

目录

第 1 章 数字 PID 控制

- 1.1 PID 控制原理
- 1.2 连续系统的模拟 PID 仿真
- 1.3 数字 PID 控制
 - 1.3.1 位置式 PID 控制算法
 - 1.3.2 连续系统的数字 PID 控制仿真
 - 1.3.3 离散系统的数字 PID 控制仿真
 - 1.3.4 增量式 PID 控制算法及仿真
 - 1.3.5 积分分离 PID 控制算法及仿真
 - 1.3.6 抗积分饱和 PID 控制算法及仿真
 - 1.3.7 梯形积分 PID 控制算法
 - 1.3.8 变速积分 PID 算法及仿真
 - 1.3.9 带滤波器的 PID 控制仿真
 - 1.3.10 不完全微分 PID 控制算法及仿真
 - 1.3.11 微分先行 PID 控制算法及仿真
 - 1.3.12 带死区的 PID 控制算法及仿真
 - 1.3.13 基于前馈补偿的 PID 控制算法及仿真
 - 1.3.14 步进式 PID 控制算法及仿真

第 2 章 常用的 PID 控制系统

- 2.1 单回路 PID 控制系统
- 2.2 串级 PID 控制
 - 2.2.1 串级 PID 控制原理
 - 2.2.2 仿真程序及分析
- 2.3 纯滞后系统的大林控制算法
 - 2.3.1 大林控制算法原理
 - 2.3.2 仿真程序及分析
- 2.4 纯滞后系统的 Smith 控制算法
 - 2.4.1 连续 Smith 预估控制
 - 2.4.2 仿真程序及分析
 - 2.4.3 数字 Smith 预估控制
 - 2.4.4 仿真程序及分析

第 3 章 专家 PID 控制和模糊 PID 控制

- 3.1 专家 PID 控制
 - 3.1.1 专家 PID 控制原理
 - 3.1.2 仿真程序及分析
- 3.2 模糊自适应整定 PID 控制
 - 3.2.1 模糊自适应整定 PID 控制原理
 - 3.2.2 仿真程序及分析
- 3.3 模糊免疫 PID 控制算法
 - 3.3.1 模糊免疫 PID 控制算法原理
 - 3.3.2 仿真程序及分析

第 4 章 神经 PID 控制

- 4.1 基于单神经网络的 PID 智能控制
 - 4.1.1 几种典型的学习规则
 - 4.1.2 单神经元自适应 PID 控制
 - 4.1.3 改进的单神经元自适应 PID 控制
 - 4.1.4 仿真程序及分析
 - 4.1.5 基于二次型性能指标学习算法的单神经元自适应 PID 控制
 - 4.1.6 仿真程序及分析
- 4.2 基于 BP 神经网络整定的 PID 控制
 - 4.2.1 基于 BP 神经网络的 PID 整定原理
 - 4.2.2 仿真程序及分析
- 4.3 基于 RBF 神经网络整定的 PID 控制
 - 4.3.1 RBF 神经网络模型
 - 4.3.2 RBF 网络 PID 整定原理
 - 4.3.3 仿真程序及分析
- 4.4 基于 RBF 神经网络辨识的单神经元 PID 模型参考自适应控制
 - 4.4.1 神经网络模型参考自适应控制原理
 - 4.4.2 仿真程序及分析
- 4.5 基于 CMAC (神经网络) 与 PID 的并行控制
 - 4.5.1 CMAC 概述
 - 4.5.2 CMAC 与 PID 复合控制算法
 - 4.5.3 仿真程序及分析
- 4.6 CMAC 与 PID 并行控制的 Simulink 仿真
 - 4.6.1 Simulink 仿真方法
 - 4.6.2 仿真程序及分析
- 第 5 章 基于遗传算法整定的 PID 控制**
 - 5.1 遗传算法的基本原理
 - 5.2 遗传算法的优化设计
 - 5.2.1 遗传算法的构成要素
 - 5.2.2 遗传算法的应用步骤
 - 5.3 遗传算法求函数极大值
 - 5.3.1 遗传算法求函数极大值实例
 - 5.3.2 仿真程序
 - 5.4 基于遗传算法的 PID 整定
 - 5.4.1 基于遗传算法的 PID 整定原理
 - 5.4.2 基于实数编码遗传算法的 PID 整定
 - 5.4.3 仿真程序
 - 5.4.4 基于二进制编码遗传算法的 PID 整定
 - 5.4.5 仿真程序
 - 5.5 基于遗传算法摩擦模型参数辨识的 PID 控制
 - 5.5.1 仿真实例
 - 5.5.2 仿真程序
- 第 6 章 先进 PID 多变量解耦控制**
 - 6.1 PID 多变量解耦控制
 - 6.1.1 PID 解耦控制原理

- 6.1.2 仿真程序及分析
- 6.2 单神经元 PID 解耦控制
 - 6.2.1 单神经元 PID 解耦控制原理
 - 6.2.2 仿真程序及分析
- 6.3 基于 DRNN 神经网络整定的 PID 解耦控制
 - 6.3.1 基于 DRNN 神经网络参数自学习 PID 解耦控制原理
 - 6.3.2 DRNN 神经网络的 Jacobian 信息辨识
 - 6.3.3 仿真程序及分析

第 7 章 几种先进 PID 控制方法

- 7.1 基于干扰观测器的 PID 控制
 - 7.1.1 干扰观测器设计原理
 - 7.1.2 连续系统的控制仿真
 - 7.1.3 离散系统的控制仿真
- 7.2 非线性系统的 PID 鲁棒控制
 - 7.2.1 基于 NCD 优化的非线性优化 PID 控制
 - 7.2.2 基于 NCD 与优化函数结合的非线性优化 PID 控制
- 7.3 一类非线性 PID 控制器设计
 - 7.3.1 非线性控制器设计原理
 - 7.3.2 仿真程序及分析
- 7.4 基于重复控制补偿的高精度 PID 控制
 - 7.4.1 重复控制原理
 - 7.4.2 基于重复控制补偿的 PID 控制
 - 7.4.3 仿真程序及分析
- 7.5 基于零相差前馈补偿的 PID 控制
 - 7.5.1 零相差控制原理
 - 7.5.2 基于零相差前馈补偿的 PID 控制
 - 7.5.3 仿真程序及分析
- 7.6 基于卡尔曼滤波器的 PID 控制
 - 7.6.1 卡尔曼滤波器原理
 - 7.6.2 仿真程序及分析
 - 7.6.3 基于卡尔曼滤波器的 PID 控制
 - 7.6.4 仿真程序及分析
- 7.7 单级倒立摆的 PID 控制
 - 7.7.1 单级倒立摆建模
 - 7.7.2 单级倒立摆控制
 - 7.7.3 仿真程序及分析
- 7.8 吊车-双摆系统的控制
 - 7.8.1 吊车-双摆系统的建模
 - 7.8.2 吊车-双摆系统的仿真

第 8 章 灰色 PID 控制

- 8.1 灰色控制原理
 - 8.1.1 生成数列
 - 8.1.2 GM 灰色模型
- 8.2 灰色 PID 控制

- 8.2.1 灰色 PID 控制的理论基础
- 8.2.2 连续系统灰色 PID 控制
- 8.2.3 仿真程序及分析
- 8.2.4 离散系统灰色 PID 控制
- 8.2.5 仿真程序及分析
- 8.3 灰色 PID 的位置跟踪
- 8.3.1 连续系统灰色 PID 位置跟踪
- 8.3.2 仿真程序及分析
- 8.3.3 离散系统灰色 PID 位置跟踪
- 8.3.4 仿真程序及分析

第 9 章 伺服系统 PID 控制

- 9.1 伺服系统低速摩擦条件下 PID 控制
- 9.1.1 Stribeck 摩擦模型描述
- 9.1.2 一个典型伺服系统描述
- 9.1.3 仿真程序及分析
- 9.2 伺服系统三环的 PID 控制
- 9.2.1 伺服系统三环的 PID 控制原理
- 9.2.2 仿真程序及分析
- 9.3 二质量伺服系统的 PID 控制
- 9.3.1 二质量伺服系统的 PID 控制原理
- 9.3.2 仿真程序及分析

第 10 章 PID 实时控制的 C + + 语言设计及应用

- 10.1 M 语言的 C + + 转化
- 10.2 基于 C + + 的三轴飞行模拟转台伺服系统 PID 实时控制
- 10.2.1 控制系统构成
- 10.2.2 系统各部分功能的软件设计
- 10.2.3 仿真程序及分析

参考文献

第 1 章 数字 PID 控制

自从计算机进入控制领域以来，用数字计算机代替模拟计算机调节器组成计算机控制系统，不仅可以用软件实现 PID 控制算法，而且可以利用计算机的逻辑功能，使 PID 控制更加灵活。数字 PID 控制在生产过程中是一种最普遍采用的控制方法，在机电、冶金、机械、化工等行业中获得了广泛的应用。将偏差的比例 (P)、积分 (I) 和微分 (D) 通过线性组合构成控制量，对被控对象进行控制，故称 PID 控制器。

