 1 has a zero or when an entire row is zero. You can teach this algorithm (including the exceptions) to a computer, as some have, but it is easy enough by-hand for many systems.

Let the denominator of a closed-loop transfer function, with real coefficients  $a_i$  be

$$a_0s^n + a_1s^{n-1} + \dots + a_{n-1}s + a_n,$$

where  $n$  a finite integer greater than or equal to the order of the numerator polynomial and  $a_0 > 0$  (if it is not, make it so by multiplication by  $-1$ ). Perform the following two steps.

7. For the interested reader, see [this stackexchange discussion](#).

8. Edward John Routh and Adolf Hurwitz were their names.

9. It is noteworthy that the criterion is based on the [Routh-Hurwitz theorem](#).

**Table routh.1:** the general form of the Routh table. Empty cells are always zero.

	1	2	3	4	...	...
$s^n$	$a_0$	$a_2$	$a_4$	$a_6$	...	...
$s^{n-1}$	$a_1$	$a_3$	$a_5$	$a_7$	...	...
$s^{n-2}$	$b_1$	$b_2$	$b_3$	$b_4$	...	...
$s^{n-3}$	$c_1$	$c_2$	$c_3$	$c_4$	...	...
$s^{n-4}$	$d_1$	$d_2$	$d_3$	$d_4$	...	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		
$s^2$	$e_1$	$e_2$				
$s^1$	$f_1$					
$s^0$	$g_1$					

First, construct a Routh table. The procedure is to fill in the general form of the Routh table, shown in **Table routh.1**, with the definitions:

$$\begin{aligned}
 b_1 &= -\frac{1}{a_1} \begin{vmatrix} a_0 & a_2 \\ a_1 & a_3 \end{vmatrix}, & b_2 &= -\frac{1}{a_1} \begin{vmatrix} a_0 & a_4 \\ a_1 & a_5 \end{vmatrix}, & b_3 &= -\frac{1}{a_1} \begin{vmatrix} a_0 & a_6 \\ a_1 & a_7 \end{vmatrix}, & \dots & (1) \\
 c_1 &= -\frac{1}{b_1} \begin{vmatrix} a_1 & a_3 \\ b_1 & b_2 \end{vmatrix}, & c_2 &= -\frac{1}{b_1} \begin{vmatrix} a_1 & a_5 \\ b_1 & b_3 \end{vmatrix}, & c_3 &= -\frac{1}{b_1} \begin{vmatrix} a_1 & a_7 \\ b_1 & b_4 \end{vmatrix}, & \dots & \\
 d_1 &= -\frac{1}{c_1} \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}, & d_2 &= -\frac{1}{c_1} \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix}, & d_3 &= -\frac{1}{c_1} \begin{vmatrix} b_1 & b_4 \\ c_1 & c_4 \end{vmatrix}, & \dots & \\
 \vdots & & \vdots & & \vdots & & \\
 g_1 &= -\frac{1}{f_1} \begin{vmatrix} e_1 & e_2 \\ f_1 & 0 \end{vmatrix}, & g_2 &= -\frac{1}{f_1} \begin{vmatrix} e_1 & 0 \\ f_1 & 0 \end{vmatrix}, & g_3 &= -\frac{1}{f_1} \begin{vmatrix} 0 & 0 \\ 0 & 0 \end{vmatrix}.
 \end{aligned}$$

Note the pattern that emerges in **Equation 1**. The number of rows and potentially nonzero columns are  $n + 1$  and  $\lceil (n + 1)/2 \rceil$ . Potentially nonzero values hug Column 1. Descending rows, the number of potentially nonzero coefficients decreases.

The second step is to interpret the Routh table. For the basic Routh table, no poles lie on the imaginary axis (which excludes marginal stability), so interpretation is simple: the number of sign changes in Column 1 is equal to the number of poles in the right half-plane—and all others are in the left half-plane. Therefore, the system is strictly stable if its Routh array is of the basic type and has no sign changes in Column 1.

**Example stab.routh-1**

Given the closed-loop transfer function

$$\frac{s + 7}{s^3 + 3s^2 + s + k} \tag{2}$$

- where  $k$  is a design parameter, using the Routh-Hurwitz criterion, find the range of  $k$  for which
- the closed-loop system is stable.

**re: Basic Routh table with an unknown parameter**



Let's build the Routh table in **Table routh.2**.  
 The lower entries were computed from **Equation 1** (n.b. we knew  $b_2 = 0$ , but compute it for demonstrative purposes) as follows:

$$b_1 = -\frac{1}{a_1} \begin{vmatrix} a_0 & a_2 \\ a_1 & a_3 \end{vmatrix} = -\frac{1}{a_1} \begin{vmatrix} - & - \\ - & - \end{vmatrix} = \underline{\hspace{2cm}},$$

$$b_2 = -\frac{1}{a_1} \begin{vmatrix} a_0 & a_4 \\ a_1 & a_5 \end{vmatrix} = -\frac{1}{a_1} \begin{vmatrix} - & - \\ - & - \end{vmatrix} = \underline{\hspace{1cm}}, \text{ and}$$

$$c_1 = -\frac{1}{b_1} \begin{vmatrix} a_1 & a_3 \\ b_1 & b_2 \end{vmatrix} = -\frac{1}{b_1} \begin{vmatrix} - & - \\ - & - \end{vmatrix} = \underline{\hspace{2cm}}.$$

**Table routh.2:** Routh table for **Example stab.routh-1**.

	1	2	3
$s^3$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^2$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^1$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^0$	$\underline{\hspace{1cm}}$	0	0

→

	1	2	3
$s^3$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^2$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^1$	$\underline{\hspace{1cm}}$	$\underline{\hspace{1cm}}$	0
$s^0$	$\underline{\hspace{1cm}}$	0	0

Now we must interpret the result. Since the first two entries in Column 1 are positive, the last two must be in order for the system stability. The conditions are:

$$\underline{\hspace{2cm}} > 0 \Rightarrow \underline{\hspace{2cm}} \text{ and } k > \underline{\hspace{1cm}}.$$

Therefore, the range for stability is  $\underline{\hspace{2cm}}$ .  
 Expressed as an interval,  $k \in \underline{\hspace{2cm}}$ .

## stab.exe Exercises for Chapter stab

### Exercise stab.saginate

A closed-loop transfer function has denominator

$$s^9 + s^8 + s^7 + 3s^6 + 9s^5 + 4s^4 + 7s^3 + (a-7)s^2 + s + 3$$

for some  $a \in \mathbb{R}$ . Do not determine necessary and sufficient conditions for stability. Rather, find a single necessary condition for stability in terms of  $a$  by inspection.

### Exercise stab.splenculus

Consider the block diagram of Fig. exe.1. What is the closed-loop transfer function; that is, the transfer function from the command  $R(s)$  to the output  $Y(s)$ ? Let the plant  $G$  have transfer function

$$G(s) = \frac{10(s-1)}{(s+5)(s+1)}, \tag{1}$$

the feedback transfer function  $H(s) = 1$ , and the controller  $C$  have transfer function

$$C(s) = K \tag{2}$$

where  $K \in \mathbb{R}$  is some gain. Determine the range of stable controller gains  $K$ .

### Exercise stab.break

Given the system shown in the diagram below, find:

- the closed loop transfer function, and
- the number of poles in the right half plane.

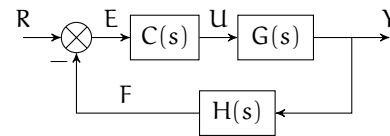
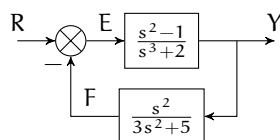


Figure exe.1: a block diagram with a controller  $C(s)$ .

\_\_\_\_\_/35 p.

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

## Exercise stab.relax

Given a closed loop transfer function with a denominator,

$$s^8 + 8s^7 + 27s^6 + 124s^5 + 168s^4 - 208s^3 - 272s^2 - 2304s - 2304,$$

find,

1. the number of poles in the right half plane,  
and
2. the number of poles on the imaginary axis.

## Transient response performance

Stable system time responses are often described in terms of two intervals, loosely defined as transient—the first part during which the effects of initial conditions remain significant—and steady-state—the second part during which the response has “settled” near its final value or final amplitude of oscillation. In this chapter, we consider performance in terms of the transient response; in the next, we will consider it in terms of the steady-state response—specifically as steady-state error. Transient response characteristics are typically found via two methods:

1. analytically and
  - a) precisely for first- and second-order systems without zeros and
  - b) approximately for first- and second-order systems with zeros and higher-order systems that have dominant poles relatively close to the imaginary complex-plane axis and
2. numerically, in simulation.

The analytical method is especially advantageous for design. Design methods we will learn in [Chapter rldesign](#) require we “place” the closed-loop poles in the complex plane. The transient response depends very much on this placement, and exactly how is something we can better understand from

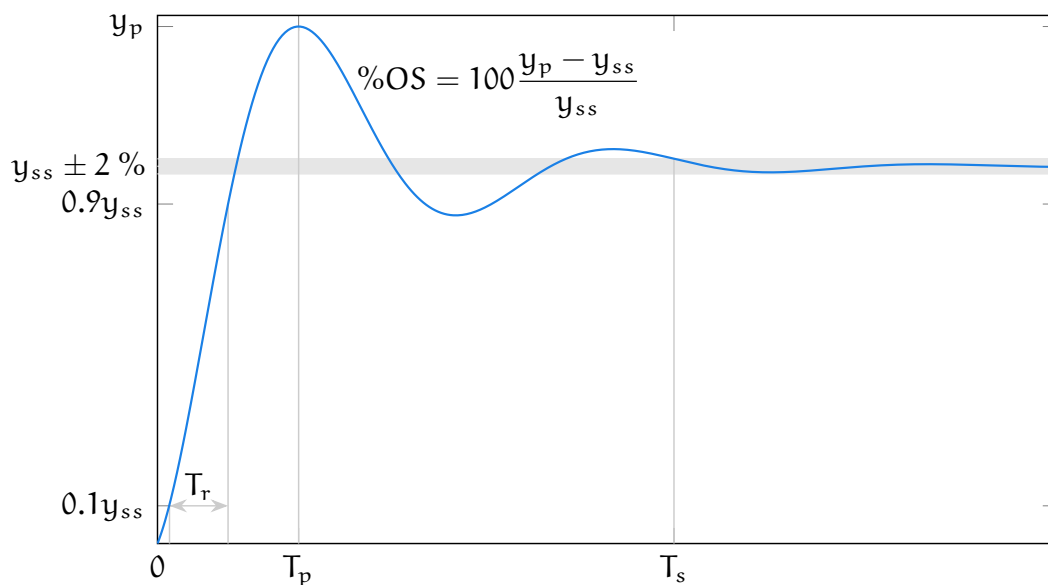
studying first- and second-order system response. We can only simulate systems defined by concrete numbers, so simulation, although powerful, is typically more helpful to fine-tune a controller rather than design it “from scratch.”

## trans.char Transient response characteristics

We define four transient response characteristics, all defined in terms of a system's step input response. For the following, please refer to the illustration in Fig. char.1.

1. The rise time  $T_r$  is the duration from the time the response reaches 10 % to the time it reaches 90 % of its final value.
2. The peak time  $T_p$  is the time at which the response reaches its first or maximum peak.<sup>1</sup>
3. The percent overshoot %OS expresses the amount the response overshoots its steady-state value, expressed as a percentage of the steady-state value.
4. The settling time  $T_s$  is the time at which the response reaches, and thereafter remains within,  $\pm 2\%$  of its steady-state value.<sup>1</sup>

1. This definition assumes the step input occurs at  $t = 0$ . Otherwise, subtract the nonzero initial time.



**Figure char.1:** transient response characteristics rise time  $T_r$ , peak time  $T_p$ , percent overshoot %OS, and settling time  $T_s$  in terms of a response's steady-state  $y_{ss}$  and peak  $y_p$ .

## trans.exact Exact analytical trans response char of first– and second–order sys

First-order systems without zeros

A first-order system without zeros has a transient response characterized by a time-constant  $\tau$  that appears in the general response as

$$1 - e^{-t/\tau} + \dots \quad (1)$$

The transient exponential decays such that in three time constants  $3\tau$  only 5 % of the term remains; in  $5\tau$ , less than 1 %.

There is neither peak nor overshoot for this type of response. However, the rise time for these systems is found by solving the time-domain differential equation

$$\tau \dot{y}(t) + y(t) = ku(t) \quad (2)$$

with output variable  $y$ , input variable  $u$ , and real constant  $k$ . It is easily shown that the solution to Eq. 2 in Eq. 2 is, for a unit step input,

$$y(t) = k \left( 1 - e^{-t/\tau} \right), \quad (3)$$

from which we discover that the steady-state value is

$$y_{ss} = \lim_{t \rightarrow \infty} y(t) \quad (4a)$$

$$= k. \quad (4b)$$

The rise time is, by definition, the duration of the time interval  $[t_1, t_2]$  such that

$$y(t_1) = 0.1y_{ss} \quad (5a)$$

$$y(t_2) = 0.9y_{ss}. \quad (5b)$$

The first of these yields

$$k \left( 1 - e^{-t_1/\tau} \right) = 0.1k \Rightarrow \quad (6a)$$

$$t_1 = -\tau \ln 0.9 \quad (6b)$$

$$\approx 0.1054\tau. \quad (6c)$$

Solving in an analogous fashion, we find  $t_2 \approx 2.3026\tau$ . The interval, then, is  $t_2 - t_1 = 2.1972\tau$ .

**Equation 7 first-order system rise time**

Finally, the settling time can be derived in a fashion similar to the rise time.

**Equation 8 first-order system settling time**

Second-order systems without zeros

Second-order system transient responses are characterized by a natural (angular) frequency  $\omega_n$  and damping ratio  $\zeta$ . It is helpful to recall the complex-plane graphical representation of the pole-zero plot for a second-order system without zeros, as shown in Fig. exact.1.

Following a procedure very similar to that for first-order systems, the following relationships can be derived.

The rise time  $T_r$  does not have an analytical solution in terms of  $\omega_n$  and  $\zeta$ . However, Fig. exact.2 shows numerical solutions for  $T_r$  scaled by  $\omega_n$  for  $\zeta \in (0, 1)$ .

The peak time  $T_p$  has the following, simple expression

$$T_p = \frac{\pi}{\omega_d}, \tag{9}$$

where  $\omega_d = \omega_n \sqrt{1 - \zeta^2}$  is the damped natural frequency.

The percent overshoot %OS is related directly to

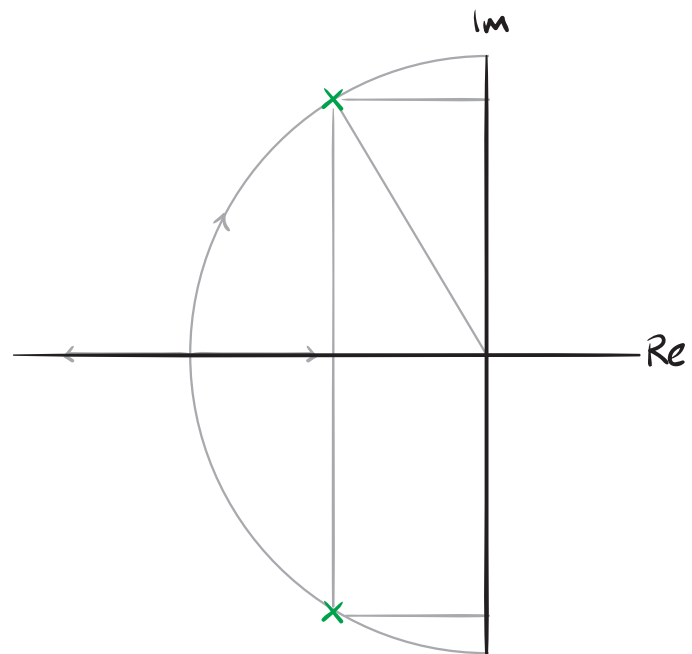


Figure exact.1: the relationship between the pole-zero plot of a second-order system with no zeros and  $\omega_n$  and  $\zeta$ .

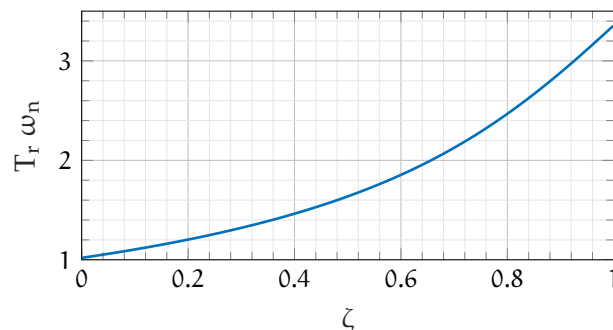


Figure exact.2: the relationship between rise time, natural frequency, and damping ratio.



$\zeta$  as follows

$$\%OS = 100 \exp \frac{-\zeta\pi}{\sqrt{1-\zeta^2}} \Leftrightarrow \quad (10)$$

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}. \quad (11)$$

Finally, the settling time  $T_s$  is expressed as

$$T_s = \frac{4}{\zeta\omega_n}. \quad (12)$$

## **trans.approx** Approx analytical transient response characteristics

Certain higher-order systems can be approximated as second-order systems and can be characterized by the parameters in the preceding section. This includes systems with zeros (in the preceding section we assumed the second-order system had no zeros).

These conditions for “good approximation” are as follows. Each is necessary but by individually insufficient; together, they are sufficient.

### **higher-order poles and zeros are significantly leftward**

Higher-order poles and any zeros have significantly more negative real parts than the dominant second-order poles. A typical guideline is that they should be at least five-times as negative. These poles and zeros contribute a relatively small amount to the transient response, since they decay much faster.

### **nearby higher-order poles and zeros nearly cancel**

All poles near the dominant second-order pole pair are nearly canceled by nearby higher-order zeros and vice-versa (i.e. zeros near a dominant second-order pole pair are nearly cancelled by higher-order poles).

## trans.sim Simulation

Many control systems are not in the class of those we have described, analytically: first- or second-order and without zeros. In order to evaluate their transient performance, regardless of how well they are approximated by the analytic solutions from before, we will simulate their step responses.

Matlab has several built-in and Control Systems Toolbox functions for analyzing the transient response of a system represented by a transfer function system model. We'll explore a few, here.

Consider, for instance, a closed-loop system with transfer function

$$F(s) = \frac{\omega_n^2(-s/z + 1)}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad (1)$$

where  $z$  is some real zero we'll move around, later;  $\omega_n = 20\pi$  rad/s is the natural frequency; and  $\zeta = 0.3$  is the damping ratio.

Let's explore this system's transient response. Clearly, for large negative values of  $z$ , the response should be approximately congruent with the exact analytic solutions of [Lecture trans.exact](#). Specifically, the rise time  $T_r$  will be described by [Figure exact.2](#), the peak time  $T_p = \pi/\omega_d$ , the percent overshoot will be

$$\%OS = 100 \exp \frac{-\zeta\pi}{\sqrt{1-\zeta^2}}, \quad (2)$$

and the settling time  $T_s$  will be

$$T_s = \frac{4}{\zeta\omega_n}. \quad (3)$$

Let's compute these analytic values.

```
z = 0.3;
w_n = 20*pi;
w_d = w_n*sqrt(1-z^2);

T_r_an = 1.39/w_n % s ... analytic, from Figure 03.3
T_p_an = pi/w_d % s ... analytic
```

```
OS_an = 100*exp(-z*pi/sqrt(1-z^2)) % %, analytic
T_s_an = 4/(z*w_n) % s ... analytic
```

```
T_r_an =
    0.0221
T_p_an =
    0.0524
OS_an =
    37.2326
T_s_an =
    0.2122
```

Now, let's define the transfer function object.

```
z_a = -z*w_n*[1.5,3,5];
n_z = length(z_a);
F = stack(1,tf(1,1)); % init model array
for i = 1:n_z % for each zero!
    F(:,i) = tf(... % tf def transfer func object
        w_n^2*[-1/z_a(i),1],... % num. polyn. coef's
        [1,2*z*w_n,w_n^2]... % den. polyn. coef's
    );
end
```

For a step input  $u(t) = u_s(t)$  and initial value  $y(0) = 0$ , let's simulate. The step function would be the easiest way to solve for the step response. However, we choose the more-general `lsim` for demonstration purposes. We must do so for each zero location  $z$ .

```
t_a = linspace(0,.3,100); % time array
u = @(t) ones(size(t)); % input for t>=0
y_0 = 0; % initial condition
y_t = NaN*ones(n_z,length(t_a)); % preallocate
for i = 1:n_z
    y_t(i,:) = lsim(F(:,i),u(t_a),t_a,y_0);
end
```

This total solution is shown in Fig. sim.1.

```
figure;
for i = 1:n_z
    p(i) = plot(t_a,y_t(i,:),...
        'displayname', ...
        ['z \approx ', sprintf('%0.2g',z_a(i))], ...
        'linewidth',1.5);hold on
end
hold off
xlabel('time (s)');
```

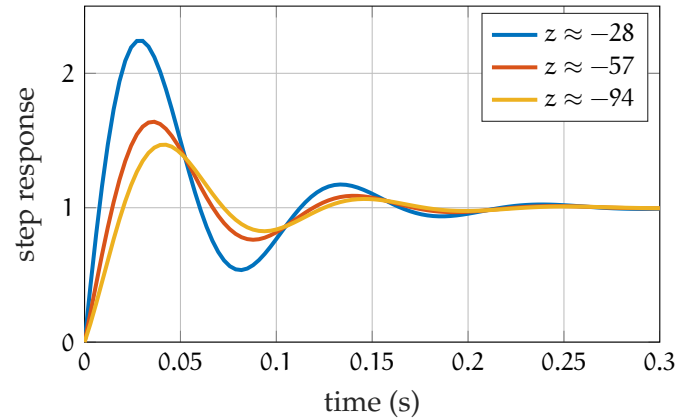
```
ylabel('step response');
grid on
l = legend(p);
```

Now, this indicates that the zero location definitely matters, and that the deviation is worse the closer the zero gets to the poles. Quantifying the response characteristics is tedious, visually, but Matlab has a built-in tool that helps: `stepinfo`.

```
for i = 1:n_z
    si(i) = stepinfo(y_t(i,:),t_a); % struct
    rt(i) = 1e3*si(i).RiseTime;
    pt(i) = 1e3*si(i).PeakTime;
    os(i) = si(i).Overshoot;
    st(i) = 1e3*si(i).SettlingTime;
    row_names{i} = sprintf('z @ %0.2g',z_a(i));
end
labels = {'T_r','T_p','OS','T_s'};
disp('Time in ms:')
disp(table(rt',pt',os',st',...
    'variablenames',labels,...
    'rownames',row_names ...
))
```

Time in ms:	T_r	T_p	OS	T_s
z @ -28	6.09	30.303	125.86	251.63
z @ -57	11.395	36.364	64.707	205.52
z @ -94	15.635	42.424	47.35	203.21

So that zero location drastically affects the overshoot and rise time, but has relatively little effect on settling and peak times. The takeaway here is not so much this specific result, but the tools one can use to find such results and the importance of doing so.



**Figure sim.1:** step response for different zero locations, from 1s im.

## trans.exe Exercises for Chapter trans

Exercise trans.apiarian

A control system has dominant closed-loop poles at  $-8 \pm j3$ . Under the second-order assumption, what is its settling time?

Exercise trans.pericentral

Consider the block diagram of Fig. exe.1. Let the plant  $G$  have transfer function

$$G(s) = \frac{9}{(s + 4)(s^2 + 3s + 9)}, \tag{1}$$

the feedback transfer function  $H(s) = 1$ , and the controller  $C$  have transfer function

$$C(s) = K \tag{2}$$

where  $K \in \mathbb{R}$  is some gain.

1. Determine the closed loop transfer function  $Y(s)/R(s)$ .
2. For  $K = 4$  and a unit step input to the closed-loop system, what are the second-order approximations of the peak time  $T_p$ , rise time  $T_r$ , settling time  $T_s$ , and percent overshoot %OS?
3. For  $K = 4$  and a unit step input to the closed-loop system, simulate to estimate peak time  $T_p$ , rise time  $T_r$ , settling time  $T_s$ , and percent overshoot %OS.
4. Compare the second-order approximations with the simulated results. Explain the differences and similarities.

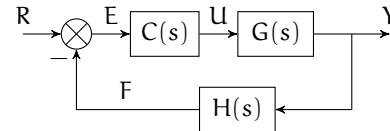


Figure exe.1: a block diagram with a controller  $C(s)$ .

Exercise trans.rest

For a second order system with a 10% overshoot and a 0.1 second rise time, find: \_\_\_\_\_ / 10 p.

1. the damping ratio  $\zeta$ ,

2. the natural frequency  $\omega_n$ , and
3. the location of the closed loop poles  $\Psi$ .

## Steady–state response performance

After the transient response has settled—that is, reached steady-state—the system may or may not be in a desirable state. If the response asymptotically approaches any state other than that commanded, it is said to have steady-state error. These arise from three primary sources:

1. nonlinearities, like backlash in gears—we won't explore this one;
2. disturbances, like those from the environment; and
3. input (command) type and the plant dynamics.

We will focus our attention on [item 3](#); [item 2](#) is similar.



