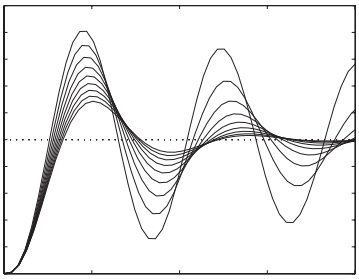


Chapter 6

PID Controller Design

PID (proportional integral derivative) control is one of the earlier control strategies [59]. Its early implementation was in pneumatic devices, followed by vacuum and solid state analog electronics, before arriving at today's digital implementation of microprocessors. It has a simple control structure which was understood by plant operators and which they found relatively easy to tune. Since many control systems using PID control have proved satisfactory, it still has a wide range of applications in industrial control. According to a survey for process control systems conducted in 1989, more than 90 of the control loops were of the PID type [60]. PID control has been an active research topic for many years;see the monographs [60–64]. Since many process plants controlled by PID controllers have similar dynamics it has been found possible to set satisfactory controller parameters from less plant information than a complete mathematical model. These techniques came about because of the desire to adjust controller parameters in situ with a minimum of effort, and also because of the possible difficulty and poor cost benefit of obtaining mathematical models. The two most popular PID techniques were the step reaction curve experiment, and a closed-loop "cycling" experiment under proportional control around the nominal operating point.

In this chapter, several useful PID-type controller design techniques will be presented, and implementation issues for the algorithms will also be discussed. In Sec. 6.1, the proportional, integral, and derivative actions are explained in detail, and some variations of the typical PID structure are also introduced. In Sec. 6.2, the well-known empirical Ziegler–Nichols tuning formula and modified versions will be covered. Approaches for identifying the equivalent first-order plus dead time model, which is essential in some of the PID controller design algorithms, will be presented. A modified Ziegler–Nichols algorithm is also given. Some other simple PID setting formulae such as the Chien–Hrones–Reswick formula, Cohen–Coon formula, refined Ziegler–Nichols tuning, Wang–Juang–Chan formula and Zhuang–Atherton optimum PID controller will be presented in Sec. 6.3. In Sec. 6.4, the PID tuning formulae for FOIPDT (first- order lag and integrator plus dead time) and IPDT (integrator plus dead time) plant models, rather than the FOPDT (first-order plus dead time) model, will be given. A graphical user interface (GUI) implementing hundreds of PID controllers tuning formulae for FOPDT model will be given in Sec. 6.5. In Sec. 6.6, an optimization-based design algorithm, together with a GUI for optimal controller design, is

given. In Sec. 6.7, some of the advanced topics on PID control will be presented, such as integrator windup phenomenon and prevention, and automatic tuning techniques. Finally, some suggestions on controller structure selections for practical process control are provided.

6.1 Introduction

6.1.1 The PID Actions

A typical structure of a PID control system is shown in Fig. 6.1, where it can be seen that in a PID controller, the error signal $e(t)$ is used to generate the proportional, integral, and derivative actions, with the resulting signals weighted and summed to form the control signal $u(t)$ applied to the plant model. A mathematical description of the PID controller is

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau) \mathrm{d}\tau + T_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \right], \tag{6.1}$$

where $u(t)$ is the input signal to the plant model, the error signal $e(t)$ is defined as $e(t) = r(t) - y(t)$, and $r(t)$ is the reference input signal.

The behavior of the proportional, integral, and derivative actions will be demonstrated individually through the following example.

Example 6.1. Consider a third-order plant model given by $G(s) = 1/(s+1)^3$. If a proportional control strategy is selected, i.e., $T_i \to \infty$ and $T_d \to 0$ in the PID control strategy, for different values of K_p, the closed-loop responses of the system can be obtained using the following MATLAB statements:

```
>> G=tf(1,[1,3,3,1]);
   for Kp=[0.1:0.1:1], G_c=feedback(Kp*G,1); step(G_c), hold on; end
   figure; rlocus(G,[0,15])
```

The closed-loop step responses are obtained as shown in Fig. 6.2(a), and it can be seen that when K_p increases, the response speed of the system increases, the overshoot of the closed-loop system increases, and the steady-state error decreases. However when K_p is large enough, the closed-loop system becomes unstable, which can be directly concluded from the root locus analysis in Sec. 3.4. The root locus of the example system is shown

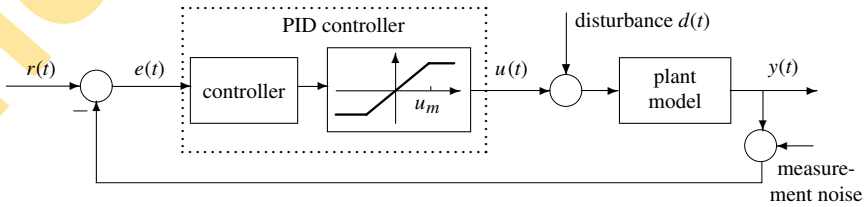
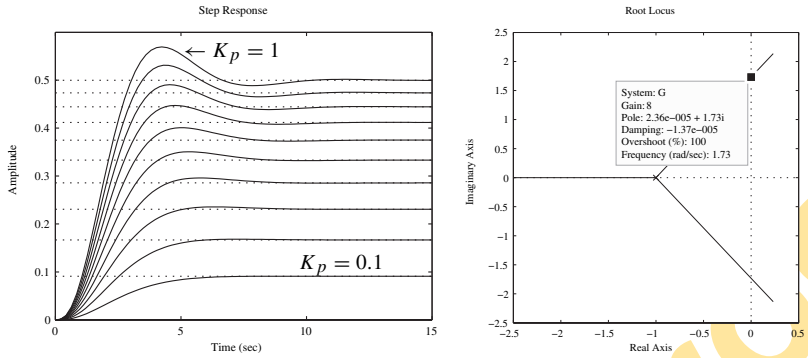
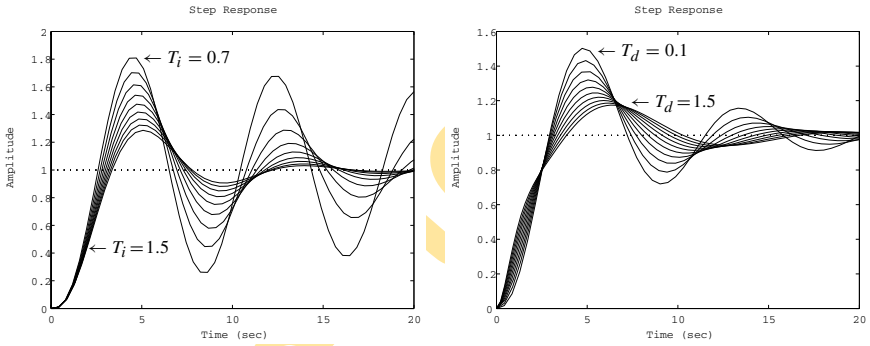


Figure 6.1. *A typical PID control structure.*



(a) closed-loop step response (b) root locus

Figure 6.2. *Closed-loop step responses.*



(a) PI control (b) PID control

Figure 6.3. *Closed-loop step responses.*

in Fig. 6.2(b), where it is seen that when K_p is outside the range of $(0, 8)$, the closed-loop system becomes unstable.

If we fix $K_p = 1$ and apply a PI (proportional plus integral) control strategy for different values of T_i, we can use the following MATLAB statements:

```
>> Kp=1; s=tf('s');
   for Ti=[0.7:0.1:1.5]
     Gc=Kp*(1+1/Ti/s); G_c=feedback(G*Gc,1); step(G_c), hold on
   end
```

to generate the closed-loop step responses of the example system shown in Fig. 6.3(a). The most important feature of a PI controller is that there is no steady-state error in the step response if the closed-loop system is stable. Further examination shows that if T_i is smaller than 0.6, the closed-loop system will not be stable. It can be seen that when T_i increases, the overshoot tends to be smaller, but the speed of response tends to be slower.

Fixing both K_p and T_i at 1, i.e., $T_i = K_p = 1$, when the PID control strategy is used, with different T_d, we can use the MATLAB statements

```
>> Kp=1; Ti=1; s=tf('s');
   for Td=[0.1:0.2:2]
      Gc=Kp*(1+1/Ti/s+Td*s); G_c=feedback(G*Gc,1); step(G_c), hold on
   end
```

to get the closed-loop step response shown in Fig. 6.3(b). Clearly, when T_d increases the response has a smaller overshoot with a slightly slower rise time but similar settling time.

In practical applications, the pure derivative action is never used, due to the "derivative kick" produced in the control signal for a step input, and to the undesirable noise amplification. It is usually replaced by a first-order low pass filter. Thus, the Laplace transformation representation of the approximate PID controller can be written as

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{sT_d}{1 + s\frac{T_d}{N}} \right) E(s). \tag{6.2}$$

The effect of N is illustrated through the following example.

Example 6.2. Consider the plant model in Example 6.1. The PID controller parameters are $K_p = 1$, $T_i = 1$, and $T_d = 1$. With different selections of N, we can use the MATLAB commands

```
>> Td=1; Gc=Kp*(1+1/Ti/s+Td*s); step(feedback(G*Gc,1)), hold on
   for N=[100,1000,10000,1:10]
      Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N)); G_c=feedback(G*Gc,1); step(G_c)
   end
   figure; [y,t]=step(G_c); err=1-y; plot(t,err)
```

to get the closed-loop step response with the approximate derivative terms as shown in Fig. 6.4(a). The error signal $e(t)$ when $N = 10$ is shown in Fig. 6.4(b). It can be seen that with $N = 10$, the approximation is fairly satisfactory.

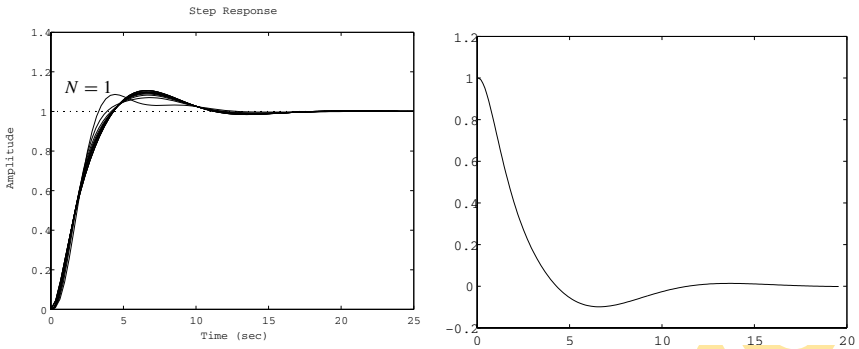
6.1.2 PID Control with Derivative in the Feedback Loop

From Fig. 6.4(b), it can be seen that there exists a jump when $t = 0$ in the error signal of the step response. This means that the derivative action may not be desirable in such a control strategy.

Thus, in practice, the derivative term may be preferred in the feedback path. Since the output does not change instantaneously for a step input a smoother signal is produced by taking the derivative of the output. This PID control strategy, which will be denoted PI-D, is shown in Fig. 6.5.

Recall the typical feedback control structure shown in Fig. 1.2. The controller and feedback transfer functions can be equivalently written as

$$G_c(s) = K_p \left(1 + \frac{1}{T_i s} \right), \tag{6.3}$$



(a) output signal (b) error signal

Figure 6.4. *PID control with approximate derivatives.*

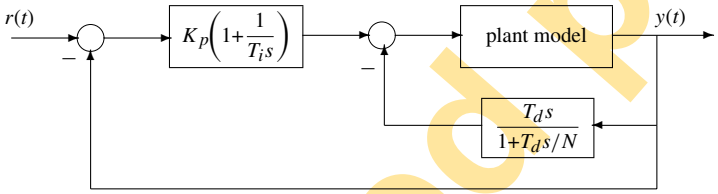


Figure 6.5. *PID control with derivative on output signal.*

$$H(s) = \frac{(1 + K_p/N)T_i T_d s^2 + K_p(T_i + T_d/N) + K_p}{K_p(T_i s + 1)(T_d s/N + 1)}. \tag{6.4}$$

The following example is designed to illustrate the consequence of using the derivative in the feedback path.

Example 6.3. For the plant model in Example 6.1, by the following MATLAB statements:

```
>> G=tf(1,[1,3,3,1]); Ti=1; Td=1; Kp=1; N=10; s=tf('s');
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
   G_c=feedback(G*Gc,1); Gc1=Kp*(1+1/s/Ti);
   H=((1+Kp/N)*Ti*Td*s^2+Kp*(Ti+Td/N)*s+Kp)/(Kp*(Ti*s+1)*(Td/N*s+1));
   G_c1=feedback(G*Gc1,H); step(G_c,G_c1)
```

the closed-loop step responses for the system with PID and PI-D are obtained and compared in Fig. 6.6. By observation, the response with the PI-D controller is slower and the overshoot larger for this particular example.

6.2 Ziegler–Nichols Tuning Formula

6.2.1 Empirical Ziegler–Nichols Tuning Formula

A very useful empirical tuning formula was proposed by Ziegler and Nichols in early 1942 [10]. The tuning formula is obtained when the plant model is given by a first-order plus

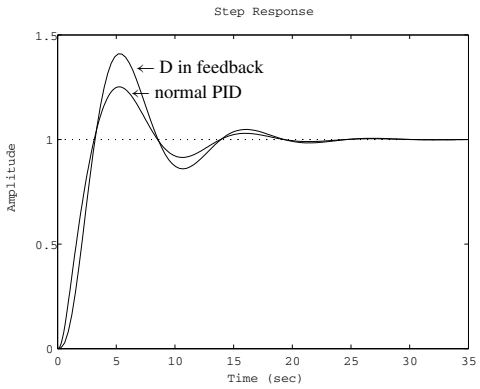


Figure 6.6. *The closed-loop step responses comparison.*

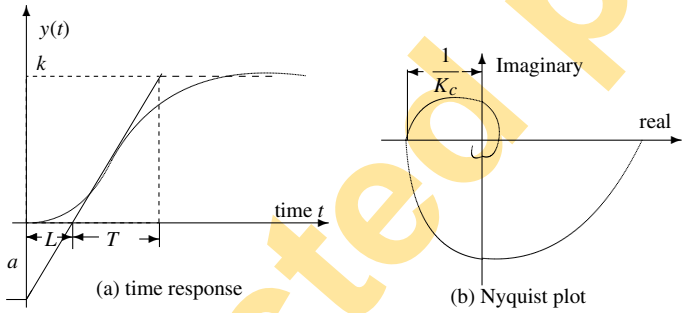


Figure 6.7. *Sketches of the responses of an FOPDT model.*

dead time (FOPDT) which can be expressed by

$$G(s) = \frac{k}{1+sT}\,\mathrm{e}^{-sL}. \tag{6.5}$$

In real-time process control systems, a large variety of plants can be approximately modeled by (6.5). If the system model cannot be physically derived, experiments can be performed to extract the parameters for the approximate model (6.5). For instance, if the step response of the plant model can be measured through an experiment, the output signal can be recorded as sketched in Fig. 6.7(a), from which the parameters of k, L, and T (or a, where $a = kL/T$) can be extracted by the simple approach shown. More sophisticated curve fitting approaches can also be used. With L and a, the Ziegler–Nichols formula in Table 6.1 can be used to get the controller parameters.

If a frequency response experiment can be performed, the crossover frequency ω_c and the ultimate gain K_c can be obtained from the Nyquist plot as shown in Fig. 6.7(b). Let $T_c = 2\pi/\omega_c$. The PID controller parameters can also be retrieved from Table 6.1. It should be noted that Table 6.1 applies for the design of P (proportional) and PI controllers in addition to the PID controller with the same set of experimental data from the plant. Since only the 180° point on the Nyquist locus is used in this approach, Ziegler and Nichols

suggested it can be found by putting the controller in the proportional mode and increasing the gain until an oscillation takes place. The point is then obtained from measurement of the gain and the oscillation frequency. This result, however, is based on linear theory, and although the technique has been used in practice, it does have major problems.