1.1 PID 控制原理

在模拟控制系统中，控制器最常用的控制规律是 PID 控制。模拟 PID 控制系统原理框图如图 1-1 所示。系统由模拟 PID 控制器和被控对象组成。

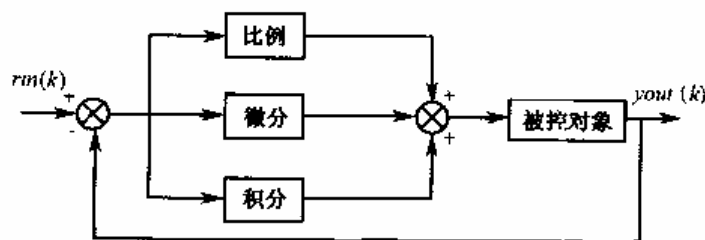


图 1-1 模拟 PID 控制系统原理框图

PID 控制器是一种线性控制器，它根据给定值 $rin(t)$ 与实际输出值 $yout(t)$ 构成控制偏差 $error(t) = rin(t) - yout(t)$ (1.1)

PID 的控制规律为

$$u(t) = k_p(error(t) + \frac{1}{T_i} \int_0^t error(t) dt + \frac{T_D derror(t)}{dt}) \quad (1.2)$$

或写成传递函数的形式

$$G(s) = \frac{U(s)}{E(s)} = k_p(1 + \frac{1}{T_i s} + T_D s) \quad (1.3)$$

式中， k_p ——比例系数； T_i ——积分时间常数； T_D ——微分时间常数。

简单说来，PID 控制器各校正环节的作用如下：

(1) 比例环节：成比例地反映控制系统的偏差信号 $error(t)$ ，偏差一旦产生，控制器立即产生控制作用，以减少偏差。

(2) 积分环节：主要用于消除静差，提高系统的无差度。积分作用的强弱取决于积分时间常数 T_i ， T_i 越大，积分作用越弱，反之则越强。

(3) 微分环节：反映偏差信号的变化趋势（变化速率），并能在偏差信号变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减少调节时间。

1.2 连续系统的模拟 PID 仿真

以三阶线性传递函数为被控对象，进行模拟 PID 控制。在信号发生器中选择正弦信号，仿真时取 $k_p = 60$ ， $k_i = 1$ ， $k_d = 3$ ，输入指令为 $rin(t) = A\sin(2\pi Ft)$ ，其中 $A = 1.0$ ， $F = 0.20$ Hz。采用 ODE45 迭代方法，仿真时间为 10s。

仿真程序：chap1_1.mdl，如图 1-2 和图 1-3 所示。

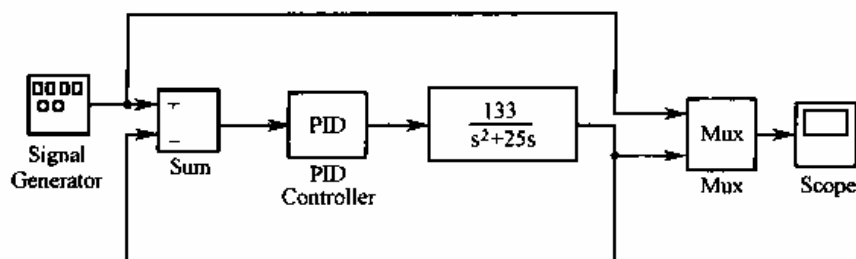


图 1-2 连续系统 PID 的 Simulink 仿真

在 PID 控制器采用封装的形式，其内部结构如图 1-3 所示。

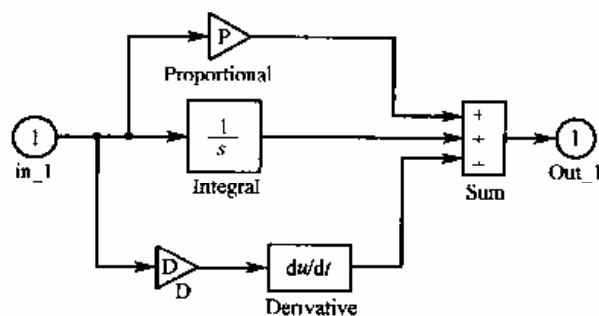


图 1-3 模拟 PID 控制器

连续系统的模拟 PID 控制正弦响应结果如图 1-4 所示。

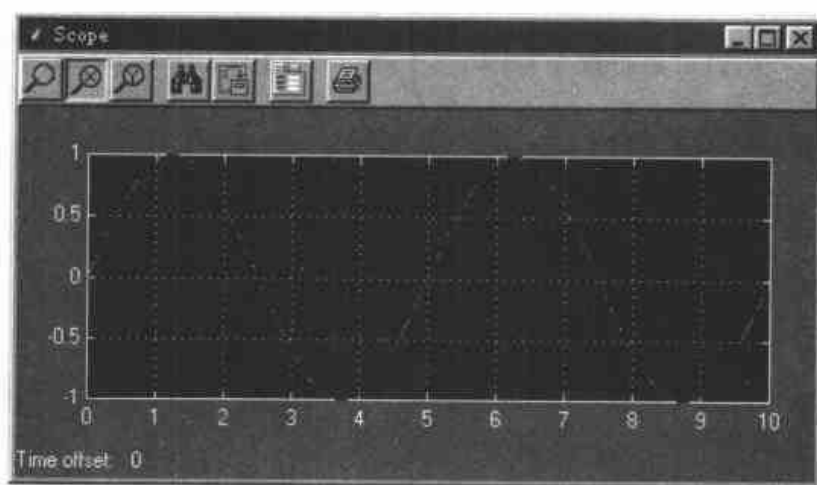


图 1-4 连续系统的模拟 PID 控制正弦响应

1.3 数字 PID 控制

计算机控制是一种采样控制，它只能根据采样时刻的偏差值计算控制量。因此，连续

PID 控制算法不能直接使用，需要采用离散化方法。在计算机 PID 控制中，使用的是数字 PID 控制器。

1.3.1 位置式 PID 控制算法

按模拟 PID 控制算法，以一系列的采样时刻点 kT 代表连续时间 t ，以矩形法数值积分近似代替积分，以一阶后向差分近似代替微分，即

$$\begin{cases} t \approx kT & (k=0,1,2,\dots) \\ \int_0^t error(t)dt \approx T \sum_{j=0}^k error(jT) = T \sum_{j=0}^k error(j) \\ \frac{derror(t)}{dt} \approx \frac{error(kT) - error((k-1)T)}{T} = \frac{error(k) - error(k-1)}{T} \end{cases} \quad (1.4)$$

可得离散 PID 表达式

$$\begin{aligned} u(k) &= k_p (error(k) + \frac{T}{T_i} \sum_{j=0}^k error(j) + \frac{T_D}{T} (error(k) - error(k-1))) \\ &= k_p error(k) + k_i \sum_{j=0}^k error(j)T + k_d \frac{error(k) - error(k-1)}{T} \end{aligned} \quad (1.5)$$

式中， $k_i = \frac{k_p}{T_i}$ ， $k_d = k_p T_D$ 。 T 为采样周期， k 为采样序号， $k=1,2,\dots$ ， $error(k-1)$ 和 $error(k)$ 分别为第 $(k-1)$ 和第 k 时刻所得的偏差信号。

位置式 PID 控制系统如图 1-5 所示。

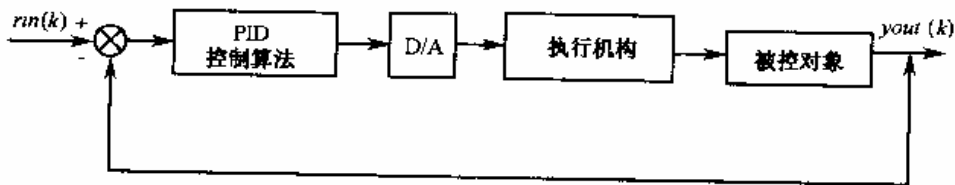


图 1-5 位置式 PID 控制系统

根据位置式 PID 控制算法得到其程序框图如图 1-6 所示。

在仿真过程中，可根据实际情况，对控制器的输出进行限幅： $[-10, +10]$ 。

1.3.2 连续系统的数字 PID 控制仿真

仿真之一

采用 MATLAB 语句形式进行仿真。被控对象为一电机模型传递函数

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

式中， $J = 0.0067$ ， $B = 0.10$ 。

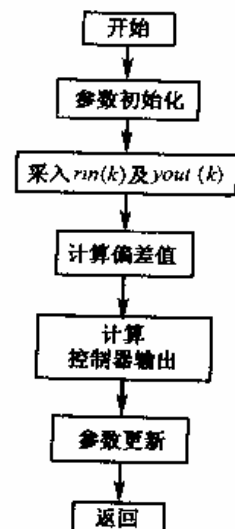


图 1-6 位置式 PID 控制算法程序框图

采用 M 函数的形式，利用 ODE45 的方法求解连续对象方程，输入指令信号为 $r_{in}(k) = 0.50\sin(2\pi t)$ ，采用 PID 控制方法设计控制器，其中 $k_p = 20.0, k_d = 0.50$ 。PID 正弦跟踪结果如图 1-7 所示。