## steady.error Steady–state error for unity feedback systems

It is uncommon for a feedback system to be truly “unity.” However nonunity feedback systems can be re-written and evaluated in terms of unity feedback counterparts.<sup>1</sup> For this reason, we will focus on unity feedback systems. First we recall the final value theorem. Let  $f(t)$  be a function of time that has a “final value”  $f(\infty) = \lim_{t \rightarrow \infty} f(t)$ . Then, from the Laplace transform of  $f(t)$ ,  $F(s)$ , the final value is  $f(\infty) = \lim_{s \rightarrow 0} sF(s)$ .

Let’s consider the unity feedback system of [Figure error.1](#) with command  $R$ , controller transfer function  $G_1$ , plant transfer function  $G_2$ , and error  $E$ . Recall that we call  $e(t)$  or (its Laplace transform)  $E(s)$  the error. We want to know the steady-state error, which, from the final value theorem, is

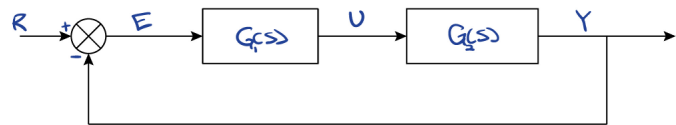
$$e(\infty) = \lim_{s \rightarrow 0} sE(s). \tag{1}$$

Now all we need is to express  $E(s)$  in more convenient terms. For the analysis that follows, we combine the controller and plant:  $G(s) = G_1(s)G_2(s)$ . From the block diagram, we can develop the transfer function from the command  $R$  to the error  $E$ .

**Equation 2 error transfer function**

Given a specific command  $R$  and forward-path transfer function  $G$ , we could take inverse Laplace transform of  $E(s)$  to find  $e(t)$  and take the limit. However, it is much easier to use the final value theorem:

1. For more details, see N. S. Nise (2011, Section 7.6).



**Figure error.1:** unity feedback block diagram with controller  $G_1(s)$  and plant  $G_2(s)$ .



This last expression is the best we can do without a specific command  $R$ . Three different commands are typically considered canonical. The first is now developed in detail, and the results of the other two are given below. First, consider a unit step command, which has Laplace transform  $R(s) = 1/s$ .



where we let  $K_p = \lim_{s \rightarrow 0} G(s)$ . We call  $K_p$  the position constant. If  $K_p$  is large, the steady-state error is small. If  $K_p$  is infinitely large, the steady-state error is zero. If  $K_p$  is small, the steady-state error is a finite constant.

The form of  $G(s)$  has implications for  $K_p$ .  $G(s)$  has a factor  $1/s^n$  where  $n$  is some nonnegative integer. Since we are concerned about what happens to  $G(s)$  when we take its limit as  $s \rightarrow 0$ , this factor is of particular importance. If  $n > 0$ ,  $K_p = \lim_{s \rightarrow 0} G(s) = \infty$ . We call the transfer function  $1/s$  an integrator, which is the inverse of the transfer function  $s$ , the differentiator.

We needn't solve for  $E$  explicitly, then. All we need to know is the command  $R$  and the number of integrators  $n$  in the forward-path transfer function  $G(s)$  (we call this the system type).

The steady-state error for other commands and system type can be derived in the same manner. The results for the canonical inputs are shown in [Table error.1](#).

**Example steady.error-1**

**re: steady-state error**

Let a system have forward-path transfer function

$$G(s) = \frac{10(s + 3)(s + 4)}{s(s + 1)(s^2 + 2s + 5)}$$

For commands  $r_1(t) = 2u_s(t)$ ,  $r_2(t) = 6tu_s(t)$ , and  $r_3(t) = 7t^2u_s(t)$ , what are the steady-state errors?

**Table error.1:** the static error constants and steady-state error for canonical commands  $r(t)$  and systems of Types 0, 1, 2, and  $n$  (the general case). Note that the faster the command changes, the more integrators are required for finite or zero steady-state error.

$r(t)$	Type $n$		Type 0		Type 1		Type 2	
	error const.	$e(\infty)$	error const.	$e(\infty)$	error const.	$e(\infty)$	error const.	$e(\infty)$
$u_s(t)$	$K_p = \lim_{s \rightarrow 0} G(s)$	$\frac{1}{1 + K_p}$	$K_p$	$\frac{1}{1 + K_p}$	$\infty$	0	$\infty$	0
$tu_s(t)$	$K_v = \lim_{s \rightarrow 0} sG(s)$	$\frac{1}{K_v}$						
$\frac{1}{2}t^2u_s(t)$	$K_a = \lim_{s \rightarrow 0} s^2G(s)$	$\frac{1}{K_a}$						

**steady.exe Exercises for Chapter steady**

## Exercise steady.hypnomancy

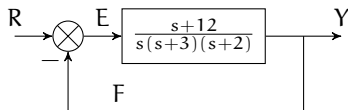
If a control system responds to a command  $r(t) = 1$  such that its output  $y(t)$  quickly settles near 0.95, what can be said about the control system's stability, steady-state response, and transient response?

## Exercise steady.nap

Given the system show below with an input  $u(t) = 0.2tu_s(t)$ , find:

\_\_\_\_\_/10 p.

1. the system type,
2. the correct static error constant, and
3. the steady state error.



## Root locus analysis

The root locus is a graphical technique for designing for closed-loop transient response from open-loop knowledge—and some cleverness.<sup>1</sup> A system's transient response is dominated by its poles. For a system with feedback, solving for these closed-loop poles is challenging, as we will see in [Lec. rlocus.def](#). Due to the use of complex analysis in this chapter, it is recommended that the reader review [Appendix A.01](#) before proceeding.

1. The root locus technique was developed by Evans (1950).

## rlocus.def Root locus definition

Closed-loop poles are hard to find

A feedback system’s closed-loop poles determine its stability and transient response. These poles depend on parameters in the controller, often the gain  $K$ , as shown in Fig. def.1.

We define the forward transfer function to be the transfer function from a loop’s error  $E$  to its output  $Y$ ; for the loop in Fig. def.2, this is simply  $KG(s)$ . Furthermore, the feedback transfer function is defined to be the transfer function from the output to the feedback summation,  $H(s)$  in the example. Finally, the open-loop transfer function is defined to be the product of the two—in the case of our example:  $KG(s)H(s)$ .

We break down  $G(s)$  and  $H(s)$  into numerators and denominators:

$$G(s) = \frac{G_n(s)}{G_d(s)} \quad \text{and} \quad H(s) = \frac{H_n(s)}{H_d(s)}. \quad (1)$$

Now we can see how these affect the closed-loop transfer function

$$\frac{KG_n(s)H_d(s)}{G_d(s)H_d(s) + KG_n(s)H_n(s)}. \quad (2)$$

We make the following observations:

1. The closed-loop poles depend on  $K$ , but for controller design,  $K$  is that for which we are solving.
  - a) An analytic solution for the closed-loop poles is intractable for systems of order greater than three.
  - b) For a given value of  $K$ , a numerical root finder is very effective.

That is, the closed-loop poles are hard to find!

2. As  $K \rightarrow 0$ , the closed-loop poles approach the open-loop poles.

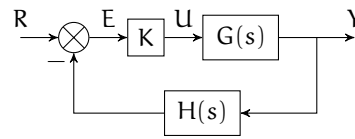


Figure def.1: a block diagram for a proportional controller  $K$ .

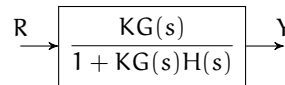


Figure def.2: a block diagram with the corresponding closed-loop transfer function block.

3. As  $K \rightarrow \infty$ , the closed-loop poles approach the open-loop zeros.

These last two observations give us the “start” and “finish” for closed-loop pole locations when  $K$  is varied from 0 to  $\infty$ , a procedure we will now define—as the root locus!

#### Definition

The root locus is the collection of closed-loop pole locations for varying proportional controller gain  $K$ . Recall that for a feedback system with plant  $G(s)$  and feedback transfer function  $H(s)$ , the closed-loop transfer function is

$$\frac{KG(s)}{1 + KG(s)H(s)} \quad (3)$$

and that finding the poles is difficult, in general. However, let us consider our observations from [Lec. rlocus.def](#). We know our “starting points”: at  $K = 0$ , the closed-loop poles are equal to the open-loop poles. And we know our “end points”: as  $K \rightarrow \infty$ , the closed-loop poles are equal to the open-loop zeros. Therefore, we can consider the root locus to be a collection of curves that begin at the open-loop poles and terminate at the open-loop zeros.

#### The magnitude and phase criteria

The closed-loop poles can be found by setting the denominator of the closed-loop transfer function to zero and solving for the values of  $s$  that satisfy this condition. Examining the closed-loop transfer function, we see this is equivalent to

$$1 + KG(s)H(s) = 0 \quad (4)$$

[Eq. 4](#) gives rise to the magnitude and phase criteria.

**Equation 5 magnitude criterion****Equation 6 phase criterion**

These criteria are always satisfied and so they will be our guide to understanding, sketching, and designing with the root locus.

What about negative gains and positive feedback?

We typically consider only nonnegative gains for the root locus, since negative gains typically lead to instability. However, this is only true for systems with positive open-loop transfer functions! When encountering a negative open-loop transfer function, it is advisable to temporarily treat it as positive, proceed with the controller design, then multiply the controller by  $-1$  (or use positive feedback). If one suspects a negative gain might be of service in a specific controller (often, slightly negative gains can remain stable) or if one is building an unstable system intentionally, develop the root locus for negative gains (or, equivalently, positive feedback); for these occasions, see N. Nise (2015).



## rlocus.sketch Sketching the root locus

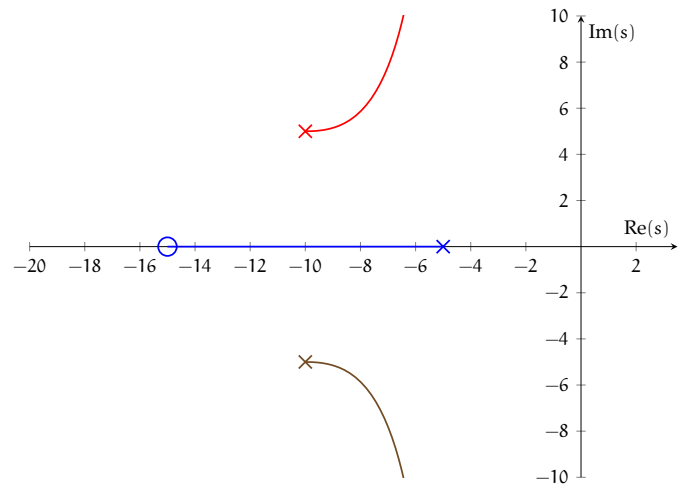
It is easy to get lost in the detailed rules of manual root locus construction. In the “old days” accurate root locus construction was critical, but now it is useful only for gaining intuition for how a given system will behave given its open-loop transfer function—which is extremely useful for design. If a detailed root locus is desired, we should use the computer tools of [Lec. rlocus.comp](#).

We will construct a procedure for sketching a root locus from the following rules. In what follows phrases such as “has locus” are used to describe curves in the complex plane for which the root locus is defined. That is, everywhere in the complex plane for which the root locus is defined is said to “have locus.” Note that some of the following rules only apply for  $K > 0$ .

- R1. The root locus begins at open-loop poles, where  $K = 0$ , and approaches open-loop zeros, where  $K \rightarrow \infty$ . This was shown in [Lec. rlocus.def](#) to follow from the form of the closed-loop transfer function.
- R2. The number of branches of the root locus is equal to the number of closed-loop poles. The number of closed-loop poles is equal to the number of open-loop poles or zeros, whichever is greater.
- R3. The root locus is symmetric about the real-axis. This is due to the fact that poles can “leave” the real-axis only as conjugate pairs.
- R4. On the real-axis, there is locus wherever an odd number of open-loop zeros and poles are on the real-axis, to the right—and no locus, elsewhere. This is a consequence of the phase criterion, [Eq. 6](#). Recall, from [Appendix A.01](#), the geometric evaluation of transfer functions. The phase

criterion states that, for locus,  $\angle KG(s)H(s)$  must always be  $\pi$  or its equivalent, so, for a test point  $\psi$ , the sum of the angles from each of the open-loop poles and zeros to  $\psi$  must be  $\pi$  or its equivalent, as can be illustrated in Fig. sketch.1. Due to the fact that every off-axis pair of open-loop poles or zeros contributes no net angle (because their angles are equally opposite), only poles and zeros on the real axis contribute to the phase of a given point on the real axis. When the point is to the right of every real-axis pole and zero, all the angle contributions are zero, and the phase criterion is not met. Moving the point leftward and it passes a pole or zero, the angle becomes  $\pm\pi$ , satisfying the angle criterion. Continuing leftward, each time it crosses a pole or zero,  $\pm\pi$  is added, toggling satisfaction of the angle criterion.

- R5. "Missing" poles and zeros are paired with infinite zeros and poles, asymptotically. An open-loop transfer function with a different number of poles and zeros is said to have "missing" poles or zeros. This is because the root locus begins at open-loop poles and approaches open-loop zeros—but what about systems with missing open-loop poles or zeros? For these situations, the root locus begins or ends at poles or zeros at infinity. For a system with more poles than zeros, which is quite common, some poles approach zeros at infinity, asymptotically. Conversely, for a system with more zeros than poles, which is uncommon and is called a non-causal system, some branches of the root locus begin asymptotically from poles at infinity. Asymptotes originate at a single real-axis intercept  $\sigma_a$ , which can be shown to be related to the finite poles  $p_i$



**Figure sketch.1:** a root locus example for illustrating the geometric interpretation of the phase criterion on the real axis.

and zeros  $z_j$ , with  $n_p$  and  $n_z$  the number of poles and zeros, as follows.

Equation	1	root	locus
asymptote	real	-axis	intercept

Note that the imaginary parts of the poles and zeros cancel, so they needn't be considered. With  $N \equiv n_p - n_z$ , the number of asymptotes is  $|N|$ . Each is a ray that originates at  $\sigma_a$ , and all that remains undetermined is the angle of each ray, which can be shown to be as follows, for all  $m \in \mathbb{Z}$ .

Equation	2	root	locus
asymptote	angles		

Note that these repeat every  $|N|$  consecutive values of  $\theta_m$ .

Every root locus (with  $K > 0$ ) will satisfy the rules above. They will help us construct sketches with the following procedure.

- RL1. Sketch the open-loop poles and zeros. According to **Rule R1**, the root locus starts at the open-loop poles and ends at the open-loop zeros.
- RL2. Sketch real-axis locus in accordance with **Rule R4**. Let's start with a win. Begin at the right of all real-axis poles and zeros (where there is never locus) and move leftward, toggling for each pole or zero, "no locus, locus, no locus, locus ...."
- RL3. If applicable, determine poles or zeros at infinity and draw asymptotes. Determine the number of finite poles  $n_p$  and finite

zeros  $n_z$ . Compute  $N = n_p - n_z$ . If  $N > 0$ , there are  $|N|$  zeros at infinity; if  $N < 0$ , there are  $|N|$  poles at infinity; and if  $N = 0$ , there are neither poles nor zeros at infinity and the rest of this step should be skipped. Compute the asymptote real-axis intercept  $\sigma_a$  from Eq. 1. Compute  $|N|$  asymptote angles  $\theta_0, \theta_1, \dots$  from Eq. 2. Sketch the asymptotes.

RL4. Finish the root locus sketch, respecting all rules. Typically, a qualitatively accurate sketch can now be constructed, which is our goal.

**Example rlocus.sketch-1**

**re: sketching the root locus**

Sketch the root locus for the open-loop transfer function

$$\frac{3(s + 1)}{(s + 3)(s + 5)}$$

**Example rlocus.sketch-2**

**re: sketching the root locus**

Sketch the root locus for the open-loop transfer function

$$\frac{53}{(s + 5)(s^2 + 2s + 2)}$$



## rlocus.comp Generating the root locus via a computer

### Matlab

In Matlab, the command `rlocus` generates a root locus plot from a linear system model object defined by `tf`, `zpk`, or `ss`. The data cursor has particularly useful information associated with it, including the gain required for the closed-loop pole of a given branch to be located at the selected point. Here are a few examples.

#### Example rlocus.comp-1

re: rlocus using zpk

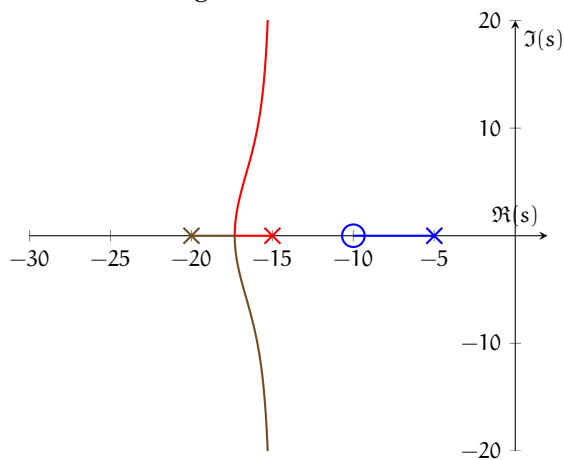
Use Matlab and `zpk` to generate a root locus plot from the open-loop transfer function

$$\frac{s + 10}{(s + 5)(s + 15)(s + 20)}$$

The following code generates the root locus plot.

```
1 sys=zpk([-10], [-5, -15, -20], 1);
2 figure
3 rlocus(sys)
```

The figure it generates should look something like the following.



#### Example rlocus.comp-2

re: rlocus using tf and custom gains

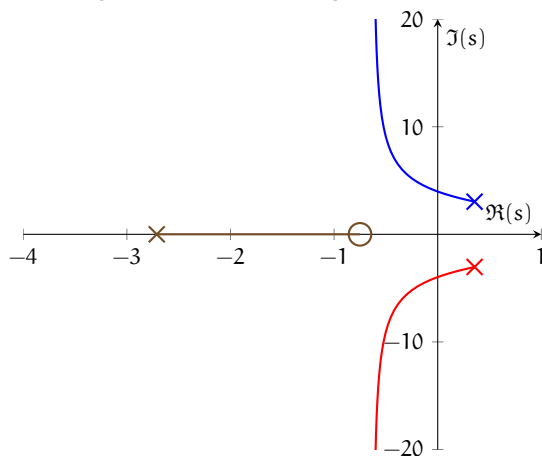
Use Matlab and `tf` to generate a root locus plot from the open-loop transfer function

$$\frac{4s + 3}{s^3 + 2s^2 + 7s + 25}$$

The following code generates the root locus plot.

```
1 sys=tf([4,3],[1,2,7,25]);
2 k=sort([3.5,logspace(-1,3,50),Inf
   ]); % custom gains
3 figure
4 rlocus(sys,k)
```

Note the use of custom gain values. Sometimes this is useful, especially if a specific gain value or range is important. In the code above, a specific gain of 3.5 is chosen; most gains (50 of them) are generated logarithmically from  $10^{-1}$  to  $10^3$ , `logspace(-1,3,50)`; and the final gain of  $\infty$ , `Inf`, is included. The array is sorted such that 3.5 is placed in the correct order in the array. The figure the code generates should look something like the following.



## Python

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

```
notebook filename: python_root_locus.ipynb
notebook kernel: python3
```

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!
```

Let's draw the root locus for the transfer function

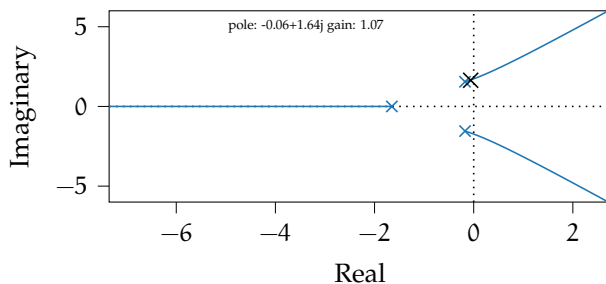
$$\frac{1}{s^3 + 2s^2 + 3s + 4} \quad (1)$$

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

```
transfer_function = c.TransferFunction(1,[1,2,3,4])
```

Now `transfer_function` is a transfer function object. We use the `root_locus` method of the Control Systems module.