A MATLAB function `ziegler()` exists to design PI/PID controllers using the Ziegler–Nichols tuning formulas:

```
function [Gc,Kp,Ti,Td,H]=ziegler(key,vars)
Ti=[]; Td=[]; H=1;
if length(vars)==4,
   K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T;
   if key==1,  Kp=1/a;
   elseif key==2, Kp=0.9/a; Ti=3.33*L;
   elseif key==3 | key==4, Kp=1.2/a; Ti=2*L; Td=L/2; end
elseif length(vars)==3,
   K=vars(1); Tc=vars(2); N=vars(3);
   if key==1, Kp=0.5*K;
   elseif key==2, Kp=0.4*K; Ti=0.8*Tc;
   elseif key==3 | key==4, Kp=0.6*K; Ti=0.5*Tc; Td=0.12*Tc; end
elseif length(vars)==5,
   K=vars(1); Tc=vars(2); rb=vars(3); N=vars(5);
   pb=pi*vars(4)/180; Kp=K*rb*cos(pb);
   if key==2, Ti=-Tc/(2*pi*tan(pb));
   elseif key==3|key==4, Ti=Tc*(1+sin(pb))/(pi*cos(pb)); Td=Ti/4; end
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

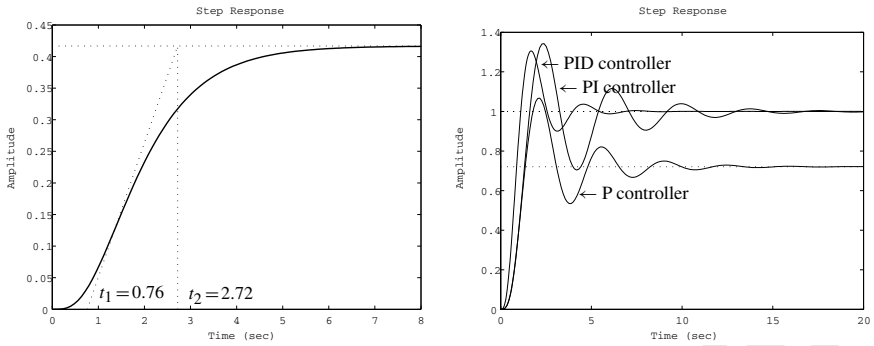
There is a low-level function `writepid()` which can be used in the design function; the content of the function is

```
function [Gc,H]=writepid(Kp,Ti,Td,N,key)
switch key
case 1, Gc=Kp;
case 2, Gc=tf(Kp*[Ti,1],[Ti,0]); H=1;
case 3, nn=[Kp*Ti*Td*(N+1)/N,Kp*(Ti+Td/N),Kp];
   dd=Ti*[Td/N,1,0]; Gc=tf(nn,dd); H=1;
case 4,  d0=sqrt(Ti*(Ti-4*Td)); Ti0=Ti; Kp=0.5*(Ti+d0)*Kp/Ti;
   Ti=0.5*(Ti+d0); Td=Ti0-Ti; Gc=tf(Kp*[Ti,1],[Ti,0]);
   nH=[(1+Kp/N)*Ti*Td, Kp*(Ti+Td/N), Kp];
   H=tf(nH,Kp*conv([Ti,1],[Td/N,1]));
case 5, Gc=tf(Kp*[Td*(N+1)/N,1],[Td/N,1]); H=1;
end
```

It seems that this function is quite lengthy for the simple Ziegler–Nichols formula given in Table 6.1. In fact, the MATLAB function also embeds a design formula discussed

Table 6.1. *Ziegler–Nichols tuning formulae.*

Controller	from step response			from frequency response		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$1/a$			$0.5K_c$		
PI	$0.9/a$	$3L$		$0.4K_c$	$0.8T_c$	
PID	$1.2/a$	$2L$	$L/2$	$0.6K_c$	$0.5T_c$	$0.12T_c$



(a) open-loop step response (b) closed-loop step response

Figure 6.8. *Controller design and responses with time domain parameters.*

later in this chapter. Here we shall consider only the syntax for the simple Ziegler–Nichols tuning rule

$$[G_c, K_p, T_i, T_d] = \texttt{ziegler(key,vars)},$$

where `key` determines the controller type with `key` $= 1$ for the P controller, `key` $= 2$ for the PI controller, and `key` $= 3$ for the PID controller. When step response data are available, one should specify `vars` $= [K, L, T, N]$, while `vars` $= [K_c, T_c, N]$ are designed for the given frequency response data.

Example 6.4. Consider a fourth-order plant

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

Enter the following MATLAB statements:

```
>> s=tf('s'); G=10/(s+1)/(s+2)/(s+3)/(s+4);
   step(G); k=dcgain(G)
```

The open-loop step response is shown in Fig. 6.8(a), with a steady-state value of 0.4167. From the step response, the parameters of the approximate FOPDT model are $k = 0.2941$, $L = 0.76$, and $T = 2.72 - 0.76 = 1.96$, based on which the P, PI, and PID controllers can be designed using the following MATLAB statements:

```
>> L=0.76; T=2.72-L; [Gc1,Kp1]=ziegler(1,[k,L,T,10])
   [Gc2,Kp2,Ti2]=ziegler(2,[k,L,T,10])
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
```

The P, PI, and PID controllers designed are, respectively,

$$G_{\rm p}(s) = 6.1895,\ G_{\rm PI}(s) = 5.57\left(1 + \frac{1}{2.5308s}\right),\ G_{\rm PID}(s) = 7.4274\left(1 + \frac{1}{1.52s} + 0.38s\right).$$

The closed-loop responses for these different controllers are obtained using the MATLAB statements

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

and they are shown in Fig. 6.8(b). It can be observed that the steady-state error exists when the P controller is used, and the response of the PID controller is faster than that of the PI controller.

If the frequency response of the plant model can be measured, the ultimate gain K_c and the crossover frequency ω_c can be read from the Nyquist plot as shown in Fig. 6.7(b). With K_c and ω_c, the parameters of different PID-type controllers can be obtained from Table 6.1. In this case, the MATLAB function `ziegler()` can still be used.

In fact, since the crossover frequency ω_c and the ultimate gain K_c are the gain margin of the open-loop plant model, one can directly obtain the parameters using the `margin()` function.

Example 6.5. Consider the plant model in Example 6.4. By the MATLAB statements

```
>> G=tf(10, [1,10,35,50,24]);
   nyquist(G); axis([-0.2,0.6,-0.4,0.4])
   [Kc,pp,wg,wp]=margin(G); [Kc,wg], Tc=2*pi/wg;
   [Gc1,Kp1]=ziegler(1,[Kc,Tc,10]); Kp1
   [Gc2,Kp2,Ti2]=ziegler(2,[Kc,Tc,10]); [Kp2,Ti2]
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[Kc,Tc,10]); [Kp3,Ti3,Td3]
```

the gain margin and its crossover frequency are found to be, respectively, 12.6, and 2.2361 rad/sec. The controllers are designed as

$$G_{\rm p}(s)=6.3,\quad G_{\rm PI}(s)=5.04\left(1+\frac{1}{2.2479s}\right),\quad G_{\rm PID}(s)=7.56\left(1+\frac{1}{1.405s}+0.3372s\right).$$

The Nyquist plot of the system can be obtained and is shown in Fig. 6.9(a). With these different controllers, the closed-loop system responses can be obtained using the MATLAB statements

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

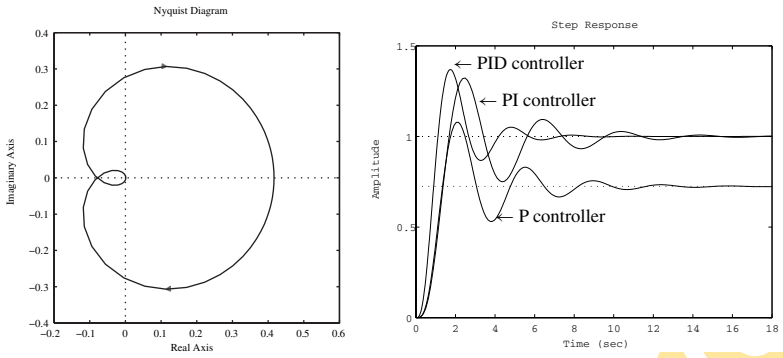
and the step responses of the closed-loop system are shown in Fig. 6.9(b).

6.2.2 Derivative Action in the Feedback Path

Assume that the derivative action is placed in the feedback path; then the normal PID parameters (K_p, T_i, T_d) can be obtained from [65] as

$$K_p = K_p'\left(1+\frac{T_d'}{T_i'}\right),\quad T_i = T_i'+T_d',\quad T_d=\frac{T_i'T_d'}{T_i'+T_d'},\tag{6.6}$$

where (K_p', T_i', T_d') are the PID parameters with derivative in the feedback path.



(a) Nyquist plots (b) closed-loop step response

Figure 6.9. *Controller design and responses.*

In other words, if a PID controller, with derivative action in a forward path, is designed, then an equivalent PID controller with the derivative action in the feedback path can be obtained by solving the following algebraic equation:

$$x^2 - T_i x + T_i T_d = 0, \quad \Rightarrow \quad x_{1,2} = \frac{T_i \pm \sqrt{T_i(T_i - 4T_d)}}{2}. \tag{6.7}$$

It is reasonable to assume in most PID controller designs that $T_i > 4T_d$. In this case, the above equation will have real roots $x_{1,2}$. Thus, from (K_p, T_i, T_d), the equivalent PID parameters for the new structure, i.e., with derivative in the feedback path, can be computed as follows:

$$T_i' = \frac{T_i + \sqrt{T_i(T_i - 4T_d)}}{2}, \quad T_d' = \frac{T_i - \sqrt{T_i(T_i - 4T_d)}}{2}, \quad K_p' = \frac{2T_i K_p}{T_i + \sqrt{T_i(T_i - 4T_d)}}. \tag{6.8}$$

The MATLAB function `ziegler()` can still be used to design such a PID controller. The syntax of the function now becomes

```
[Gc,Kp,Ti,Td,H]=ziegler(key,vars)
```

with `key` $= 4$ and H is the equivalent feedback transfer function object.

Example 6.6. Consider the plant model in Example 6.4. The normal PID controller can be designed using the Ziegler–Nichols algorithm. An effective design of a PID controller with a derivative in the feedback path can also be obtained with the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); N=10; [Kc,Pm,wc,wp]=margin(G);
   Tc=2*pi/wc; [Gc1,Kp1,Ti1,Td1]=ziegler(3,[Kc,Tc,N]),
   [Gc2,Kp2,Ti2,Td2,H]=ziegler(4,[Kc,Tc,N]),
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,H); step(G_c1,G_c2)
```

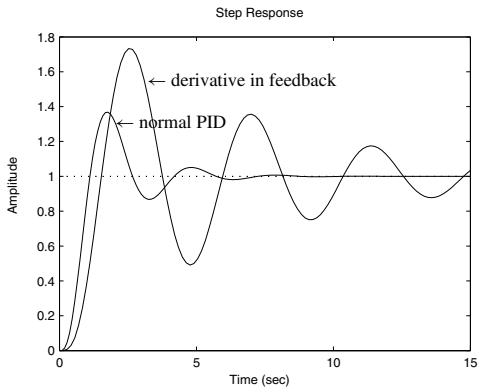


Figure 6.10. *PID controllers comparison.*

The controllers designed are $G_{\text{PID}}(s) = 7.5600\,(1 + 1/1.4050s + 0.3372s)$, with $K_p' = 4.5360$, $T_i' = 0.8430$, $T_d' = 0.5620$, and the step response comparison is shown in Fig. 6.10(a).

It can be seen that although the PID controller with derivative in the feedback path might be easier and faster to be implemented compared to the normal PID controller, its performance may not be very satisfactory. Sometimes, such a PID controller should be designed using a dedicated algorithm to ensure a good control performance.

6.2.3 Methods for First-Order Plus Dead Time Model Fitting

It can be seen that the model (6.5) is useful for designing a PID controller because of the availability of a simple formula. The method in Sec. 6.2.1 for finding L and T of a given plant is simple to use with the graph of a plant step response. Although in modern computation it is not necessary to reduce a model to this form to find suitable PID controller parameters, which may be found by using the original model with one of many possible approaches, nevertheless it can be useful. Given the plant transfer function, we can use one of the model reduction methods described in Chapter 3. For example, the suboptimal reduction method [47] is very effective at the expense of an affordable heavy computational load. The optimal reduced-order model can be obtained with the function `opt_app()`, covered in Sec. 3.6. In this section, two other effective and frequently used algorithms will be introduced.

Frequency response method

Consider the frequency response of a first-order model

$$G(\mathrm{j}\omega) = \frac{k}{Ts+1}\mathrm{e}^{-Ls}\Big|_{s=\mathrm{j}\omega} = \frac{k}{T\mathrm{j}\omega+1}\mathrm{e}^{-\mathrm{j}\omega L}. \tag{6.9}$$

The ultimate gain K_c at the crossover frequency ω_c is actually the first intersection of a Nyquist plot with the negative part of the real axis, i.e.,

$$\begin{cases} \dfrac{k(\cos\omega_c L - \omega_c T\sin\omega_c L)}{1+\omega_c^2 T^2} = -\dfrac{1}{K_c}, \\ \sin\omega_c L + \omega_c T\cos\omega_c L = 0, \end{cases} \tag{6.10}$$

where k is the steady-state value or DC (direct current) gain of the system which can be directly evaluated from the given transfer function. Define two variables $x_1 = L$ and $x_2 = T$ satisfying

$$\begin{cases} f_1(x_1,x_2) = kK_c(\cos\omega_c x_1 - \omega_c x_2\sin\omega_c x_1) + 1 + \omega_c^2 x_2^2 = 0, \\ f_2(x_1,x_2) = \sin\omega_c x_1 + \omega_c x_2\cos\omega_c x_1 = 0. \end{cases} \tag{6.11}$$

The Jacobian matrix is that

$$\begin{aligned} \boldsymbol{J} &= \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} \\ &= \begin{bmatrix} -kK_c\omega_c\sin\omega_c x_1 - kK_c\omega_c^2 x_2\cos\omega_c x_1 & 2\omega_c^2 x_2 - kK_c\omega_c\sin\omega_c x_1 \\ \omega_c\cos\omega_c x_1 - \omega_c^2 x_2\sin\omega_c x_1 & \omega_c\cos\omega_c x_1 \end{bmatrix}. \end{aligned} \tag{6.12}$$

So, (x_1, x_2) can be solved using any quasi-Newton algorithm. The MATLAB function `[K,L,T]=getfod(G)` is written for solving x_1 and x_2 in order to find the parameters K, L, T of the system.

```
function [K,L,T]=getfod(G,method)
K=dcgain(G);
if nargin==1
   [Kc,Pm,wc,wcp]=margin(G); ikey=0; L=1.6*pi/(3*wc); T=0.5*Kc*K*L;
   if finite(Kc), x0=[L;T];
      while ikey==0, u=wc*x0(1); v=wc*x0(2);
         FF=[K*Kc*(cos(u)-v*sin(u))+1+v^2; sin(u)+v*cos(u)];
         J=[-K*Kc*wc*sin(u)-K*Kc*wc*v*cos(u), -K*Kc*wc*sin(u)+2*wc*v;
            wc*cos(u)-wc*v*sin(u), wc*cos(u)];
         x1=x0-inv(J)*FF;
         if norm(x1-x0)<1e-8, ikey=1; else, x0=x1;
      end, end
      L=x0(1); T=x0(2);
   end
elseif nargin==2 & method==1
   [n1,d1]=tfderv(G.num{1},G.den{1}); [n2,d2]=tfderv(n1,d1);
   K1=dcgain(n1,d1); K2=dcgain(n2,d2);
   Tar=-K1/K; T=sqrt(K2/K-Tar^2); L=Tar-T;
end
function [e,f]=tfderv(b,a)
f=conv(a,a); na=length(a); nb=length(b);
e1=conv((nb-1:-1:1).*b(1:end-1),a);
e2=conv((na-1:-1:1).* a(1:end-1),b); maxL=max(length(e1),length(e2));
e=[zeros(1,maxL-length(e1)) e1]-[zeros(1,maxL-length(e2)) e2];
```

Transfer function method

Consider the first-order model with delay given by

$$G_n(s) = \frac{ke^{-Ls}}{Ts+1}.$$

Taking the first- and second-order derivatives of $G_n(s)$ with respect to s, one can immediate find that

$$\frac{G_n'(s)}{G_n(s)} = -L - \frac{T}{1+Ts}, \quad \frac{G_n''(s)}{G_n(s)} - \left(\frac{G_n'(s)}{G_n(s)}\right)^2 = \frac{T^2}{(1+Ts)^2}.$$

Evaluating the values at $s = 0$ yields

$$T_{\text{ar}} = -\frac{G_n'(0)}{G_n(0)} = L + T, \quad T^2 = \frac{G_n''(0)}{G_n(0)} - T_{\text{ar}}^2, \tag{6.13}$$

where T_{ar} is also referred to as the average residence time. From the former equation, one has $L = T_{\text{ar}} - T$. Again, the DC gain k can be evaluated from $G_n(0)$.

The solution for the FOPDT model is thus obtained by using the derivatives of its transfer function $G(s)$ in the above formula.

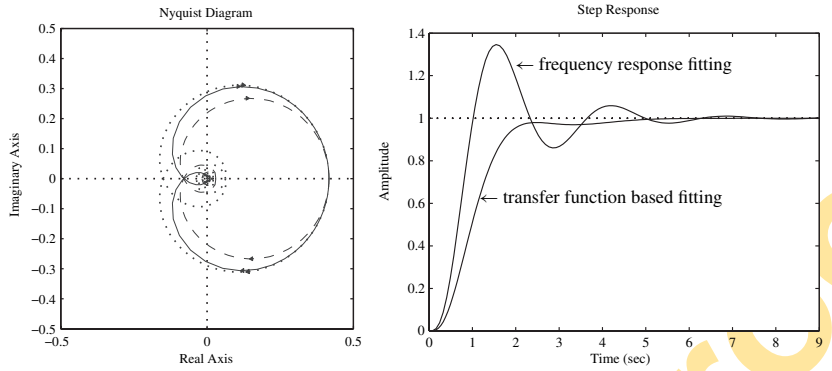
The MATLAB function `getfod()` listed earlier can be used with the syntax `[K,L,T]= getfod(G,1)` to find the parameters K, L, T of the system.

Example 6.7. Consider the fourth-order model used in Example 6.4. The parameters of its approximate FOPTD model can be obtained using the MATLAB statements

```
>> G=tf(10,[1,10,35,50,24]);
   [k,L,T]=getfod(G); G1=tf(k,[T 1]); G1.ioDelay=L;
   [Gc1,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
   [k,L,T]=getfod(G,1), G2=tf(k,[T 1]); G2.ioDelay=L;
   nyquist(G,'-',G1,'--',G2,':'); figure
   [Gc2,Kp4,Ti4,Td4]=ziegler(3,[k,L,T,10])
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1); step(G_c1,G_c2)
```

The Nyquist plot comparisons of the plant model and the two approximations are shown in Fig. 6.11(a).