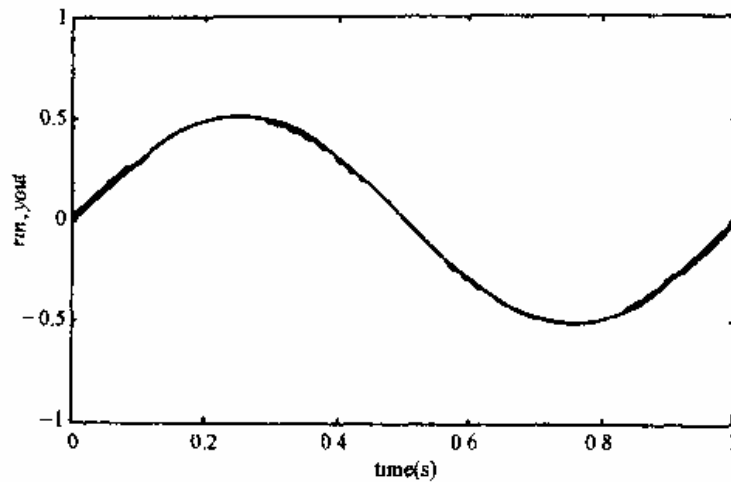


图 1-7 PID 正弦跟踪

控制主程序: chap1_2.m

```
%Discrete PID control for continuous plant
clear all;
close all;

ts=0.001; %Sampling time
xk=zeros(2,1);
e_1=0;
u_1=0;

for k=1:1:2000
time(k) = k*ts;

rin(k)=0.50*sin(1*2*pi*k*ts);

para=u_1;
tSpan=[0 ts];
[tt,xx]=ode45('chap1_2f',tSpan,xk,[],para);
xk = xx(length(xx),:);
yout(k)=xk(1);

e(k)=rin(k)-yout(k);
de(k)=(e(k)-e_1)/ts;

u(k)=20.0*e(k)+0.50*de(k);
```



```

%Control limit
if u(k)>10.0
    u(k)=10.0;
end
if u(k)<-10.0
    u(k)=-10.0;
end

u_1=u(k);
e_1=e(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)'),ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)'),ylabel('error');

```

连续对象子程序: chap1_2f.m

```

function dy = PlantModel(t,y,flag,para)
u=para;
J=0.0067;B=0.1;

dy=zeros(2,1);
dy(1) = y(2);
dy(2) = -(B/J)*y(2) + (1/J)*u;

```

仿真之二

采用 Simulink 进行仿真。被控对象为三阶传递函数, 采用 Simulink 模块与 M 函数相结合的形式, 利用 ODE45 的方法求解连续对象方程, 主程序由 Simulink 模块实现, 控制器由 M 函数实现。输入指令信号为一采样周期 1ms 的正弦信号。采用 PID 方法设计控制器, 其中 $k_p = 1.5, k_i = 2.0, k_d = 0.05$ 。误差的初始化是通过时钟功能实现的, 从而在 M 函数中实现了误差的积分和微分。

控制主程序: chap1_3.mdl, 如图 1-8 所示。

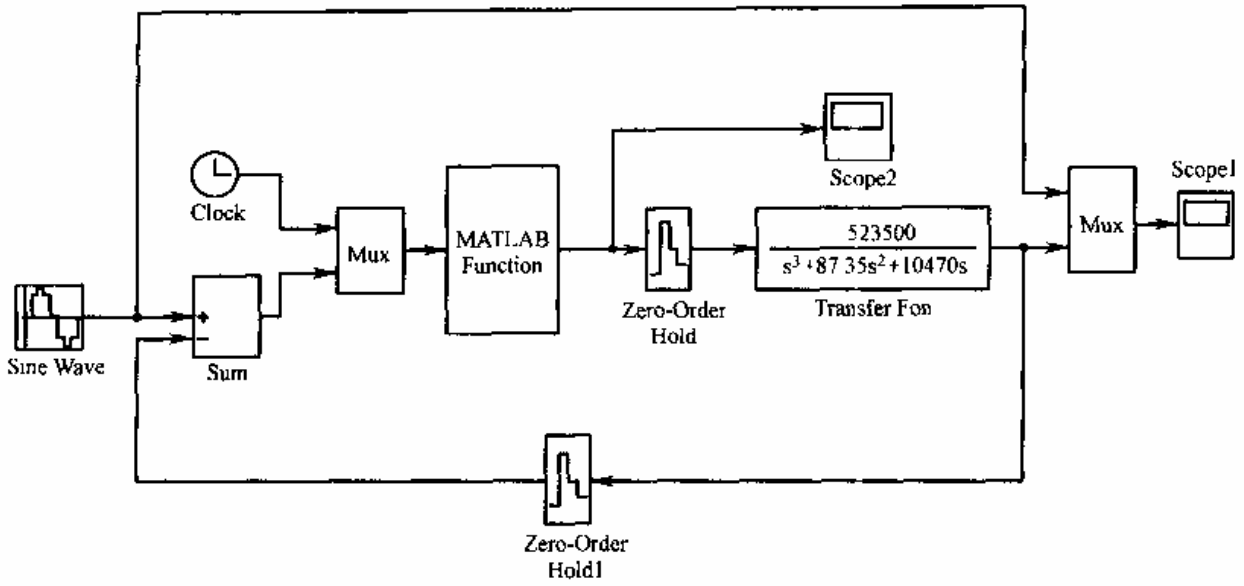


图 1-8 PID 正弦跟踪

控制器子程序: chap1_3f.m

```
function [u]=pidsimf(u1,u2)
persistent pidmat errori error_1

if u1==0
    errori=0
    error_1=0
end

ts=0.001;
kp=1.5;
ki=2.0;
kd=0.05;

error=u2;
error_d=(error-error_1)/ts;
error_i=error_i+error*ts;

u=kp*error+kd*error_d+ki*error_i;
error_1=error;
```

PID 正弦跟踪结果如图 1-9 所示。

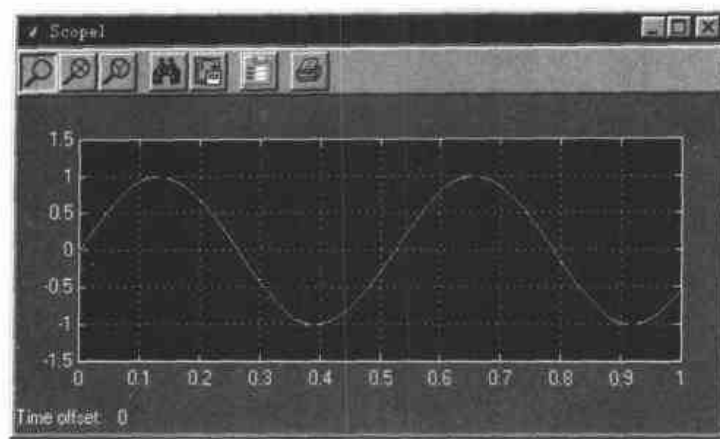


图 1-9 PID 正弦跟踪

1.3.3 离散系统的数字 PID 控制仿真

控制对象为

$$G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms，采用 z 变换进行离散化，经过 z 变换后的离散化对象为

$$yout(k) = -den(2)yout(k-1) - den(3)yout(k-2) - den(4)yout(k-3) + num(2)u(k-1) + num(3)u(k-2) + num(4)u(k-3)$$

仿真之一

针对离散系统的阶跃信号、正弦信号和方波信号的位置响应，设计离散 PID 控制器。其中，S 为信号选择变量，S=1 时为阶跃跟踪，S=2 时为方波跟踪，S=3 时为正弦跟踪，PID 阶跃跟踪结果如图 1-10 至图 1-12 所示。

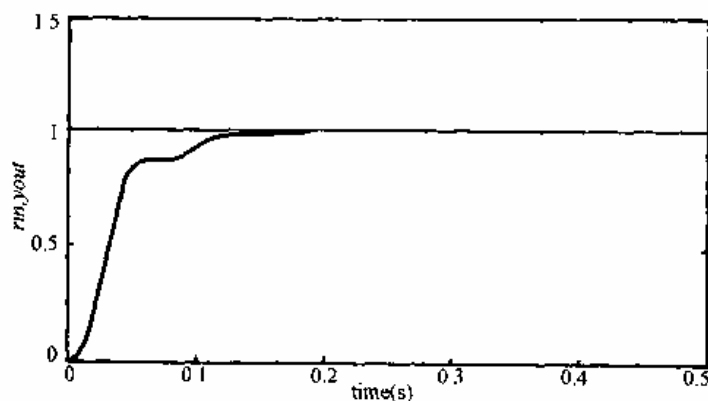


图 1-10 PID 阶跃跟踪 (S=1)