```
p1 = c.rlocus(transfer_function) # compute root locus
plt.show() # display the plot
```



Notice that double-clicking the locus yields a data cursor that gives the complex coordinate and corresponding gain! For instance, at the coordinate  $-0.10 + j1.61$ , the gain is 0.67. Therefore, to place a closed-loop pole at this location, we would choose  $K = 0.67$ .

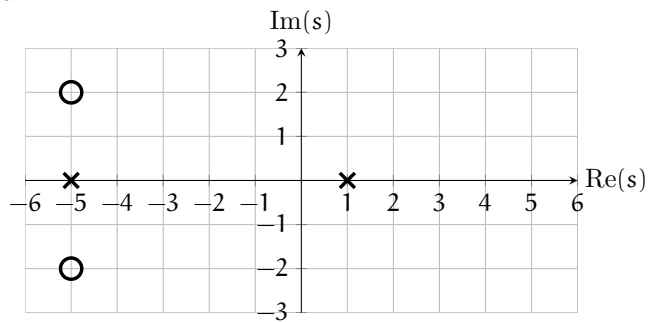


# rlocus.exe Exercises for Chapter rlocus

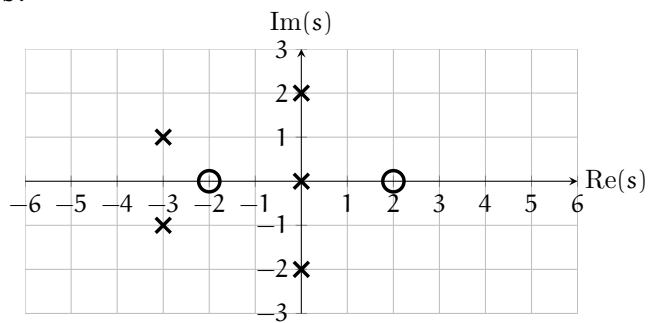
Exercise rlocus.burritosteve

Given the open-loop pole-zero plots below, sketch the root locus plots (use this sheet) for positive controller gain  $K$ .

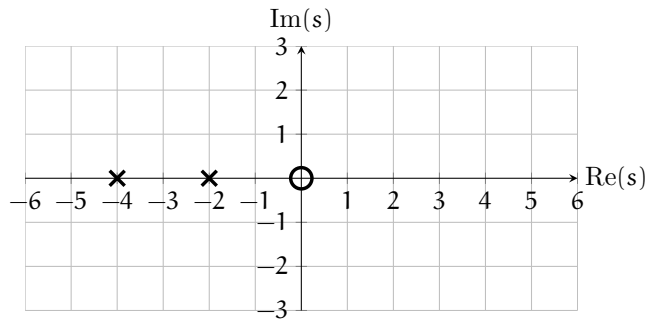
a.



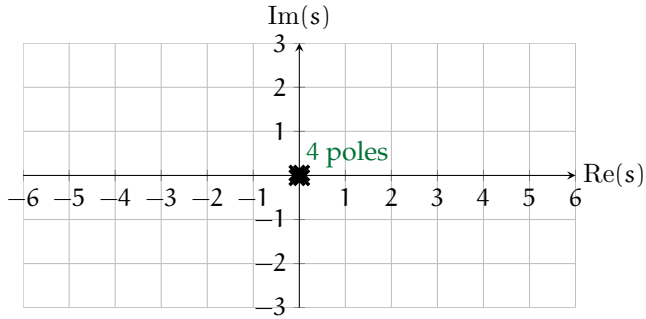
b.



c.



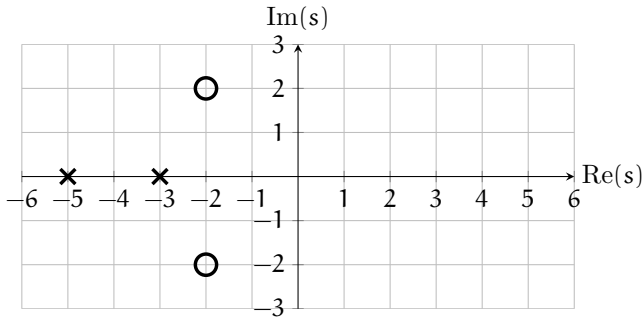
d.



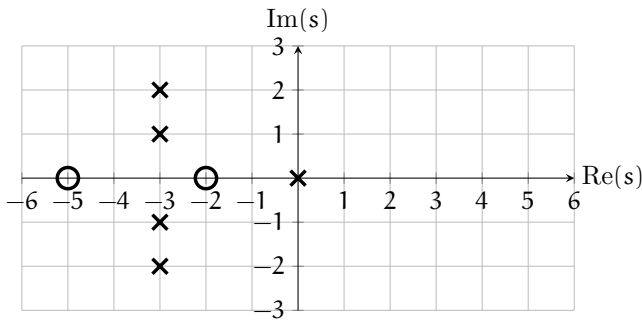
Exercise rlocus.dunnage

Given the open-loop pole-zero plots below, sketch the root locus plots (use this sheet) for positive controller gain  $K$ .

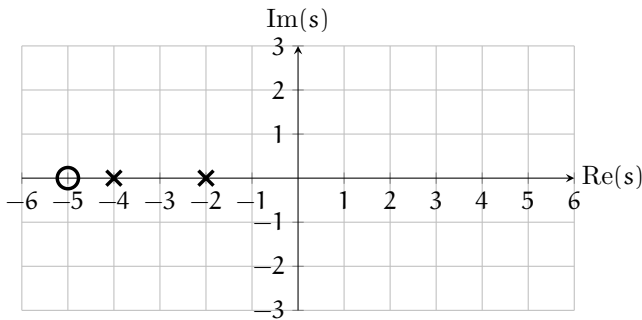
1.



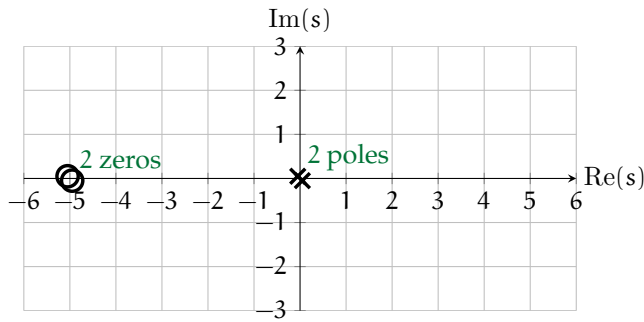
2.



3.



4.

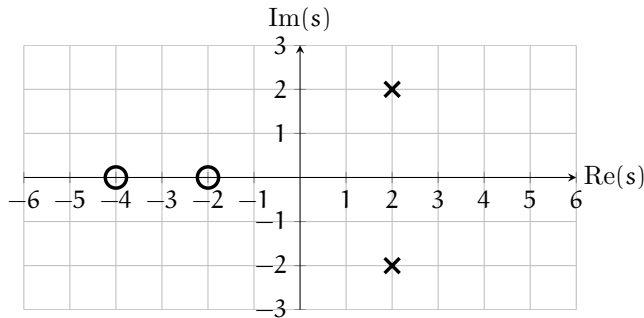


Exercise rlocus.respite

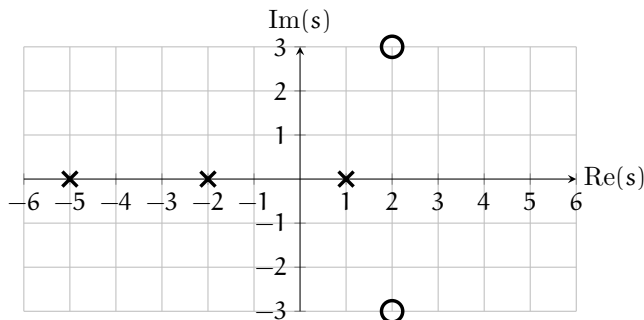
\_\_\_\_\_/20 p.

Given the open-loop pole-zero plots below, sketch the root locus plots (use this sheet) for positive controller gain  $K$ . Comment on the stability of each system. For example, the system is stable for all gain  $K$ , or it becomes unstable as  $K$  increases.

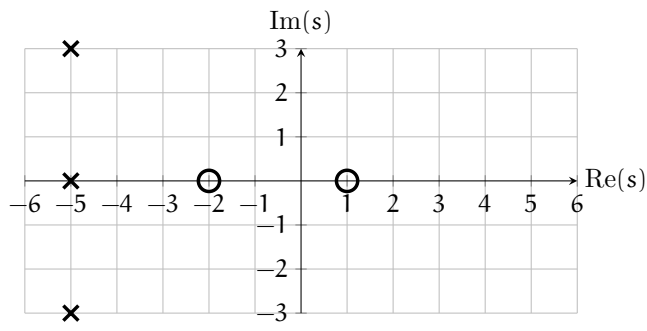
1.



2.



3.



## Root–locus design

In root locus design, our task is to place the dominant closed-loop poles such that the closed-loop system

1. is stable ([Chapter stab](#)),
2. has desirable transient response performance characteristics ([Chapter trans](#)), and
3. has desirable steady-state response characteristics ([Chapter steady](#)).

Several types of controllers can be designed using these techniques. The most basic is gain control ([Lec. rldesign.P](#)), which gives us a single parameter—the loop gain—for controller design. The others we consider here are of two main types: proportional-integral-derivative (PID) and proportional-lead-lag. The two are quite similar, but the latter can be implemented with passive circuits, whereas the former require active circuits.

## rldesign.gain Gain from the root locus

In what follows, we will be primarily concerned with locations on the root locus that yield desirable transient response characteristics. But knowing the location on the root locus at which we would like to operate requires that we know the implicit gain associated with placing the closed-loop poles at that location. This lecture demonstrates two methods of finding the gain associated with a given location on the root locus.

Gain, analytically and geometrically

Recall that in [Lec. rlocus.def](#), we defined the root locus magnitude criterion to be

$$(1)$$

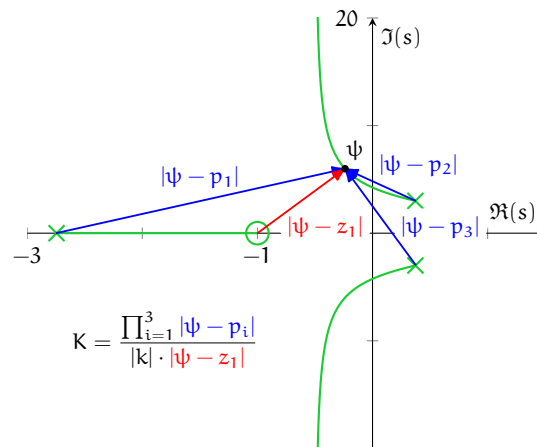
Solving this for  $K$ , we obtain the following result, which is valid for a test point  $s = \psi$  on the root locus.

Equation 2 gain from the magnitude criterion

Consider this in terms of the gain  $k$ , poles  $p_j$ , and zeros  $z_i$  of  $G(s)H(s)$  (note that  $k$  is exclusive of the controller gain  $K$ ). [Eq. 2](#) can be written as a function of a test point  $\psi$  on the root locus. (See [Appendix A.01](#) for details on this representation.)

Equation 3 geometric gain

From the discussion of [Appendix A.01](#), we obtain the geometric interpretation: the gain is equal to the the product of the lengths of vectors



**Figure gain.1:** an example of a geometric interpretation of the the loop gain  $K$  for a closed-loop system with transfer function poles  $p_{1,2,3}$ , zero  $z_1$ , and gain  $k$  at a complex value  $s = \psi$  on the root locus.

that originate at poles and terminate at the test point divided by the product of  $k$  with the lengths of vectors that originate at the zeros and terminate at the test point. This is illustrated in Fig. gain.1.

### Gain, the easy way

Finding the loop gain  $K$  that yields a closed-loop pole at a specific point on the root locus is much easier to find numerically. Recall that for a given value of gain  $K$ , the closed-loop poles are easily found numerically. Software typically generates root locus plots in a brute-force way: by simply computing the closed-loop pole locations for a range of gains. Therefore, the gain corresponding to each point on these plots is then known a priori.

In programs such as MATLAB, the data cursor will typically display the gain corresponding to each point. If insufficient resolution is available, more can typically be specified with an optional argument, as in MATLAB's `rlocus` as follows.

```
G = zpk([-10,-20],[-5,5,-3],1); % transfer function
k = [logspace(-1,4,1e4),Inf]; % custom gains
figure;
rlocus(G,k) % plot with custom gains
```

## 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

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

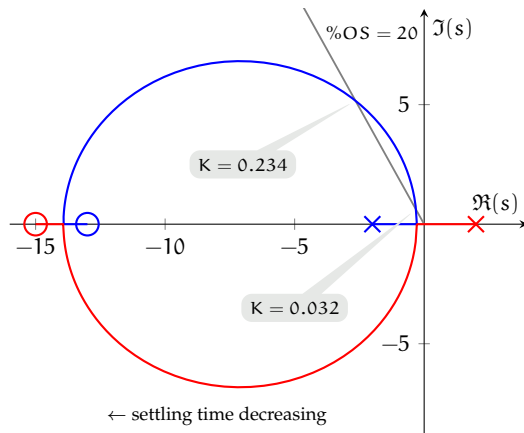
### re: proportional controller design for percent overshoot

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
```



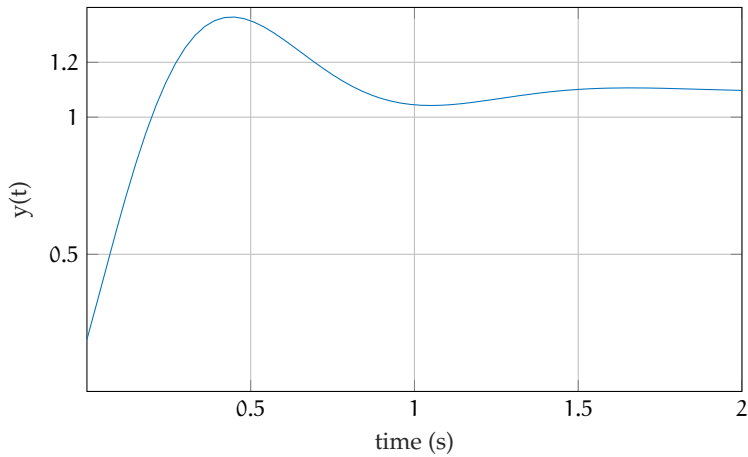
From the figure, we can see that when the gain is either  $K = 0.032$  or  $K = 0.234$ , according to the second-order approximation, the %OS is 20. We prefer the latter because of our requirement to minimize the settling time, which decreases as the closed-loop poles move leftward. Now we must find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

```
Gc1=feedback(K*G,1);
```

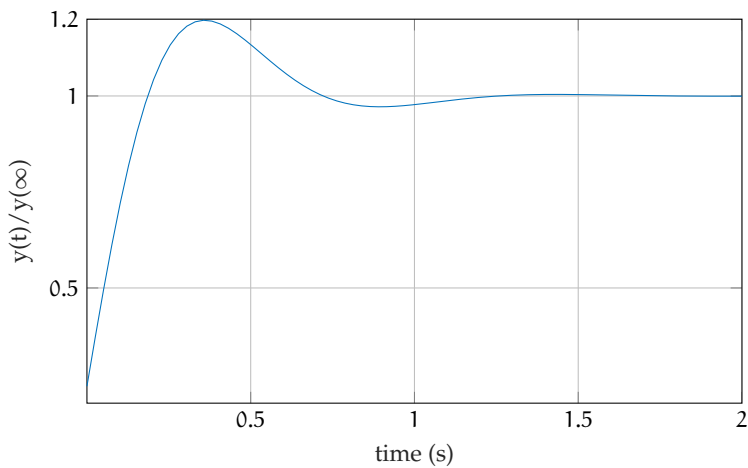
Now we are ready to simulate the step response to evaluate the actual transient response.

```
t1=5; % final time
[y,t]=step(Gc1,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.<sup>a</sup>



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 $\psi$

Let's determine a target point  $\psi$  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  $\text{Re}(\psi)$  and the

overshoot a specific angle  $\angle\psi$ . The real part is found from the expressions

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

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

The angle is found via the equations

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

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

Remarkably simple expressions result:

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

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

So, in the final analysis, the desired pole location  $\psi$  (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,pOS)
print("psi = %0.3g + j %0.3g" % (np.real(psi),np.imag(psi)))
```

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

Design with the root locus

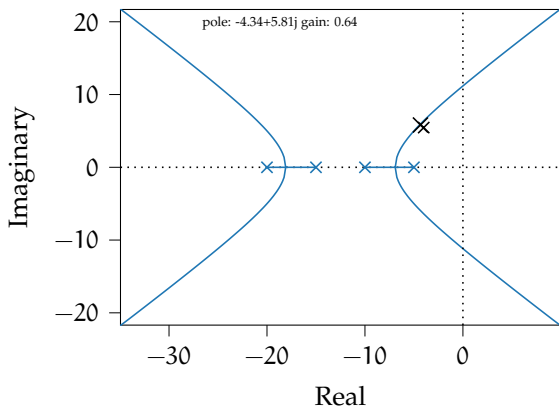
Defining a transfer function in Python is straightforward with the Control Systems module (documentation [here](#)).

```
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  $\psi$ , 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

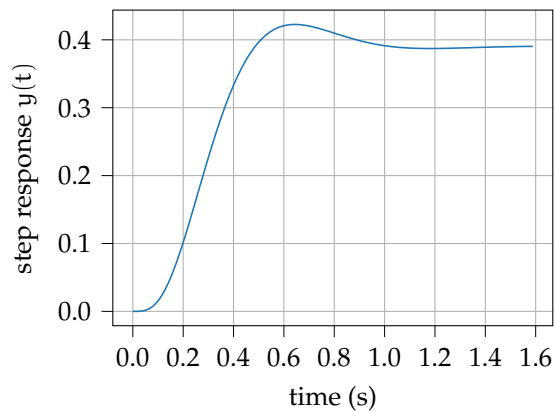
We use the Control Systems module's `feedback` method to find the closed-loop transfer function.

```
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.



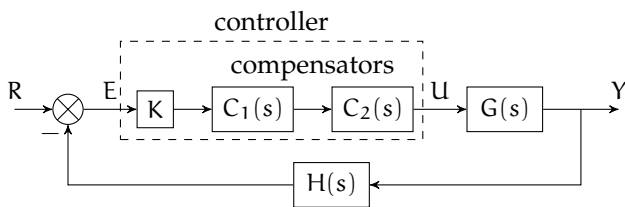
## rldesign.beyondP Beyond proportional design

Using proportional design, the closed-loop poles are restricted to the root locus. Often the root locus does not pass through the closed-loop pole location specified by performance requirements. Therefore, design techniques that can move the poles to desirable locations are indicated.

We consider two classes of controller: proportional-integral-derivative (PID) and proportional-lead-lag. PID controllers use “ideal” integrators ( $s^{-1}$ ) and differentiators ( $s$ ) and therefore require active circuits for instantiation. Proportional-lead-lag controllers can be considered approximations of PID controllers, and these can be realized in passive circuits.<sup>1</sup>

We will build controllers incrementally by cascade compensation, which is illustrated in Fig. beyondP.1. This means we will begin with proportional controller design, then add cascade compensation to achieve different performance requirements. For instance, we will begin with a gain (P) control design, then cascade an integral compensator (now the controller is PI), and finally cascade a derivative compensator (now it is PID).

1. When describing “active” and “passive” controllers, we have in mind analog circuit instantiations. However, the vast majority of modern controllers are actually instantiated in digital circuits via microcontrollers. Due to the high rates of analog-to-digital (ADC) and digital-to-analog (DAC) conversion in modern controllers, often digital controller performance is nearly identical to that of a corresponding analog controller. A consequence of this is that continuous-time (analog) controller design, as we learn in this chapter, can be applied in the discrete-time (digital) case with minor alteration.



**Figure beyondP.1:** block diagram illustrating cascade compensation via compensators  $C_1$  and  $C_2$ .

## rldesign.PI Proportional–integral (PI) controller design

When studying steady-state error, we discovered that the more integrators ( $s^{-1}$ ) in the open-loop transfer function, the better the steady-state error. PI control includes integrator compensation to a proportional controller without significantly affecting the transient response. Later, we will deal with how to design for transient response.

Here's the plan: include an integrator (i.e. pole at the origin) in the controller and a nearby zero to counter the pole's (slowing) effects on the transient response.

Why does the integrator affect the transient response? Adding a pole at the origin completely changes the root locus, and therefore the location of the closed-loop poles, and therefore the transient response.

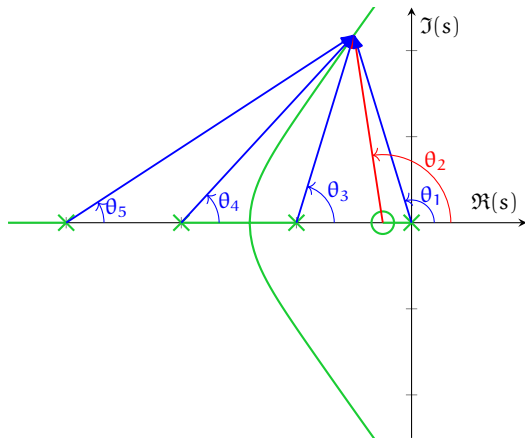
In order to mitigate this, we place a zero near the origin, which nearly cancels the integrator's effect on the root locus. To see this, recall that the root locus must meet the phase criterion (Lec. rlocus.def). Let us meditate on Fig. PI.1, in which a system is presented that initially contained the three left half-plane poles and no zeros. Including the integral pole at the origin (integrator) and zero nearby, we obtain the root locus shown. From the phase criterion,

$$-\theta_1 + \theta_2 - \theta_3 - \theta_4 - \theta_5 = \pi \pm 2\pi m \quad (m \in \mathbb{Z}). \quad (1)$$

If the compensator zero is placed close to the pole, then  $\theta_2 - \theta_1 \approx 0$  and the root locus is mostly unchanged from its pre-compensation state.

### Design procedure

The following design procedure can guide us through this typically straightforward controller design.



**Figure PI.1:** the effect of the integral compensator on the root locus's angle condition.

1. Design a proportional controller to meet transient response requirements by choosing the gain  $K$  for the dominant closed-loop poles to be  $p_{1,2}$ .
2. Include cascade integral compensation and a real zero near  $\text{Re}(p_{1,2})/10$ .
3. Tune the gain  $K$  such that the close-loop poles are as desirable as possible.
4. Simulate the time response to see if it meets specs. Tune. If the steady-state compensation is too slow, try moving the zero leftward.

### Example rldesign.PI-1

For a plant with transfer function

$$\frac{10}{(s + 2)(s + 5)}$$

design a unity feedback PI controller such that the system has  $\%OS = 20$  and zero steady-state error for step inputs.

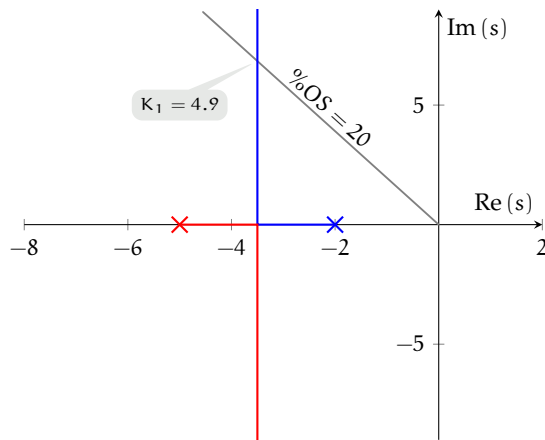
We will use MATLAB. First, let us observe that with no integrators in the plant (Type 0 system), the system will have a finite steady-state error to step inputs. Therefore, we require integral compensation. Let's define the transfer function.

### re: PI control for percent overshoot

```
sys1 = zpk([], [-2, -5], 10);
```

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(sys1, sort([0, .225, 4:.1:10, Inf])
);
ylim([-10, 10])
grid on
```



From the figure, we can see that when the gain is  $K_1 = 4.9$ , according to the second-order approximation, the %OS is 20. This occurs at the test point  $s_{tp} = -3.5 + j6.84$ . If we were designing simply a P controller, we would now simulate the closed-loop response with this gain. Before we simulate, let's apply integral compensation. We put a pole at the origin as the integrator and compensate with a nearby zero. We start with that zero at  $\text{Re}(s_{tp})/10 = -0.35$ . Our compensator has the transfer function

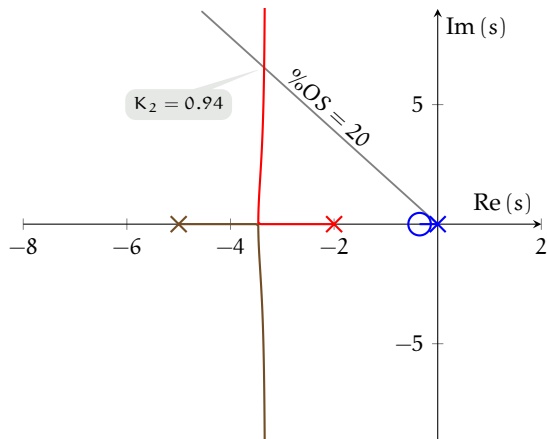
$$\frac{s + 0.35}{s}$$

and can be applied to the open-loop transfer function as follows.

```

sReal = -3.5;
zeroc = sReal/10;
comp = zpk([zeroc],[0],1); %
    compensator
sys2 = K1*comp*sys1; % controlled open
    -loop tf
    
```

Now a new root locus analysis is required in order to determine the new gain required to get back near the test point. This is shown below.



We see that the cascade gain required to return to the overshoot ray is  $K_2 = 0.94$ .

Now we must find the closed-loop transfer functions for each controller design, which can be found as follows.

```

sys1c1 = feedback(K1*sys1,1);
sys2c1 = feedback(K2*sys2,1);
    
```

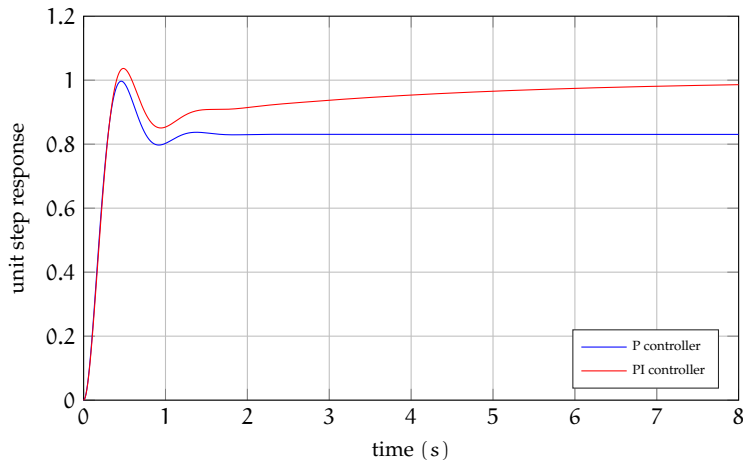
Now we are ready to simulate the closed-loop step response to evaluate the actual step response for each controller design.

```

tvec = 0:.01:8;
y1 = step(sys1c1,tvec);
y2 = step(sys2c1,tvec);
stepinfo(y1,tvec)
    
```

The command `stepinfo` computes the simulated transient response characteristics. The result is  $\%OS = 20.0$ , good! Note that we used the P controller for this evaluation. The strict definition of this gives a skewed value due to the steady-state error compensation.

- Let's take a look at a plot comparing the two step responses.



Note that the PI-controlled system has steady-state error approaching zero and that the two systems have similar transient response characteristics, per our expectation. The steady-state error does respond relatively “slowly” because the third closed-loop pole introduced by the integral compensator is relatively close to the imaginary axis. Moving the compensator zero leftward can speed this response, but transient responses will be increasingly effected. In this case, the settling time determined by the complex closed-loop poles (we could call this the “transient settling time”) will increase as we move the zero leftward. However, the settling time determined by the integrator (we could call this the “steady-state settling time”) will simultaneously decrease. Specific system requirements would determine how we balance these considerations.

Our final controller design has transfer function

$$K_1 K_2 \cdot \frac{s + 0.35}{s} = 4.61 \cdot \frac{s + 0.35}{s}.$$

## 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.

### 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  $\alpha$ .

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

$$K_2 \frac{s - z_c}{s - p_c}, \quad (1)$$

where  $p_c < 0$  is a real pole near the origin;  $z_c$  is a real zero near  $\alpha p_c$ ; and, initially,  $K_2 = 1$ . For minimal effect on the original transient response design,  $\text{Re}(p_{1,2}) \ll z_c$ , 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 desirable as possible. This step can often be omitted.

2. There are more precise ways to compute a location of  $z_c$  based on a specified factor  $\alpha$  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.

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

$$K_1 K_2 \frac{s - z_c}{s - p_c}. \quad (2)$$

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

$$\frac{s + 10}{s^2 + 8s + 25}. \quad (3)$$

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 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)
```

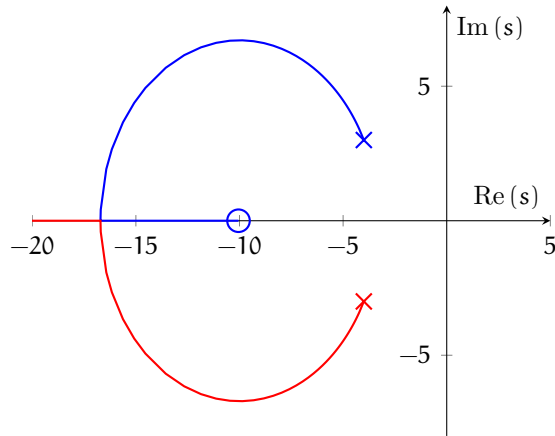
Let's use the second-order approximation that

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

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

$$K_1 = 12. \quad (5)$$





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

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 long 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
```

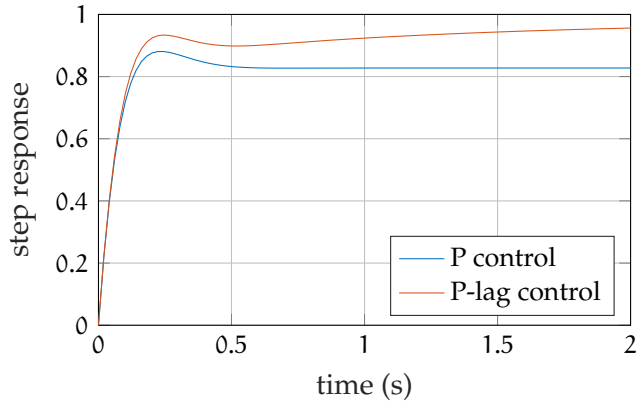
Let's look at the simulation results, shown in [Figure PLag.2](#). The settling time for the proportional controller looks about right, but the steady-state error is about 18%. We'd like it to be about 1.8%. The lag compensator has a similar transient and a slow steady-state error decrease. It's so slow 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);
```

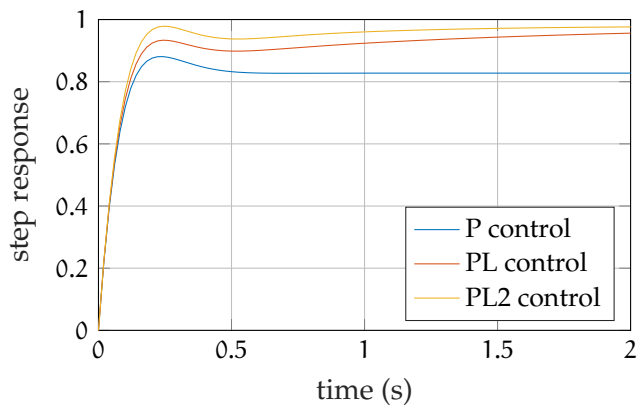
```
C_L2 =
    (s+2)
    -----
    (s+0.2)

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

From [Figure PLag.3](#), we see that there's improvement. Let's try increasing the gain  $K_2$



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

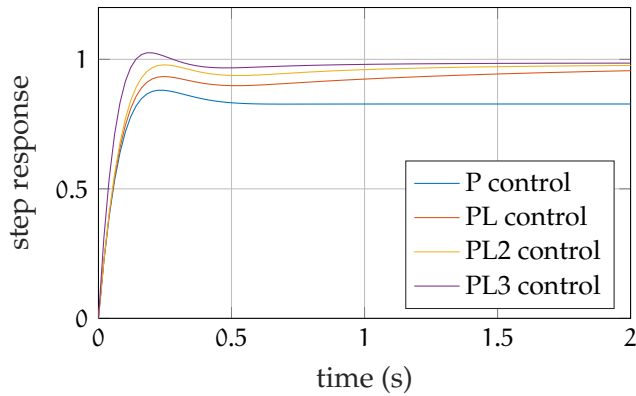


**Figure P Lag.3:** step responses for proportional and proportional-lag controllers (secondary design).

and moving the compensator pole and zero leftward more aggressively to see if we can 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.
```



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

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.