With the frequency response method, the K, L, T parameters are obtained as 0.4167, 0.7882, 2.3049. The PID controller designed with the Ziegler–Nichols formulas is $G_{c1}(s) = 8.4219\,(1 + 1/1.5764s + 0.3941s)$. While the parameters using the transfer function method are 0.4167, 0.8902, 1.1932, the PID controller is $G_{c2}(s) = 3.8602\,(1 + 1/1.7804s + 0.4451s)$. The closed-loop step responses with the above two PID controllers are shown in Fig. 6.11(b).



(a) Nyquist plots (b) closed-loop step responses

Figure 6.11. *PID controller responses.*

It can be seen that although the PID controller designed with the transfer function identification algorithm looks better, it does not reflect the usual overshoot characteristics of Ziegler–Nichols tuning, presumably due to the inaccurately identified parameters of an FOPDT model.

With the use of the suboptimal model reduction technique presented in Sec. 3.6.3, the parameters can be extracted with the following statements and the controller can better be designed:

```
Gr=opt_app(G,0,1,1); [n,d]=tfdata(G,'v');
K=dcgain(G); T=d(1)/d(2); L=Gr.ioDelay;
```

6.2.4 A Modified Ziegler–Nichols Formula

Consider the Nyquist frequency response shown in Fig. 6.12(a), where for a selected point A on the Nyquist plot, the control effects of the P, I, and D terms are shown in the appropriate directions. Thus, with properly chosen K_p, T_i, and T_d, it is possible to move the given point A on the Nyquist curve of the uncontrolled plant to an arbitrary position on the Nyquist plot of the controlled system. The typical Nyquist plot under PID control is shown in Fig. 6.12(b), where A_1 corresponds to the point A in Fig. 6.12(a).

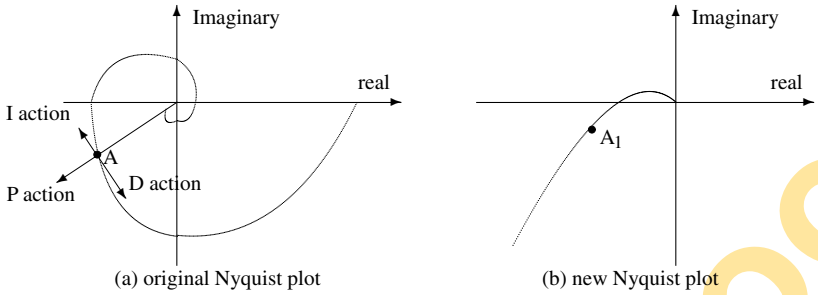
Denote point A in the complex plane as $G(j\omega_0) = r_a e^{j(\pi+\phi_a)}$. Suppose A is to be moved to A_1 which is represented by $G_1(j\omega_0) = r_b\, e^{j(\pi+\phi_b)}$. Assume that the PID controller at frequency ω_0 is $G_c(s) = r_c e^{j\phi_c}$. Then, obviously,

$$r_b e^{j(\pi+\phi_b)} = r_a r_c e^{j(\pi+\phi_a+\phi_c)}. \tag{6.14}$$

Therefore, $r_c = r_b/r_a$ and $\phi_c = \phi_b - \phi_a$. So, based on the above analysis, PI and PID controllers can be designed as follows:

- *PI control*: The controller can be designed such that

$$K_p = \frac{r_b \cos(\phi_b - \phi_a)}{r_a}, \quad T_i = \frac{1}{\omega_0 \tan(\phi_a - \phi_b)}, \tag{6.15}$$



(a) original Nyquist plot (b) new Nyquist plot

Figure 6.12. *Sketches of FOPDT model.*

which means that $\phi_a > \phi_b$ for a positive T_i.

As a special case, the Ziegler–Nichols algorithm design is by

$$K_p = K_c r_b \cos\phi_b, \quad T_i = -\frac{T_c}{2\pi \tan\phi_b}, \tag{6.16}$$

where $T_c = 2\pi/\omega_c$, $r_a = 1/K_c$, and $\phi_a = 0$.

- *PID control*: The controller can be designed such that

$$K_p = \frac{r_b \cos(\phi_b - \phi_a)}{r_a}, \quad \omega_0 T_d - \frac{1}{\omega_0 T_i} = \tan(\phi_b - \phi_a). \tag{6.17}$$

Clearly, T_i and T_d are not unique according to (6.17). To get a unique PID design, it is a usual practice to set $T_d = \alpha T_i$, where α is a constant. Given an α, T_i and T_d can be obtained uniquely from

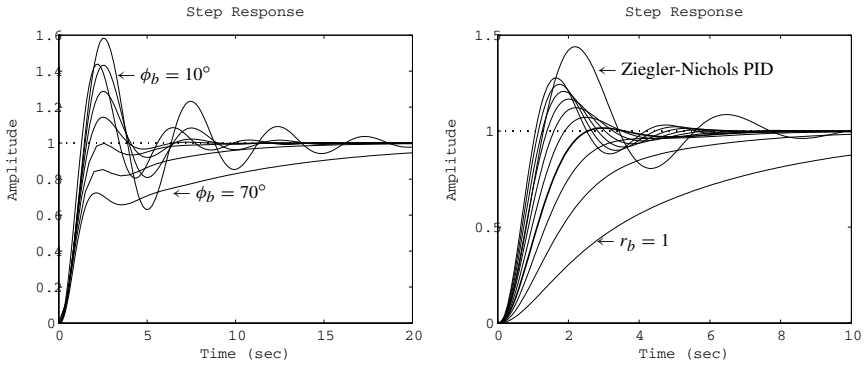
$$T_i = \frac{1}{2\alpha\omega_0}\left(\tan(\phi_b - \phi_a) + \sqrt{4\alpha + \tan^2(\phi_b - \phi_a)}\right), \quad T_d = \alpha T_i. \tag{6.18}$$

By inspection, it is seen that the Ziegler–Nichols tuning formula is a special case when $\alpha = 1/4$. The Ziegler–Nichols tuning formula can be rewritten as follows:

$$K_p = K_c r_b \cos\phi_b, \quad T_i = \frac{T_c}{\pi}\left(\frac{1+\sin\phi_b}{\cos\phi_b}\right), \quad T_d = \frac{T_c}{4\pi}\left(\frac{1+\sin\phi_b}{\cos\phi_b}\right), \tag{6.19}$$

where $r_a = 1/K_c$, $\phi_a = 0$, and $\alpha = 1/4$.

It can be seen that the PI or PID controllers can be designed by a suitable choice of r_b and ϕ_b. The design problem is then one of selecting suitable values for these two parameters to give the appropriate performance. This is called a modified Ziegler–Nichols PI/PID tuning formula, which has been implemented in the MATLAB function `ziegler()`, too. The only difference is that `vars` $= [K_c, T_c, r_b, \phi_b, N]$.



(a) for different ϕ_b (b) for different r_b

Figure 6.13. *Closed-loop step responses.*

Example 6.8. Consider the plant model given by $G(s) = 1/(s+1)^3$. The PID controller by the original Ziegler–Nichols tuning method can be obtained as follows:

```
>> G=tf(1,[1,3,3,1]); [Kc,pp,wg,wp]=margin(G); Tc=2*pi/wg;
   [Gc1,Kp1,Ti1,Td1]=ziegler(3,[Kc,Tc,10])
```

and the controller $G(s) = 4.8007\,(1 + 1/1.8137s + 0.4353s)$ is obtained. Now, let us illustrate the flexibility of the modified Ziegler–Nichols PI/PID tuning formula. First, fix $r_b = 0.5$ and change ϕ_b. By the following MATLAB statements:

```
>> G_c=feedback(G*Gc1,1); step(G_c,20); rb=0.5; hold on
   for pb=[10:10:70]
      [Gc2,Kp2,Ti2,Td2]=ziegler(3,[Kc,Tc,rb,pb,10]);
      G_c2=feedback(G*Gc2,1); step(G_c2,20);
   end
```

the closed-loop step responses of the system for different values of ϕ_b are shown in Fig. 6.13(a). Clearly, when ϕ_b increases, the overshoot and oscillation become smaller. When ϕ_b is larger than 60°, there is no overshoot, but the response becomes too sluggish. A good choice for the phase angle based on these responses is approximately 45°.

Now, fix ϕ_b at $\phi_b = 45°$ and change r_b. By the MATLAB statements

```
>> G_c=feedback(G*Gc1,1); step(G_c,10); pb=45; hold on;
   for rb=[0.1:0.1:1]
      [Gc2,Kp2,Ti2,Td2]=ziegler(3,[Kc,Tc,rb,pb,10]);
      G_c2=feedback(G*Gc2,1); step(G_c2,10);
   end
```

the closed-loop step responses of the system for different r_b are compared in Fig. 6.13(b). It can be seen that the smaller the r_b, the smaller the overshoot and the slower the response. Clearly, $r_b = 0.45$, and $\phi_b = 45°$ can be considered as a good choice for this example with almost no overshoot and with a reasonably fast response.

It can be concluded that the modified tuning method is advantageous over the original Ziegler–Nichols PI/PID tuning technique.

6.3 Other PID Controller Tuning Formulae

Many variants of the traditional Ziegler–Nichols PID tuning methods have been proposed. Several of these are given in the following section.

6.3.1 Chien–Hrones–Reswick PID Tuning Algorithm

The Chien–Hrones–Reswick (CHR) method [66] emphasizes the set-point regulation or disturbance rejection. In addition one qualitative specifications on the response speed and overshoot can be accommodated. Compared with the traditional Ziegler–Nichols tuning formula, the CHR method uses the time constant T of the plant explicitly.

The CHR PID controller tuning formulas are summarized in Table 6.2 for set-point regulation. The more heavily damped closed-loop response, which ensures, for the ideal plant model, the "quickest response without overshoot" is labeled "with 0% overshoot," and the "quickest response with 20% overshoot" is labeled "with 20% overshoot."

Similarly, Table 6.3 is used to design controllers for disturbance rejection purposes.

A MATLAB function `chrPID()` is written which can be used to design different controllers using the CHR algorithms:

```
function [Gc,Kp,Ti,Td,H]=chrpid(key,tt,vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T; Ti=[]; Td=[];
ovshoot=vars(5); if tt==1, TT=T; else TT=L; tt=2;  end
if ovshoot==0,
   KK=[0.3,0.35,1.2,0.6,1,0.5; 0.3,0.6,4,0.95,2.4,0.42];
else,
   KK=[0.7,0.6,1,0.95,1.4,0.47; 0.7,0.7,2.3,1.2,2,0.42];
end
switch key
case 1, Kp=KK(tt,1)/a;
case 2, Kp=KK(tt,2)/a; Ti=KK(tt,3)*TT;
case {3,4}, Kp=KK(tt,4)/a; Ti=KK(tt,5)*TT; Td=KK(tt,6)*L;
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

Table 6.2. *CHR tuning formulae for set-point regulation.*

Controller	with 0% overshoot			with 20% overshoot		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$0.3/a$			$0.7/a$		
PI	$0.35/a$	$1.2T$		$0.6/a$	T	
PID	$0.6/a$	T	$0.5L$	$0.95/a$	$1.4T$	$0.47L$

Table 6.3. *CHR tuning formulae for disturbance rejection.*

Controller	with 0% overshoot			with 20% overshoot		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$0.3/a$			$0.7/a$		
PI	$0.6/a$	$4L$		$0.7/a$	$2.3L$	
PID	$0.95/a$	$2.4L$	$0.42L$	$1.2/a$	$2L$	$0.42L$

The syntax of the `chrpid()` function is

```
[Gc,Kp,Ti,Td]=chrPID(key,typ,vars)
```

where the returned variables are defined similar to those in `ziegler()`. `key` = 1, 2, 3 is for P, PI, and PID controllers, respectively. The variable `typ` denotes the type of criteria used with `typ` = 1 for set-point control and any other value for disturbance rejection. `vars` = [k, L, T, N, O_s] with $O_s = 0$ denotes no overshoot, and any other value denotes 20% overshoot.

Example 6.9. Consider the plant model in Example 6.4. The Ziegler–Nichols PID controller and the four CHR controllers for different controller types and specifications are obtained using the following statements:

```
>> s=tf('s'); G=10/((s+1)*(s+2)*(s+3)*(s+4));
   [k,L,T]=getfod(G); N=10; [Gc1,Kp,Ti,Td]=ziegler(3,[k,L,T,N])
   [Gc2,Kp,Ti,Td]=chrpid(3,1,[k,L,T,N,0])
   [Gc3,Kp,Ti,Td]=chrpid(3,1,[k,L,T,N,20])
   [Gc4,Kp,Ti,Td]=chrpid(3,2,[k,L,T,N,0]);
```

The four PID controllers designed are, respectively,

$$G_1(s) = 8.4219\left(1+\frac{1}{1.5764s}+0.3941s\right),\quad G_2(s) = 4.2110\left(1+\frac{1}{2.3049s}+0.3941s\right),$$

$$G_3(s) = 6.6674\left(1+\frac{1}{3.2268s}+0.3704s\right),\quad G_4(s) = 6.6674\left(1+\frac{1}{1.8917s}+0.3310s\right).$$

For the different controllers designed in the above, the step response of the closed-loop systems can be obtained using the following MATLAB statements:

```
>> step(feedback(G*Gc1,1),feedback(G*Gc2,1),feedback(G*Gc3,1),...
       feedback(G*Gc4,1),10)
```

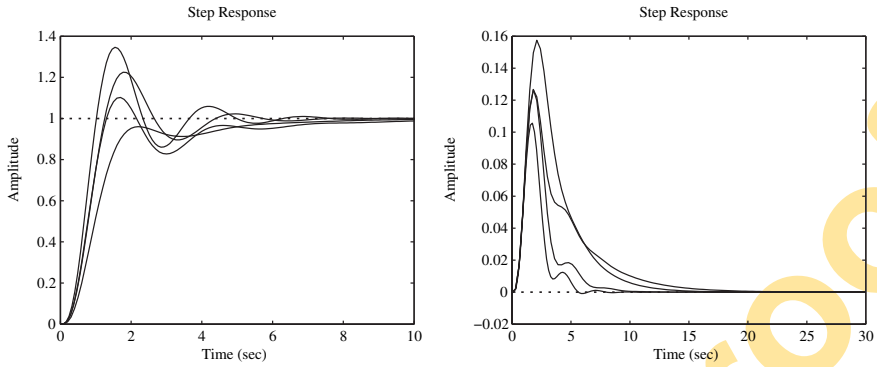
as summarized in Fig. 6.14(a). It can be seen that the set-point regulation controller with 0% overshoot gives a satisfactory result. Similarly, with the following MATLAB statements:

```
>> step(feedback(G,Gc1),feedback(G,Gc2),feedback(G,Gc3),...
       feedback(G,Gc4),30)
```

the closed-loop responses to a step disturbance signal can be obtained as shown in Fig. 6.14(b). Clearly, compared with the traditional Ziegler–Nichols controller, the effect of the disturbance signal can be significantly reduced by a CHR controller.

6.3.2 Cohen–Coon Tuning Algorithm

Another Ziegler–Nichols type tuning algorithm is the Cohen–Coon tuning formula [67]. Referring to the FOPDT model (6.5) approximately obtained from experiments, denote



(a) set-point step response (b) disturbance step response

Figure 6.14. *Closed-loop step responses of CHR controllers.*

$a = kL/T$ and $\tau = L/(L+T)$. The different controllers can be designed by the direct use of Table 6.4.

A MATLAB function `cohenpid()` is written which can be used to design a PID controller using the Cohen–Coon tuning formulas:

```
function [Gc,Kp,Ti,Td,H]=cohenpid(key,vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4);
a=K*L/T; tau=L/(L+T); Ti=[]; Td=[];
switch key
case 1,Kp=(1+0.35*tau/(1-tau))/a;
case 2,
   Kp=0.9*(1+0.92*tau/(1-tau))/a; Ti=(3.3-3*tau)*L/(1+1.2*tau);
case {3,4}, Kp=1.35*(1+0.18*tau/(1-tau))/a;
   Ti=(2.5-2*tau)*L/(1-0.39*tau); Td=0.37*(1-tau)*L/(1-0.81*tau);
case 5
   Kp=1.24*(1+0.13*tau/(1-tau))/a; Td=(0.27-0.36*tau)*L/(1-0.87*tau);
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

The syntax is `[`G_c`,`K_p`,`T_i`,`T_d`,`H`]=cohenpid(key,vars)`, where the `vars` arguments should be written as `vars` $= [k, L, T, N]$.