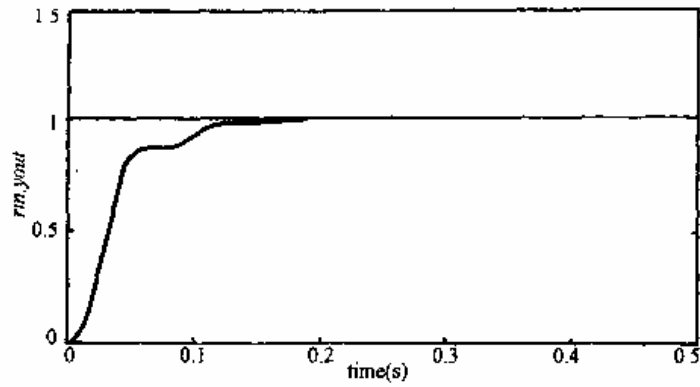


图 1-11 PID 方波跟踪 ($S=2$)

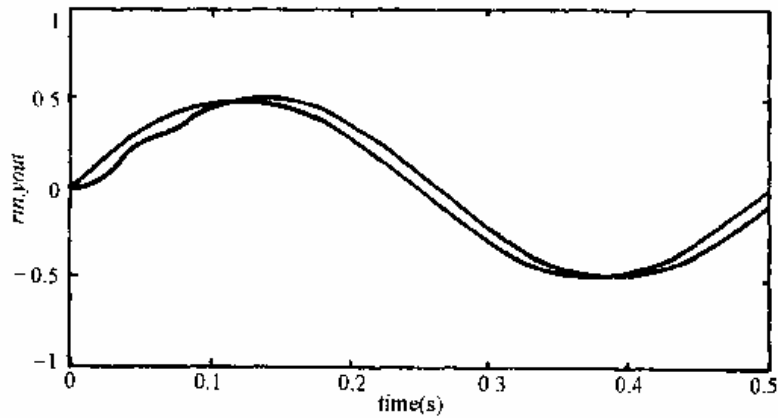


图 1-12 PID 正弦跟踪 ($S=3$)

仿真程序: chap1_4.m

%PID Controller

clear all;

close all;

ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]);

dsys=c2d(sys,ts,'z');

[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;

y_1=0.0;y_2=0.0;y_3=0.0;

x=[0,0,0]';

error_1=0;

for k=1:1:500

time(k)=k*ts;

S=3;

if S==1

kp=0.50;ki=0.001;kd=0.001;

```

        rin(k)=1;                %Step Signal
elseif S==2
    kp=0.50;ki=0.001;kd=0.001;
    rin(k)=sign(sin(2*2*pi*k*ts)); %Square Wave Signal
elseif S==3
    kp=1.5;ki=1.0;kd=0.01;      %Sine Signal
    rin(k)=0.5*sin(2*2*pi*k*ts);
end

u(k)=kp*x(1)+kd*x(2)+ki*x(3); %PID Controller
%Restricting the output of controller
if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);

%Return of parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                %Calculating P
x(2)=(error(k)-error_1)/ts; %Calculating D
x(3)=x(3)+error(k)*ts;      %Calculating I

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)'),ylabel('rin,yout');

```

仿真之二

针对离散系统的三角波、锯齿波和随机信号位置响应，设计离散 PID 控制器，各信号的跟踪结果如图 1-13 至图 1-15 所示，其中 S 代表输入指令信号的类型。 $S=1$ 时为三角波， $S=2$ 时为锯齿波， $S=3$ 时为随机波。在仿真过程中，如果 $D=1$ ，则通过 `pause` 命令实现动态演示仿真。在随机信号跟踪中，对随机信号的变化速率进行了限制。

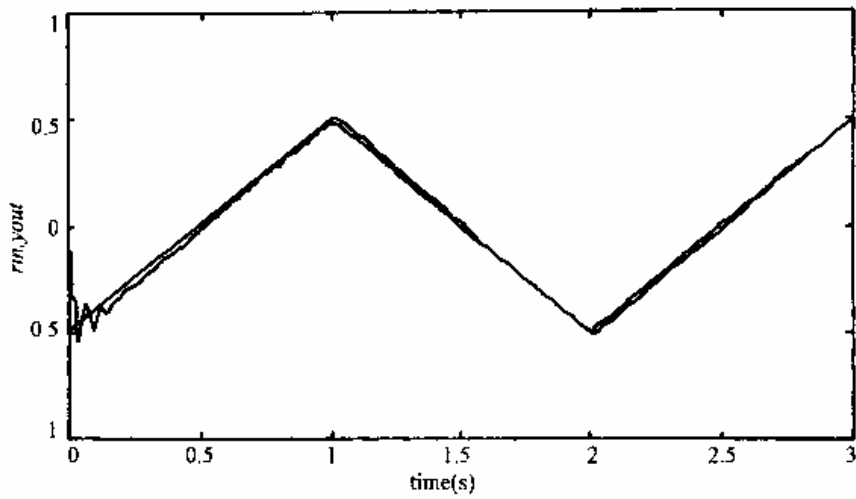


图 1-13 PID 三角波跟踪 ($S=1$)

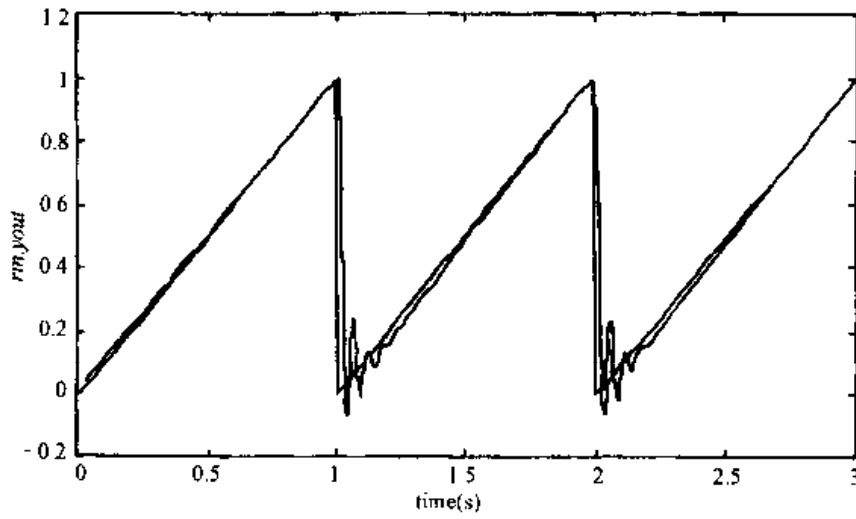


图 1-14 PID 锯齿波跟踪 ($S=2$)

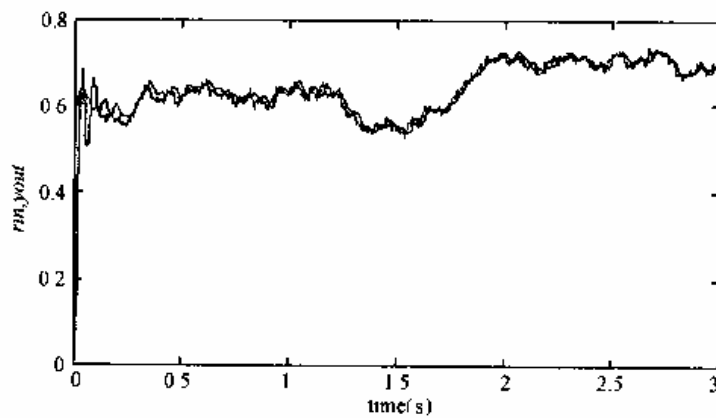


图 1-15 PID 随机信号跟踪 ($S=3$)

```

仿真程序: chap1_5.m
%PID Controller
clear all;
close all;

```

```

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
r_1=rand;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';
error_1=0;

for k=1:1:3000
time(k)=k*ts;

kp=1.0;ki=2.0;kd=0.01;

S=1;
if S==1 %Triangle Signal
    if mod(time(k),2)<1
        rin(k)=mod(time(k),1);
    else
        rin(k)=1-mod(time(k),1);
    end
    rin(k)=rin(k)-0.5;
end
if S==2 %Sawtooth Signal
    rin(k)=mod(time(k),1.0);
end
if S==3 %Random Signal
    rin(k)=rand;
    vr(k)=(rin(k)-r_1)/ts; %Max speed is 5.0
    while abs(vr(k))>=5.0
        rin(k)=rand;
        vr(k)=abs((rin(k)-r_1)/ts);
    end
end

u(k)=kp*x(1)+kd*x(2)+ki*x(3); %PID Controller