## rldesign.PD Proportional–derivative (PD) controller design

Thus far, our designs have been restricted to closed-loop pole locations on the original root locus. We could add integral or lag compensation for steady-state error performance and vary the gain for transient response performance. But what if we desire closed-loop poles  $p_{1,2}$  to be in a location that the root locus does not intersect?

Among many possible methods to address this, we pursue the following: a derivative compensator with zero location  $z_c$  chosen such that the root locus intersects  $p_{1,2}$ , with form

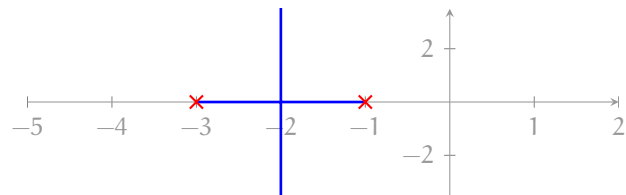
$$K(s - z_c), \quad (1)$$

where  $K \in \mathbb{R}$  is a gain. This compensator is called “derivative” because its primary effect on the overall controller’s operation on the error  $e$  is a new factor of  $s$ , yielding a scaling of the term  $sE(s) = \dot{e}(t)$ .

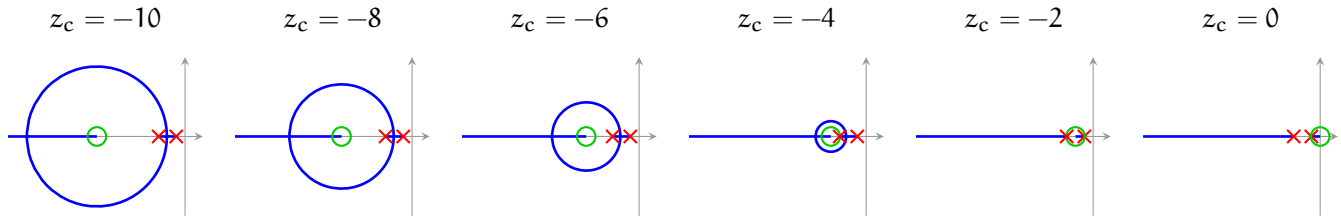
The effect of this zero is to pull the locus toward it. Consider the simple plant of Fig. PD.1.

Suppose we would like to speed up the closed-loop response, but cannot because, no matter how much gain we use, the settling time is fixed by the vertical asymptotes. If we use a compensator zero at  $z_c$ , we can pull the locus leftward, as shown in Fig. PD.2. Varying  $z_c$  from  $-\infty$  to 0, we see that any location left of  $-2$  can be intersected. In fact, if we consider both positive and negative gains for this example, we can place a desired closed-loop pole at any location in the complex plane!

A way to approach designing a controller for a plant  $G$  with a derivative compensator  $C$  is to consider the compensator zero’s effect on the phase criterion, which must always be satisfied



**Figure PD.1:** root locus for a simple plant with two poles.



**Figure PD.2:** root locus (blue) for plant with poles (red) compensated with a zero (green) at  $z_c$ . Note that varying  $z_c$  yields root loci that can intersect any point in the complex plane if negative gains are considered. An animation corresponding to this figure can be found at [https://youtu.be/VZbT\\_2bT2xU](https://youtu.be/VZbT_2bT2xU).

at points on the root locus:

$$\angle(G(s)C(s)) = \pi. \tag{2}$$

In order for a desired point  $s = \psi$  to be on the root locus, then,<sup>3</sup>



3. The  $2\pi$  modulo in these expressions is suppressed for clarity.

Let this angle  $\angle(\psi - z_c)$ , called the compensator angle, be given the symbol

$$\theta_c \equiv \angle(\psi - z_c). \tag{3}$$

Then

$$z_c = \text{Re}(\psi) - \text{Im}(\psi) / \tan \theta_c \quad (\theta_c \in [-\pi, \pi]), \tag{4}$$

where we have limited the application of this result to  $\theta_c \in [-\pi, \pi]$  because a single zero can contribute angles in this interval only.<sup>4,5</sup> This result is to be used in the design procedure that follows. It can be understood geometrically as the position of  $z_c$  such that the angle of the vector with tail at  $z_c$  and head at  $\psi$  is  $\theta_c$ .

4. See [Lec. rldesign.multd](#) for how to handle required angle compensations beyond  $\pm\pi$ .

5. Note that  $\theta_c \in [-\pi, 0)$  is possible only when  $\text{Im} \psi < 0$  and  $\theta_c \in (0, \pi]$  is possible only when  $\text{Im} \psi > 0$ .

### Design procedure

The following procedure provides a starting-point for proportional-derivative controller design. Let's assume the transient

response specification is such that we desire a closed-loop pole to be located at  $s = \psi$ .

1. Design a proportional controller to meet transient response requirements by choosing the gain  $K_1$  for the dominant closed-loop poles to be as close as possible to  $\psi$ .
2. Include a cascade derivative compensator of the form

$$K_2(s - z_c), \quad (5)$$

where, initially,  $K_2 = 1$  and  $z_c$  is a real zero that satisfies Eq. 4. For convenience, we repeat the two key formulas:

$$\theta_c = \pi - \angle G(\psi) \quad \text{and} \\ z_c = \text{Re}(\psi) - \text{Im}(\psi) / \tan \theta_c \quad (\theta_c \in [-\pi, \pi]).$$

3. Use a new root locus to tune the gain  $K_2$  such that a closed-loop pole is at  $\psi$ .
4. Construct the closed-loop transfer function with the controller

$$K_1 K_2 (s - z_c). \quad (6)$$

5. Simulate the time response to see if it meets specifications. Tune.

### A design example

Let a system have plant transfer function

$$\frac{1}{(s + 2)(s + 6)(s + 11)}. \quad (7)$$

Design a PD controller such that the closed-loop settling time is about 0.8 seconds and the overshoot is about 15%.

### Determining $\psi$

We use Matlab for the design.<sup>6</sup> First, we must determine what the specified transient response criteria imply for the locations of our

6. See [ricopic.one/control/source/pd\\_controller\\_design\\_example.m](http://ricopic.one/control/source/pd_controller_design_example.m) for the source.

closed-loop poles. Let one of these desired pole locations be called  $\psi$ . The transient response performance criteria are as follows.

```
Ts = .8; % sec ... spec settling time
OS = 15; % percent ... spec overshoot
```

The second-order approximation from [Chapter trans](#) tells us that the settling time specification implies a specific  $\text{Re}(\psi)$  and the overshoot a specific angle  $\angle\psi$ . The real part is found from the expressions

$$T_s = \frac{4}{\zeta\omega_n} \quad \text{and} \quad \text{Re}(\psi) = -\zeta\omega_n \Rightarrow \quad (8)$$

$$\text{Re}(\psi) = -\frac{4}{T_s}. \quad (9)$$

The angle is found via the equations

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}, \quad (10)$$

$$\tan(\angle\psi) = \frac{\sqrt{1-\zeta^2}}{\zeta}, \quad \text{and} \quad \tan(\angle\psi) = -\text{Im}(\psi)/\text{Re}(\psi). \quad (11)$$

A remarkably simple expression results:

$$\text{Im}(\psi) = -\text{Re}(\psi) \frac{\sqrt{1-\zeta^2}}{\zeta} \quad (12a)$$

$$\text{Im}(\psi) = -\text{Re}(\psi) \frac{\pi}{\ln(100/\%OS)}. \quad (12b)$$

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

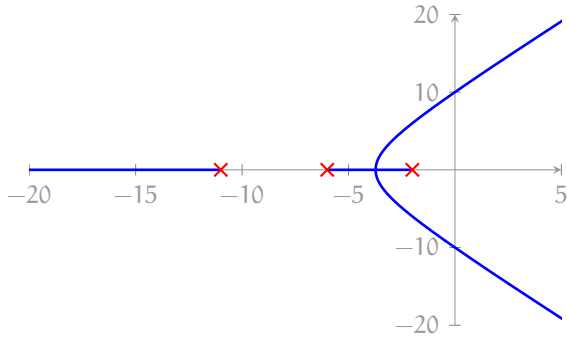
$$\psi = -\frac{4}{T_s} \left( 1 - j \frac{\pi}{\ln(100/\%OS)} \right). \quad (13)$$

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

```
psi_fun = @(Ts,pOS) -4/Ts*(1-1j*pi/log(100/pOS));
psi = psi_fun(Ts,OS);
disp(sprintf('psi = %0.3g + j %0.3g',real(psi),imag(psi)))
```

```
| psi = -5 + j 8.28
```





**Figure PD.3:** root locus without compensation.

### P control

We design a proportional controller that gets us as close as possible to  $\psi$ . The root locus is shown in [Figure PD.3](#).

```
G = zpk([], [-2, -6, -11], 1);
figure
rlocus(G)
```

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

$$K_1 = 240. \tag{14}$$

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

```
K1 = 240;
G_P = feedback(K1*G, 1);
```

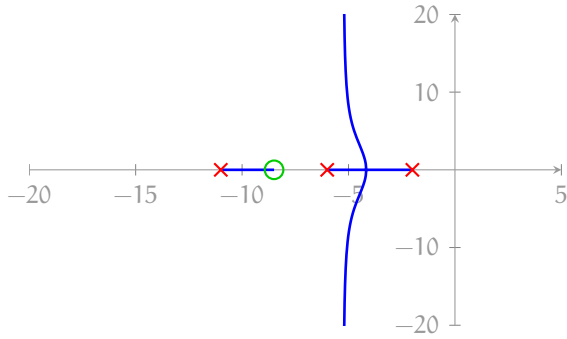
### Derivative compensation

Now, we use cascade derivative compensation with compensator

$$K_2(s - z_c). \tag{15}$$

For now, we set  $K_2 = 1$ . From [Equation 4](#), we compute the compensator zero

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



**Figure PD.4:** root locus with compensation.

```
theta_c = pi - angle(evalfr(G,psi));
z_c = real(psi) - abs(imag(psi))/tan(theta_c);
disp(sprintf('theta_c = %0.3g deg',rad2deg(theta_c)))
disp(sprintf('z_c = %0.3g',z_c))
```

```
theta_c = 67.1 deg
z_c = -8.5
```

Let's construct the compensator sans tuned gain  $K_2$  and tune it up using another root locus.

```
C_sans = zpk(z_c, [], 1);
figure
rlocus(K1*C_sans*G)
```

The resulting root locus of [Figure PD.4](#) intersects  $\psi$ ! (I mean, we knew it would, but we had our doubts.) The corresponding gain is, from [Equation 2](#) (or we could use the data cursor),

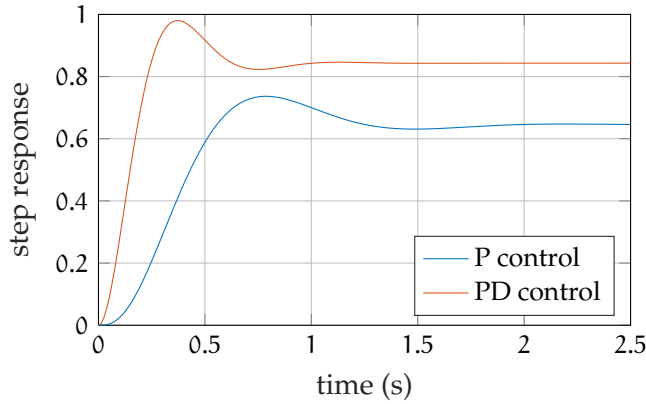
$$K_2 = \frac{1}{|(\psi - z_c)G(\psi)|} \tag{16}$$

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

```
K2 = 1/abs(evalfr(K1*C_sans*G,psi));
C = K1*K2*C_sans;
G_PD = feedback(C*G,1);
```

**Simulate**

Our placement of the  $\psi$  depended on the second-order approximation's accuracy, which



**Figure PD.5:** step responses for proportional and proportional-derivative controllers.

in this case is questionable, due to the proximity of a third closed-loop pole. In any case, we simulate the step response to test the efficacy of the PD controller design and to compare it 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_PD = step(G_PD,t_a); % PD controlled step response
```

```
figure
plot(t_a,y_P);
hold on;
plot(t_a,y_PD);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','location','southeast');
```

The responses, shown in [Figure P Lag.3](#), suggest the PD controller is at least close to meeting the transient specifications. It is a happy accident that the steady-state error also improved; derivative compensation does not always do this. Let's use `stepinfo` to compute more accurate transient response characteristics of the PD-controlled system.

```
si_PD = stepinfo(y_PD,t_a);  
disp(sprintf('settling time: %0.3g',si_PD.SettlingTime))  
disp(sprintf('percent overshoot: %0.3g',si_PD.Overshoot))
```

```
settling time: 0.82  
percent overshoot: 16.2
```

This is quite close to the specification. If desired, the gain  $K_2$  and the zero location  $z_c$  could be tuned, iteratively.

## rldesign.PLead Proportional–lead design

Similar to how proportional-lag controllers can be considered passively realizable PI controllers, proportional-lead controllers can be considered passively realizable PD controllers. The idea is to choose a design point  $\psi$  through which we construct the root locus to pass. As with PD control, this point is chosen to meet primarily transient response characteristics, and the controller contributes the proper phase such that the root locus passes through the point; however, we have both a pole and a zero to set in the compensator:

$$C(s) = K_2 \frac{s - z_c}{s - p_c}. \quad (1)$$

We will “arbitrarily” choose either  $p_c$  or  $z_c$  and the phase criterion for our design point  $\psi$  will set the other. However, the “arbitrary” selection of  $p_c$  or  $z_c$  in fact affects both the transient response and the steady-state error (if it is finite).

Let’s work out the details. A way to approach designing a controller for a plant  $G$  with lead compensator  $C$  is to consider the compensator effects on the phase criterion, which must always be satisfied at points on the root locus:

$$\angle(G(s)C(s)) = \pi. \quad (2)$$

In order for a desired point  $s = \psi$  to be on the root locus, then,<sup>7</sup>

7. The  $2\pi$  modulo in these expressions is suppressed for clarity.

$$\begin{aligned} \angle(G(\psi)C(\psi)) &= \pi \\ \angle G(\psi) + \angle C(\psi) &= \pi \Rightarrow \\ \angle C(\psi) &= \pi - \angle G(\psi) \Rightarrow \\ \angle(\psi - z_c) - \angle(\psi - p_c) &= \pi - \angle G(\psi). \end{aligned}$$

Let this angle  $\angle(\psi - z_c) - \angle(\psi - p_c)$ , called the compensator angle, be given the symbol

$$\theta_c \equiv \angle(\psi - z_c) - \angle(\psi - p_c). \quad (3)$$

So we can choose to arbitrarily set the location of either  $z_c$  or  $p_c$  and the other will be set by the phase criterion. Therefore we have either

$$\angle(\psi - p_c) = \underbrace{\angle(\psi - z_c)}_{\text{arbitrary}} - \theta_c \quad \text{or} \quad (4a)$$

$$\angle(\psi - z_c) = \theta_c + \underbrace{\angle(\psi - p_c)}_{\text{arbitrary}}. \quad (4b)$$

And, from trigonometry,

$$p_c = \text{Re}(\psi) - |\text{Im}(\psi)| / \tan(\angle(\psi - z_c) - \theta_c) \quad \text{or} \quad (5a)$$

$$z_c = \text{Re}(\psi) - |\text{Im}(\psi)| / \tan(\theta_c + \angle(\psi - p_c)). \quad (5b)$$

This result is to be used in the design procedure that follows.

### Design procedure

The following procedure provides a starting-point for proportional-lead controller design. Let's assume the transient response requirement is such that, according to the second-order approximation, we desire a closed-loop pole to be located at  $s = \psi$ .

1. Design a proportional controller to meet transient response requirements by choosing the gain  $K_1$  for the dominant closed-loop poles to be as close as possible to  $\psi$ .
2. Include a cascade lead compensator of the form

$$K_2 \frac{s - z_c}{s - p_c}, \quad (6)$$

where we arbitrarily set either  $z_c$  or  $p_c$ ; initially,  $K_2 = 1$ . The other parameter must be chosen to satisfy Eq. 5a or Eq. 5b. For convenience, we repeat the key formulas:

$$\theta_c = \pi - \angle G(\psi) \quad \text{and, after setting arbitrarily } z_c \text{ or } p_c,$$

$$p_c = \text{Re}(\psi) - |\text{Im}(\psi)| / \tan(\angle(\psi - z_c) - \theta_c) \quad \text{or}$$

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

- By construction  $\psi$  is on the root locus, so the gain can be computed directly from Eq. 2:

$$K_2 = \frac{1}{|K_1 C(\psi) G(\psi)|}. \tag{7}$$

- Construct the closed-loop transfer function with the controller

$$K_1 K_2 \frac{s - z_c}{s - p_c}. \tag{8}$$

- Simulate the time response to see if it meets specifications. Tune.

### A design example

Let a system have plant transfer function

$$\frac{37500}{s^4 + 70s^3 + 1625s^2 + 14000s + 37500}. \tag{9}$$

Design a P-lead controller such that the closed-loop settling time is about 0.4 seconds and the overshoot is about 10%.

### Determining $\psi$

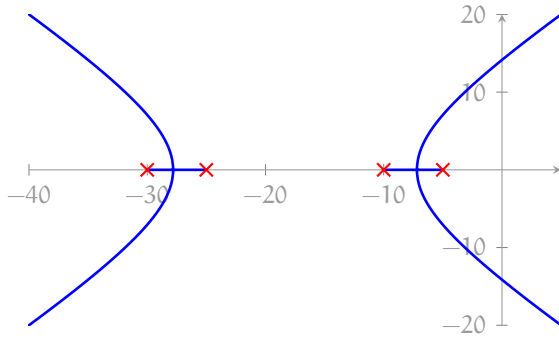
We use Matlab for the design.<sup>8</sup> First, we must determine what the specified transient response criteria imply for the locations of our closed-loop poles. Let one of these desired pole locations be called  $\psi$ . The transient response performance criteria are as follows.

8. See [ricopic.one/control/source/plead\\_controller\\_design\\_example.m](http://ricopic.one/control/source/plead_controller_design_example.m) for the source.

```
Ts = .4; % sec ... spec settling time
OS = 10; % percent ... spec overshoot
```

The second-order approximation from [Chapter trans](#) tells us that the settling time specification implies a specific  $\text{Re}(\psi)$  and the overshoot a specific angle  $\angle\psi$ . From previous results, the desired pole location  $\psi$  (assuming the second-order approximation is valid) is given by the expression

$$\psi = -\frac{4}{T_s} \left( 1 - j \frac{\pi}{\ln(100/\%OS)} \right). \tag{10}$$



**Figure PLead.1:** root locus without compensation.

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

```
psi_fun = @(Ts,pOS) -4/Ts*(1-1j*pi/log(100/pOS));
psi = psi_fun(Ts,OS);
disp(sprintf('psi = %0.3g + j %0.3g',real(psi),imag(psi)))
```

| psi = -10 + j 13.6

### P control

We design a proportional controller that gets us as close as possible to  $\psi$ . The root locus is shown in [Figure PLead.1](#).

```
G = tf([37500],[1,70,1625,14000,37500]);
figure
rlocus(G)
```

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

$$K_1 = 1.1. \tag{11}$$

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

```
K_1 = 1.1;
G_P = feedback(K_1*G,1);
```



## Lead compensation

Now, we use cascade lead compensation with compensator

$$K_2 \frac{s - z_c}{s - p_c}. \quad (12)$$

For now, we set  $K_2 = 1$ . Let's also set  $p_c = -40, -100,$  and  $-400$  to see how we fair with different "arbitrary" choices. 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_c = [-40,-100,-400];
theta_c = pi - angle(evalfr(G,psi));
theta_p_c = angle(psi*ones(size(p_c))-p_c);
z_c = real(psi) - abs(imag(psi))./tan(theta_c + theta_p_c);
disp(sprintf('theta_c = %0.3g deg',rad2deg(theta_c)))
for i = 1:length(p_c)
    disp(sprintf(...
        'pole phase contribution = %0.3g deg',...
        rad2deg(theta_p_c(i))...
    ))
    disp(sprintf('z_c = %0.3g',z_c(i)))
end
```

```
theta_c = 96.7 deg
pole phase contribution = 24.5 deg
z_c = -1.75
pole phase contribution = 8.62 deg
z_c = -6.26
pole phase contribution = 2 deg
z_c = -7.91
```

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

```
C_sans = stack(1,tf(1,1)); % initialize model array
C = stack(1,tf(1,1)); % initialize model array
for i = 1:length(p_c)
    C_sans(i) = zpk(z_c(i),p_c(i),1); % without gain
    K_2(i) = 1/abs(evalfr(K_1*C_sans(i)*G,psi));
    C(i) = K_1*K_2(i)*C_sans(i);
    disp(sprintf('K_2 = %0.3g',K_2(i)))
end
```

```
K_2 = 4.88
K_2 = 15.2
K_2 = 66.7
```

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

```
G_Plead = stack(1,tf(1,1));
for i = 1:length(p_c)
    G_Plead(i) = feedback(C(i)*G,1);
end
```

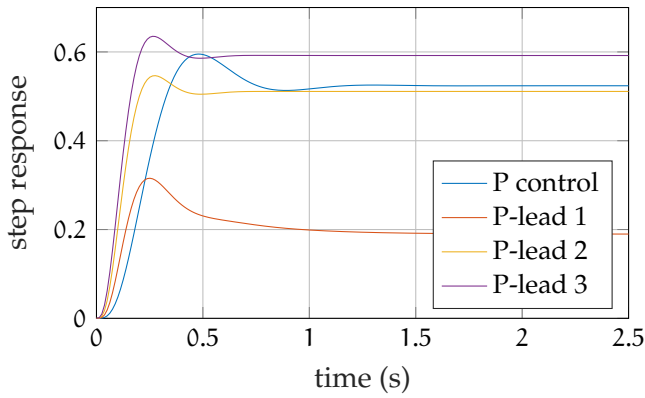
Simulate

Our placement of the  $\psi$  depended on the second-order approximation's accuracy, which in this case is questionable. In any case, we simulate the step response to test the efficacy of the P-lead controller design and to compare it 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
for i = 1:length(p_c)
    y_Plead(:,i) = step(G_Plead(i),t_a); % P-lead step resp.
end
```

```
figure
plot(t_a,y_P);
hold on;
for i = 1:length(p_c)
    plot(t_a,y_Plead(:,i));
end
xlabel('time (s)');
ylabel('step response');
grid on
legend(...
    'P control','P-lead 1','P-lead 2','P-lead 3',...
    'location','southeast'...
);
```

The responses, shown in [Figure PLead.2](#), suggest the lead-compensated controllers are at least close to meeting the transient specifications. The steady-state error is worse for compensator locations that are less-negative and better for those that are more-negative. For this reason, we remember that our “arbitrary” choice of one of our compensator parameters still affects the steady-state (and sometimes transient) response. Let's use `stepinfo` to



**Figure PLead.2:** step responses for proportional and proportional-lead controllers.

compute more accurate transient response characteristics for the different controllers.