Table 6.4. *Controller parameters of Cohen–Coon method.*

Controller	K_p	T_i	T_d
P	$\frac{1}{a}\left(1+\frac{0.35\tau}{1-\tau}\right)$		
PI	$\frac{0.9}{a}\left(1+\frac{0.92\tau}{1-\tau}\right)$	$\frac{3.3-3\tau}{1+1.2\tau}L$	
PD	$\frac{1.24}{a}\left(1+\frac{0.13\tau}{1-\tau}\right)$		$\frac{0.27-0.36\tau}{1-0.87\tau}L$
PID	$\frac{1.35}{a}\left(1+\frac{0.18\tau}{1-\tau}\right)$	$\frac{2.5-2\tau}{1-0.39\tau}L$	$\frac{0.37-0.37\tau}{1-0.81\tau}L$

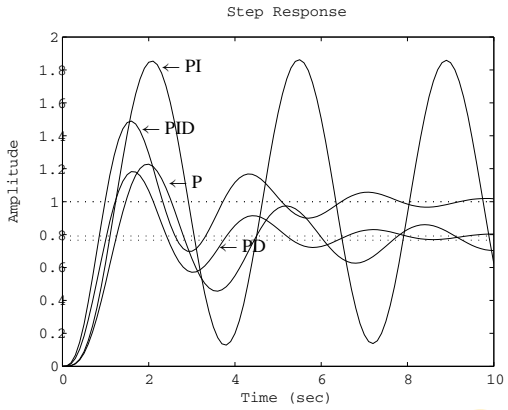


Figure 6.15. *Step responses under controllers of the Cohen–Coon method.*

Example 6.10. Consider the plant model given in Example 6.4 with its P, PI, PD, and PID controllers designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [k,L,T]=getfod(G);
   [Gc1,Kp1]=cohenpid(1,[k,L,T,10])
   [Gc2,Kp2,Ti2]=cohenpid(2,[k,L,T,10])
   [Gc3,Kp3,Ti3,Td3]=cohenpid(5,[k,L,T,10])
   [Gc4,Kp4,Ti4,Td4]=cohenpid(3,[k,L,T,10])
```

and the controllers are obtained as

$$G_1(s) = 7.8583,\ G_2(s) = 8.3036\,(1 + 1/1.5305s)\,,$$
$$G_3(s) = 9.0895(1 + 0.1805s),\ G_4(s) = 10.0579\,(1 + 1/1.7419s + 0.2738s)\,.$$

With the following MATLAB statements:

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); G_c4=feedback(G*Gc4,1);
   step(G_c1,G_c2,G_c3,G_c4,10)
```

the closed-loop step responses of the systems with the different controllers are shown in Fig. 6.15.

6.3.3 Refined Ziegler–Nichols Tuning

Since the PID controller designed by the conventional Ziegler–Nichols tuning formulas often exhibits rather strong oscillation in the set-point response and a large overshoot, a refinement to such a PID controller tuning algorithm can be obtained with the use of set-point weighting [68]:

$$u(t) = K_p\left[(\beta u_c - y) + \frac{1}{T_i}\int e\mathrm{d}t - T_d\frac{\mathrm{d}y}{\mathrm{d}t}\right], \tag{6.20}$$

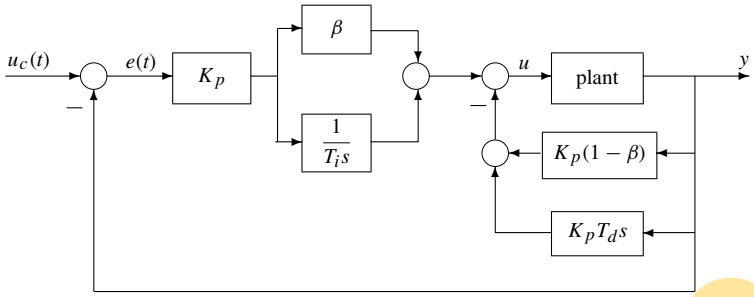


Figure 6.16. *Refined PID control structure.*

where the derivative action is performed on the output signal and a fraction of the input signal is added to the control signal. Usually, $\beta < 1$. The control law can be rewritten as

$$u(t) = K_p\left(\beta e + \frac{1}{T_i}\int e\mathrm{d}t\right) - K_p\left[(1-\beta)y + T_d\frac{\mathrm{d}y}{\mathrm{d}t}\right]. \tag{6.21}$$

The block diagram representation of the control system can be constructed as shown in Fig. 6.16. Compared with the typical feedback control structure shown in Fig. 1.2, after some transfer function block manipulations, the controller $G_c(s)$ and the feedback $H(s)$ can be easily obtained as follows:

$$G_c(s) = K_p\left(\beta + \frac{1}{T_i s}\right), \tag{6.22}$$

$$H(s) = \frac{T_i T_d \beta(N+2-\beta)s^2/N + (T_i + T_d/N)s + 1}{(T_i\beta s + 1)(T_d s/N + 1)}. \tag{6.23}$$

Define the normalized delay constant τ as $\tau = L/T$ and a constant κ by $\kappa = K_c k$. For different ranges of the variables τ and κ, PID controller parameters were suggested as follows:

- If $2.25 < \kappa < 15$ or $0.16 < \tau < 0.57$, use the original Ziegler–Nichols design parameters. To ensure that the overshoot is less than 10% or 20%, β should be evaluated, respectively, from

$$\beta = \frac{15-\kappa}{15+\kappa} \text{ or } \beta = \frac{36}{27+5\kappa}. \tag{6.24}$$

- If $1.5 < \kappa < 2.25$ or $0.57 < \tau < 0.96$, the integral parameter T_i in the Ziegler–Nichols controller should be changed to $T_i = 0.5\mu T_c$, where

$$\mu = \frac{4}{9}\kappa \text{ and } \beta = \frac{8}{17}(\mu - 1). \tag{6.25}$$

- If $1.2 < \kappa < 1.5$, in order to keep the overshoot less than 10%, the parameters of the PID should be refined as

$$K_p = \frac{5}{6}\left(\frac{12+\kappa}{15+14\kappa}\right), \quad T_i = \frac{1}{5}\left(\frac{4}{15}\kappa + 1\right). \tag{6.26}$$

A MATLAB function `rziegler()` is written which can be used to design a refined PID controller:

```
function [Gc,Kp,Ti,Td,beta,H]=rziegler(vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T; Kp=1.2/a;
Ti=2*L; Td=L/2; Kc=vars(5); Tc=vars(6); kappa=Kc*K; tau=L/T; H=[];
if (kappa > 2.25 & kappa<15) | (tau>0.16 & tau<0.57)
   beta=(15-kappa)/(15+kappa);
elseif (kappa<2.25 & kappa>1.5) | (tau<0.96 & tau>0.57)
   mu=4*jappa/9; beta=8*(mu-1)/17; Ti=0.5*mu*Tc;
elseif (kappa>1.2 & kappa<1.5),
   Kp=5*(12+kappa)/(6*(15+14*kappa)); Ti=0.2*(4*kappa/15+1); beta=1;
end
Gc=tf(Kp*[beta*Ti,1],[Ti,0]); nH=[Ti*Td*beta*(N+2-beta)/N,Ti+Td/N,1];
dH=conv([Ti*beta,1],[Td/N,1]); H=tf(nH,dH);
```

The syntax of the function is `[`$G_c, K_p, T_i, T_d, \beta, H$`]=rziegler(vars)`, where `vars` = $[k, L, T, N, K_c, T_c]$.

Example 6.11. Consider the plant model in Example 6.4. The refined PID controller can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [k,L,T]=getfod(G);
   [Kc,p,wc,m]=margin(G);  Tc=2*pi/wc;
   [Gc,Kp,Ti,Td,beta,H]=rziegler([k,L,T,10,Kc,Tc])
   G_c=feedback(G*Gc,H); [Gc1,Kp1,Ti1,Td1]=ziegler(3,[k,L,T,10]);
   G_c1=feedback(G*Gc1,1); step(G_c,G_c1);
```

The parameters of the refined PID controller should be taken as $K_p = 8.4219$, $T_i = 1.5764$, $T_d = 0.3941$, $\beta = 0.4815$. The closed-loop step responses under the refined Ziegler–Nichols PID controller are shown in Fig. 6.17, with a comparison to the response from the conventional Ziegler–Nichols PID controller. The response is significantly improved but not as good as the responses using other tuning algorithms such as the modified Ziegler–Nichols method with $r_b = 0.45$, and $\phi_b = 45°$.

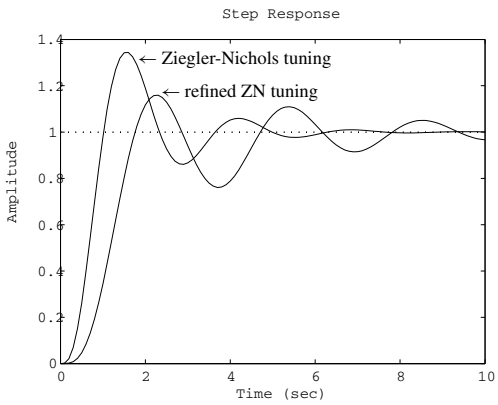


Figure 6.17. *Step responses under refined Ziegler–Nichols controller.*

6.3.4 The Wang–Juang–Chan Tuning Formula

Based on the optimum ITAE criterion, the tuning algorithm proposed by Wang, Juang, and Chan [69] is a simple and efficient method for selecting the PID parameters. If the k, L, T parameters of the plant model are known, the controller parameters are given by

$$K_p = \frac{(0.7303 + 0.5307T/L)(T + 0.5L)}{K(T + L)}, \qquad T_i = T + 0.5L, \quad T_d = \frac{0.5LT}{T + 0.5L}. \tag{6.27}$$

A MATLAB function `wjcpid()` is written for the PID controller design, using the Wang–Juang–Chan tuning formula:

```
function [Gc,Kp,Ti,Td]=wjcpid(vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); Td=0.5*L*T/(T+0.5*L);
Kp=(0.7303+0.5307*T/L)*(T+0.5*L)/(K*(T+L)); Ti=T+0.5*L;
s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
```

where `vars` $= [k, L, T, N]$.

6.3.5 Optimum PID Controller Design

Optimum setting algorithms for a PID controller were proposed by Zhuang and Atherton [70] for various criteria. Consider the general form of the optimum criterion

$$J_n(\theta) = \int_0^\infty [t^n e(\theta, t)]^2 \mathrm{d}t, \tag{6.28}$$

where $e(\theta, t)$ is the error signal which enters the PID controller, with θ the PID controller parameters. For the system structure shown in Fig. 6.1, two setting strategies are proposed: one for the set-point input and the other for the disturbance signal $d(t)$. In particular, three values of n are discussed, i.e., for $n = 0, 1, 2$. These three cases correspond, respectively, to three different optimum criteria: the integral squared error (ISE) criterion, integral squared time weighted error (ISTE) criterion, and the integral squared time-squared weighted error (IST^{2}E) criterion [65]. The expressions given were obtained by fitting curves to the optimum theoretical results.

Set-Point optimum PID tuning

If the plant can be represented by the FOPDT model in (6.5), the typical PI controller can be empirically represented as

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2 + b_2(L/T)}, \tag{6.29}$$

where the (a, b) pairs can be obtained from Table 6.5. When the first-order approximation to the plant model can be obtained, the PI controller can be designed easily by the direct use of Table 6.5 and (6.29).

Table 6.5. *Set-point PI controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	0.980	0.712	0.569	1.072	0.786	0.628
b_1	−0.892	−0.921	−0.951	−0.560	−0.559	−0.583
a_2	0.690	0.968	1.023	0.648	0.883	1.007
b_2	−0.155	−0.247	−0.179	−0.114	−0.158	−0.167

Table 6.6. *Set-point PID controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.048	1.042	0.968	1.154	1.142	1.061
b_1	−0.897	−0.897	−0.904	−0.567	−0.579	−0.583
a_2	1.195	0.987	0.977	1.047	0.919	0.892
b_2	−0.368	−0.238	−0.253	−0.220	−0.172	−0.165
a_3	0.489	0.385	0.316	0.490	0.384	0.315
b_3	0.888	0.906	0.892	0.708	0.839	0.832

Table 6.7. *Set-point PID controller parameters with D in feedback path.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.260	1.053	0.942	1.295	1.120	1.001
b_1	−0.887	−0.930	−0.933	−0.619	−0.625	−0.624
a_2	0.701	0.736	0.770	0.661	0.720	0.754
b_2	−0.147	−0.126	−0.130	−0.110	−0.114	−0.116
a_3	0.375	0.349	0.308	0.378	0.350	0.308
b_3	0.886	0.907	0.897	0.756	0.811	0.813

For the PID controller, its gains can be set as follows:

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2 + b_2(L/T)}, \quad T_d = a_3 T\left(\frac{L}{T}\right)^{b_3}, \tag{6.30}$$

where for different ratios L/T, the coefficients (a, b) are defined in Table 6.6.

To include the derivative action in the output signal, the corresponding PID controller is given by

$$U(s) = K_p\left(1 + \frac{1}{T_i s}\right)E(s) - \frac{sT_d}{1 + sT_d/N}Y(s), \tag{6.31}$$

where the parameters (a, b) should be determined according to Table 6.7.

Disturbance rejection PID tuning

Sometimes one may want to design disturbance rejection PID controllers, i.e., to design a controller having a good rejection performance on the disturbance signal $d(t)$. The parameters of the PI controller should be set as

$$K_p = \frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2}\left(\frac{L}{T}\right)^{b_2}, \tag{6.32}$$

where the parameters (a, b) are obtained directly from Table 6.8.

Furthermore, for the PID controller,

$$K_p = \frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2}\left(\frac{L}{T}\right)^{b_2}, \quad T_d = a_3 T\left(\frac{L}{T}\right)^{b_3}, \tag{6.33}$$

and the (a, b) parameters are determined from Table 6.9.

A MATLAB function `optpid()` is written which can be used to get the parameters of the PID controller:

```
function [Gc,Kp,Ti,Td,H]=optPID(key,typ,vars)
k=vars(1); L=vars(2); T=vars(3); N=vars(4); Td=[];
if length(vars)==5, iC=vars(5);
   switch key
   case 2
A=[0.980,0.712,0.569,1.072,0.786,0.628; 0.892,0.921,0.951,0.560,0.559,0.583;
   0.690,0.968,1.023,0.648,0.883,1.007; 0.155,0.247,0.179,0.114,0.158,0.167];
   case 3
A=[1.048,1.042,0.968,1.154,1.142,1.061; 0.897,0.897,0.904,0.567,0.579,0.583;
   1.195,0.987,0.977,1.047,0.919,0.892; 0.368,0.238,0.253,0.220,0.172,0.165;
   0.489,0.385,0.316,0.490,0.384,0.315; 0.888,0.906,0.892,0.708,0.839,0.832];
   case 4
A=[1.260,1.053,0.942,1.295,1.120,1.001; 0.887,0.930,0.933,0.619,0.625,0.624;
   0.701,0.736,0.770,0.661,0.720,0.754; 0.147,0.126,0.130,0.110,0.114,0.116;
   0.375,0.349,0.308,0.378,0.350,0.308; 0.886,0.907,0.897,0.756,0.811,0.813];
   end
   ii=0; if (L/T>1) ii=3; end; tt=L/T; a1=A(1,ii+iC); b1=-A(2,ii+iC);
   a2=A(3,ii+iC); b2=-A(4,ii+iC); Kp=a1/k*tt^b1; Ti=T/(a2+b2*tt);
   if key==3| key==4
      a3=A(5,ii+iC); b3=A(6,ii+iC); Td=a3*T*tt^b3;
   end
else,
   Kc=vars(5); Tc=vars(6); k=vars(7);
   switch key
```

Table 6.8. *Disturbance rejection PI controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.279	1.015	1.021	1.346	1.065	1.076
b_1	−0.945	−0.957	−0.953	−0.675	−0.673	−0.648
a_2	0.535	0.667	0.629	0.552	0.687	0.650
b_2	0.586	0.552	0.546	0.438	0.427	0.442

Table 6.9. *Disturbance rejection PID controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.473	1.468	1.531	1.524	1.515	1.592
b_1	−0.970	−0.970	−0.960	−0.735	−0.730	−0.705
a_2	1.115	0.942	0.971	1.130	0.957	0.957
b_2	0.753	0.725	0.746	0.641	0.598	0.597
a_3	0.550	0.443	0.413	0.552	0.444	0.414
b_3	0.948	0.939	0.933	0.851	0.847	0.850

```
   case 2, Kp=0.361*Kc;Ti=0.083*(1.935*k+1)*Tc;
   case 3, Kp=0.509*Kc; Td=0.125*Tc; Ti=0.051*(3.302*k+1)*Tc;
   case 4, Kp=(4.437*k-1.587)/(8.024*k-1.435)*Kc;
      Ti=0.037*(5.89*k+1)*Tc; Td=0.112*Tc;
   end
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

The syntax of the function is

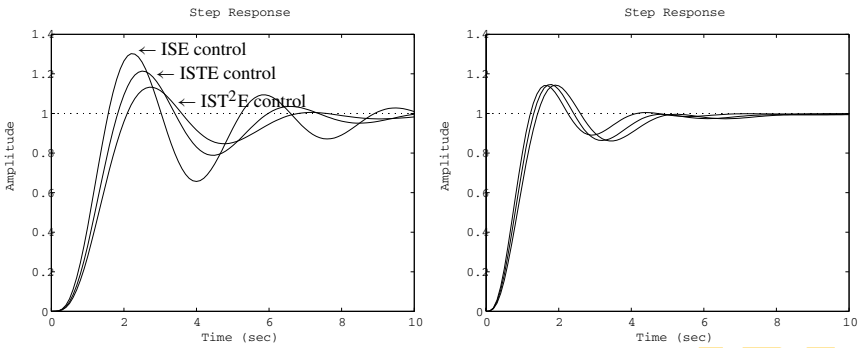
```
[Gc,Kp,Ti,Td,H]=optpid(key,typ,vars)
```

where `key` $= 2, 3, 4$ for PI, normal PID, and PID controllers with D in the feedback path, respectively, and `typ` $= 1, 2$ for set-point and disturbance rejection, respectively. The variable `vars` $= [k, L, T, N, C]$, where C is the criterion type with $C = 1, 2, 3$ for ISE, ISTE, and IST^{2}E criteria, respectively. The returned variables are G_c, the cascade controller object, and K_p,T_i,T_d are the PID controller parameters. H is returned, if `key` $= 4$, as the equivalent feedback transfer function for the structure with the derivative in the feedback path.