```

```

%Restricting the output of controller
if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);

r_1=rin(k);

u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);           %Calculating P
x(2)=(error(k)-error_1)/ts; %Calculating D
x(3)=x(3)+error(k)*ts;   %Calculating I
xi(k)=x(3);

error_1=error(k);
D=0;
if D==1 %Dynamic Simulation Display
    plot(time,rin,'b',time,yout,'r');
    pause(0.0000000000000000);
end
end
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');

```

上述 PID 控制算法的缺点是，由于采用全量输出，所以每次输出均与过去的状态有关，计算时要对 $error(k)$ 量进行累加，计算机输出控制量 $u(k)$ 对应的是执行机构的实际位置偏差，如果位置传感器出现故障， $u(k)$ 可能会出现大幅度变化。 $u(k)$ 的大幅度变化会引起执行机构位置的大幅度变化，这种情况在生产中是不允许的，在某些重要场合还可能造成重大事故。为避免这种情况的发生，采用增量式 PID 控制算法。

1.3.4 增量式 PID 控制算法及仿真

当执行机构需要的是控制量的增量（例如驱动步进电机）时，应采用增量式 PID 控制。根据递推原理可得

$$u(k-1) = k_p(\text{error}(k-1) + k_i \sum_{j=0}^k \text{error}(j) + k_d(\text{error}(k) - \text{error}(k-1))) \quad (1.6)$$

增量式 PID 控制算法

$$\Delta u(k) = k_p(\text{error}(k) - \text{error}(k-1)) + k_i \text{error}(k) + k_d(\text{error}(k) - 2\text{error}(k-1) + \text{error}(k-2)) \quad (1.7)$$

根据增量式 PID 控制算法，设计了仿真程序，被控对象如下：

$$G(s) = \frac{400}{s^2 + 50s}$$

PID 控制参数为： $k_p = 8, k_i = 0.10, k_d = 10$ 。

仿真程序： chap1_6.m

```
%Increment PID Controller
clear all;
close all;

ts=0.001;
sys=tf(400,[1,50,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';

error_1=0;
error_2=0;
for k=1:1:1000
    time(k)=k*ts;

    rin(k)=1.0;
    kp=8;
    ki=0.10;
    kd=10;

    du(k)=kp*x(1)+kd*x(2)+ki*x(3);
    u(k)=u_1+du(k);
```

```

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

error=rin(k)-yout(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error-error_1;           %Calculating P
x(2)=error-2*error_1+error_2; %Calculating D
x(3)=error;                   %Calculating I

error_2=error_1;
error_1=error;
end
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');

```

增量式 PID 阶跃跟踪结果如图 1-16 所示。

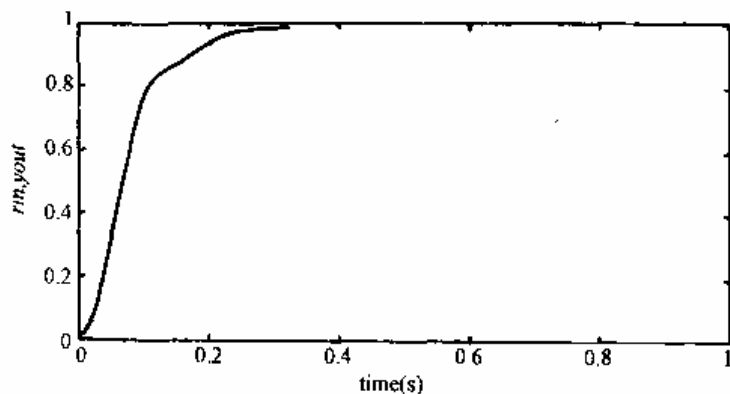


图 1-16 增量式 PID 阶跃跟踪

由于控制算法中不需要累加，控制增量 $\Delta u(k)$ 仅与最近 k 次的采样有关，所以误动作时影响小，而且较容易通过加权处理获得比较好的控制效果。

在计算机控制系统中，PID 控制是通过计算机程序实现的，因此它的灵活性很大。一些原来在模拟 PID 控制器中无法实现的问题，在引入计算机以后，就可以得到解决，于是产生了一系列的改进算法，形成非标准的控制算法，以改善系统品质，满足不同控制系统的需

要。

1.3.5 积分分离 PID 控制算法及仿真

在普通 PID 控制中引入积分环节的目的, 主要是为了消除静差, 提高控制精度。但在过程的启动、结束或大幅度增减设定时, 短时间内系统输出有很大的偏差, 会造成 PID 运算的积分积累, 致使控制量超过执行机构可能允许的最大动作范围对应的极限控制量, 引起系统较大的超调, 甚至引起系统较大的振荡, 这在生产中是绝对不允许的。

积分分离控制基本思路是: 当被控量与设定值偏差较大时, 取消积分作用, 以免由于积分作用使系统稳定性降低, 超调量增大; 当被控量接近给定值时, 引入积分控制, 以便消除静差, 提高控制精度。其具体实现步骤如下:

(1) 根据实际情况, 人为设定阈值 $\varepsilon > 0$;

(2) 当 $|error(k)| > \varepsilon$ 时, 采用 PD 控制, 可避免产生过大的超调, 又使系统有较快的响应;

(3) 当 $|error(k)| \leq \varepsilon$ 时, 采用 PID 控制, 以保证系统的控制精度。

积分分离控制算法可表示为

$$u(k) = k_p error(k) + \beta k_i \sum_{j=0}^k error(j)T + k_d (error(k) - error(k-1)) / T \quad (1.8)$$

式中, T 为采样时间, β 项为积分项的开关系数

$$\beta = \begin{cases} 1 & |error(k)| \leq \varepsilon \\ 0 & |error(k)| > \varepsilon \end{cases} \quad (1.9)$$

根据积分分离式 PID 控制算法得到其程序框图如图 1-17 所示。

仿真实例

设被控对象为--延迟对象

$$G(s) = \frac{e^{-80s}}{60s+1}$$

采样时间为 20s, 延迟时间为 4 个采样时间, 即 80s, 被控对象离散化为

$$y(k) = -den(2)y(k-1) + num(2)u(k-5)$$

取 $M=1$, 采用积分分离式 PID 控制器进行阶跃响应, 对积分分离式 PID 控制算法进行改进, 采用分段积分分离方式, 即根据误差绝对值的不同, 采用不同的积分强度。仿真中指令信号为 $rin(k)=40$, 控制器输出限制在 $[-110,110]$, 其阶跃式跟踪结果如图 1-18 所示。

取 $M=2$, 采用普通 PID 控制, 其阶跃式跟踪结果如图 1-19 所示。

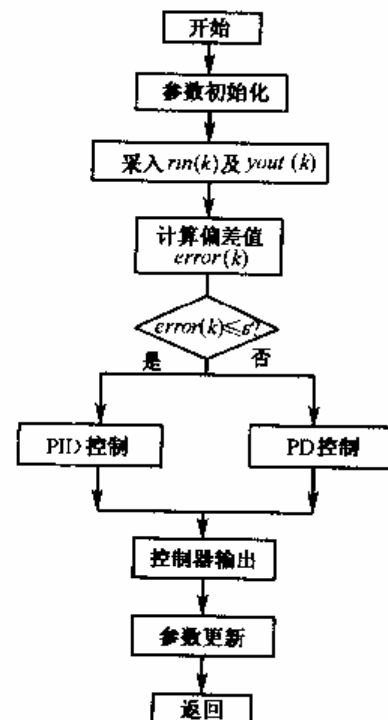


图 1-17 积分分离式 PID 控制算法程序框图

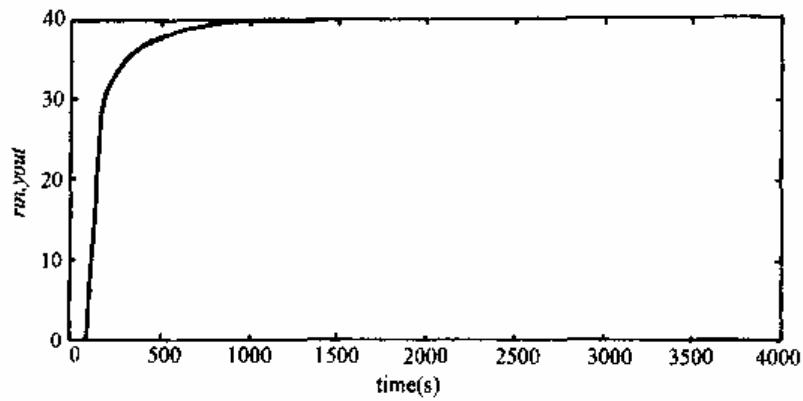


图 1-18 积分分离式 PID 阶跃跟踪 ($M = 1$)