```

disp('P control')
si_P = stepinfo(y_P,t_a);
disp(sprintf('settling time: %0.3g',si_P.SettlingTime))
disp(sprintf('percent overshoot: %0.3g\n',si_P.Overshoot))
for i = 1:length(p_c)
    si_Plead = stepinfo(y_Plead(:,i),t_a);
    disp(sprintf('p_c: %0.3g',p_c(i)))
    disp(sprintf(...
        'settling time: %0.3g',si_Plead.SettlingTime ...
    ))
    disp(sprintf(...
        'percent overshoot: %0.3g\n',si_Plead.Overshoot...
    ))
end
    
```

```

P control
settling time: 0.906
percent overshoot: 13.6

p_c: -40
settling time: 1.28
percent overshoot: 66.2

p_c: -100
settling time: 0.371
percent overshoot: 6.95

p_c: -400
settling time: 0.37
percent overshoot: 7.31
    
```

We see that most of the P-lead controllers meet the settling time and percent overshoot

requirements. However, the first one is problematic. This is mostly due to the second-order approximation being significantly violated in this case. We see from the time response that the initial overshoot happens quickly, but the return to steady-state is slow. If desired, the gain  $K_2$  and compensator pole and zero locations could be tuned, iteratively.

## rldesign.PID Prop–integral–derivative controller design

We have designed P, PI, and PD controllers.

Now we include all three terms in a single PID controller. With this, we can design for both steady-state and transient response.

A PID controller transfer function will have one pole and two zeros. One zero  $z_I$  and the pole will be specified by an integral compensator and the other zero  $z_D$  will be specified by a derivative compensator.

Our design process will yield a PID controller with transfer function

$$\underbrace{K_1}_{\text{P design}} \cdot \underbrace{K_2(s - z_D)}_{\text{D compensation}} \cdot \underbrace{K_3 \frac{(s - z_I)}{s}}_{\text{I compensation}} = K_P + K_I/s + K_D s, \quad (1)$$

where the named gains are called proportional  $K_P$ , integral  $K_I$ , and derivative  $K_D$ . The design procedure below will yield numbered gains  $K_1$  (P design),  $K_2$  (D compensation), and  $K_3$  (I compensation). They are related as follows:

$$K_P = -K_1 K_2 K_3 (z_D + z_I) \quad (2)$$

$$K_I = K_1 K_2 K_3 z_I z_D \quad (3)$$

$$K_D = K_1 K_2 K_3. \quad (4)$$

Our design procedure is as follows.

1. Check that the integral compensation of a PID controller is necessary and sufficient to meet the steady-state performance criteria.
2. From the transient performance criteria and using the second-order approximation, determine the region of the  $s$ -plane in which the dominant closed-loop poles of the root locus should appear.
3. Design a P controller and evaluate its transient response performance.

4. Apply derivative (D) compensation to improve the transient response. Simulate to verify the transient response performance.
5. Apply integrator (I) compensation to improve the steady-state error performance.
6. Check all performance criteria and adjust gains and zero locations, as-needed.
7. Determine gains: proportional  $K_p$ , integral  $K_I$ , and derivative  $K_D$ .

### A design example

Let a system have plant transfer function

$$\frac{s + 40}{s^2 + 10s + 200} \quad (5)$$

Design a PID controller with unity feedback such that the closed-loop rise time is about 0.05 seconds, the overshoot is less than 5%, and the steady-state error is zero for a step command.

### Determining $\psi$

We use Matlab for the design.<sup>9</sup> First, we see that the plant is Type 0, so integral compensation is required to yield zero steady-state error for a step command and therefore a PID controller is a good choice. Second, 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.

<sup>9</sup>. See [ricopic.one/control/source/pid\\_controller\\_design\\_example\\_01.m](https://www.ricopic.one/control/source/pid_controller_design_example_01.m) for the source.

```
Tr = .05; % sec ... spec rise time
OS = 5; % percent ... spec overshoot max
```

The second-order approximation from [Chapter trans](#) tells us, via [Fig. exact.2](#), that the rise time specification implies a specific ratio

between  $\omega_n$  and the implicit function  $f(\zeta)$  defined in Fig. exact.2:

$$T_r \omega_n = f(\zeta) \Rightarrow \quad (6a)$$

$$T_r = \frac{f(\zeta)}{\omega_n} \quad (6b)$$

$$= 0.05. \quad (\text{spec})$$

The minimum angle is determined from the overshoot specification via the relations

$$\angle\psi = \pi - \arccos \zeta \quad \text{and} \quad (7)$$

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}. \quad (8)$$

```
zeta = -log(OS/100)/sqrt(pi^2+log(OS/100)^2)
psi_angle_min = pi - acos(zeta)
```

```
zeta =
    0.6901

psi_angle_min =
    2.3324
```

With  $\zeta$  in-hand, we use Fig. exact.2 to determine  $f(\zeta)$  and apply Eq. 6a to determine  $|\psi| = \omega_n$ :

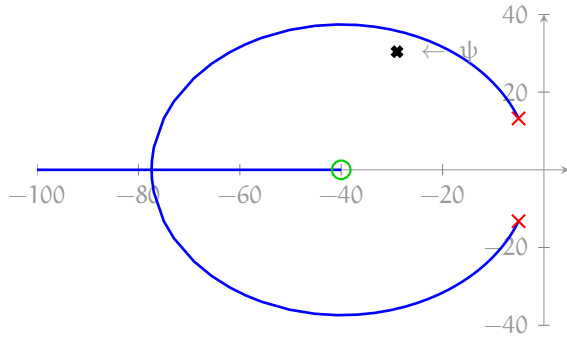
```
psi_mag = 2.1/Tr % also = omega_n
```

```
psi_mag =
    42
```

So the target magnitude  $|\psi|$  and minimum angle  $\angle\psi$  are determined. Let's convert this to rectangular coordinates:

```
psi_real = psi_mag*cos(psi_angle_min);
psi_imag = psi_mag*sin(psi_angle_min);
psi = psi_real+i*psi_imag
```

```
psi =
    -28.9845 +30.3957i
```



**Figure PID.1:** root locus without compensation.

So this is our design target for the dominant closed-loop poles. As usual, it depends on the second-order approximation, so we will need to simulate to determine the actual performance.

### P control

We design a proportional controller that gets us as close as possible to  $\psi$ . The root locus is shown in [Figure PID.1](#).

```
G = tf([1,40],[1,10,200]);
figure
rlocus(G);hold on
plot(psi,'kx','MarkerSize',5,'LineWidth',2)
text(real(psi),imag(psi),' \leftarrow \psi')
```

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

$$K_1 = 64. \tag{9}$$

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

```
K1 = 64;
G_P = feedback(K1*G,1); % closed loop transfer func
```



## Derivative compensation

Now, we try cascade derivative compensation with compensator

$$K_2(s - z_c). \quad (10)$$

For now, we set  $K_2 = 1$ . From Equation 4, we compute the compensator zero angle contribution

$$\theta_c = \pi - \angle G(\psi).$$

```
theta_c = pi - angle(evalfr(G,psi));
disp(sprintf('theta_c = %0.3g deg',rad2deg(theta_c)))
```

```
| theta_c = 13.1 deg
```

We try using the zero compensator:

$$K_2(s - z_c). \quad (11)$$

where

$$z_c = \operatorname{Re}(\psi) - |\operatorname{Im}(\psi)| / \tan(\theta_c) \quad (12)$$

```
z_c = real(psi) - abs(imag(psi))/tan(theta_c);
disp(sprintf('z_c = %0.3g',z_c))
```

```
| z_c = -159
```

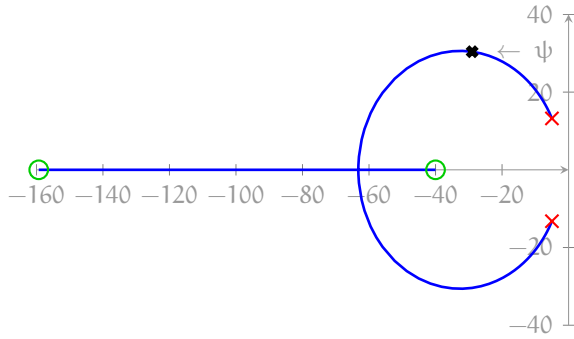
Let's construct the compensator sans tuned gain  $K_2$  and construct the corresponding root locus.

```
C_sans = zpk(z_c, [], 1);
figure
rlocus(K1*C_sans*G);hold on
plot(psi, 'kx', 'MarkerSize', 5, 'LineWidth', 2)
text(real(psi), imag(psi), ' \leftarrow \psi')
```

By construction, the resulting root locus of Fig. PID.2 intersects  $\psi$ . The corresponding gain is, from Eq. 2 (or we could use the data cursor),

$$K_2 = \frac{1}{|K_1(\psi - z_c)G(\psi)|}. \quad (13)$$

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



**Figure PID.2:** root locus with derivative compensation.

```
K2 = 1/abs(evalfr(K1*C_sans*G,psi))
C = K1*K2*C_sans;
G_PD = feedback(C*G,1);
```

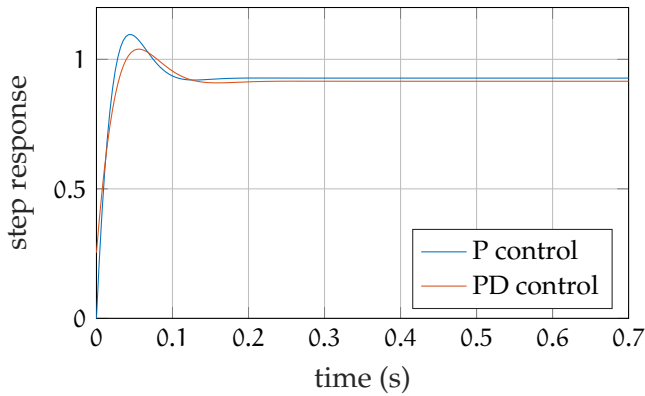
```
K2 =
    0.0053
```

Simulate Our placement of the  $\psi$  depended on the second-order approximation’s accuracy, which in this case is questionable, due to the proximity of a third closed-loop pole. In any case, we simulate the step response to test the efficacy of the PD controller design and to compare it with the P controller.

```
t_a = linspace(0,.7,200); % s ... sim time
y_P = step(G_P,t_a); % P controlled step response
y_PD = step(G_PD,t_a); % PD controlled step response
```

```
figure
plot(t_a,y_P);
hold on;
plot(t_a,y_PD);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','location','southeast');
```

The responses, shown in [Figure PID.3](#), suggest the PD controller is probably not meeting the transient performance specifications. Let’s use `stepinfo` to compute more accurate transient



**Figure PID.3:** step responses for proportional and proportional-derivative controllers.

response characteristics of the PD-controlled system.

```

si_PD = stepinfo(y_PD,t_a);
disp(sprintf('rise time: %0.3g',si_PD.RiseTime))
disp(sprintf('percent overshoot: %0.3g',si_PD.Overshoot))
    
```

```

rise time: 0.022
percent overshoot: 13.6
    
```

It's too fast and overshoots too much. Our second-order approximation that led to this design is not very accurate. Before we start tuning this design, let's fix the steady-state error by including an integral compensator. Perhaps this compensator's zero can "help" us with our fix.

### Integral compensation

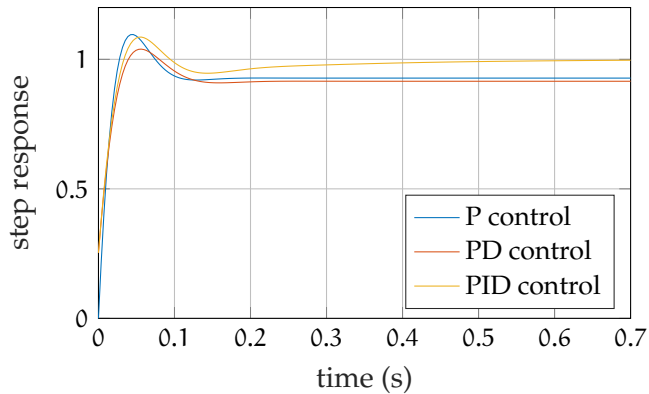
The integral compensator has its usual form

$$K_3 \frac{s - z_I}{s} \tag{14}$$

We're less concerned than usual about affecting our transient response with this compensator because we need some help doing so in any case. Let's start with  $z_I = -5$ .

```

z_I = -5;
C_I_sans = zpk(z_I,0,1);
    
```



**Figure PID.4:** step responses for proportional and proportional-derivative controllers.

Now, a root locus wouldn't be particularly helpful here, since our second-order approximation is poor and getting worse by the minute. Instead, we proceed directly to simulation.

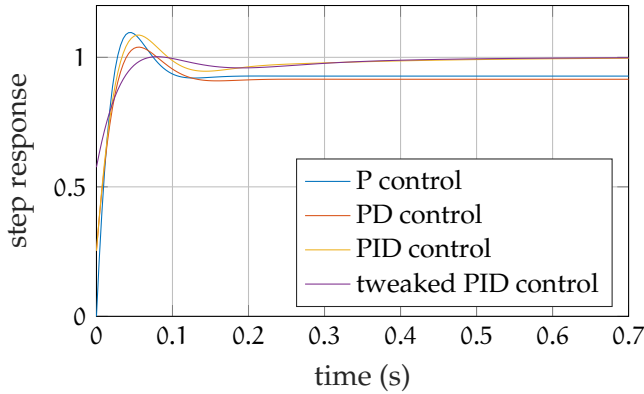
```
G_PID1 = feedback(C_I_sans*C*G,1);
y_PID1 = step(G_PID1,t_a); % PID controlled step response
```

The responses, shown in [Figure PID.4](#), show that the steady-state error has improved with integral compensation, and so has the transient response, but not enough.

```
figure
plot(t_a,y_P);
hold on;
plot(t_a,y_PD);
hold on;
plot(t_a,y_PID1);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','PID control',...
'location','southeast');
```

Let's take a look at the `stepinfo`.

```
si_PID = stepinfo(y_PID1,t_a);
disp(sprintf('rise time: %0.3g',si_PID.RiseTime))
disp(sprintf('percent overshoot: %0.3g',si_PID.Overshoot))
```



**Figure PID.5:** step responses for proportional and proportional-derivative controllers.

```
rise time: 0.0246
percent overshoot: 8.98
```

It's still too fast and overshoots too much. At this point we can directly tweak our compensator zeros and the overall gain to try to meet our specifications.

```
K3 = 4;
z_D = -25;
z_I = -8;
C_D_sans = zpk(z_D, [], 1);
C_I_sans = zpk(z_I, 0, 1);
G_PID2 = feedback(K1*K2*K3*C_I_sans*C_D_sans*G, 1);
y_PID2 = step(G_PID2, t_a); % PID controlled step response
```

```
figure
plot(t_a, y_P); hold on;
plot(t_a, y_PD); hold on;
plot(t_a, y_PID1); hold on;
plot(t_a, y_PID2);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control', 'PD control', 'PID control', ...
      'tweaked PID control', 'location', 'southeast');
```

```
si_PID = stepinfo(y_PID2, t_a);
disp(sprintf('rise time: %0.3g', si_PID.RiseTime))
disp(sprintf('percent overshoot: %0.3g', si_PID.Overshoot))
```

```
rise time: 0.0422
percent overshoot: 0.391
```

It turns out to be difficult to meet both specifications, even with the massively tweaked controller design. Whenever one attempts to increase the rise time, the overshoot also increases. However, we've done a serviceable job, considering.

Compute the PID gains.

$$\begin{aligned} K_P &= -K_1 * K_2 * K_3 * (z_D + z_I) \\ K_I &= K_1 * K_2 * K_3 * z_D * z_I \\ K_D &= K_1 * K_2 * K_3 \end{aligned}$$

KP =

44.8013

KI =

271.5228

KD =

1.3576

## rldesign.PLeLa Proportional–lead–lag controller design

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.

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

### A design example

Let a system have plant transfer function

$$\frac{200}{s^3 + 29s^2 + 170s - 200} \quad (1)$$

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>10</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.

10. See [ricopic.one/control/source/plaglead\\_controller\\_design\\_example.m](http://ricopic.one/control/source/plaglead_controller_design_example.m) for the source.

```
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,  $\angle\psi$ :

$$\angle\psi = \pi - \arccos \zeta. \quad (2)$$

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

$$T_S = -4 / \text{Re}(\psi). \quad (3)$$

```
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)
```

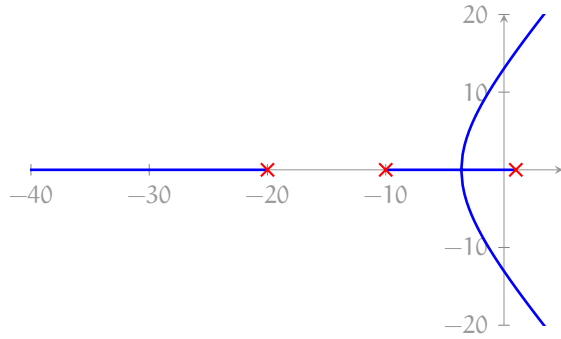
Although we cannot get close to  $\psi$  on the root locus, we can at least meet our %OS specification by choosing a gain of about

$$K_1 = 5. \quad (4)$$

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);
```





**Figure PLeLa.1:** root locus without compensation.

### Lead compensation

Now, we use cascade lead compensation with compensator

$$K_2 \frac{s - z_{ld}}{s - p_{ld}} \tag{5}$$

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 = zpk(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  $C_{\text{lead}}$ , and the closed-loop transfer function  $G_{\text{lead}}$ .

```
G_Plead = feedback(C_ld*G,1);
```

Lag compensation

Now, we use cascade lag compensation with compensator

$$K_3 \frac{s - z_{lg}}{s - p_{lg}}. \quad (6)$$

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.

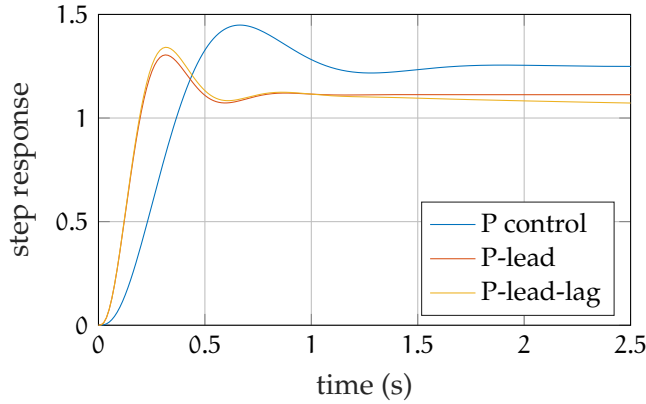
```
alpha = abs(ess_ld)/sse
```

```
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  $G_{\text{PLL}}$ .

```
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);
```



**Figure PLeLa.2:** step responses for proportional, proportionalLead, and proportionalLead-lag controllers.

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);
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.

## rldesign.multd Multiple derivative compensators

Lec. rldesign.PD shows how to design a derivative compensator such that the compensated root locus of a control system can be made to include some test point  $\psi \in \mathbb{C}$  where the designer would like a closed-loop pole (typically to satisfy transient response requirements). This derivative compensator has the form

$$C_D = K(s - z_c), \tag{1}$$

for gain  $K \in \mathbb{R}$  and zero  $z_c \in \mathbb{R}$ . The crux of the design procedure is to compute via the root locus phase criterion<sup>11</sup> the required compensator phase contribution:

$$\theta_c = \pi - \angle GH(\psi) \tag{2}$$

for open-loop transfer function  $GH(s)$ . A trigonometric analysis shows that, for  $\theta_c \in [-\pi, \pi]$ , the compensator zero must be

$$z_c = \text{Re}(\psi) - \text{Im}(\psi) / \tan \theta_c. \tag{3}$$

The obvious limitation here is that if the required compensation  $\theta_c$  is beyond  $\pm\pi$ , the derivative compensator of Eq. 1 cannot contribute sufficient phase. The strategy we adopt here is to augment the derivative compensator to include as many (equal) zeros as we need:

$$C_m = K(s - z_m)^m, \tag{4}$$

where  $z_m$  is a zero of multiplicity  $m$ . We call this a multiple derivative compensator or  $m$ -derivative compensator.

How do we select the compensator zero  $z_m$  and multiplicity  $m$  for a given  $\theta_c$ ? First, we determine  $m$  by determining how many  $\pi$  (or  $-\pi$ ) contributions are required:<sup>12,13</sup>

$$m = \left\lceil \frac{|\theta_c|}{\pi} \right\rceil. \tag{5}$$

11. The phase criterion was defined in Lec. rlocus.def, Eq. 6.

---

Algorithm multd.1 the multiple derivative compensator algorithm.

---

```
function d_comp_m(ψ, GH(s))
    θ_c ← π - ∠GH(ψ)    ▷ required phase comp
    m ← ceiling(θ_c/π)    ▷ zeros needed
    θ_m ← θ_c/m          ▷ divide contributions
    z_m ← Re(ψ) - Im(ψ)/tan θ_m    ▷ trig
    C'_m ← (s - z_m)^m    ▷ comp sans gain
    K_m ← |C'_m(ψ)GH(ψ)|-1    ▷ angle criterion
    C_m ← K_m C'_m        ▷ comp with gain
    return C_m
end function
```

---

12. The function  $\lceil \cdot \rceil$  is called the ceiling function and rounds up to the nearest integer.

13. Note that if  $\theta_c \in [-\pi, \pi]$ , the multiplicity  $m = 1$  and the compensator is a regular derivative compensator.

With this, we can divide-up the the required phase contribution  $\theta_c$  among the  $m$  zeros:

$$\theta_m = \theta_c/m. \tag{6}$$

By construction,  $\theta_m \in [-\pi, \pi]$ , so the compensator zeros should be located at

$$z_m = \text{Re}(\psi) - \text{Im}(\psi)/\tan \theta_m. \tag{7}$$

This is summarized in [Algorithm multd.1](#).

### Causality

A complication can arise when derivative compensation yields a closed-loop transfer function with more zeros than poles—a type of system called non-causal (non-non-causal systems are called causal). Non-causal systems are those that depend on future states, something classically<sup>14</sup> impossible to instantiate in real-time, and therefore a controller that creates such a control system is of no practical use.<sup>15</sup> Adding multiple zeros to a controller can easily yield such undesirable systems.

To mitigate this, we can include  $\iota$  pure integrators  $1/s$  into the compensator. They will obviously affect the root locus, so their effects must be taken into account during the zero compensator calculations. This is done by treating the open-loop transfer function as if it already had the compensator integrators  $1/s^\iota$ .

[Algorithm multd.2](#) summarizes this approach.

### Example rldesign.multd-1

Design a controller to meet the

14. It gets complicated when considering relativity and quantum mechanics, which we do not, here.

15. Non-causal system models are useful for digital signal post-processing, but these are always a posteriori—i.e. “future” time is known because it is in the analytic past. Controllers do not have this luxury.

---

### Algorithm multd.2 the multiple derivative compensator algorithm with $\iota$ integrators.

---

```
function d_comp_mi(ψ, GH(s), ι)
    θ_c ← π - ∠GH(ψ)/sι      ▷ required phase
    comp
    m ← ceiling(θ_c/π)          ▷ zeros needed
    θ_m ← θ_c/m                ▷ divide contributions
    z_m ← Re(ψ) - Im(ψ)/tan θ_m    ▷ trig
    C'_m ← (s - z_m)m/sι      ▷ comp sans gain
    K_m ← |C'_m(ψ)GH(ψ)|-1    ▷ angle criterion
    C_m ← K_m C'_m             ▷ comp with gain
    return C_m
end function
```

---

## 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

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

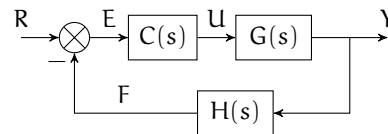


Figure exe.1: a block diagram with a controller  $C(s)$ .



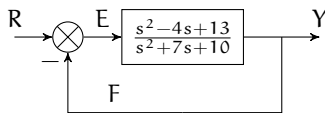
1. Design a PID controller  $C(s)$  such that the closed-loop overshoot is less than 30%, peak time is close to 0.25 seconds, and the steady-state error is zero for a ramp command.
2. Demonstrate the controller performance by simulating and plotting both a step response and a ramp response.
3. Compute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



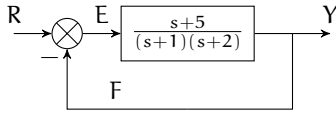
Exercise rldesign.sleep

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

$$\frac{200}{s^3 + 29s^2 + 170s - 200} \tag{4}$$

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%.

16. See [ricopic.one/control/source/plaglead\\_controller\\_design\\_example.m](http://ricopic.one/control/source/plaglead_controller_design_example.m) for the source.