Example 6.12. Consider the plant model in Example 6.4. The optimal PI and PID controllers can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); N=10; [k,L,T]=getfod(G);
   f1=figure; f2=figure;
   for iC=1:3
      [Gc,Kp,Ti,Td]=optpid(2,1,[k,L,T,N,iC]);
      figure(f1), G_c=feedback(G*Gc,1); step(G_c,10), hold on,
      [Gc,Kp,Ti,Td]=optpid(3,1,[k,L,T,N,iC]);
      figure(f2), G_c=feedback(G*Gc,1); step(G_c,10), hold on,
   end
```

The relevant closed-loop step responses are shown in Figs. 6.18(a) and (b).



(a) PI control (b) PID control

Figure 6.18. *Closed-loop step responses of optimal controllers.*

PID Design based on ultimate frequency and gain

When the crossover frequency ω_c and the ultimate gain K_c are known, with $T_c = 2\pi/\omega_c$, three types of PID controllers are summarized in Table 6.10, where $\kappa = kK_c$ is the normalized gain of the plant model [70]. The values given were deduced from the relationship between the FOPDT plant parameters and its ultimate gain and frequency.

The corresponding values for the PI controller are given in Table 6.11.

When the relay automatic tuning strategy is used, which will be discussed later in this chapter, the oscillation frequency ω_0 and the magnitude a_0 can be measured. Then, $T_0 = 2\pi/\omega_0$ and $K_0 = 4h/(a_0\pi)$. Assume that $\kappa_0 = kK_0$. ω_0 and K_0 are approximations to ω_c and K_c, but more accurate results can be obtained for the PID controller parameters from Table 6.12.

The PI controllers for disturbance rejection can also be obtained with the direct use of Table 6.13.

Improved gain-phase approach

The gain-phase assignment algorithm can be used to design a PID controller

$$K_p = \frac{m\cos\phi}{|\,G(j\omega_c)\,|} = mK_c\cos\phi,\ T_d = \frac{\tan\phi+\sqrt{4/\alpha+\tan^2\phi}}{2\omega_c},\ T_i = \alpha T_d \tag{6.34}$$

Table 6.10. *PID controller parameters for ISTE criterion.*

PID	Set-point	Disturbance rejection	D in feedback
K_p	$0.509K_c$	$\dfrac{4.434\kappa - 0.966}{5.12\kappa + 1.734}K_c$	$\dfrac{4.437\kappa - 1.587}{8.024\kappa - 1.435}K_c$
T_i	$0.051(3.302\kappa + 1)T_c$	$\dfrac{1.751\kappa - 0.612}{3.776\kappa + 1.388}T_c$	$0.037(5.89\kappa + 1)T_c$
T_d	$0.125T_c$	$0.144T_c$	$0.112T_c$

Table 6.11. *PI controller parameters for ISTE criterion.*

PI	Set-point	Disturbance rejection
K_p	$\dfrac{4.264-0.148\kappa}{12.119-0.432\kappa}K_c$	$\dfrac{1.892\kappa+0.244}{3.249\kappa+2.097}K_c$
T_i	$0.083(1.935\kappa+1)T_c$	$\dfrac{0.706\kappa-0.227}{0.7229\kappa+1.2736}T_c$

Table 6.12. *PID controller parameters for ISTE criterion for autotuning.*

PID	Set-point	Disturbance rejection	D on output
K_p	$0.604K_0$	$\dfrac{6.068\kappa_0-4.273}{5.758\kappa_0-1.058}K_0$	$\dfrac{2.354\kappa_0-0.696}{3.363\kappa_0+0.517}K_0$
T_i	$0.04(4.972\kappa_0+1)T_0$	$\dfrac{1.1622\kappa_0-0.748}{2.516\kappa_0-0.505}T_0$	$0.271\kappa_0T_0$
T_d	$0.130T_0$	$0.15T_0c$	$0.1162T_0c$

Table 6.13. *PI controller parameters for ISTE criterion for autotuning.*

PI	Set-point	Disturbance rejection
K_p	$\dfrac{1.506\kappa_0-0.177}{3.341\kappa_0+0.606}K_0$	$\dfrac{6.068\kappa_0-4.273}{5.758\kappa_0-1.058}K_0$
T_i	$0.055(3.616\kappa_0+1)T_0$	$\dfrac{5.352\kappa_0-2.926}{5.539\kappa_0 5.536}T_0$

where $\alpha = 0.413(3.302\kappa + 1)$ or $\alpha = 1.687\kappa_0$. The constants ϕ and m can be obtained from one of the following two cases:

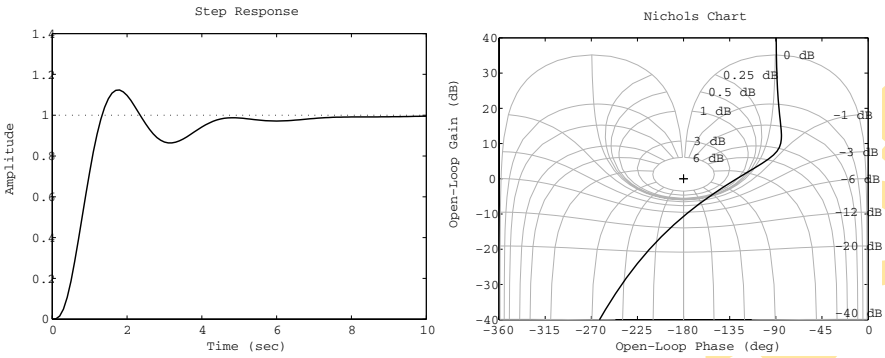
- For the normalized gain κ,

$$\phi = 33.8^\circ(1 - 0.97\mathrm{e}^{-0.45\kappa}), \quad m = 0.614(1 - 0.233\mathrm{e}^{-0.347\kappa}). \tag{6.35}$$

- If the frequency and the gain under automatic tuning are measured, the following approach can be used:

$$\phi = 33.2^\circ(1 - 1.38\,\mathrm{e}^{-0.68\kappa_0}), \quad m = 0.613(1 - 0.262\,\mathrm{e}^{-0.44\kappa_0}). \tag{6.36}$$

The MATLAB function `optpid()` can be used again to solve for the PID controller parameters with the improved gain-phase method. The syntax of the function, for the particular design tasks with this algorithm, is `[`G_c`,`K_p`,`T_i`,`T_d`,`H`]=optpid(key,typ,vars)` where `vars` $= [k, L, T, N, K_c, T_c, \kappa]$ are the relevant parameters of the plant model. As before, if the value of `key` is selected as `key` $= 4$, the effective PID controller, with derivative action in the feedback path, can be designed.



(a) step responses (b) Nichols charts

Figure 6.19. *Responses for the optimal gain-phase margins design.*

Example 6.13. Consider again the plant model in Example 6.4. The PID controller can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [Kc,pm,wc,wm]=margin(G);
   Tc=2*pi/wc; kappa=dcgain(G)*Kc; [k,L,T]=getfod(G);
   N=10; vars=[k,L,T,N,Kc,Tc,kappa];
   [Gc,Kp,Ti,Td,H]=optpid(3,1,vars); G_c=feedback(G*Gc,1); step(G_c),
   figure, nichols(G*Gc); grid; axis([-360,0,-40,40])
```

the controller is

$$G_c(s) = 6.4134\left(1 + \frac{2.6276}{s} + 0.3512s\right).$$

The closed-loop step response and the Nichols chart of the system are obtained as shown in Figs. 6.19(a) and (b), respectively. It can be seen that the responses are satisfactory, compared with the controllers designed using other approaches.

Example 6.14. Let us revisit the original Ziegler–Nichols tuning algorithm. We have seen in Sec. 6.2 that the original Ziegler–Nichols parameter setting formula does not achieve a very satisfactory PID control performance. In this example, we will show, via redesigning the PID controller for the plant model in Example 6.4, a new Ziegler–Nichols parameter setting procedure can give a much improved performance which is close to that achieved by the optimum PID parameter setting method.

Before applying the original Ziegler–Nichols parameter setting formula, the optimal reduced-order model is obtained first to extract the characteristics of the plant model. Then, with this optimally reduced FOPDT model, a PID controller can be designed using the Ziegler–Nichols algorithm. By the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); Gr=opt_app(G,0,1,1); L=Gr.ioDelay;
   T=Gr.den{1}(1)/Gr.den{1}(2); K=Gr.num{1}(2)/Gr.den{1}(2);
   Gc=ziegler(3,[K,L,T,10]); Gc1=optpid(3,1,[K,L,T,10,2]);
   step(feedback(G*Gc,1),feedback(G*Gc1,1))
```

the new Ziegler–Nichols PID controller and the optimum PID controller can be designed. Their step responses are compared in Fig. 6.20. We can see that the new Ziegler–Nichols

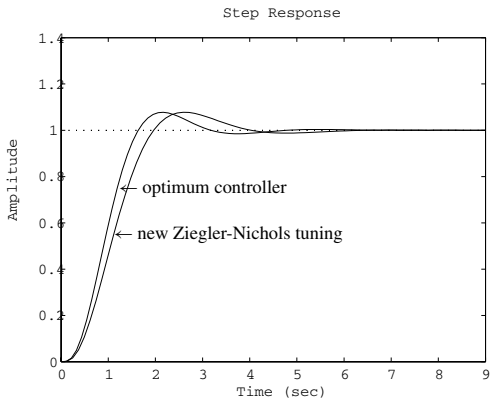


Figure 6.20. *Step responses comparison of two PID controllers.*

parameter setting procedure gives a much improved performance compared with that presented in Example 6.4. In fact, this new Ziegler–Nichols PID controller performs similarly to the optimum PID controller in terms of step response speed and overshoot.

6.4 PID Controller Tuning Algorithms for Other Types of Plants

All the PID tuning algorithms discussed in the previous sections are based on the FOPDT plant models; they cannot be used for many other plant models in practice. A great many PID tuning algorithms have been collected in the handbook [71], where apart from the FOPDT-based algorithms, tuning algorithms for other plant models are also given. Here only a few PID controller algorithms are summarized, with their MATLAB implementations.

6.4.1 PD and PID Parameter Setting for IPDT Models

A widely encountered plant model is described by a mathematical description $G(s) = K\mathrm{e}^{-Ls}/s$, which is referred to as the integrator plus dead time (IPDT) model. This kind of plant model cannot be controlled by the PD and PID controllers using the setting algorithms given in the previous sections.

Since there already exists an integrator in the plant model, an extra integrator in the controller is not required to remove a steady-state error to a step input, but it is needed to remove the output error caused by a steady disturbance at the plant input. PD controllers may also be used to avoid large overshoot. The mathematical models of PD and PID controllers are, respectively,

$$G_{\mathrm{PD}}(s) = K_p(1 + T_d s), \quad G_{\mathrm{PID}}(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right). \tag{6.37}$$

11.5

Table 6.14. *The coefficients of the controller for IPDT models.*

criterion	a_1	a_2	a_3	a_4	a_5
ISE	1.03	0.49	1.37	1.49	0.59
ITSE	0.96	0.45	1.36	1.66	0.53
ISTSE	0.9	0.45	1.34	1.83	0.49

PD and PID parameter setting algorithms were presented in [72], based on various performance indices, and given as

$$\begin{aligned} &\text{PD controller } K_p = \frac{a_1}{KL}, \quad T_d = a_2 L, \\ &\text{PID controller } K_p = \frac{a_3}{KL}, \quad T_i = a_4 L, \quad T_d = a_5 L, \end{aligned} \tag{6.38}$$

where for different criteria, the coefficients a_i can be selected as shown in Table 6.14. The following MATLAB function can be written to implement the above algorithms:

```
function [Gc,Kp,Ti,Td]=ipdtctrl(key,key1,K,L,N)
a=[1.03,0.49,1.37,1.49,0.59; 0.96,0.45,1.36,1.66,0.53;
    0.9,0.45,1.34,1.83,0.49]; s=tf('s'); Ti=inf;
if key==1
   Kp=a(key1,1)/K/L; Td=a(key1,2)*L; Gc=Kp*(1+Td*s/(1+Td/N*s));
else
   Kp=a(key1,3)/K/L; Ti=a(key1,4)*L; Td=a(key1,5)*L;
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
end
```

In the function, `key` is the switch for PD and PID controller selections, with `key` $= 1$ for PD, 2 for PID. The argument for `key` $= 1$ is to set for ISE, ITSE, and ISTSE selections.

6.4.2 PD and PID Parameters for FOIPDT Models

Another category of plant model is defined by a first-order lag and integrator plus dead time (FOIPDT) whose mathematical model is

$$G(s) = \frac{K\mathrm{e}^{-Ls}}{s(Ts+1)}.$$

Since an integrator is contained in the model, an extra integrator is not necessary in the controller to remove the steady-state error to a set point change. Thus, a PD controller may be used if there is no steady state disturbance at the plant. A PD controller setting algorithm is included in [71, 73]:

$$K_p = \frac{2}{3KL}, \quad T_d = T. \tag{6.39}$$

Also a PID setting algorithm is included in [71, 74] such that

$$K_p = \frac{1.111T}{KL^2} \frac{1}{\left[1+(T/L)^{0.65}\right]^2}, \quad T_i = 2L\left[1+\left(\frac{T}{L}\right)^{0.65}\right], \quad T_d = \frac{T_i}{4}. \tag{6.40}$$

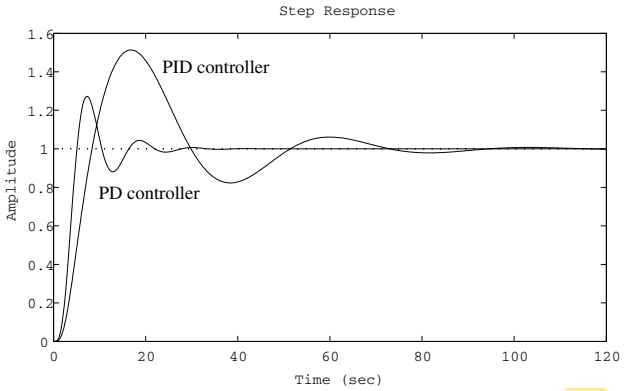


Figure 6.21. *Comparisons of the PID and PD controllers.*

A control design function `foipdt()` is written to implement the two algorithms, where `key` is used to select the structure of the controller, i.e., 1 for PD and 2 for PID. If the parameters K, L, T, N are known, the controller can immediately be designed.

```
function [Gc,Kp,Ti,Td]=foipdt(key,K,L,T,N)
s=tf('s');
if key==1
   Kp=2/3/K/L; Td=T; Ti=inf; Gc=Kp*(1+Td*s/(1+Td*s/N));
else
   a=(T/L)^0.65; Kp=1.111*T/(K*L^2)/(1+a)^2;
   Ti=2*L*(1+a); Td=Ti/4; Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
end
```

Example 6.15. Consider the plant model

$$G(s) = \frac{1}{s(s+1)^4},$$

where there exists an integrator and the rest of the model can be described by an FOPDT model. Thus, the original model can be approximated by an FOIPDT model. The following statements can be used to design PD and PID controllers. The step response of the closed-loop systems are obtained as shown in Fig. 6.21.