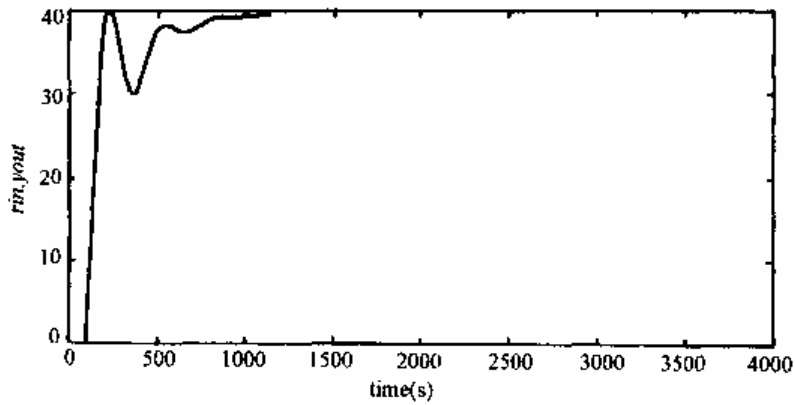


图 1-19 采用普通 PID 阶跃跟踪 ($M = 2$)

仿真程序: chap1_7.m

```
%Integration Separation PID Controller
```

```
clear all;
```

```
close all;
```

```
ts=20;
```

```
%Delay plant
```

```
sys=tf([1],[60,1],'inputdelay',80);
```

```
dsys=c2d(sys,ts,'zoh');
```

```
[num,den]=tfdata(dsys,'v');
```

```
u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
```

```
y_1=0;y_2=0;y_3=0;
```

```
error_1=0;error_2=0;
```

```
ei=0;
```

```
for k=1:1:200
```

```
time(k)=k*ts;
```

```
%Delay plant
```

```

yout(k)=-den(2)*y_1+num(2)*u_5;

%I separation
rin(k)=40;
error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

M=2;
if M==1          %Using integration separation
    if abs(error(k))>=30&abs(error(k))<=40
        beta=0.3;
    elseif abs(error(k))>=20&abs(error(k))<=30
        beta=0.6;
    elseif abs(error(k))>=10&abs(error(k))<=20
        beta=0.9;
    else
        beta=1.0;
    end
elseif M==2
    beta=1.0;    %Not using integration separation
end

kp=0.80;
ki=0.005;
kd=3.0;
u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+beta*ki*ei;

if u(k)>=110      % Restricting the output of controller
    u(k)=110;
end
if u(k)<=-110
    u(k)=-110;
end

u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end

```

```

figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');

```

由仿真结果可以看出，采用积分分离方法控制效果有很大的改善。值得注意的是，为保证引入积分作用后系统的稳定性不变，在输入积分作用时比例系数 k_p 可作相应变化。此外， β 值应根据具体对象及要求而定，若 β 过大，则达不到积分分离的目的；若 β 过小，则会导致无法进入积分区。如果只进行 PD 控制，会使控制出现余差。

1.3.6 抗积分饱和 PID 控制算法及仿真

1. 积分饱和现象

所谓积分饱和现象是指若系统存在一个方向的偏差，PID 控制器的输出由于积分作用的不断累加而加大，从而导致执行机构达到极限位置 X_{\max} (例如阀门开度达到最大)，如图 1-20 所示，若控制器输出 $u(k)$ 继续增大，阀门开度不可能再增大，此时就称计算机输出控制量超出了正常运行范围而进入了饱和区。一旦系统出现反向偏差， $u(k)$ 逐渐从饱和区退出。进入饱和区愈深则退出饱和区所需时间愈长。在这段时间内，执行机构仍停留在极限位置而不能随偏差反向立即做出相应的改变，这时系统就像失去控制一样，造成控制性能恶化。这种现象称为积分饱和现象或积分失控现象。

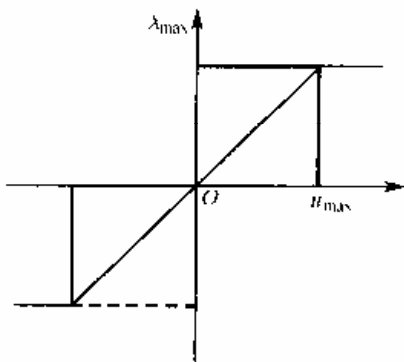


图 1-20 执行机构饱和特性

2. 抗积分饱和算法

作为防止积分饱和的方法之一就是抗积分饱和法，该方法的思路是在计算 $u(k)$ 时，首先判断上一时刻的控制量 $u(k-1)$ 是否已超出限制范围：

若 $u(k-1) > u_{\max}$ ，则只累加负偏差；若 $u(k-1) < u_{\min}$ ，则只累加正偏差。这种算法可以避免控制量长时间停留在饱和区。

仿真实例

控制对象为

$$G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms，取 $M=1$ ，采用抗积分饱和算法进行离散系统阶跃响应，仿真结果如图 1-21 所示。取 $M=2$ ，采用普通 PID 算法进行离散系统阶跃响应，其阶跃响应结果如图 1-22 所示。由仿真结果可以看出，采用抗积分饱和 PID 方法，可以避免控制量长时间停留在饱和区，防止系统产生超调。

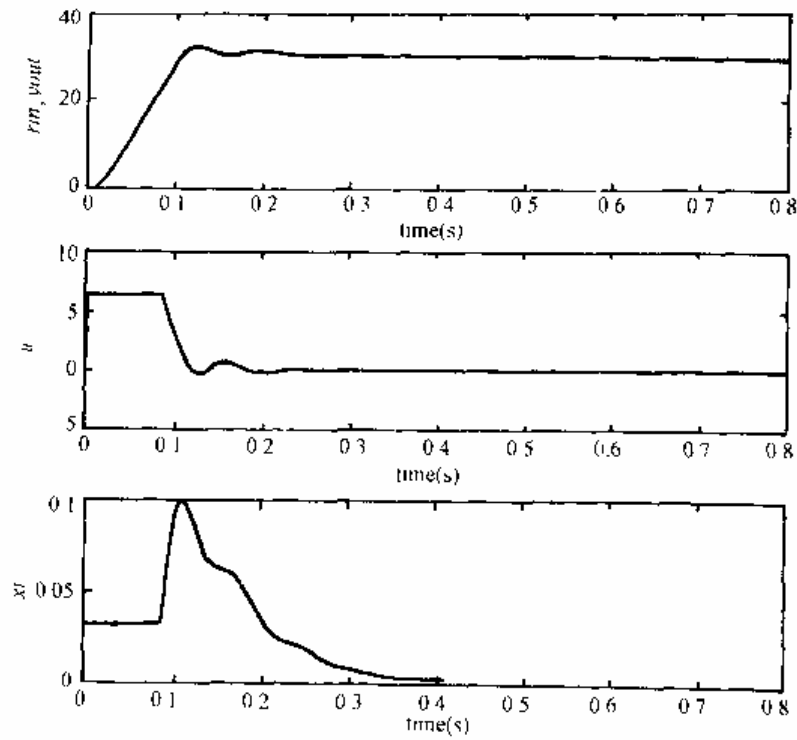


图 1-21 抗积分饱和仿真结果 ($M=1$)

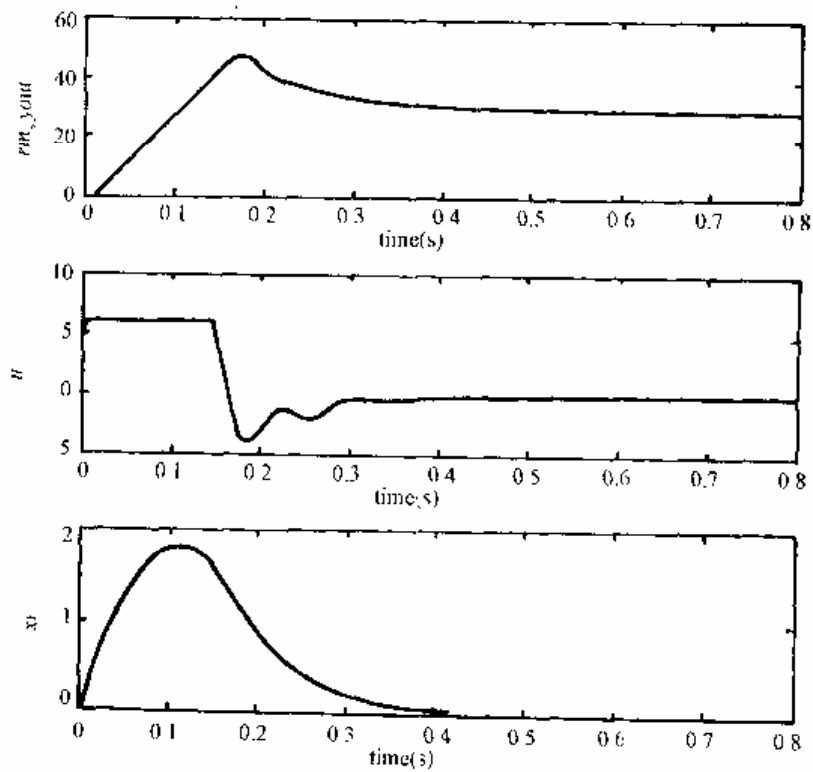


图 1-22 普通 PID 算法进行离散系统阶跃响应结果 ($M=2$)