## 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

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

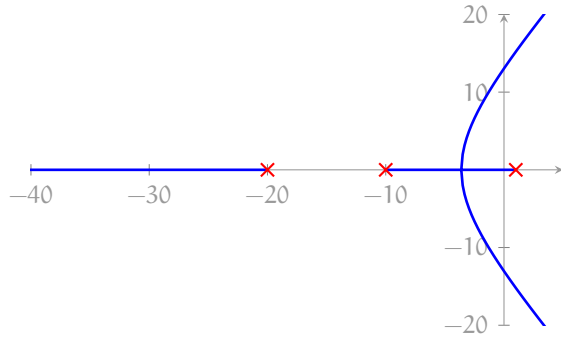
```
zeta = -log(OS/100)/sqrt(pi^2+(log(OS/100))^2);
psi_angle = pi - acos(zeta);
psi_re = -4/Ts;
psi_im = psi_re*tan(psi_angle);
psi = psi_re + j*psi_im;
disp(sprintf('psi = %0.3g + j %0.3g',real(psi),imag(psi)))
```

```
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 = zpk(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

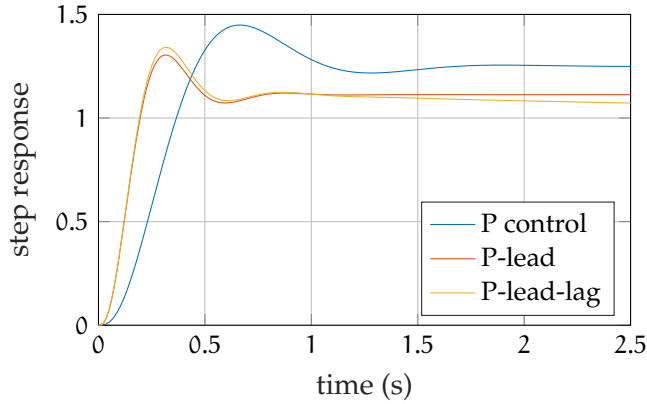
Our placement of the  $\psi$  depended on the second-order approximation's accuracy. In any case, we simulate the step response to test the efficacy of the P-lead and P-lead-lag controller designs and compare them with the P controller.

```
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, proportional-lead, and proportional-lead-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.



## 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.29

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.30

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

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

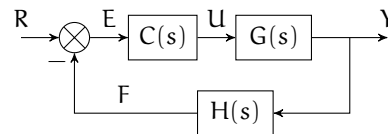


Figure exe.1: a block diagram with a controller  $C(s)$ .

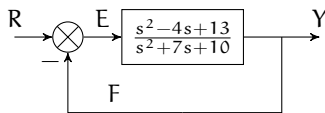
1. Design a PID controller  $C(s)$  such that the closed-loop overshoot is less than 30%, peak time is close to 0.25 seconds, and the steady-state error is zero for a ramp command.
2. Demonstrate the controller performance by simulating and plotting both a step response and a ramp response.
3. Compute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



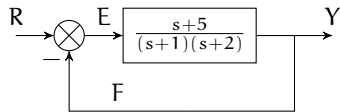
Exercise rldesign.sleep

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

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



freq

---

# Frequency response analysis

## freq.intro Introduction

The frequency response function  $H(j\omega)$  is a complex function that relates a system's input  $u$  to its output  $y$  in terms of the input's frequency content. Given a transfer function  $H(s)$  (which also relates  $u$  to  $y$ ), the frequency response function can be found by the substitution

$$H(j\omega) = H(s)|_{s \rightarrow j\omega}. \quad (1)$$

It can be shown that, for a system with input  $u(t) = A \sin(\omega t + \psi)$ , with  $A, \omega, \psi \in \mathbb{R}$  being the amplitude, angular frequency, and phase of the input, and frequency response function  $H(j\omega)$ , the steady-state output is

### Equation 2 frequency-dependent sinusoidal response

where  $|H(j\omega)|$  and  $\angle H(j\omega)$  are the magnitude (i.e. norm) and phase of  $H(j\omega)$ , respectively. There are three striking aspects of this equation:

1. the output is also a sinusoid at the same frequency as the input;
2. the output amplitude is the input amplitude scaled by  $|H(j\omega)|$ ; and
3. the output phase is the input phase plus  $\angle H(j\omega)$ .

With Fourier Series and Fourier Transform representations of signals, we can consider the input to be composed of sinusoids. For LTI systems, the principle of superposition allows us to construct a corresponding output representation.

In [Lec. freq.bode](#) and [Lec. freq.nyquist](#), we introduce the two primary ways  $H(j\omega)$  is plotted. [Lec. freq.nystab](#) explores what we can learn about system stability from  $H(j\omega)$  and its

plots. Finally, we learn how the different time-domain and frequency-domain representations of a system are related

Lec. [freq.freqtime](#).

The frequency response methods of this chapter were actually developed before root locus methods, and are equivalent in many ways. We learn these methods for two reasons: first, they give us a deeper understanding of control systems and second there are a few situations for which frequency response methods are preferred:

1. when constructing a transfer function from measurement data,
2. when designing a controller for transient and steady-state response characteristics with lead compensation (sans lag compensation), and
3. when determining the stability of a nonlinear system.

## freq.bode Bode plots

1 Given Eq. 2, we are often most-interested in the magnitude  $|H(j\omega)|$  and phase  $\angle H(j\omega)$  of the frequency response function. Each of these is a function of angular frequency  $\omega$ , so plotting  $|H(j\omega)|$  vs.  $\omega$  and  $\angle H(j\omega)$  vs.  $\omega$  is quite useful. Bode plots are such plots with axes scaled in a specific manner.

2 A Bode plot is a useful graphical representation of the frequency response of a system. Let  $|U(\omega)|$  and  $|Y(\omega)|$  be the complex amplitudes of the input and the output, respectively. Recall that the magnitude of the frequency response function  $|H(j\omega)|$  can be expressed as

**Equation 1 frequency response function as an amplitude ratio**

3 This is a ratio of amplitudes, and so it is akin to amplitude ratios commonly expressed in decibels (dB). However, the magnitude ratio of Eq. 1 is not dimensionless, and therefore cannot be expressed as decibel in the strict sense. Nevertheless, it is standard usage in system dynamics and control theory use the familiar formula to compute the logarithmic magnitude

**Equation 2 logarithmic magnitude of  $H(j\omega)$  in "dB"**

4 The phase is usually plotted in degrees, and the  $\omega$ -axis is logarithmic in both plots. The two plots are typically tiled vertically with the magnitude plot above the phase. We now work a simple example.

**Example freq.bode-1**

Let a system have transfer function  $H(s) = s$ , a single zero at the origin. Find the frequency response function and draw the Bode plot for the system.

**re: A simple Bode plot**



## freq.bodesimp Bode plots for simple transfer functions

1 Although we have defined Bode plots in terms of the frequency response function  $H(j\omega)$ , it turns out that, due to its similarity, we can just as easily talk about the Bode plot of a transfer function. Since this is common convention, we proceed in kind.

2 It turns out that bode plots, both magnitude and phase, given their logarithmic scale (recall that the  $\omega$ -axes are also plotted logarithmically), are quite asymptotic to straight-lines for first- and second-order systems. Furthermore, higher-order system transfer functions can be re-written as the product of those of first- and second-order. For instance,

$$H(s) = \frac{\underline{m}s + \underline{c}}{s^3 + \underline{a}s^2 + \underline{b}s + \underline{d}} \quad (1a)$$

$$= \underline{m} \cdot (\underline{c}s + 1) \cdot \frac{1}{\underline{a}s + 1} \cdot \frac{1}{s^2 + \underline{b}s + \underline{d}} \quad (1b)$$

3 Recall (from, for instance, phasor representation) that for products of complex numbers, phases  $\phi_i$  add and magnitudes  $M_i$  multiply. For instance,

$$M_1 \angle \phi_1 \cdot \frac{1}{M_2 \angle \phi_2} \cdot \frac{1}{M_3 \angle \phi_3} = \frac{M_1}{M_2 M_3} \angle (\phi_1 - \phi_2 - \phi_3). \quad (2)$$

And if one takes the logarithm of the magnitudes, they add; for instance,

$$\log \frac{M_1}{M_2 M_3} = \log M_1 - \log M_2 - \log M_3. \quad (3)$$

There is only one more link in the chain: first- and second-order Bode plots depend on a handful of parameters that can be found directly from transfer functions. There is no need to compute  $|H(j\omega_0)|$  and  $\angle H(j\omega_0)$ !

4 In a manner similar to [Example freq.bode-1](#), we construct Bode plots for several simple transfer functions in this lecture. Once we have

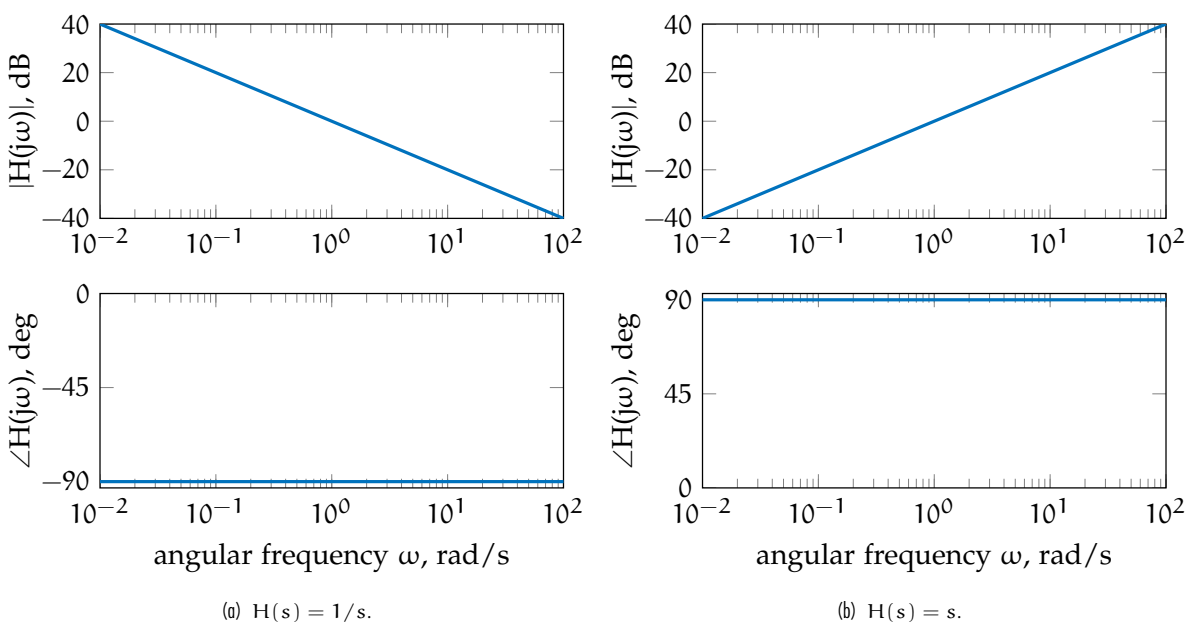
these simple “building blocks,” we will be able to construct sketches of higher-order systems by graphical addition because logarithmic magnitudes and phases combine by summation, as shown in [Lec. freq.bodesketch](#).

### Constant gain

5 For a transfer function that is simply a constant real gain  $H(s) = K$ , the frequency response function is trivially  $H(j\omega) = K$ . Its magnitude  $|H(j\omega)| = |K|$ . For positive gain  $K$ , the phase is  $\angle H(j\omega) = 0$ , and for negative  $K$ , the phase is  $\angle H(j\omega) = 180$  deg.

### Pole and zero at the origin

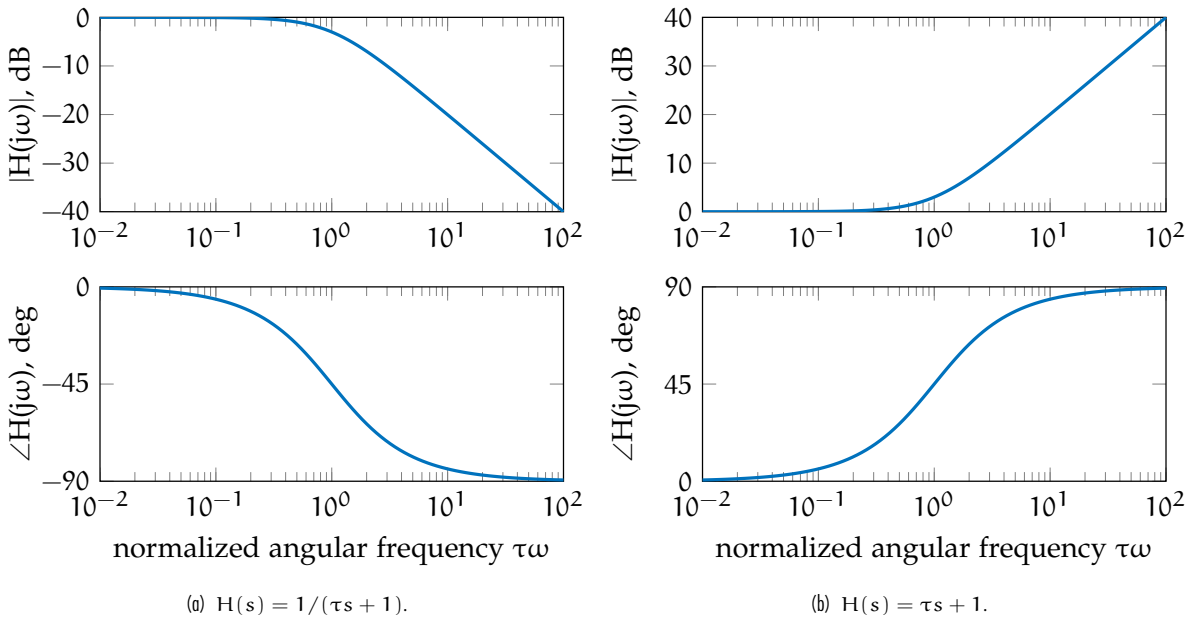
6 In [Example freq.bode-1](#), we have already demonstrated how to derive from the transfer function  $H(s) = s$ , a zero at the origin, the frequency response function plotted in [Fig. bodesimp.1](#). Similarly, for  $H(s) = 1/s$ , a pole at the origin, the frequency response function plotted in [Fig. bodesimp.1](#).



**Figure bodesimp.1:** Bode plots for (a) a pole at the origin and (b) a zero at the origin.

Real pole and real zero

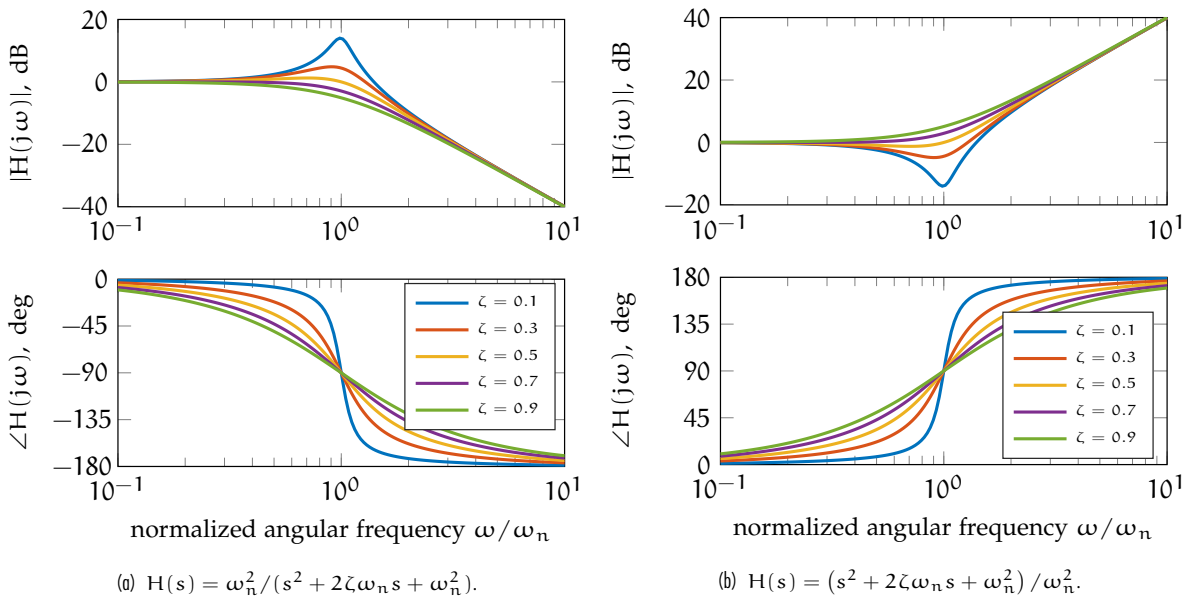
7 The derivations for real poles and zeros are not included, but the resulting Bode plots are shown in Fig. bodesimp.2.



**Figure bodesimp.2:** Bode plots for (a) a single real pole and (b) a single real zero.

Complex conjugate pole pairs and zero pairs

8 The derivations for complex conjugate pole pairs and zero pairs are not included, but the resulting Bode plots are shown in Fig. bodesimp.3.



**Figure bodesimp.3:** Bode plots for (a) a complex conjugate pole pair and (b) a complex conjugate zero pair.

## freq.bodesketch Sketching Bode plots

1 We can use MATLAB's `bode` command to create Bode plots from LTI system models. However, we must understand how these plots relate to their transfer functions. In this section, we learn to sketch Bode plots in order to deepen our intuition of this relationship.

2 Let  $H(s) = \prod_i H_i(s)$ ; that is, let  $H(s)$  be the product of several factors  $H_i(s)$ . The magnitude and phase are

$$|H(s)| = \prod_i |H_i(s)| \quad \text{and} \quad \angle H(s) = \sum_i \angle H_i(s). \quad (1)$$

The Bode plot consists of plots of  $20 \log_{10}|H(s)|$  and  $\angle H(s)$  with  $s \mapsto j\omega$ . The magnitude and phase expressions, become

$$20 \log_{10}|H(j\omega)| = \sum_i 20 \log_{10}|H_i(j\omega)| \quad \text{and} \quad \angle H(j\omega) = \sum_i \angle H_i(j\omega). \quad (2)$$

This result means we can graphically sum both the magnitude and phase Bode plots of the individual factors of  $H(s)$ , as long as we are adding magnitudes in dB.

### Example freq.bodesketch-1

re: a transfer function under analysis

Given the transfer function

$$H(s) = \frac{200000(s+1)}{s^3 + 110s^2 + 11000s + 100000}$$

answer the following questions and imperatives.

- Sketch a Bode plot on [Fig. bodesketch.1](#).
- Confirm the accuracy of the sketch in Matlab, using the functions `bode` and `tf`.
- If the input to a system with this transfer function is  $5 \sin(\omega t + \pi/7)$ , what is the output amplitude and phase for

- i  $\omega = 1$  rad/s,
- ii  $\omega = 10$  rad/s, and
- iii  $\omega = 1000$  rad/s?

Use Matlab's function `evalfr` to perform the calculations.

a

To sketch the transfer function, we must decompose the transfer function into multiple simple factors. First, we can find the poles:

$$-10, -50 + j86.6, -50 - j86.6,$$

which tells us we have a complex conjugate pair and a single real pole. Factoring, accordingly,

$$\begin{aligned} H(s) &= 200000(s + 1) \cdot \frac{1}{s + 1} \cdot \frac{1}{s^2 + 100s + 10000} \\ &= 2(s + 1) \cdot \frac{1}{s/10 + 1} \cdot \frac{100^2}{s^2 + 2 \cdot 0.5 \cdot 100s + 100^2} \end{aligned}$$

The sketch is shown in Fig. [bodesketch.1](#).

b

See the code listing below.

```
sys = 2e5*...
tf(...
    [1,1],...
    [1,110,11000,1e5]...
);
bode(sys);
```

c

The output amplitude is always  $5|H(j\omega)|$  and output phase is always  $\pi/7 + \angle H(j\omega)$ . We could estimate them from the Bode plot sketch, but we instead choose to evaluate the Matlab transfer function, as in the listing below.

```
in_amp = 5;
in_phase = pi/7; % rad
omega_a = [1,10,1e3]; % rad/s
for i = 1:length(omega_a)
```

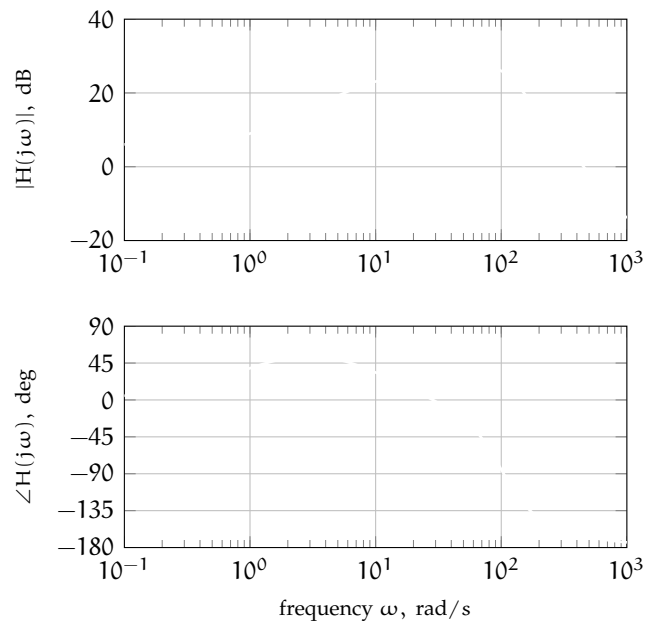


Figure bodesketch.1: a Bode plot for Example [freq.bodesketch-1](#).

```
H_eval = evalfr(sys,j*omega_a(i));
H_mag = abs(H_eval);
H_phase = angle(H_eval);
out_amp = 5*H_mag;
out_phase = in_phase + H_phase;
sprintf(...
    ['For input angular freq %0.2g,\n',...
      ' input amplitude %0.2g,\n',...
      ' input phase %0.2g,\n',...
      ' H magnitude %0.2g, and\n',...
      ' H phase %0.2g,\n',...
      ' the output amplitude is %0.2g and\n',...
      ' the output phase is %0.2g.\n'...
    ],...
    omega_a(i),...
    in_amp,...
    in_phase,...
    H_mag,...
    H_phase,...
    out_amp,...
    out_phase...
  )
end
```

The output amplitudes are 14, 71, and 1 and the output phases are 1.1, 1,  $-2.6$  rad.

## freq.nyquist Nyquist criterion

### Introduction

Consider a feedback control system. Let  $G(s)$  be the forward-loop transfer function and let  $H(s)$  be the feedback transfer function. The Nyquist plot is a parametric plot of the frequency response function  $G(j\omega)H(j\omega)$  of the open-loop transfer function  $G(s)H(s)$ .

The Nyquist criterion allows us to gain insight about closed-loop stability from the open-loop frequency response (Nyquist and Bode plots) and open-loop pole location. Additionally, insight into transient response and steady-state error response characteristics can be determined from Nyquist plots. In this sense, the Nyquist plot is analogous to the root-locus plot.

### A description of the Nyquist criterion

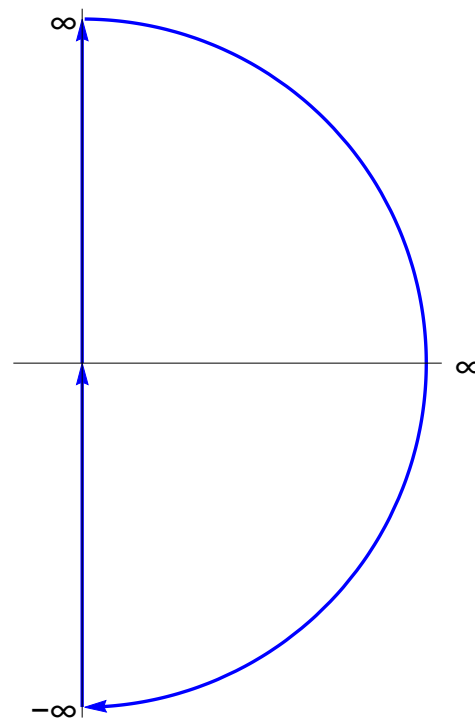
A rigorous derivation of the Nyquist criterion is beyond the scope of this work. However, a motivating description is included. Before we begin, please review complex functions, as described in [Appendix A.01](#).

The full Nyquist plot is the mapping of a contour  $\Gamma_N$  that contains the right-half plane and is defined as beginning at the origin, moving vertically along the  $j\omega$ -axis “to infinity,” encircling the right-half-plane with a semicircle “to negative infinity,” and returning vertically to the origin, as shown in [Fig. nyquist.1](#).

The Nyquist plot is now defined, but what remains is describing the Nyquist criterion and why it works. There are several important insights required to understand it.