```
>> s=tf('s'); G1=1/(s+1)^4; G=G1/s; Gr=opt_app(G1,0,1,1);
   K=Gr.num{1}(2)/Gr.den{1}(2); L=Gr.ioDelay; T=1/Gr.den{1}(2);
   [Gc1,Kp1,Ti1,Td1]=foipdt(1,K,L,T,10);
   [Gc2,Kp2,Ti2,Td2]=foipdt(2,K,L,T,10);
   step(feedback(G*Gc1,1),feedback(G*Gc2,1))
```

The controllers are

$$G_{\mathrm{PD}}(s) = 0.3631\left(1+\frac{2.3334s}{1+0.23334s}\right), \quad G_{\mathrm{PID}}(s) = 0.1635\left(1+\frac{1}{7.9638s}+\frac{1.9910s}{1+0.1991s}\right).$$

It can be seen from the control results that the PD controller is significantly better than the PID controller. This is because the 180° lag given by two integrators makes good control more difficult.

11.5

Table 6.15. *The coefficients of the controller for unstable FOPDT models.*

Criterion	a_1	b_1	a_2	b_2	a_3	b_3	γ
ISE	1.32	0.92	4	0.47	3.78	0.84	0.95
ITSE	1.38	0.9	4.12	0.9	3.62	0.85	0.93
ISTSE	1.35	0.95	4.52	1.13	3.7	0.86	0.97

6.4.3 PID Parameter Settings for Unstable FOPDT Models

In practical control systems, the plant model may approximate an unstable FOPDT model, i.e.,

$$G(s) = \frac{Ke^{-Ls}}{Ts-1}.$$

The following algorithms may be used to design the PID controller, [72].

$$K_p = \frac{a_1}{K}A^{b_1}, \quad T_i = a_2 T A^{b_2}, \quad T_d = a_3 T\left[1 - b_3 A^{-0.02}\right]A^{\gamma}, \tag{6.41}$$

where $A = L/T$. For different criterion, the coefficients a_i, b_i, γ of the PID controller can be obtained in Table 6.15. Based on the algorithm, a PID controller design function for unstable FOPDT models can be written such that

```
function [Gc,Kp,Ti,Td]=ufolpd(key,K,L,T,N)
Tab=[1.32, 0.92, 4.00, 0.47, 3.78, 0.84, 0.95;
     1.38, 0.90, 4.12, 0.90, 3.62, 0.85, 0.93;
     1.35, 0.95, 4.52, 1.13, 3.70, 0.86, 0.97];
a1=Tab(key,1); b1=Tab(key,2); a2=Tab(key,3); b2=Tab(key,4);
a3=Tab(key,5); b3=Tab(key,6); gam=Tab(key,7); A=L/T;
Kp=a1*A^b1/K; Ti=a2*T*A^b2; Td=a3*T*(1-b3*A^(-0.02))*A^gam;
s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
```

6.5 PID_Tuner: A PID Controller Design Program for FOPDT Models

Hundreds of PID parameter tuning algorithms have been collected in the handbook [71]. Many of the methods are based on the FOPDT plant models. Thus, a GUI is designed, which can be used to design PID-type controllers, and also a closed-loop simulation for the designed controllers can be obtained. With the interface, the following procedures can be used to design PID controllers:

1. Enter `pid_tuner` under the MATLAB prompt. The interface in Fig. 6.22 is given, which can be used to design PID-type controllers.
2. Click the Plant model button; a dialog box will be given to prompt you to enter the plant model. Any single input–single output (SISO) continuous model, with or without time delays, can be defined. The button Modify Plant Model can be used to modify the plant models.
3. Once the plant model is specified, the Get FOPDT parameters button can be clicked to extract the FOPDT parameters, i.e., to find the parameters K, L, T. Many different

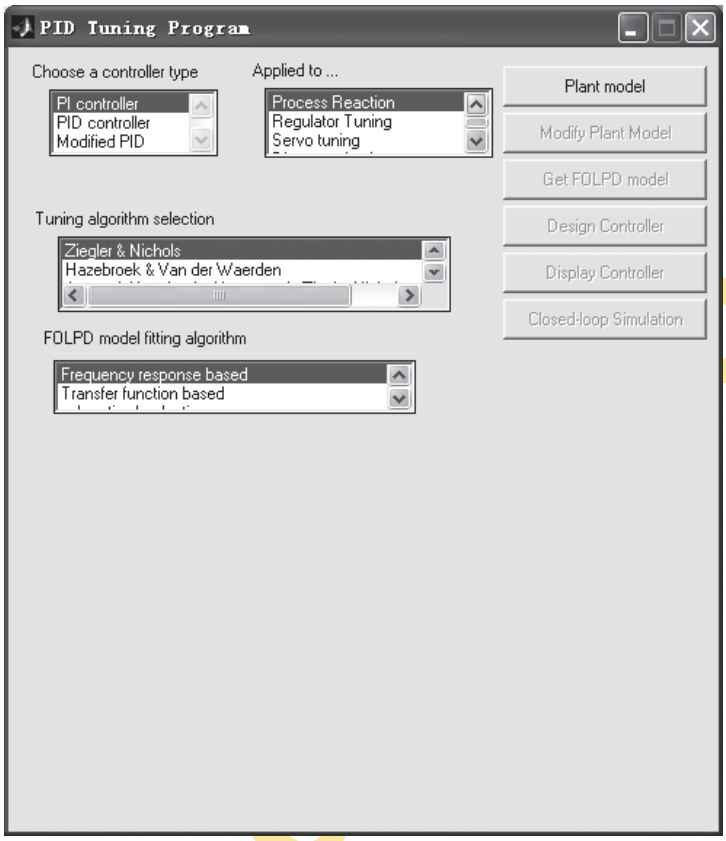


Figure 6.22. *PID controller design interface.*

methods can be used to extract the parameters, for instance, using the optimum fitting methods. The fitting algorithms can be selected via the FOPDT model parameters fitting list box.

4. With the K, L, T parameters, the controller can be designed. The controller type can be selected by the combinations of the list boxes Choose controller type, Apply to, and Tuning algorithm selection, which provides the algorithms in [75].
5. The Design Controller button can be used to design the relevant PID controller.
6. The Closed-loop Simulation button can be used to show the closed-loop step response of the system under the controllers designed.

Example 6.16. For the plant model

$$G(s) = \frac{1}{(s+1)^6},$$

click Plant model to enter the model. The dialog box shown in Fig. 6.23 is displayed, and the numerator, denominator, coefficient vectors, and delay constant can be entered. Then click the Apply button to model the input procedure.

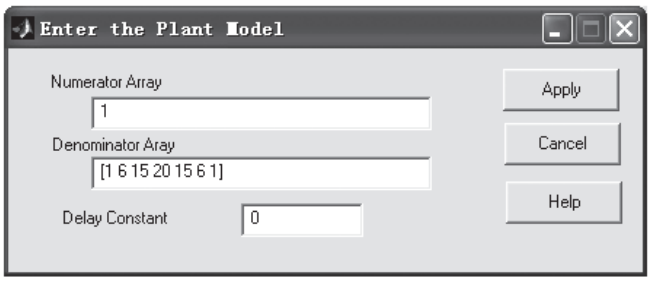


Figure 6.23. *Dialog box of plant model input.*

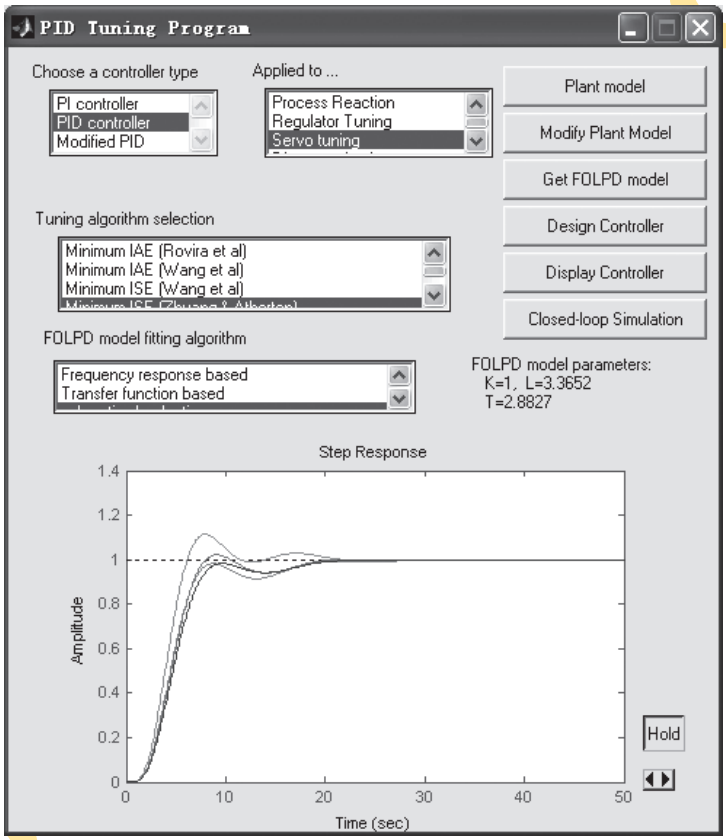


Figure 6.24. *PID controller design and display.*

To design a controller, the FOPDT parameters should be obtained first. The fitting algorithms can be selected as the sub optimal reduction item; the button Get FOPDT model can then be clicked to extract the model parameters, as shown in Fig. 6.24.

The controller can be obtained by the Design Controller button. For instance, the Minimum IAE (Wang et al) item can be used to design the controller

$$G_c(s) = 0.936172\left(1 + \frac{1}{4.565340s} + 1.062467s\right).$$

Click the Closed-loop Simulation button to show the closed-loop step response. One may click the Hold button to hold the results. The step responses under different controllers can be displayed together. So, this feature can be used to compare different algorithms, as shown in Fig. 6.24.

6.6 Optimal Controller Design

Optimal control is defined as the optimization of certain predefined performance indices. For instance, commonly used performance indices can be the ones in (3.50). Sometimes, parametric objective functions may be used, for example, the linear quadratic optimal regulator problem, where the two weighting matrices $\boldsymbol{Q}$, $\boldsymbol{R}$ need to be defined. There is as yet no universally accepted way to define these two matrices.

In this section, we first summarize and illustrate some solutions to unconstrained and constrained optimization problems using MATLAB. Then the method can be applied to optimal controller design problems. Finally, a MATLAB interface optimal controller designer (OCD) for optimal controller design is presented.

6.6.1 Solutions to Optimization Problems with MATLAB

Unconstrained optimization problems

The mathematical formulation of the unconstrained optimization problem is

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}), \tag{6.42}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$. The interpretation of the formula is: find the vector $\boldsymbol{x}$ such that the objective function $F(\boldsymbol{x})$ is minimized. If a maximization problem is treated, the objective function can be changed to $-F(\boldsymbol{x})$ such that it can be converted to a minimization problem.

A MATLAB function `fminsearch()` is provided using the well-established simplex algorithm [76]. The syntax of the function is

```
[x, fopt, key, c]=fminsearch(Fun, x0, OPT)
```

where `Fun` is a MATLAB function, an inline function, or an anonymous function to describe the objective function. The variable $\boldsymbol{x}_0$ is the starting point for the search method. The argument `OPT` contains further control options for the optimization process.

Example 6.17. If a function with two variables is given by $z = f(x, y) = (x^2-2x)\mathrm{e}^{-x^2-y^2-xy}$ and the minimum point is required, one should first introduce a vector $\boldsymbol{x}$ for the unknown variables x and y. One may select $x_1 = x$ and $x_2 = y$. The objective function can be rewritten as $f(\boldsymbol{x}) = (x_1^2 - 2x_1)\mathrm{e}^{-x_1^2-x_2^2-x_1x_2}$. The objective function can be expressed as an anonymous function such that

```
>> f=@(x)[(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2))];
```

If one selects an initial search point at $(1, 1)$, the minimum point can be found with the statements

```
>> x0=[0; 0]; x=fminsearch(f,x0.)
```

Then the solution obtained is $\boldsymbol{x} = [0.6110, -0.3055]^{\mathbf{T}}$.

Constrained optimization problems

The general form of the unconstrained optimization problem is

$$\boldsymbol{x} \text{ s.t.} \begin{cases} \min & F(\boldsymbol{x}) \\ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{B} \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq} \\ \boldsymbol{x}_{\mathrm{m}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\mathrm{M}} \\ \boldsymbol{C}(\boldsymbol{x}) \leq \boldsymbol{0} \\ \boldsymbol{C}_{eq}(\boldsymbol{x}) = \boldsymbol{0} \end{cases} \tag{6.43}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. The constraints are classified as linear equality constraints $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq}$, linear inequality constraints $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{B}$, and nonlinear constraints $\boldsymbol{C}_{eq}(\boldsymbol{x}) = \boldsymbol{0}$ and $\boldsymbol{C}(\boldsymbol{x}) \leq \boldsymbol{0}$. The upper and lower bounds of the optimization variables can also be defined such that $\boldsymbol{x}_{\mathrm{m}} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\mathrm{M}}$.

The interpretation of the optimization problem is: find the vector $\boldsymbol{x}$, which minimizes the objective function $F(\boldsymbol{x})$, while satisfying all the constraints.

A MATLAB function `fmincon()` can be used to solve constrained optimization problems. The syntax of the function is

```
[x, f_opt, key, c]=fmincon(Fun, x0, A, B, Aeq, Beq, xm, xM, CFun, OPT)
```

where `Fun` again could be M-functions, inline functions, or anonymous functions for the objective function, and $\boldsymbol{x}_0$ is the starting search point. The nonlinear constraints can be described by the MATLAB function `CFun`.

Example 6.18. Consider the following nonlinear programming problem:

$$\boldsymbol{x} \text{ s.t.} \begin{cases} \min & [1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3]. \\ x_1^2+x_2^2+x_3^2-25=0 \\ 8x_1+14x_2+7x_3-56=0 \\ x_1, x_2, x_3 \geq 0 \end{cases}$$

The objective function can be expressed with an anonymous function

```
>> f=@(x)[1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3)];
```

Also, the two constraints are equalities, one of which is nonlinear. The nonlinear constraints can be described in the following MATLAB function, where two constraint variables `ceq` and `c` are returned. Since there is no inequality constraint, the variable `c` returns an empty matrix.

```
function [c,ceq]=opt_con(x)
ceq=x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25; c=[];
```

The linear equality constraint can be expressed by the $\boldsymbol{A}_{\text{eq}}$, $\boldsymbol{B}_{\text{eq}}$ matrices, while the linear inequality matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ should be empty ones, since there is no linear inequalities in the problem. Selecting an initial search position at $\boldsymbol{x}_0 = [1, 1, 1]^{\mathbf{T}}$, the problem can then be solved using the following statements:

```
>> x0=[1;1;1]; xm=[0;0;0]; xM=[]; A=[]; B=[]; Aeq=[8,14,7]; Beq=56;
   [x,f_opt,c,d]=fmincon(f,x0,A,B,Aeq,Beq,xm,xM,'opt_con')
```

The optimum solution can then be found, where $\boldsymbol{x}^* = [3.5121, 0.2170, 3.5522]^{\mathbf{T}}$ and $f_{\text{opt}} = 961.7151$.

6.6.2 Optimal Controller Design

With the powerful tools provided in MATLAB, many optimal control problems can be converted in to conventional optimization problems. With the above-mentioned functions, some optimal controller problems can be easily solved. Although not allowing elegant analytical solutions, numerical methods are extremely powerful practical techniques for controller design.

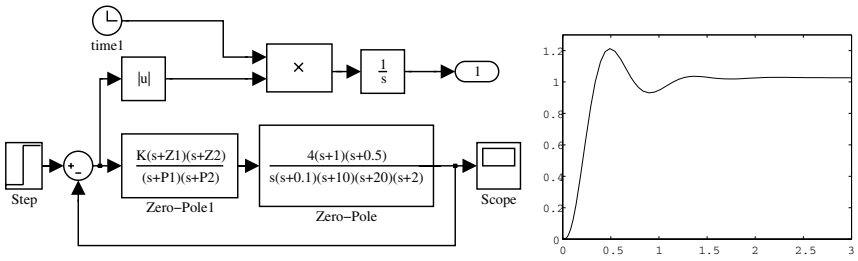
Example 6.19. Assume that

$$G(s) = \frac{10(s+1)(s+0.5)}{s(s+0.1)(s+2)(s+10)(s+20)}.$$

The phase lead-lag controllers can be designed using the method in Sec. 5.1. Here optimal controller design is explored. Integral-type criteria are very suitable for servo control problems. Given a plant model, a Simulink block diagram can be established as shown in Fig. 6.25(a), where the ITAE criterion can be evaluated as shown.

In order to minimize the ITAE criterion, the following MATLAB function can be written to describe the objective function:

```
function y=c6optml(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % assign variable into MATLAB workspace
[t,xx,yy]=sim('c6moptm1.mdl',3); y=yy(end);   % evaluate objective function
```



(a) Simulink model (file: c6moptm1.mdl) (b) closed-loop response

Figure 6.25. *Phase lead-lag controller and system response.*

The `assignin()` function can be used to assign the variables in the MATLAB workspace, and the model parameters can be defined in the optimization variable vector $\boldsymbol{x}$. The following MATLAB statements can be used to solve the optimization problem:

```
>> x0=20*ones(5,1); x=fminsearch('c6optm1',x0)
```

and the parameters are returned in the variable $\boldsymbol{x}$, from which the controller model can be written as

$$G_c(s) = 243.77\frac{(s+53)(s+66.58)}{(s+38.28)(s+62.09)}.$$

Under this controller, the step response of the system is shown in Fig. 6.25(b).

In practical calculation, when the zero of the controller is very small, the computation may become extremely slow. To solve the problem, a suitable constraint to ensure that all the five variables do not become smaller than 0.01 can be introduced. The following statements can then be used to solve the problem:

```
>> x=fmincon('c6optm1',x0,[],[],[],[],0.01*ones(5,1))
```

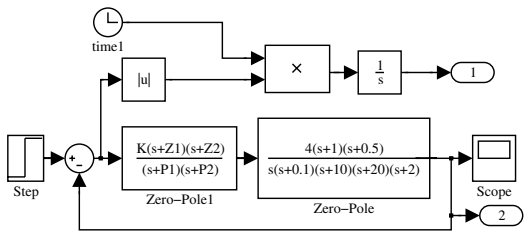
Based on the numerical optimization technique, an extra constraint can be introduced. For instance, if one wants to reduce the overshoot such that $\sigma \leq 3\%$, a new Simulink model can be established as shown in Fig. 6.26(a). The objective function can be rewritten as

```
function y=c6optm2(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % Assign variables to MATLAB workspace
[t,xx,yy]=sim('c6moptm2.mdl',3); y=yy(end,1);   % Evaluate objective function
if max(yy(:,2))>1.03, y=1.2*y; end % update objective function
```

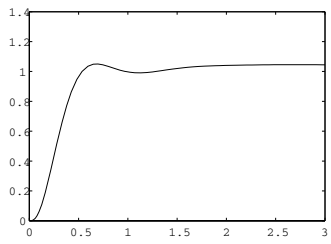
It can be seen from the last sentence that if the overshoot is too large, one can increase the objective function manually.

The following statements can be given to solve the problem, and the closed-loop step response of the system is shown in Fig. 6.26(b).