仿真程序: chap1_8.m

```
%PID Controller with intergration sturation
```

```
clear all;
```

```
close all;
```

```

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';

error_1=0;

um=6;
kp=0.85;ki=9.0;kd=0.0;
rin=30;          %Step Signal

for k=1:1:800
time(k)=k*ts;

u(k)=kp*x(1)+kd*x(2)+ki*x(3);  % PID Controller

if u(k)>=um
    u(k)=um;
end
if u(k)<=-um
    u(k)=-um;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin-yout(k);

M=2;
if M==1 %Using intergration sturation
if u(k)>=um
    if error(k)>0
        alpha=0;
    else

```



```

        alpha=1;
    end
elseif u(k)<=-um
    if error(k)>0
        alpha=1;
    else
        alpha=0;
    end
else
    alpha=1;
end

elseif M==2 %Not using intergration sturation
    alpha=1;
end

%Return of PID parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
error_1=error(k);

x(1)=error(k);           % Calculating P
x(2)=(error(k)-error_1)/ts; % Calculating D
x(3)=x(3)+alpha*error(k)*ts; % Calculating I

xi(k)=x(3);
end
figure(1);
subplot(311);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('Position tracking');
subplot(312);
plot(time,u,'r');
xlabel('time(s)');ylabel('Controller output');
subplot(313);
plot(time,xi,'r');
xlabel('time(s)');ylabel('Integration');

```

1.3.7 梯形积分 PID 控制算法

在 PID 控制律中积分项的作用是消除余差，为了减小余差，应提高积分项的运算精度，为此，可将矩形积分改为梯形积分。梯形积分的计算公式为

$$\int_0^t e(t)dt = \sum_{i=0}^k \frac{e(i) + e(i-1)}{2} T \quad (1.10)$$

1.3.8 变速积分 PID 算法及仿真

在普通的 PID 控制算法中, 由于积分系数 k_i 是常数, 所以在整个控制过程中, 积分增量不变。而系统对积分项的要求是, 系统偏差大时积分作用应减弱甚至全无, 而在偏差小时则应加强。积分系数取大了会产生超调, 甚至积分饱和, 取小了又迟迟不能消除静差。因此, 如何根据系统偏差大小改变积分的速度, 对于提高系统品质是很重要的。变速积分 PID 可较好地解决这一问题。

变速积分 PID 的基本思想是设法改变积分项的累加速度, 使其与偏差大小相对应: 偏差越大, 积分越慢, 反之则越快。

为此, 设置系数 $f(e(k))$, 它是 $e(k)$ 的函数。当 $|e(k)|$ 增大时, f 减小, 反之增大。变速积分的 PID 积分项表达式为

$$u_i(k) = k_i \left\{ \sum_{i=0}^{k-1} e(i) + f[e(k)]e(k) \right\} T \quad (1.11)$$

系数 f 与偏差当前值 $|e(k)|$ 的关系可以是线性的或非线性的, 可设为

$$f[e(k)] = \begin{cases} 1 & |e(k)| \leq B \\ \frac{A - |e(k)| + B}{A} & B < |e(k)| \leq A + B \\ 0 & |e(k)| > A + B \end{cases} \quad (1.12)$$

f 值在 $[0, 1]$ 区间内变化, 当 $|e(k)|$ 差大于所给分离区间 $A + B$ 后, $f = 0$, 不再对当前值 $e(k)$ 进行继续累加; 当偏差 $|e(k)|$ 小于 B 时, 加入当前值 $e(k)$, 即积分项变为

$u(k) = k_i \sum_{i=0}^k e(i)T$, 与一般 PID 积分项相同, 积分动作达到最高速; 而当偏差 $|e(k)|$ 在 B 与 $A + B$ 之间时, 则累加计入的是部分当前值, 其值在 $0 \sim |e(k)|$ 之间随 $|e(k)|$ 的大小而变化, 因此,

$$u(k) = k_p e(k) + k_i \left\{ \sum_{i=0}^{k-1} e(i) + f[e(k)]e(k) \right\} \cdot T + k_d [e(k) - e(k-1)] \quad (1.13)$$

这种算法对 A 、 B 两参数的要求不精确, 参数整定较容易。

仿真实例

设被控对象为一延迟对象

$$G(s) = \frac{e^{-80s}}{60s + 1}$$

采样时间为 $20s$, 延迟时间为 4 个采样时间, 即 $80s$, 取 $k_p = 0.45, k_d = 12, k_i = 0.0048$, $A = 0.4, B = 0.6$ 。取 $M = 1$, 采用变速积分 PID 控制算法进行阶跃响应, 其结果如图 1-23 和图 1-24 所示。取 $M = 2$, 采用普通 PID 控制, 其结果如图 1-25 所示。由仿真结果可以看出,

变速积分与积分分离两种控制方法很类似，但调节方式不同，前者对积分项采用的是缓慢变化，而后者则采用所谓“开关”控制。变速积分调节质量更高。

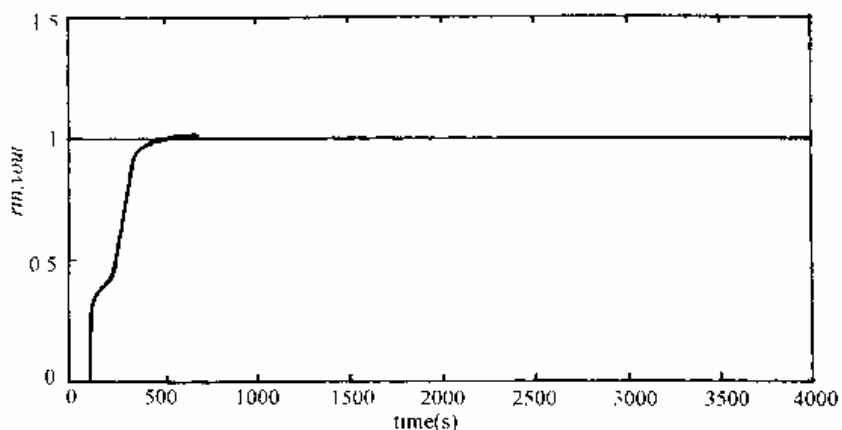


图 1-23 变速积分阶跃响应 ($M = 1$)

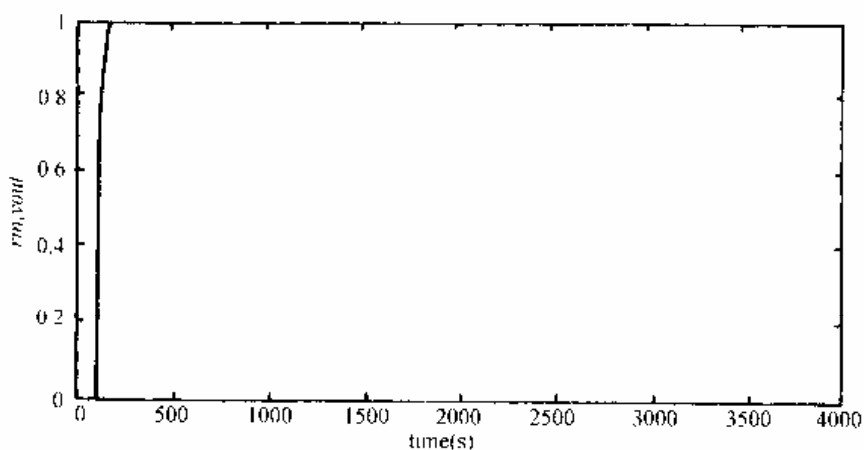


图 1-24 变速积分参数的变化

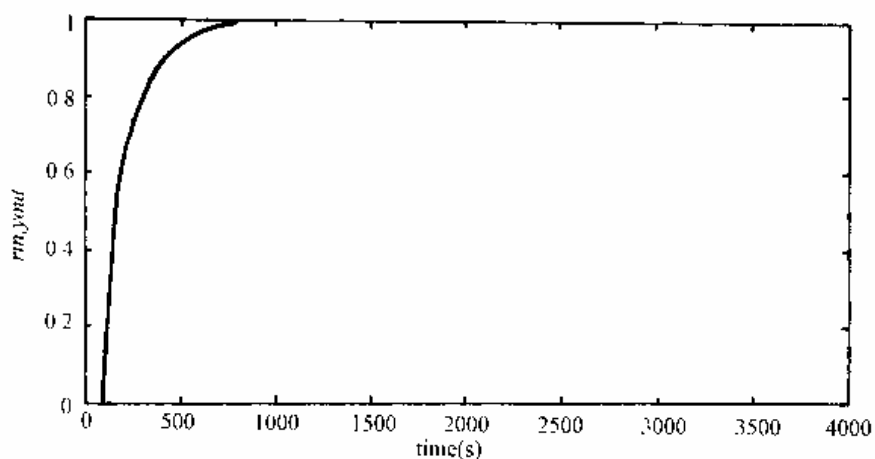


图 1-25 普通 PID 控制阶跃响应 ($M = 2$)