### encirclements of the origin give us a clue

It turns out that whenever we map  $\Gamma_N$  with a transfer function  $F$ , we find out a relationship between the number of poles



**Figure nyquist.1:** contour  $\Gamma_N$  to be mapped by transfer function.



$P$  inside  $\Gamma_N$ , the number of zeros  $Z$  inside  $\Gamma_N$ , and the number of counterclockwise encirclements  $N$  of the origin by the contour  $F(\Gamma_N)$ . The primary insight is that when a pole or zero is encircled by  $\Gamma_N$ , it contributes an entire  $\pm 2\pi$  in phase around the contour, whereas when a pole or zero is not encircled by  $\Gamma_N$ , its net contribution to phase is zero. This yields the following relationship:

$$N = P - Z. \quad (1)$$

### the open-loop transfer function mapping is close to what we want

We know the poles and zeros of the open-loop transfer function  $G(s)H(s)$ . We want to know information about the closed-loop pole locations. The closed-loop transfer function (without compensation) is

$$T(s) = \frac{G(s)}{1 + G(s)H(s)}. \quad (2)$$

Let's rewrite  $G(s)$  and  $H(s)$  in terms of numerators and denominators, as follows:

$$G(s) = \frac{G_n(s)}{G_d(s)} \quad \text{and} \quad H(s) = \frac{H_n(s)}{H_d(s)}. \quad (3)$$

Let's see what our closed-loop transfer function looks like now:

$$T(s) = \frac{G_n(s)H_d(s)}{G_d(s)H_d(s) + G_n(s)H_n(s)}. \quad (4)$$

Finally, let's consider the denominator Eq. 2 for a moment:

$$1 + G(s)H(s) = \frac{G_d(s)H_d(s) + G_n(s)H_n(s)}{G_d(s)H_d(s)}, \quad (5)$$

which, combined with Eq. 3 and Eq. 4, allows us to see two important observations:

1. the poles of  $1 + G(s)H(s)$  equal the poles of  $G(s)H(s)$  and

2. the zeros of  $1 + G(s)H(s)$  equal the poles of  $T(s)$ .

We are so close! We know the poles of  $G(s)H(s)$ , therefore we know the poles of  $1 + G(s)H(s)$ . We want to know the poles of  $T(s)$ , which are related to the zeros of  $1 + G(s)H(s)$ , which we don't have, but we have something related: the open loop transfer function mapping  $G(\Gamma_N)H(\Gamma_N)$ .

**a sidestep for all the money** What if we just map with the open-loop transfer function  $G(\Gamma_N)H(\Gamma_N)$ ? That gives us almost exactly the same image as  $1 + G(\Gamma_N)H(\Gamma_N)$ , but shifted one unit to the left. This means that if we plot  $G(\Gamma_N)H(\Gamma_N)$  and interpret it as  $1 + G(\Gamma_N)H(\Gamma_N)$ , we can determine stability of the closed loop transfer function! Let's redefine our N-P-Z relationships for the mapping  $G(\Gamma_N)H(\Gamma_N)$ .

1. Let  $N$  be the number of counterclockwise encirclements of  $-1$ .
2. Let  $P$  be the number of open-loop poles in the right-half plane.
3. Let  $Z$  be the number of closed-loop poles in the right-half plane.

If we have a plot of  $G(\Gamma_N)H(\Gamma_N)$ , we have the first two and the third is given by the Nyquist criterion:

$$Z = P - N. \quad (6)$$

We will use this to determine stability in [Lec. freq.nystab](#). However, even now, we know that the existence of right-half-plane closed-loop poles implies closed-loop instability, so we can already identify that much. Before exploring stability further, we will learn to sketch the Nyquist plot. Not because we don't have MATLAB, but rather to gain intuition.

Sketching Nyquist plots

We now begin sketching Nyquist plots. Remember that we are first-of-all interested in the number of counterclockwise encirclements of  $-1$ , which will help us determine stability via the Nyquist criterion. We proceed by example.

**Example freq.nyquist-1**

Let an open-loop transfer function be defined by

$$G(s)H(s) = \frac{30}{(s + 4)(s + 7)}$$

Sketch its Nyquist plot and apply the Nyquist criterion to determine the number of closed-loop poles in the right-half plane.

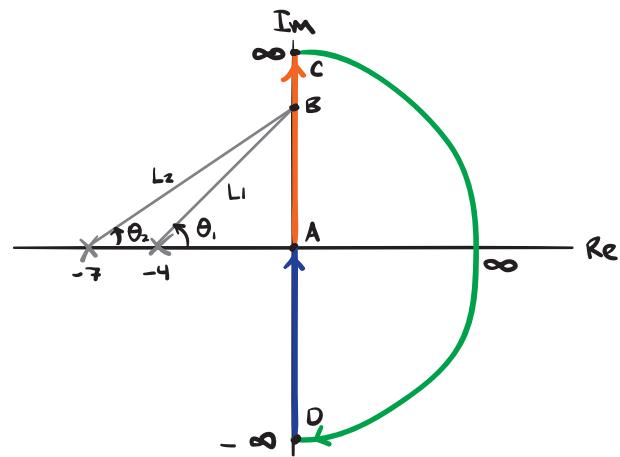
Let's sketch the contour  $\Gamma_N$  in Fig. nyquist.2. So the magnitude and phase of the mapped contour are



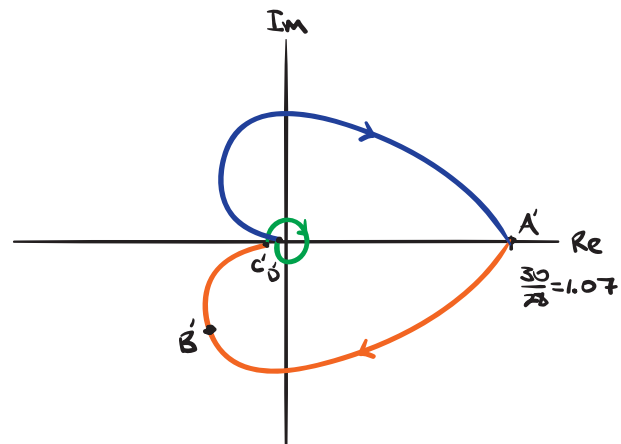
We begin at point A and map the orange contour, which is the positive  $j\omega$ -axis. At A,  $\theta_1 = \theta_2 = 0$ , so  $\angle G(A)H(A) = 0$ , and  $L_1 = 4$  and  $L_2 = 7$ , so  $|G(A)H(A)| = 30/28 \approx 1.07$ . In the Fig. nyquist.3, we sketch  $G(\Gamma_N)H(\Gamma_N)$  with point  $A' = G(A)H(A)$ . As we move to B on the orange contour, the angle becomes increasingly negative and the magnitude decreases. Finally, at C, the angle approaches  $-180$  deg and the magnitude approaches 0. Note that in the sketch we don't go quite to zero because we want to leave space to represent what occurs at zero.

What occurs at zero is that the green contour "at infinity" is mapped. The angle changes from  $+180$  deg to  $-180$  deg and the magnitude

**re: a stable open-loop system**



**Figure nyquist.2:**



**Figure nyquist.3:**

stays at 0. We sketch this by showing a 360 deg rotation back to  $+180 \text{ deg} = -180 \text{ deg}$  at  $C'$ . This doesn't always happen. Sometimes the angle with which the origin is approached is different than the angle with which it leaves. In this case, the blue contour exits at 180 deg with increasing amplitude, only to "mirror" the orange contour's return to  $A'$ . This does always occur: The Nyquist plot is always symmetric about the real axis and the  $j\omega$ -axis image is essentially a mirroring of the  $-j\omega$ -axis image. Examining the Nyquist plot sketch, there are no counterclockwise encirclements of  $-1$ , i.e.  $N = 0$ . The open-loop transfer function has no poles in the right-half-plane, i.e.  $P = 0$ . Therefore, from the Nyquist criterion,



So there are no closed-loop in the right-half-plane and the closed-loop system is stable.

What if there's an open-loop pole on the contour  $\Gamma_N$ ? The magnitude of the contour  $G(\Gamma_N)H(\Gamma_N)$  becomes infinite, but we cannot determine at which phase it does so. Therefore, in these cases we take an infinitesimal detour around the pole so that we can keep track of the phase. The magnitude still approaches infinity, but the phase information is retained. Let's consider another example that illustrates this.

### Example freq.nyquist-2

re: a system with open-loop poles on the Nyquist contour

Let an open-loop transfer function be defined by

$$G(s)H(s) = \frac{10(s+1)}{s^2+1}.$$

- Sketch its Nyquist plot and apply the Nyquist criterion to determine the number of closed-

- loop poles in the right-half plane.

Let's sketch the contour  $\Gamma_N$  in Fig. nyquist.4.

So the magnitude and phase of the mapped contour are



We begin at point A and map the orange contour, which is the positive  $j\omega$ -axis. At A,  $\theta_1 = 0$  and  $\theta_2 = -\theta_3$ , so  $\angle G(A)H(A) = 0$ , and  $L_1 = L_2 = L_3 = 1$ , so  $|G(A)H(A)| = 10$ . In Fig. nyquist.5, we sketch  $G(\Gamma_N)H(\Gamma_N)$  with point  $A' = G(A)H(A)$ . As we move to B on the orange contour,  $\theta_1 \rightarrow +45$  deg and still  $\theta_2 = -\theta_3$ , so  $\angle G(A)H(A) \rightarrow +45$  deg. But the magnitude approaches infinity because  $L_2 \rightarrow 0$ . The infinitesimal detour from B to C doesn't change the magnitude, but it does change the phase by  $-180$  deg. Finally, from C to D the only angle that changes is  $\theta_1$  by  $+45$  deg, which yields  $\angle G(A)H(A) \rightarrow -90$  deg as the magnitude approaches zero due to the denominator of the magnitude approaching infinity faster than the numerator. What occurs at zero is that the green contour "at infinity" is mapped. The angle changes from  $-90$  deg to  $+90$  deg and the magnitude stays at 0. We sketch this by showing a  $180$  deg rotation back to  $+90$  deg at C'. The blue contour exits at  $+90$  deg with increasing amplitude, only to "mirror" the orange contour's return to A'.

Examining the Nyquist plot sketch, there are no counterclockwise encirclements of  $-1$ , i.e.  $N = 0$ . The open-loop transfer function has no poles in the right-half-plane, i.e.  $P = 0$ . Therefore, from the Nyquist criterion,

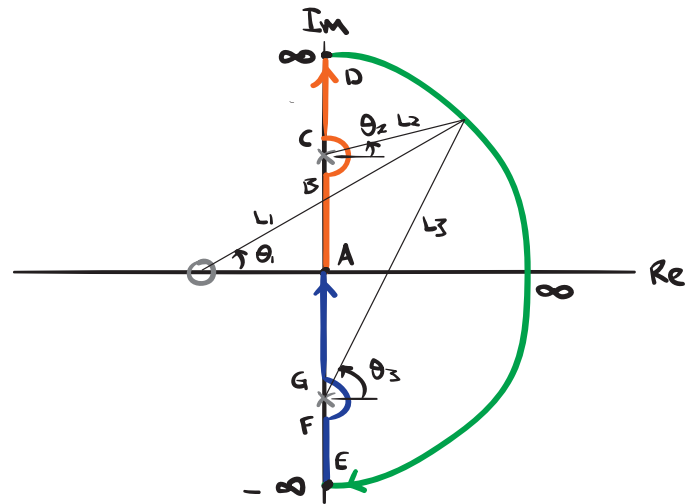


Figure nyquist.4:



Figure nyquist.5:



So there are no closed-loop in the right-half-plane and the closed-loop system is stable.

## freq.nystab Stability from the Nyquist plot

The Nyquist stability criterion has established closed-loop stability for unity controller gain. Of course, we would like to learn about the stability for any value of gain  $K$  and closed-loop transfer function

$$T(s) = \frac{G(s)}{1 + KG(s)H(s)}. \quad (1)$$

The Nyquist plot is of  $G(\Gamma_N)H(\Gamma_N)$ , and if we include the gain it becomes  $KG(\Gamma_N)H(\Gamma_N)$ . This simply scales the magnitude by  $K$ , which “stretches” the image by a factor of  $K$ . Recall that, for stability, we are concerned with encirclements of  $-1$ . So scaling  $K$  can change the number of encirclements.

For instance, let’s consider a system with open-loop transfer function

$$G(s)H(s) = \frac{(s-1)(s-2)}{(s+3)(s+5)}. \quad (2)$$

This system has  $P = 0$  open-loop poles in the right-half-plane, thus for no encirclements of  $-1$ , the closed-loop system is stable by the Nyquist criterion. Fig. nystab.1 shows the Nyquist plot for three different values of gain  $K$ :

- ( $K = 1.00$ )** this is the Nyquist plot we have thus far considered, and  $N = 0$ , so  $Z = 0$  and the closed-loop system is stable;
- ( $K = 4.00$ )** this stretches the previous plot by a factor of 4,  $N = -2$ , so  $Z = +2$  and the closed-loop system is unstable;
- ( $K = 2.67$ )** scales the original plot such that it intersects  $-1$ , at which point the system is marginally stable.

Stability from the positive  $j\omega$ -axis image, alone

It turns out we can determine stability from the image of the positive  $j\omega$ -axis, alone. Due to the

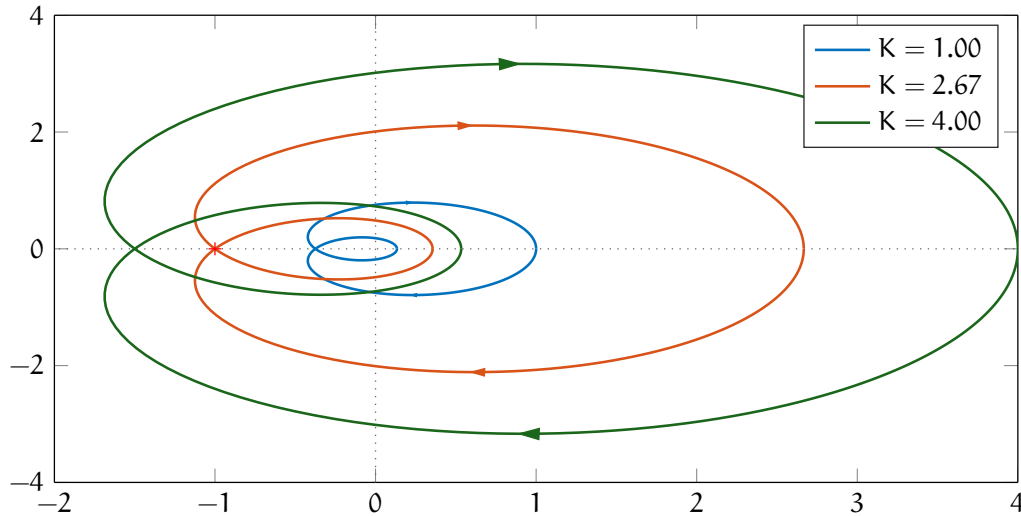


Figure nystab.1:

real-axis symmetry of the poles and zeros of transfer functions, the image of the negative  $j\omega$ -axis is simply a reflection about the real axis of the image of the positive  $j\omega$ -axis. Nothing that occurs at zero magnitude affects stability. The same is true for that which occurs at infinite magnitude, which never intersects the negative real axis.<sup>1</sup>

1. This is true for poles and zeros on the  $j\omega$  axis as long as we draw the infinitesimal detour into the right-half-plane, as we are accustomed to doing.

Gain margin and phase margin

Let us consider a system with a Nyquist plot for which smaller values of gain  $K$  yield a stable closed-loop system and higher values of  $K$  that yield one that is unstable. It turns out that these types of systems are very common. We know that the key to stability in such systems is the number of encirclements of  $-1$ . We are now ready to define two key quantities, the gain margin and the phase margin. For a given gain  $K$ , these parameters quantify “how stable” a given system is. The gain margin  $G_M$  is a logarithm of the distance from  $-1$  to the Nyquist plot’s negative real-axis intercept  $-1/a$ , as shown in Fig. nystab.2. Specifically, in dB

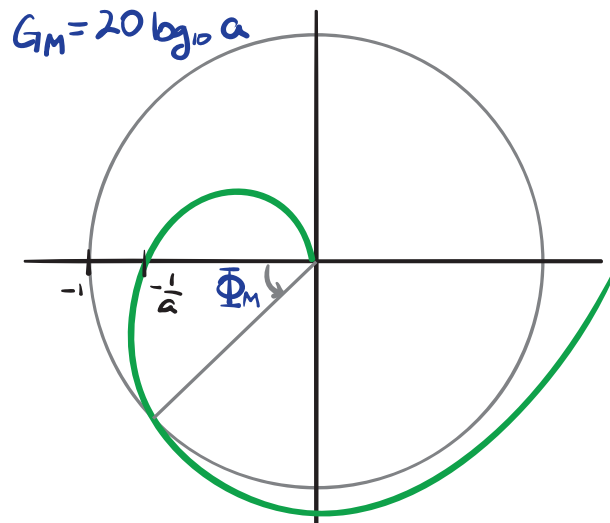


Figure nystab.2:



(the typical manner of expressing  $G_M$ ),

$$G_M = 20 \log_{10} a. \quad (3)$$

The phase margin  $\Phi_M$  is defined as the angle at which the magnitude is unity as the Nyquist plot approaches its negative real-axis intercept  $-1/a$ . This is typically difficult to find, mathematically, from a Nyquist plot. Primarily for this reason, in a moment we will switch to a Bode plot representation, in which both  $G_M$  and  $\Phi_M$  are easily found.

As already mentioned,  $G_M$  and  $\Phi_M$  can be considered measures of stability. We can consider this to mean that higher  $G_M$  and  $\Phi_M$  correspond to greater confidence that the closed-loop system will remain stable, even under small changes to its system parameters.

## freq.nybode Stability, GM, and PM from Bode plots

Bode plots are an alternative representation of the positive  $j\omega$ -axis Nyquist plot. As we established in [Lec. freq.nystab](#), this is sufficient information for stability via the Nyquist criterion. In a similar fashion, the gain and phase margins can also be found from the positive  $j\omega$ -axis Nyquist plot.

For these reasons, the stability, gain margin, and phase margin can all be found from the Bode plot. In fact, this is the preferred method for finding the gain and phase margins.

It is common, but somewhat risky, practice to simply use the Bode plot to determine stability, gain margin, and phase margin. Here is why it is risky: when we use it, we assume that the system

1. is open-loop stable;
2. with sufficient gain, has only clockwise encirclements of  $-1$ ; and
3. has a single negative-real-axis crossing.

Although these are all commonly met, there are plenty of systems for which they are not. For this reason, we encourage caution with this common practice. We proceed by describing the method.

Recall that the gain margin  $G_M$  is defined by the distance between the negative-real-axis intercept of a Nyquist plot and  $-1$ . This occurs at  $-180$  deg. On a Bode plot, such as that of [Fig. nybode.1](#), it is easy to determine the dB magnitude difference from the magnitude at  $-180$  deg and  $1 = 0$  dB.

Similarly, recall that the phase margin  $\Phi_M$  is defined as the difference between the angle at magnitude  $1 = 0$  dB and  $-180$  deg as the Nyquist plot approaches  $-180$  deg. On a Bode plot, such as that of [Fig. nybode.1](#), the magnitude  $0$  dB

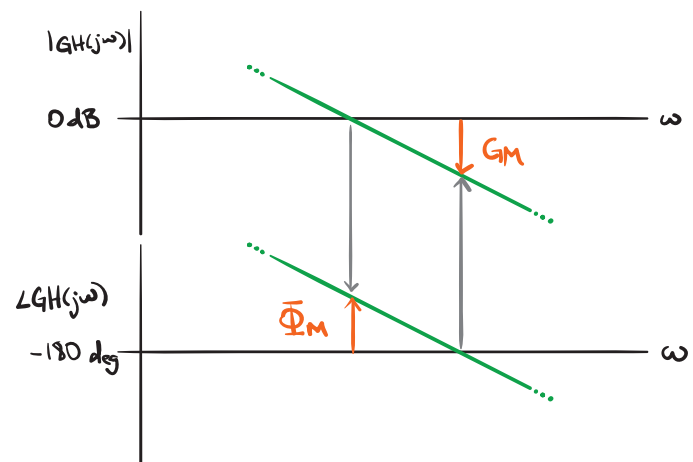


Figure nybode.1:

point near  $-180$  deg corresponds to the phase from which  $\Phi_M$  can be determined.

### Example freq.nybode-1

### re: Gain and phase margin from a Bode plot

Let the open-loop transfer function  $GH(s)$  be defined as

$$GH(s) = \frac{1}{s^3 + 2s^2 + 5s + 6}$$

Determine closed-loop stability, gain margin, and phase margin from an open-loop Bode plot.

## freq.freqtime Relations among time and frequency domain reps

The second-order assumption

As in root locus design, our transient response characteristics—such as percent overshoot %OS, settling time  $T_s$ , and peak time  $T_p$ —can be related exactly to second-order response characteristics  $\zeta$  and  $\omega_n$ , which have their own interpretations in the frequency domain and are related to key features of the Bode plot. This often gets us close enough that small iterations on the initial design can achieve the desired transient response.

The second-order approximation assumes an open-loop transfer function of the form

$$G(s) = \frac{\omega_n^2}{s(s + 2\zeta\omega_n)} \quad (1)$$

which yields a closed-loop transfer function

$$T(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad (2)$$

which has a familiar frequency response.

Bandwidth

The term bandwidth appears in many contexts, but in control theory when using the second-order assumption it has a very specific definition.

**Definition freq.1: bandwidth**

Let a system have a transfer function  $G(s)$  and frequency response function  $G(j\omega)$ . The bandwidth  $\omega_{BW}$  of the system is the angular frequency at which  $|G(j\omega)|$  is 3 dB less than  $|G(j0)|$ .

Closed-loop percent overshoot from the closed-loop bandwidth

It is straightforward to show that the bandwidth of the second-order closed-loop transfer of Eq. 2

is related to its natural frequency  $\omega_n$  and damping ratio  $\zeta$  by the expression

$$\omega_{BW} = \omega_n \left( (1 - 2\zeta^2) + (4\zeta^4 - 4\zeta^2 + 2)^{1/2} \right)^{1/2}. \quad (3)$$

So if a system behaves approximately like this second-order system, Eq. 3 relates the closed-loop frequency response characteristic  $\omega_{BW}$  and the closed-loop time response characteristics  $\omega_n$  and  $\zeta$ . This is a big step, but we often design the speed of response in terms of settling time  $T_s$ , peak time  $T_p$ , and rise time  $T_r$ . We already have relationships for these quantities and  $\omega_n$  and  $\zeta$ , the consequences of two of which when applied to Eq. 3 are shown below:

$$\omega_{BW} = \frac{4}{T_s \zeta} \left( (1 - 2\zeta^2) + (4\zeta^4 - 4\zeta^2 + 2)^{1/2} \right)^{1/2} \quad (4a)$$

$$\omega_{BW} = \frac{\pi}{T_p \sqrt{1 - \zeta^2}} \left( (1 - 2\zeta^2) + (4\zeta^4 - 4\zeta^2 + 2)^{1/2} \right)^{1/2}. \quad (4b)$$

Furthermore, it can be shown that, when it exists (which it does for  $0 < \zeta < 1/\sqrt{2}$ ), the peak magnitude  $M_p = \max_{\omega} |H(j\omega)|$  is

$$M_p = \frac{1}{2\zeta \sqrt{1 - \zeta^2}}. \quad (5)$$

Of course, percent overshoot %OS is directly related to  $\zeta$  by the equations

$$\%OS = 100 \exp \frac{-\zeta\pi}{\sqrt{1 - \zeta^2}} \iff \zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}} \quad (6)$$

so  $M_p$  can be directly related to %OS.

Closed-loop percent overshoot and damping ratio from the open-loop phase margin

From closed-loop considerations of the frequency response, we have learned to

determine some closed-loop time response characteristics. Now we learn to determine one of these characteristics—percent overshoot %OS—from open-loop frequency response. From the transfer function of Eq. 1, it is straightforward to relate the phase margin  $\Phi_M$  of the open-loop transfer function to the damping ratio  $\zeta$  via the expressions

$$\Phi_M = \arctan \frac{2\zeta}{\sqrt{-2\zeta^2 + \sqrt{1 + 4\zeta^4}}} \iff (7)$$

$$\zeta = \frac{\tan \Phi_M}{2(1 + \tan^2 \Phi_M)^{1/4}}. (8)$$

As we know from Eq. 6, percent overshoot %OS is directly related to  $\zeta$ , so  $\Phi_M$  can be directly related to %OS, as shown in Fig. freqtime.1.

Closed-loop settling and peak times from the open-loop frequency response

We introduced the concept of bandwidth in above and related the closed-loop bandwidth  $\omega_{BW}$  to settling time  $T_s$  and peak time  $T_p$  in Eq. 4. There is a method, which we present but leave underived, that allows us to find the closed-loop bandwidth of many systems from the open-loop frequency response, allowing us to relate the open-loop frequency response to  $T_s$  and  $T_p$ . The method is based on the following insight: the closed-loop bandwidth is approximately equal to the frequency at which the magnitude of the open-loop frequency response is in the interval  $[-6, -7.5]$  dB if the phase of the open-loop frequency response is in the interval  $[-135, -225]$  deg.