```
>> x=fmincon('c6optm2',x0,[],[],[],[],0.01*ones(5,1))
```



(a) modified Simulink model (file:c6moptm2.mdl)



(b) closed-loop step response

Figure 6.26. *Modified simulation model and response.*

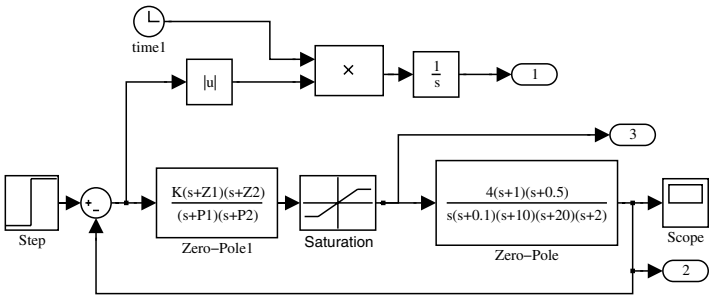


Figure 6.27. *The Simulink model with saturations (file: c 6moptm3.mdl).*

The controller model

$$G_{c_2}(s) = 161.4965\frac{(s+43.1203)(s+55.7344)}{(s+28.4746)(s+61.0652)}$$

can be designed.

Considerations on implementation of the controller are often neglected in theoretical control solutions. It can be seen that for a unit step input, this controller gives an initial output of 200, which is too high. It could cause hardware problems with a bad design and saturate the actuator leading to nonlinear operation. However, if saturation is included in the actuator, the resulting response can be easily solved using numerical methods, since one can simply add a saturation block in the Simulink model.

Example 6.20. Consider again the controller design problem. Assuming that the control signal should be kept within ± 20, the Simulink model can be modified as shown in Fig. 6.27, and the objective function can be rewritten as

```
function y=c6optm3(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % assign variables in MATLAB workspace
[t,xx,yy]=sim('c6moptm3.mdl',15); y=yy(end,1);   %   evaluate objective function
if max(yy(:,2))>1.03, y=1.4*y; end    % update the objective function
```

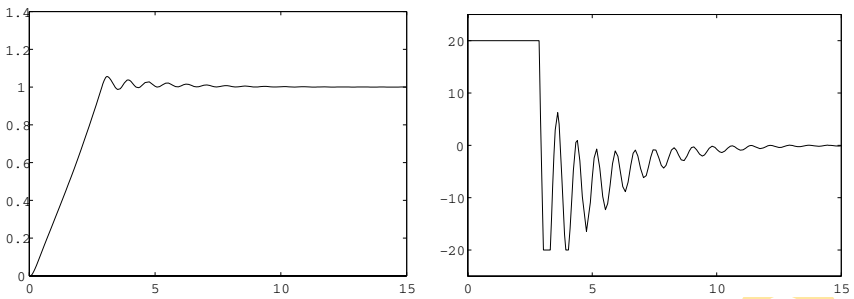
The following statements can be used to search for the optimum controller for the system:

```
>> x=fmincon('c6optm3',x0,[],[],[],[],0.01*ones(5,1))
```

and the controller

$$G_c(s) = 37.1595\frac{(s+142.6051)(s+62.6172)}{(s+20.3824)(s+27.6579)}$$

can be designed. The output signal and the control signal under such a controller can be obtained as shown in Fig. 6.28. It can be seen that the control results are satisfactory.



(a) output signal (b) control signal

Figure 6.28. *Step response of the system when saturations are introduced.*

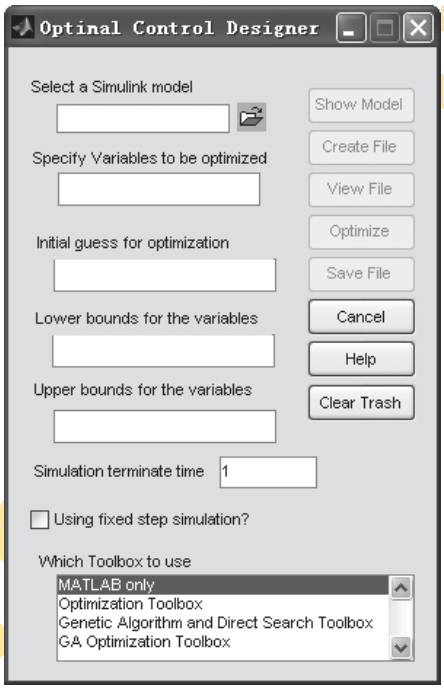


Figure 6.29. *OCD interface.*

6.6.3 A MATLAB/Simulink-Based Optimal Controller Designer and Its Applications

From the examples in the previous section, using numerical optimization algorithms, optimal controller design can be made simple. In this section, we will introduce a Matlab/Simulink-based optimal controller designer (OCD) with some application examples.

The procedures for applying the OCD program are as follows:

1. Type `ocd` at the MATLAB prompt; the main interface is shown in Fig. 6.29. The program can be used in optimal controller design.

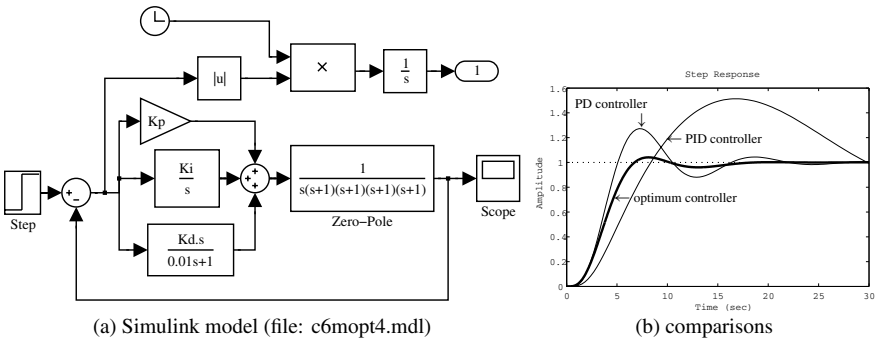
2. A Simulink model can be established and the model should contain at least two elements: the undetermined variable names and the outport reflecting the optimum criterion. For instance, in the PI controller design problem, the two variables K_p and K_i can be assigned. The ITAE criterion can be represented in the Simulink model as outport 1.
3. Fill in the Simulink model name in the Select a Simulink model edit box.
4. Fill in the variable names to be optimized in the Specify Variables to be optimized edit box, with variable names separated with commas.
5. Estimate the terminate time for the error to become zero and enter it in the Simulation terminate time edit box.
6. Click Create File to automatically generate a MATLAB function `optfun_*.m` and click Clear Trash to delete the temporary objective function files.
7. Click Optimize to start, the optimization process. The optimal variables can be obtained. Sometimes, the button should be clicked again to ensure more accurate optimum solutions. The functions `fminsearch()`, `nonlin()` and `fmincon()` can be called automatically for parameter optimization.
8. The upper and lower bounds to the variables can also be used, and initial search point can be specified, if necessary.

Example 6.21. Consider the FOIPDT-type plant model in Example 6.15; i.e., the plant model is given by

$$G(s) = \frac{1}{s(s+1)^4}.$$

The Simulink model for the PID control, with ITAE descriptions, is established as shown in Figure 6.30(a), and it is saved in the file `c6mopt4.mdl`.

Fill in the Simulink model name in the Select a Simulink model edit box, for instance, fill in `c6mopt4` for this example. The variable names to be optimized, `Kp,Ki,Kd` should be entered in the Specify Variables to be optimized edit box, and enter 30 in the Simulation terminate time edit box. Then click the Create File button to automatically generate the MATLAB function to describe the objective function



(a) Simulink model (file: c6mopt4.mdl) (b) comparisons

Figure 6.30. *PID control model and response comparisons.*

```
function y=optfun_2(x)
assignin('base','Kp',x(1));
assignin('base','Ki',x(2));
assignin('base','Kd',x(3));
[t_time,x_state,y_out]=sim('c6mopt4.mdl',[0,30.000000]);
y=y_out(end);
```

where the second, third, and fourth lines in the code will assign the variables in vector $\boldsymbol{x}$ to the variables K_p, K_i, K_d in the MATLAB workspace. Simulation is then performed to calculate the objective function.

Click the Optimize button to initiate the optimization process. In the meantime, the scope window should be opened to visualize the optimization process. After optimization, the optimum PID controller will be obtained as

$$G_c(s) = 0.2583 + \frac{0.0001}{s} + \frac{0.7159s}{0.01s+1}$$

which minimizes the ITAE criterion. It can be seen that $K_i = 0.0001$ is very small, which can be neglected, and thus a PD controller is sufficient for the system. The closed-loop step response is shown in Fig. 6.30(b). It can be seen that the control response is highly superior to the one obtained in Example 6.15.

Example 6.22. The OCD program is not restricted to simple PID controller problems. It can also be used for complicated system models such as the cascade PI control system shown in Fig. 2.11.

To solve the problem, the Simulink model shown in Fig. 6.31 can be established, and saved as `c6model2.mdl`. Note that four undetermined parameters `Kp1`, `Ki1`, `Kp2`, `Ki2` should be optimized. The ITAE criterion can be defined. Starting the OCD, the model name `c6model2` should be entered into the Select a Simulink model edit box, and in the Specify Variables to be optimized edit box, `Kp1,Ki1,Kp2,Ki2` should be filled in. Also, in the Simulation terminate time edit box, one may fill in 0.6. Click the Create File to generate the MATLAB function. One may design the controllers by clicking the Optimize button, and the controllers, which minimize the ITAE criterion, can be found as $K_{p_1} = 37.9118$,

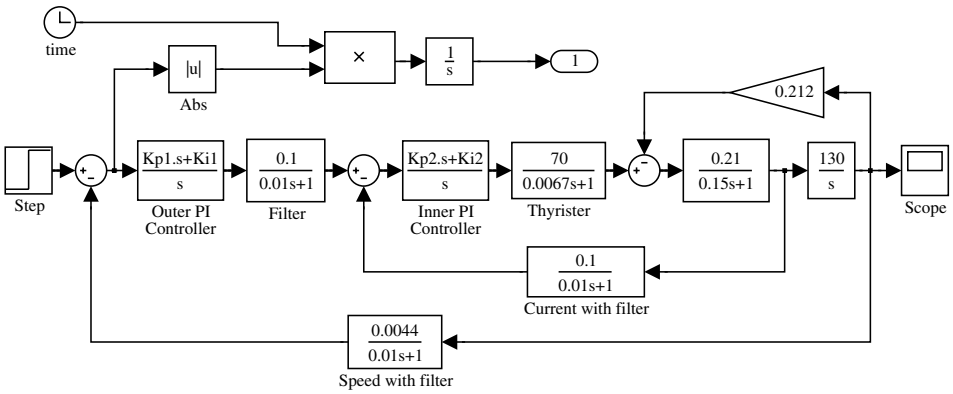


Figure 6.31. *Simulation model of cascade PI control (file: c6model2.mdl).*

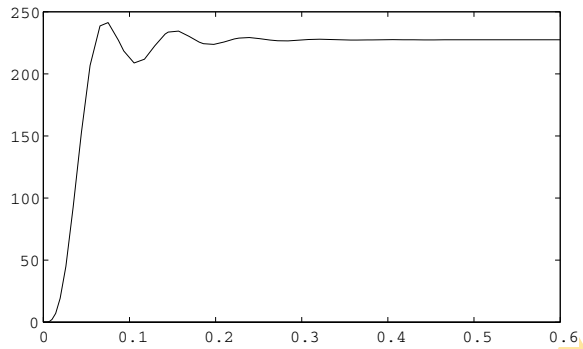


Figure 6.32. *Optimal control of the system.*

$K_{i_1} = 12.1855$, $K_{p_2} = 10.8489$, and $K_{i_2} = 0.9591$, i.e., the controllers are

$$G_{c_1}(s) = 37.9118 + \frac{12.1855}{s} \quad \text{and} \quad G_{c_2}(s) = 10.8489 + \frac{0.9591}{s}.$$

Under these controllers, the step response of the closed-loop system can be obtained as shown in Fig. 6.32. It can be seen that the response is satisfactory.

It can be seen from the previous examples that the OCD program is quite versatile in finding the optimal controllers. However, in some applications, the OCD may not find a solution due to the poorly posed problem or because a good initial search point has not been found. This can be a drawback in conventional optimization algorithms, but many such problems can be avoided by intelligent use based on an understanding of the system behavior.

The genetic algorithm (GA) [77] allows the optimization search from many initial points in a parallel manner. The Genetic Algorithm Optimization Toolbox (GAOT) [78] provides a series of MATLAB-based functions for solving optimization problems using genetic algorithms. This toolbox is used with the OCD program, and the facility is useful in solving problems where conventional optimization methods cannot easily find an initial feasible search point. The GA Optimization Toolbox is the last list box in Fig. 6.29.

Example 6.23. Consider an unstable plant model

$$G(s) = \frac{s+2}{s^4+8s^3+4s^2-s+0.4}.$$

By the direct use of the OCD program, a feasible PID controller cannot be designed. However, one may still establish a Simulink model as shown in Fig. 6.33, which is the same as the previous examples.

In order to ensure that the control action is not too large, a saturation element can be appended to the controller, with the saturation width of $\Delta = 5$. From the OCD program, with the GAOT selection, the optimal PID controller can be designed as

$$G_c(s) = 47.8313 + \frac{0.2041}{s} + \frac{55.3632s}{0.01s+1}.$$

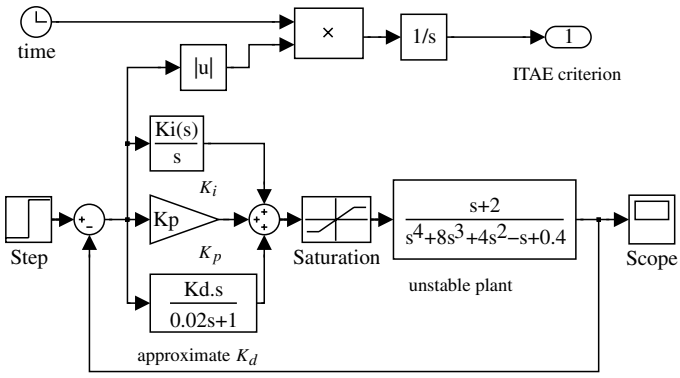


Figure 6.33. *Simulink model for PID control (file: c6munsta.mdl).*

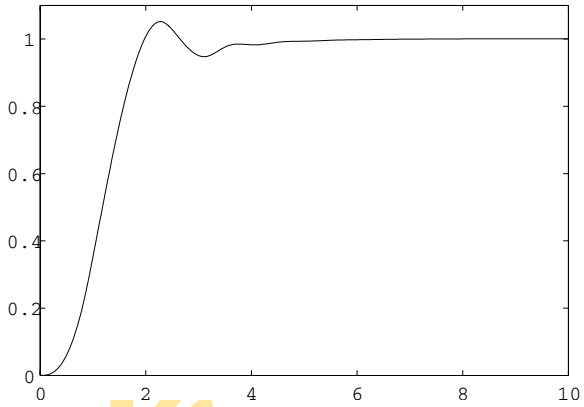


Figure 6.34. *Simulation results for an unstable plant with a PID controller.*

The step response of the closed-loop system under the optimal controller is shown in Fig. 6.34. It can be seen that the PID controller can still be designed, with the help of GAs, and the transient response is satisfactory.

6.7 More Topics on PID Control

6.7.1 Integral Windup and Anti-Windup PID Controllers

A Simulink model for the study of the phenomenon of integrator windup is shown in Fig. 6.35.

The plant model is given by

$$G(s) = \frac{10}{s^4 + 10s^3 + 35s^2 + 50s + 24},$$

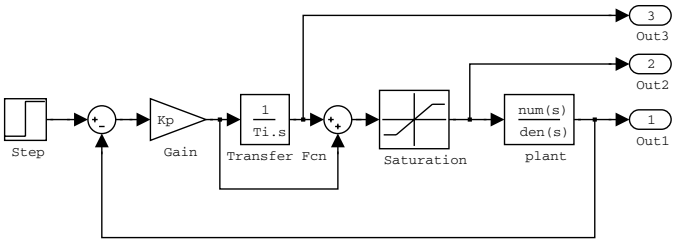


Figure 6.35. *Integrator windup demonstration (file: c6mwind.mdl).*

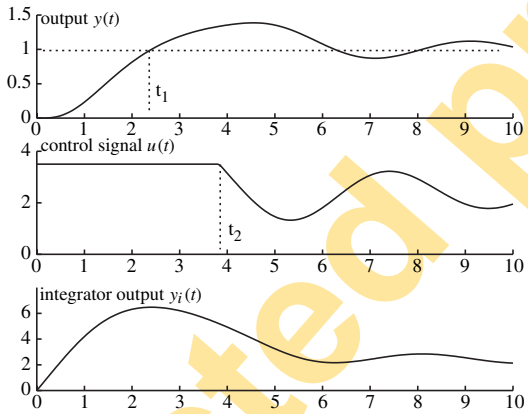


Figure 6.36. *Integrator windup demonstration.*

and the parameters for the PI controller are given by $K_p = 5.04$ and $T_i = 1.124$. With an actuator saturation nonlinear element given by $u_m = 3.5$, the related signals in the PI controlled system are shown in Fig. 6.36. When there is an initial set-point change in $r(t)$, the error signal is initially so large that the control signal $u(t)$ quickly reaches its actuator saturation limit. Even when the output signal reaches the reference value at the time t_1, which gives a negative error signal due to the large value of the integrator output, the control signal still remains at the saturation value u_m, which causes the output of the system to continuously increase until it reaches the time t_2, and the negative action of the error signal begins to have effect. This phenomenon is referred to as the integrator windup action, which is unwanted in control applications. Therefore, we need to briefly introduce different antiwindup PID controllers for use in practice. We shall use Simulink for illustration.

An antiwindup PID controller is provided as an icon in the Simulink environment, and the internal structure is shown in Fig. 6.37. The signal reflecting the actuator saturation is fed into the integrator action, which is determined by a ratio $1/T_t$. For instance, one can simulate the PID control system in the previous example using the Simulink model as shown in Fig. 6.38(a). For different T_t, the output signals are compared in Fig. 6.38(b). It can be seen that for smaller values of T_t, the windup phenomenon can be reduced more significantly.

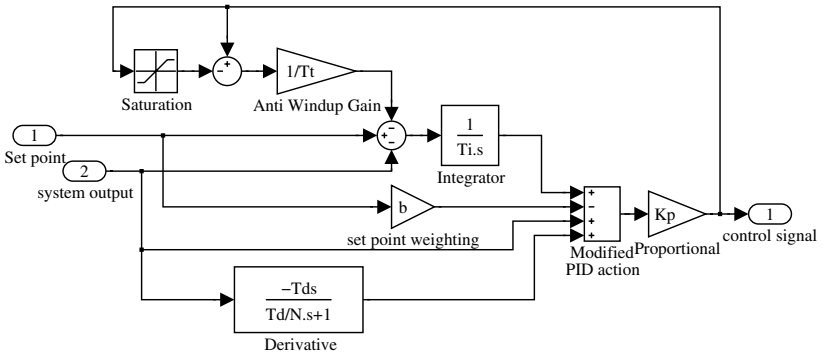
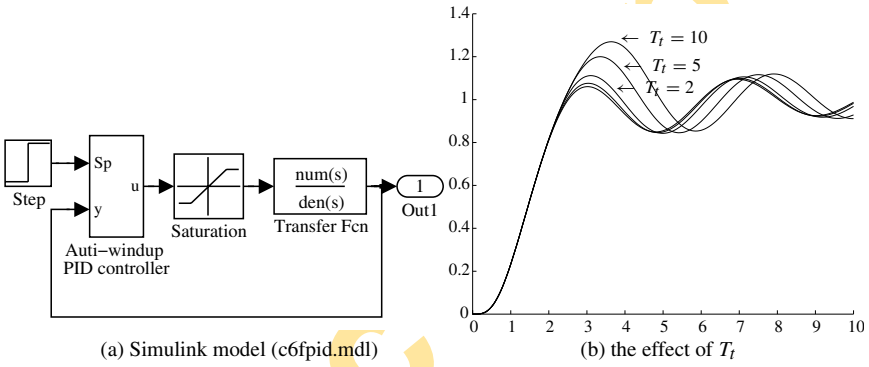


Figure 6.37. *Anti-windup PID structure (file: c6awpid.mdl).*



(a) Simulink model (c6fpid.mdl) (b) the effect of T_t

Figure 6.38. *Effect of anti-windup PI controllers.*

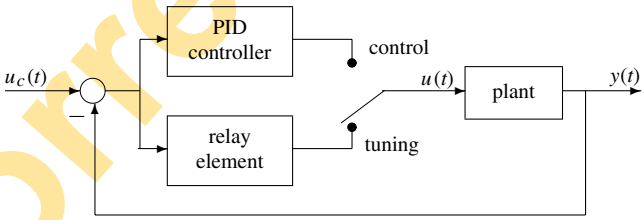
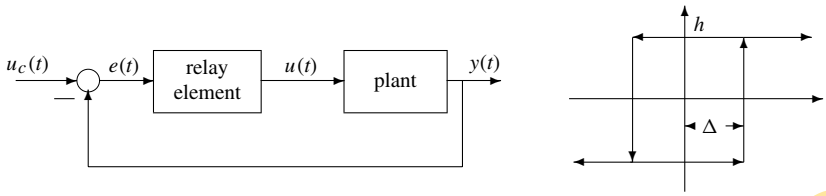


Figure 6.39. *Structure of a relay automatic tuning PID controller.*

6.7.2 Automatic Tuning of PID Controllers

An automatic tuning (also known as autotuning or autotuner) PID controller strategy is proposed by Åström and Hägglund [61]. Now the commercial automatic tuning PID controllers are available from most hardware manufacturers.

The structure of the relay-type of automatic tuning is shown in Fig. 6.39, and it can be seen that the two modes are alternated by the use of switching. When the operator feels the need to adjust the parameters of the PID controller, he or she can simply press a button



(a) relay control block diagram (b) typical relay element

Figure 6.40. *Nonlinear model of relay control.*

to switch the process to the tuning mode, and the parameters can be tuned automatically. When this tuning task is completed, the process can be switched back to normal feedback control mode.

Under the tuning mode, the system is equivalent to the structure shown in Fig. 6.40(a), and the typical relay nonlinearity is shown in Fig. 6.40(b). Several approaches can be used to determine the crossover frequency ω_c and the ultimate gain K_c. The describing function approach is the theoretical basis for relay autotuning analysis, and Tsypkin's method (see Atherton [51]) can also be applied as described below.

Determining ω_c and K_c with the describing function method

In the describing function approach [51], one can approximately represent the static nonlinear element by an equivalent gain in analyzing the so-called limit cycles. Such a gain is referred to as the describing function of the nonlinearity and is in fact input amplitude dependent. For different nonlinear functions, the describing functions may also be different; a comprehensive study of describing functions can be found in [51].

The limit cycle, or oscillation, can be approximately determined by finding the intersection of the Nyquist plot of the plant model with the negative reciprocal of the describing function $N(a)$, as illustrated in Fig. 6.41(a), which means that the conditions when the oscillation occurs are

$$1 + N(a)G(s)\,|_{s=\mathrm{j}\omega_c} = 0, \quad \text{i.e.,} \quad G(\mathrm{j}\omega_c) = -\frac{1}{N(a)}. \tag{6.44}$$

The describing function of the system with relay nonlinearity given in Fig. 6.40(b) is that

$$N(a) = \frac{4h}{\pi a^2}\left(\sqrt{a^2 - \Delta^2} - \mathrm{j}\Delta\right), \tag{6.45}$$

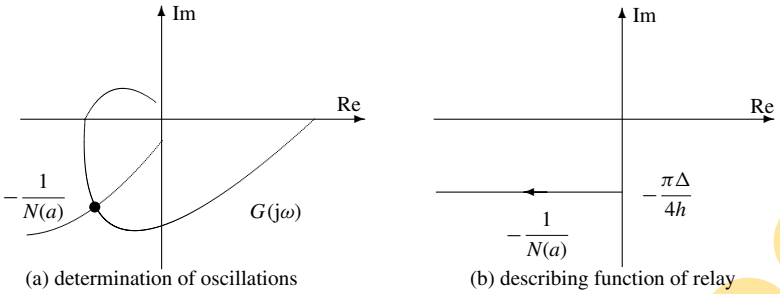
from which the negative reciprocal of the describing function $N(a)$ is simply

$$-\frac{1}{N(a)} = -\frac{\pi}{4h}\sqrt{a^2 - \Delta^2} - \mathrm{j}\frac{\pi\Delta}{4h}, \tag{6.46}$$

which is just a straight line as shown in Fig. 6.41 (b).

The crossover frequency ω_c and the ultimate gain K_c can be obtained. For simplicity, assume that $\Delta = 0$. Then, the describing function can be simplified to $N(a) = 4h/(\pi a)$. So, immediately, one has

$$K_c = \frac{4h}{\pi a}, \quad T_c = \frac{2\pi}{\omega_c}. \tag{6.47}$$



(a) determination of oscillations (b) describing function of relay

Figure 6.41. *Determination of the magnitude and frequency of oscillations.*

Determining ω_c and K_c with Tsypkin's method

The describing function method is essentially based on the principle of fundamental harmonic equivalence. Tsypkin's method, on the other hand, can be used when more accurate analysis of relay systems is required, where the higher-order harmonics need to be considered apart from the fundamental one, for relay nonlinearities.

The Fourier series expansion of the square wave signal, which is the output of the relay action, can be written as

$$y(t) = \sum_{n=1(2)}^{\infty} \frac{4h}{n\pi} \sin n\omega(t - t_1), \tag{6.48}$$

where "(2)" represents a step of 2, i.e., only odd harmonics are considered since the relay function is an odd function. The Fourier series expansion of the output signal can then be written as

$$c(t) = \sum_{n=1(2)}^{\infty} \frac{4h}{n\pi} g_n \sin[n\omega(t - t_1) + \phi_n] \tag{6.49}$$

with g_n and ϕ_n the magnitude and phase of the plant model, respectively, i.e., $G(nj\omega) = g_n e^{j\phi_n}$. If the external input to the system is 0, then $x(t) = -c(t)$, and the switching point satisfies $x(t_1) = \delta$, $\dot{x}(t_1) < 0$. The locus $A(\omega)$ can be defined as

$$\mathrm{Re}[A_G(\theta, \omega)] = \sum_{n=1(2)}^{\infty} \Big[V_G(n\theta) \sin(n\theta) + U_G(n\theta) \cos(n\theta) \Big], \tag{6.50}$$

$$\mathrm{Im}[A_G(\theta, \omega)] = \sum_{n=1(2)}^{\infty} \left[\frac{1}{n} V_G(n\theta) \cos(n\theta) - U_G(n\theta) \sin(n\theta) \right], \tag{6.51}$$

where $G(nj\omega) = U_G(n\omega) + jV_G(n\omega)$. Assume that $t_1 = 0$. The magnitude and frequency of the limit cycles can be solved from

$$\mathrm{Im}[A_G(0, \omega) + A_G(\omega\Delta t, \omega)] = -\frac{\pi\delta}{2h} \tag{6.52}$$

and with the constraints $\mathrm{Re}[A_G(0, \omega) - A_G(\omega\Delta t, \omega)] < 0$. If the relay element is symmetrical, then one has

$$\mathrm{Im}[A_G(0, \omega)] = -\frac{\pi\delta}{4h}. \tag{6.53}$$

6.7.3 Control Strategy Selections

It has been pointed out in some references, such as [60], that PID controllers can be used only for plants with relatively small time delay (or equivalent delays). When the delay constant increases, the PID controller cannot guarantee good responses. In fact, apart from the traditional PID control structure, other control strategies may also be used to deal with such cases. This leaves us with the following question: In practical applications, what kind of controller structure should be used to design a usable controller for a given plant model?

Such a question is well studied in [79], where the normalized parameters τ and κ are introduced, from which different control strategies are suggested, as summarized in Table 6.16, where apart from the τ and κ parameters, τ_2 and κ_2 are also introduced for the plant model given by

$$G(s) = \frac{K_v}{s(1+sT_v)} e^{-sL}$$

with the relations

$$\tau_2 = \frac{L}{T_v}, \quad \kappa_2 = \frac{\lim_{s\to 0} sG(s)}{\omega_c |G(j\omega_c)|} = \frac{1}{2\pi} K_v K_c T_c, \quad \text{and} \quad \tau_2 = \frac{\frac{2}{\pi} + \text{atan}\sqrt{\kappa_2^2 - 1}}{\sqrt{\kappa_2^2 - 1}}. \tag{6.54}$$

It can be seen that Table 6.16, in some sense, can be used as a guide for choosing a suitable controller structure for a given plant model.

Table 6.16. *Controller selection from the plant model.*

Ranges of τ or κ	No precise control necessary	Precise control needed		
		High noise	Low saturation	Low measurement noise
$\tau > 1, \kappa < 1.5$	I control	I+B+C	PI+B+C	PI+B+C
$0.6 < \tau < 1$ $1.5 < \kappa < 2.25$	I or PI control	I+A	PI+A	PI+A+C or PID+A+C
$0.15 < \tau < 0.6$ $2.25 < \kappa < 15$	PI control	PI	PI or PID	PID
$\tau < 0.15, \kappa > 15$or $\tau_2 > 0.3, \kappa_2 < 2$	P or PI control	PI	PI or PID	PI or PID
$\tau_2 < 0.3, \kappa_2 > 2$	PD+E	F	PD+E	PD+E

A represents forward compensation suggested

B represents forward compensation necessary

C represents dead-zone compensation suggested

D represents dead-zone compensation necessary

E represents set-point weighting necessary

F represents for pole placement

Problems

1. For the plant models

$$\text{(a) } G_a(s) = \frac{1}{(s+1)^3}, \quad \text{(b) } G_b(s) = \frac{1}{(s+1)^5}, \quad \text{(c) } G_c(s) = \frac{-1.5s+1}{(s+1)^3},$$

design PID (or PI) controllers using different design algorithms from this chapter and compare the closed-loop behaviors of the controlled systems.

2. Find the FOPDT approximations to the plant models given by

$$\text{(a) } G(s) = \frac{12(s^2-3s+6)}{(s+1)(s+5)(s^2+3s+6)(s^2+s+2)},$$

$$\text{(b) } G(s) = \frac{-5s+2}{(s+1)^2(s+3)^3}\mathrm{e}^{-0.5s},$$

$$\text{(c) } G(z) = \frac{1.0569\times 10^{-5}(z+18.42)(z+1.841)(z+0.3406)(z+0.03405)}{(z-0.8025)(z-0.7866)(z-0.7711)(z-0.7558)(z-0.6703)}, \quad T=0.1,$$

using various algorithms discussed in this chapter. Compare the closeness of the approximation using relevant time and frequency domain analysis techniques.

3. Investigate the disturbance rejection properties of the controllers designed for the plants in Problem 1. Assume that the disturbances are added in the steady-state responses. If any of the controllers does not perform well for disturbance rejection, design a new PID controller to improve the disturbance rejection performance and check whether the new PID controller is suitable for set-point control.

4. For different PID controllers, analyze the compensated systems with time and frequency domain tools. When the derivative term in the controller is disabled, what will happen with the control performance?

5. Using the PID tuner program, compare the PID controllers designed from different design approaches for the plant model

$$G(s) = \frac{1}{(s+1)^6},$$

and find a good PID controller.

6. Construct a Simulink model for PID control system structures with the plant model containing a pure delay term. Design different PID controllers for the plant models given below:

$$\text{(a) } G_a(s) = \frac{1}{(s+1)(2s+1)}\mathrm{e}^{-s}, \quad \text{(b) } G_b(s) = \frac{1}{(17s+1)(6s+1)}\mathrm{e}^{-30s},$$

$$\text{(c) } G_c(s) = \frac{s+2}{(s+1)(4s+1)}\mathrm{e}^{-0.1s}, \quad \text{(d) } G(z) = \frac{0.01752z+0.01534}{z^2-1.637z+0.6703}z^{-10}, \; T=0.2.$$

Compare the simulation results with the approximate results when the pure delay term is replaced by a Padé approximation. also try to analyze the system under Smith predictor control.

7. Design PID controllers for the plants

$$\text{(a) } G(s)=\frac{15}{s(s+1)(s+2)^2(s+5)},\quad \text{(b) } G(s)=\frac{5(s-5)}{s(s+5)^4}.$$

8. Solve the unconstrained optimization problem

$$\min_{\boldsymbol{x}} \begin{array}{l} 100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)+(1-x_3^2)^2 \\ +10.1\left[(x_2-1)^2+(x_4-1)^2\right]+19.8(x_2-1)(x_4-1). \end{array}$$

9. Solve the constrained optimization problems

$$\text{(a)} \quad \min_{\boldsymbol{x}\ \text{s.t.}\begin{cases} x_1+x_2\le 0 \\ -x_1x_2+x_1+x_2\ge 1.5 \\ x_1x_2\ge -10 \\ -10\le x_1,x_2\le 10 \end{cases}} \mathrm{e}^{x_1}(4x_1^2+2x_2^2+4x_1x_2+2x_2+1),$$

$$\text{(b)} \quad \max_{\boldsymbol{x}\ \text{s.t.}\begin{cases} 0.003079x_1^3x_2^3x_5-\cos^3 x_6\ge 0 \\ 0.1017x_3^3x_4^3-x_5^2\cos^3 x_6\ge 0 \\ 0.09939(1+x_5)x_1^3x_2^2-\cos^2 x_6\ge 0 \\ 0.1076(31.5+x_5)x_3^3x_4^2-x_5^2\cos^2 x_6\ge 0 \\ x_3x_4(x_5+31.5)-x_5[2(x_1+5)\cos x_6+x_1x_2x_5]\ge 0 \\ 0.2\le x_1\le 0.5, 14<\le x_2\le 22, 0.35\le x_3\le 0.6 \\ 16\le x_4\le 22, 5.8\le x_5\le 6.5, 0.14\le x_6\le 0.2618 \end{cases}} \frac{1}{2\cos x_6}\left[x_1x_2(1+x_5)+x_3x_4\left(1+\frac{31.5}{x_5}\right)\right].$$

10. Using ITAE, IAE, and ISE criteria, design optimal PID controllers for the open-loop plants

$$\text{(a) } G_a(s)=\frac{1}{(s+1)(2s+1)}\mathrm{e}^{-s},\quad \text{(b)} G_b(s)=\frac{1}{(17s+1)(6s+1)}\mathrm{e}^{-30s}$$

and comments on which criterion will usually lead to the best control results.

11. For a time varying plant model $\ddot{y}(t)+\mathrm{e}^{-0.2t}\dot{y}(t)+\mathrm{e}^{-5t}\sin(2t+6)y(t)=u(t)$, design an optimal PI control which minimizes the ITAE criterion. Analyze the closed-loop behavior of the system.

12. For the plant model

$$G(s) = \frac{1 + \frac{3e^{-s}}{s+1}}{s+1},$$

design an optimal PID controller and analyze the step response of the closed-loop system.

13. In the OCD examples, the selection of simulation terminate time t_f is quite important. Please summarize how the t_f should be selected.