This gives us a method to approximate closed-loop  $T_s$  and  $T_p$  by inspecting the open-loop frequency response. Here's the method:

1. estimate the closed-loop bandwidth  $\omega_{BW}$  by finding the frequency at which the

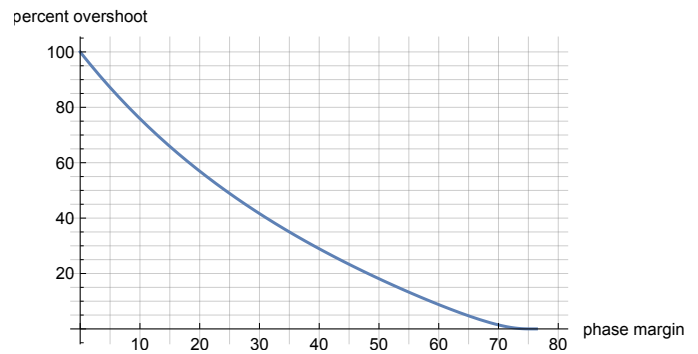


Figure freqtime.1: percent overshoot %OS versus phase margin  $\Phi_M$ .

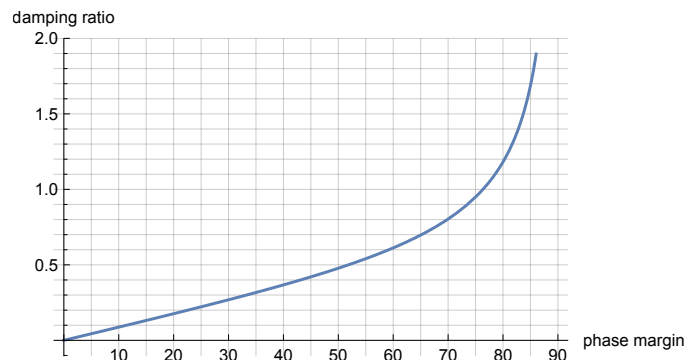


Figure freqtime.2: damping ratio  $\zeta$  versus phase margin  $\Phi_M$ .

- magnitude of the open-loop frequency response is in the interval  $[-6, -7.5]$  dB;
2. verify that open-loop phase at  $\omega_{BW}$  is in the interval  $[-135, -225]$  deg;
  3. determine  $\zeta$  via the phase margin (Eq. 7, Fig. freqtime.2); and
  4. estimate  $T_s$  and  $T_p$  via Eq. 4.

## **freq.exe** Exercises for Chapter freq



## Frequency response design

The analytical techniques of [Chapter freq](#) allow us to develop design procedures for controllers in the frequency domain. There are few advantages to frequency response design over root locus design. Frequency response design proceeds primarily from the easily-sketched bode plot, which makes it easier to execute by-hand than root locus design. One clear advantage is that, when designing a lead controller, it is possible with frequency response design to not only tune %OS and response time, but desirable steady-state error can also be achieved, something for which root locus design is not conducive.

## freqd.gain Transient response design by adjusting the gain

The following design procedure allows us to design for a desired percent overshoot. A similar procedure could be followed to design for a desired damping ratio.

1. Generate open-loop Bode plots with some convenient initial gain  $K_i$ .
2. Use either Fig. [freqtime.1](#) or Eq. 6 and Eq. 7 to find the desired phase margin  $\Phi_M$ .
3. From the Bode phase plot, determine the frequency  $\omega_{\Phi_M}$  at which (180 deg minus the absolute value of) the phase is equal to the desired phase margin.
4. Change the gain to be such that the magnitude plot would intersect 0 dB at  $\omega_{\Phi_M}$ .

### Example freqd.gain–1

Design a unity feedback gain controller for a system with the plant

$$G(s) = \frac{10}{(s + 90)(s + 30)}$$

such that the percent overshoot %OS is approximately 20%.

re: Percent overshoot design by adjusting the gain

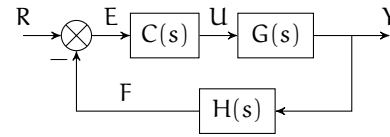
## freqd.exe Exercises for Chapter freqd

Exercise freqd.libricide

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

$$G(s) = \frac{1}{s^3 + 11s^2 + 39s + 29}. \quad (1)$$

1. Use frequency response methods to design a gain controller such that the unity feedback closed-loop overshoot is about 10%.
2. Demonstrate the controller performance by simulating and plotting a step response.
3. Compute the simulated overshoot (via the step response).



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

## State–space design

Root-locus and frequency-domain design techniques have their own strengths, but they cannot be applied to broad classes of systems, including nonlinear systems and multiple-input multiple-output (MIMO) systems. Moreover, these techniques can only attempt to specify the locations of the dominant closed-loop poles. Therefore, those systems for which higher-order poles significantly affect the response do not respond well to these techniques.

These are some of the reasons why “modern” control theory uses state-space design techniques. Drawbacks of these techniques are limited, but include that less insight can be gained from them and that closed-loop zeros cannot be specified directly.

For an  $n$ -th order plant,  $n$  poles must be placed. This requires  $n$  adjustable parameters, which are gains. We will learn a technique in this chapter for placing these  $n$  poles with  $n$  gains for a certain class of systems.

## ss.sfdbck Controller design method

We will consider single-input single-output (SISO) control plants that can be written with input  $u$ ; state vector  $x$ ; output  $y$ ; state model matrices  $A$ ,  $B$ ,  $C$ , and  $D$ ; and state and output equations

$$\dot{x} = Ax + Bu \tag{1a}$$

$$y = Cx + Du. \tag{1b}$$

Plants of this form can be written in block diagram form, as illustrated in Fig. sfdbck.1. In general, SISO systems are of order  $n$  with  $n$  state variables.

Let us consider the following feedback control method called state feedback control. We will feed back the state vector  $x$ , operate on it with a  $1 \times n$  vector of gains  $K \in \mathbb{R}^n$ , and subtract the result from the command  $r$ , the result of which becomes the input  $u$ , as shown in Fig. sfdbck.2. The control problem for state feedback control is to determine the  $n$  gains in  $K$  such that the closed-loop poles are located in desirable positions. The gain  $N \in \mathbb{R}$  is provided for steady-state error considerations, which will be addressed in Lec. ss.sfdbck. A new state model can be derived for the closed-loop system as follows. Let us consider the command  $r$  to be our new “input,” instead of  $u$ , which is now the control effort. From the block diagram,

$$u = Nr - Kx, \tag{2}$$

which can be substituted into Eq. 1 to define the new state model

$$\dot{x} = (A - BK)x + NBr \tag{3a}$$

$$y = (C - DK)x + NDr. \tag{3b}$$

The eigenvalues of  $A - BK$ , which can be found from equating zero and the closed-loop characteristic polynomial

$$P_K = \det (sI - A + BK), \tag{4}$$

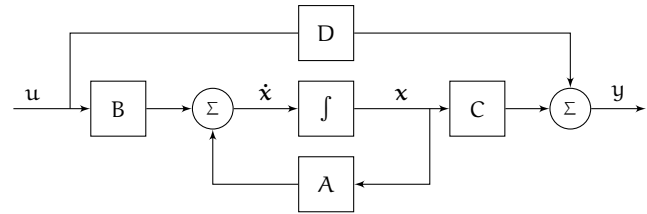


Figure sfdbck.1: the plant state model of Eq. 1 written in block diagram form.

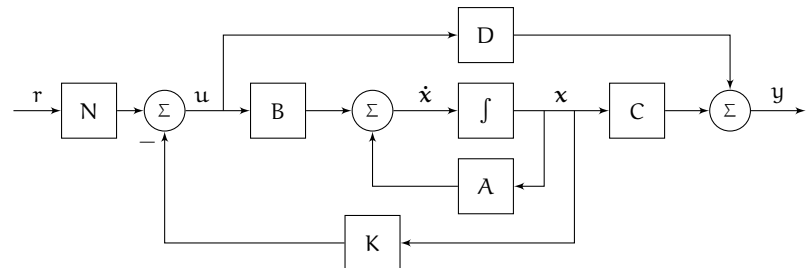


Figure sfdbck.2: the state feedback control block diagram.

are equal to the closed-loop poles, which we would like to place in specific locations. Those specific locations can be specified by the design characteristic polynomial  $P_d$ .  $P_K$  depends on the  $n$  gains  $K_i$ , and  $n$  equations can be found by equating the polynomial coefficients of  $P_K$  and  $P_d$ .

Solving for  $K_i$  is straightforward but can be very tedious in the general case. Let the coefficients of  $P_d$  be  $\delta_i$  and those of  $P_K$  be denoted  $\kappa_i$ . Then the  $n \times 1$  vector containing  $\kappa_i$  can be expressed as a linear combination of  $K_i$  as

$$\kappa = \mathcal{K} \mathbf{K}^T, \quad (5)$$

where  $\mathcal{K}$  is an  $n \times n$  matrix of coefficients that were derived from  $A$  and  $B$ . Let  $\delta$  be the  $n \times 1$  vector of components  $\delta_i$ . Since the vector  $\delta$  is specified by our design requirements, we can solve for  $\mathbf{K}$  as follows.

$$\kappa = \delta, \quad (6)$$

and therefore,

$$\begin{aligned} \mathcal{K} \mathbf{K}^T = \delta &\implies \\ \mathbf{K}^T = \mathcal{K}^{-1} \delta &\implies \\ \mathbf{K} = (\mathcal{K}^{-1} \delta)^T. &\quad (7) \end{aligned}$$

Eq. 7 is valid for all cases in which  $\mathcal{K}$  is invertible.<sup>1</sup> However, there is a special form of the original state-space model that always yields a simple solution for  $\mathbf{K}$ : the phase-variable canonical form (see [Appendix B.02](#)).

1. We leave the following as an open question: under what conditions is  $\mathcal{K}$  invertible?

Solving for the gain via the phase-variable canonical form

The phase-variable canonical form of the original system is:

$$\dot{\mathbf{x}}_c = \mathbf{A}_c \mathbf{x}_c + \mathbf{B}_c \mathbf{u} \quad (8a)$$

$$\mathbf{y} = \mathbf{C}_c \mathbf{x}_c + \mathbf{D}_c \mathbf{u} \quad (8b)$$

where

$$A_c = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{8c}$$

$$C_c = [c_1 \quad c_2 \quad \cdots \quad c_n], \text{ and} \quad D_c = [d_1], \tag{8d}$$

where the components  $a_i$  are defined by the original characteristic polynomial

$$P = \det(sI - A) = s^n + a_{n-1}s^{n-1} + \cdots + a_1s + a_0. \tag{9}$$

With  $A_c$  defined, the form of the feedback state model with feedback row vector  $K_c$  is:

$$A'_c = A_c - B_c K_c, \quad B'_c = B_c, \tag{10a}$$

$$C'_c = C_c - D_c K_c, \text{ and} \quad D'_c = D_c. \tag{10b}$$

$A'_c$  deserves further attention. The special canonical form of  $A_c$  and  $B_c$  makes the expression for  $A'_c$  simply

$$A'_c = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & 0 \\ 0 & 0 & \cdots & 1 \\ -(a_0 + K'_1) & -(a_1 + K'_2) & \cdots & -(a_{n-1} + K'_n) \end{bmatrix}, \tag{11}$$

where  $K'_i$  is the row vector of gains in the phase-variable canonical basis. The design characteristic polynomial coefficients  $\delta_i$  must equal the characteristic polynomial coefficients

$$\delta_i = a_i + K'_{i+1}, \tag{12}$$

which gives

$$K'_i = \delta_{i-1} - a_{i-1}. \tag{13}$$

This yields  $K'$ . If we equate the feedback

$$\begin{aligned} Kx &= K'x_c \implies \\ K &= K'T_c. \end{aligned} \tag{14}$$

Let  $u$  and  $u_c$  be the controllability matrices for the original basis and the phase-variable canonical basis, respectively. From [Appendix B.02](#), we can compute the transformation matrix to be

$$T_c = u_c u^{-1}. \tag{15}$$

### Steady-state error

We can use the gain  $N$  to drive the closed-loop steady-state error to zero for step inputs. The idea is that we can scale the input by the reciprocal of the closed-loop steady-state error. Let  $G_{CL}(s)$  be the closed-loop transfer function. From the final value theorem for a unit step input,

$$N = \lim_{s \rightarrow 0, N \rightarrow 1} 1/G_{CL}(s). \tag{16}$$

If  $N$  is nonzero and finite, the response will have zero steady-state error. Although it is derived from unit step inputs, we can apply this formula to slowly varying inputs as well.

### Example ss.sfdbck-1

### re: state feedback pole placement design

Given the state-space model

$$\begin{aligned} A &= \begin{bmatrix} -1 & 0 & -1 \\ -1 & -1 & 0 \\ 0 & -1 & -1 \end{bmatrix} & B &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ C &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} & D &= \begin{bmatrix} 0 \end{bmatrix}, \end{aligned}$$

- design a controller with 15% overshoot and a
- settling time of 1 sec.









## **ss.exe** Exercises for Chapter ss

A

---

## Mathematical topics

## A.01 Complex functions

A complex function  $f$  maps a subset of the complex plane to the complex plane (i.e.  $f : \mathbb{C} \rightarrow \mathbb{C}$ ). For instance, a complex function  $f$  can map a single complex number  $s_0$  to another  $s_1 = f(s_0)$ .

A curve in the complex plane is defined as a continuous function mapping a closed interval of the reals to the complex plane. A contour is defined as a directed curve consisting of a finite set of directed smooth curves, the final endpoint of which is identical to the starting point (Fig. A.01.1 shows a plot of a contour  $\Gamma$ ).

A contour can be mapped by a complex function, and this is our primary concern. The image of a contour  $\Gamma$  mapped by a complex function  $f$  is itself a contour  $f(\Gamma)$ , as shown in Fig. A.01.2.

Complex functions are of interest in control theory because transfer functions, one of the central mathematical objects of control theory, are complex functions. The utility of evaluating and mapping contours with complex functions arises especially in root-locus design and frequency response design (especially for the Nyquist stability criterion).

### Example A.01 – 1

Map the complex point  $s = 1 + j3$  with the transfer (complex) function

$$H(s) = \frac{s + 4}{s - 1}.$$

Sometimes we say that we are “evaluating” the transfer function at the point  $s = 1 + j3$ .

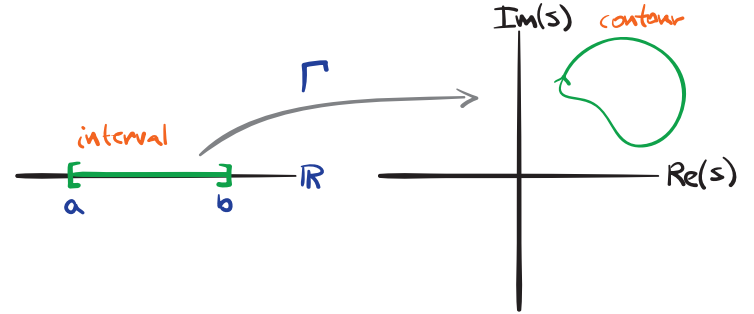


Figure A.01.1: illustrating the definition of a complex contour  $\Gamma$ .

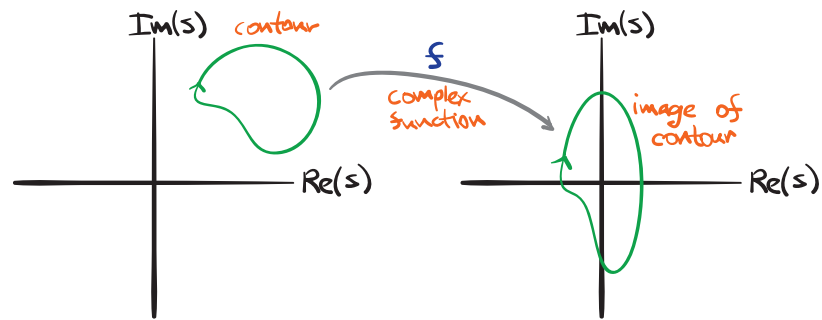


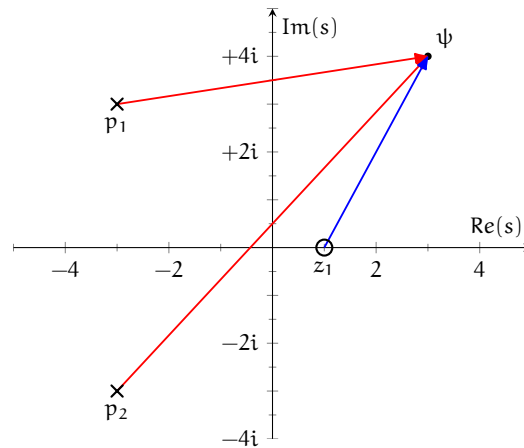
Figure A.01.2: a representation of a complex function  $f$  mapping a contour.

### re: transfer function mapping a single point

A geometric interpretation of complex functions

It is often helpful to interpret the complex mapping of a point or a contour geometrically. Let us consider a transfer (complex) function  $H(s)$  with complex zeros  $z_i$ , complex poles  $p_j$ , and real scaling factor  $k$ . Considering each factored term of the transfer function in terms of its magnitude and phase, we can write the magnitude and phase of the transfer function as follows.

**Equation 1 magnitude and phase of a transfer function**



**Figure A.01.3:** an example of a geometric interpretation of the evaluation of a complex function with poles  $p_{1,2}$  and zero  $z_1$  at a complex value  $s = \psi$ .

We can interpret this geometrically as follows. Let us consider the evaluation of Eq. 1 at a specific complex value  $\psi$ . The differences  $\psi - z_i$  and  $\psi - p_i$  can be thought of as vectors in the complex plane with tails at  $z_i$  and  $p_i$  and heads at  $\psi$ . Fig. A.01.3 shows this geometric interpretation with  $p_{1,2} = -3 \pm j3$ ,  $z_1 = 1$ , and  $\psi = 3 + j4$ .

**Example A.01 – 2**

Let  $I = [0, 2\pi]$ . Let the contour  $\Gamma : I \rightarrow \mathbb{C}$  be defined parametrically, with  $t \in I$ , as

$$\Gamma(t) = \sin t + j \cos t.$$

Map  $\Gamma$  with the transfer function

$$H(s) = \frac{s}{s^2 + 2s + 2}$$

and plot the result.

re: transfer function mapping a contour

```
1 \[CapitalGamma][t_] := {Sin[t], Cos[t]}
2 H[s_] := (s + 1)/(s^2 + 2*s + 2);
3
4 ps = {Blue, Arrowheads[{0, .05, .05, .05}]};
5 mappingcontour = Animate[
6   {
7     ParametricPlot[
8       \[CapitalGamma][t], {t, 0, T},
9       PlotRange -> {-1, 1},
10      PlotStyle -> ps,
11      PlotLabel -> "\[CapitalGamma]"
12    ] /.
13     Line -> Arrow,
14    ParametricPlot[
15      H[Complex @@ \[CapitalGamma][t]] // {Re[#], Im[#]} &,
16      {t, 0.001, T},
17      PlotRange -> {-1.5, 1.5},
18      PlotStyle -> ps,
19      PlotLabel -> "H(\[CapitalGamma])"
20    ] /.
21     Line -> Arrow
22  } // GraphicsRow,
23  {T, 0, 2*\[Pi]}
24 ]
```

**Figure A.01.5:** a basic Mathematica script for visualizing the transfer function mapping of Example A.01-2. A more thorough notebook is available [here](#).



---

## **Linear systems theory topics**

## B.01 Controllability, observability, and stabilizability

The three topics controllability, observability, and stabilizability are three topics of central concern to linear systems theory.

### Controllability

Controllability is defined as follows.

#### Definition B.1: controllable and uncontrollable

If there exists some input to a linear system such that any initial state in its state space can be evolved in finite time to any final state in its state space, the system is controllable. Otherwise, the system is uncontrollable.

A given system's controllability can be determined from the following.

#### Definition B.2: controllability matrix

Let a linear system of order  $n$  and number of inputs  $r$  have state space  $\{A, B, C, D\}$ . We define the  $n \times nr$  controllability matrix to be

$$\mathcal{U} = [B \mid AB \mid A^2B \mid \dots \mid A^{n-1}B].$$

The following well-known theorem, left unproven here, allows us to easily determine the controllability of a given system.

#### Theorem B.3: controllability

A linear system is controllable if its controllability matrix has full rank. If it is less than full rank, the linear system is uncontrollable.

## B.02 Canonical forms of the state model

There are several canonical forms for the state equations, all of which can be found via basis transformations from other forms.

### Phase-variable canonical form

The phase-variable canonical form is represented by the SISO<sup>1</sup> state model

$$\dot{\mathbf{x}}_c = \mathbf{A}_c \mathbf{x}_c + \mathbf{B}_c u \quad (1a)$$

$$y = \mathbf{C}_c \mathbf{x}_c + D_c u \quad (1b)$$

1. There are phase-variable canonical forms for MIMO systems as well, but these are less standardized.

where

$$\mathbf{A}_c = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}, \quad \mathbf{B}_c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad (1c)$$

$$\mathbf{C}_c = [c_1 \quad c_2 \quad \cdots \quad c_n], \text{ and} \quad D_c = [d_1]. \quad (1d)$$

In order to transform a SISO system  $\{A, B, C, D\}$  with state vector  $\mathbf{x}$  to phase-variable canonical form, we change bases via the substitution of  $\mathbf{x} = \mathbf{T}_c \mathbf{x}_c$  into the original system, which gives

$$\mathbf{A}_c = \mathbf{T}_c^{-1} \mathbf{A} \mathbf{T}_c, \quad \mathbf{B}_c = \mathbf{T}_c^{-1} \mathbf{B}, \quad (2a)$$

$$\mathbf{C}_c = \mathbf{C} \mathbf{T}_c, \text{ and} \quad D_c = D. \quad (2b)$$

The special form of [Equation 1](#) yields the following characteristic polynomial:

$$s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0. \quad (3)$$

Recall that eigenvalues of a system are invariant to basis change, and therefore so is its characteristic polynomial. From this we can conclude that  $\mathbf{A}_c$  can be completely determined by finding the characteristic polynomial of the

original matrix  $A$ .  $B_c$  is already fully determined, but  $C_c$  and  $D_c$  remain undetermined. They may be found by discovering the transformation matrix  $T_c$  and substituting it into [Equation 2](#).

Finding the phase-variable canonical transformation

The phase-variable canonical transformation matrix  $T_c$  can be found by relating the controllability matrices of the original form and the canonical form.

**Theorem B.4: phase-variable canonical transformation**

The transformation matrix from a system representation with controllability matrix  $\mathcal{U}$  to a phase-variable canonical transformation with controllability matrix  $\mathcal{U}_c$  is

$$T_c = \mathcal{U}_c \mathcal{U}^{-1}. \quad (4)$$

By the Definition of the controllability matrix, the original controllability matrix is

$$\mathcal{U} = [B \mid AB \mid A^2B \mid \dots \mid A^{n-1}B] \quad (5)$$

and that of the canonical form is

$$\mathcal{U}_c = [B_c \mid A_c B_c \mid A_c^2 B_c \mid \dots \mid A_c^{n-1} B_c]. \quad (6)$$

Note that  $\mathcal{U}$  and  $\mathcal{U}_c$  are both known from above. We relate the two forms by applying [Equation 2](#) to [Equation 6](#) to yield

$$\mathcal{U}_c = [T_c^{-1}B \mid T_c^{-1}AB \mid T_c^{-1}A^2B \mid \dots \mid T_c^{-1}A^{n-1}B] \quad (7a)$$

$$= T_c \mathcal{U}, \quad (7b)$$

to yield

$$T_c = \mathcal{U}_c \mathcal{U}^{-1}.$$

C

---

## Physical topics

## **C.01 Decibels**

---

## Bibliography

- Agarwal, A. and J. Lang (2005). Foundations of Analog and Digital Electronic Circuits. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science. isbn: 9780080506814.
- Authority, Tennessee Valley and Tomia (2018). Hydroelectric dam—Wikipedia, The Free Encyclopedia. [Online; accessed 13-February-2018].
- Baldursson, Stefán (2005). ?BLDC Motor Modelling and Control – A Matlab®/Simulink®Implementation? mathesis. Chalmers University.
- Booton, Richard C. and Simon Ramo (july 1984). ?The development of systems engineering? inIEEE Transactions on Aerospace and Electronic Systems: AES–20, pages 306–9.
- Brogan, William L (1991). Modern Control Theory. Third. Prentice Hall.
- Evans, W. R. (january 1948). ?Graphical Analysis of Control Systems? inTransactions of the American Institute of Electrical Engineers: 67.1, pages 547–551. issn: 0096-3860. doi: 10.1109/T-AIEE.1948.5059708.
- (january 1950). ?Control System Synthesis by Root Locus Method? inTransactions of the American Institute of Electrical Engineers: 69.1, pages 66–69. issn: 0096-3860. doi: 10.1109/T-AIEE.1950.5060121.

- Horowitz, P and W Hill (2015). The Art of Electronics. Cambridge University Press. isbn: 9780521809269.
- Hsu, H.P. (1967). Fourier Analysis. Simon & Schuster. isbn: 9780671270377.
- Nise, N.S. (2015). Control Systems Engineering, 7th Edition. Wiley. isbn: 9781118800829.
- Nise, Norman S. (2011). Control Systems Engineering. Sixth. John Wiley & Sons, Inc.
- NOAA (august 2017). Monthly Average Precipitation 1951–2008 Olympia Regional Airport—NOAA Station.
- Picone, Rico A.R. (2018). State.  
<https://github.com/ricopicone/state>.
- Rowell, Derek and David N. Wormley (1997). System Dynamics: An Introduction. Prentice Hall.