仿真程序: chap1_9.m

```
%PID Controller with changing integration rate
clear all;
```

```

close all;

%Big time delay Plant
ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;
ei=0;

for k=1:1:200
time(k)=k*ts;

rin(k)=1.0; %Step Signal

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;
error(k)=rin(k)-yout(k);

kp=0.45;kd=12;ki=0.0048;
A=0.4;B=0.6;

%T type integration
ei=ei+(error(k)+error_1)/2*ts;

M=2;
if M==1 %Changing integration rate
if abs(error(k))<=B
f(k)=1;
elseif abs(error(k))>B&abs(error(k))<=A+B
f(k)=(A-abs(error(k))+B)/A;
else
f(k)=0;
end

elseif M==2 %Not changing integration rate
f(k)=1;

```

```

end

u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+ki*f(k)*ei;

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
%Return of PID parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,f,'r');
xlabel('time(s)');ylabel('Integration rate f');

```

1.3.9 带滤波器的PID控制仿真

仿真实例 1

验证低通滤波器的滤波性能。

设低通滤波器为

$$Q(s) = \frac{1}{0.04s + 1}$$

采样时间为 1ms，输入信号为带有高频正弦噪声（100Hz）的低频（0.2Hz）正弦信号。采用低通滤波器滤掉高频正弦信号。滤波器的离散化采用 Tustin 变换，其 Bode 图如图 1-26 和图 1-27 所示。仿真结果表明，该滤波器对高频信号具有很好的滤波作用。

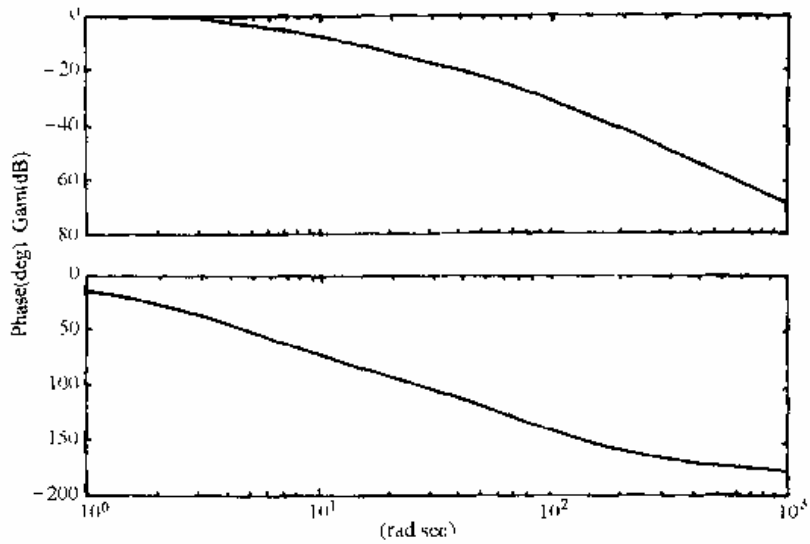


图 1-26 低通滤波器 Bode 图

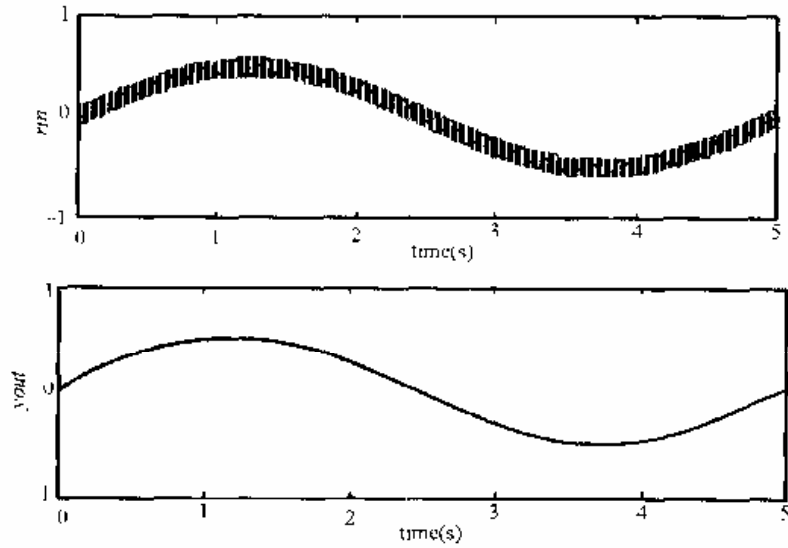


图 1-27 原始信号及滤波后信号

仿真程序: chap1_10.m

```
%Low Pass Filter
```

```
clear all;
```

```
close all;
```

```
ts=0.001;
```

```
Q=tf([1],[0.04,1]); %Low Freq Signal Filter
```

```
Qz=c2d(Q,ts,'tustin');
```

```
[num,den]=tfdata(Qz,'v');
```

```
y_1=0;y_2=0;
```

```
r_1=0;r_2=0;
```

```
for k=1:1:5000
```

```
time(k)=k*ts;
```

```

%Input Signal with disturbance
D(k)=0.10*sin(100*2*pi*k*ts); %Disturbance signal
rin(k)=D(k)+0.50*sin(0.2*2*pi*k*ts); %Input Signal

yout(k)= den(2)*y_1+num(1)*rin(k)+num(2)*r_1;

y_2=y_1;y_1=yout(k);
r_2=r_1;r_1=rin(k);
end
figure(1);bode(Q);
figure(2);
subplot(211);
plot(time,rin,'r');
xlabel('time(s)');ylabel('rin');
subplot(212);
plot(time,yout,'b');
xlabel('time(s)');ylabel('yout');

```

仿真实例 2

采用低通滤波器的 PID 控制。

被控对象为三阶传递函数

$$G_p(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

低通滤波器为

$$Q(s) = \frac{1}{0.04s + 1}$$

采样时间为 1ms，干扰信号加在对象的输出端。分三种情况进行仿真： $M=1$ 时，为未加干扰信号； $M=2$ 时，为加干扰信号未加滤波； $M=3$ 时，为加干扰信号加滤波。阶跃响应结果如图 1-28 至图 1-30 所示。

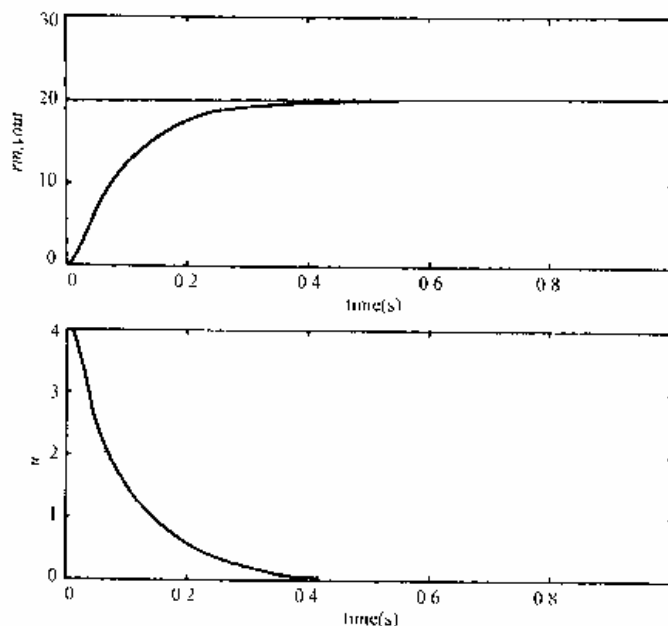


图 1-28 普通 PID 控制阶跃响应 ($M=1$)

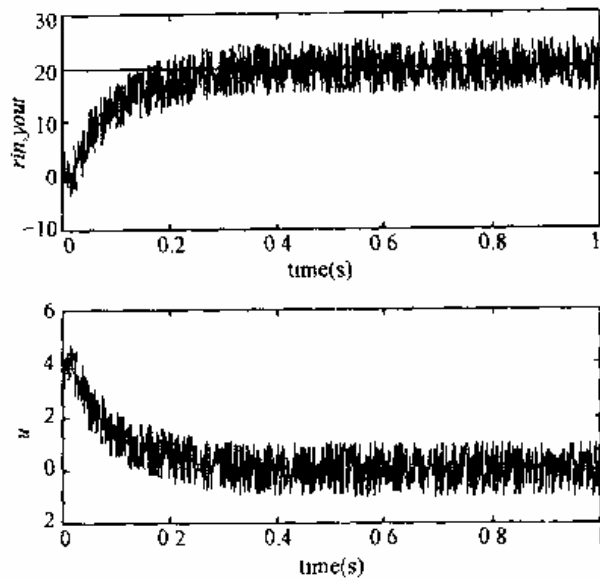


图 1-29 无滤波器时 PID 控制阶跃响应 ($M = 2$)

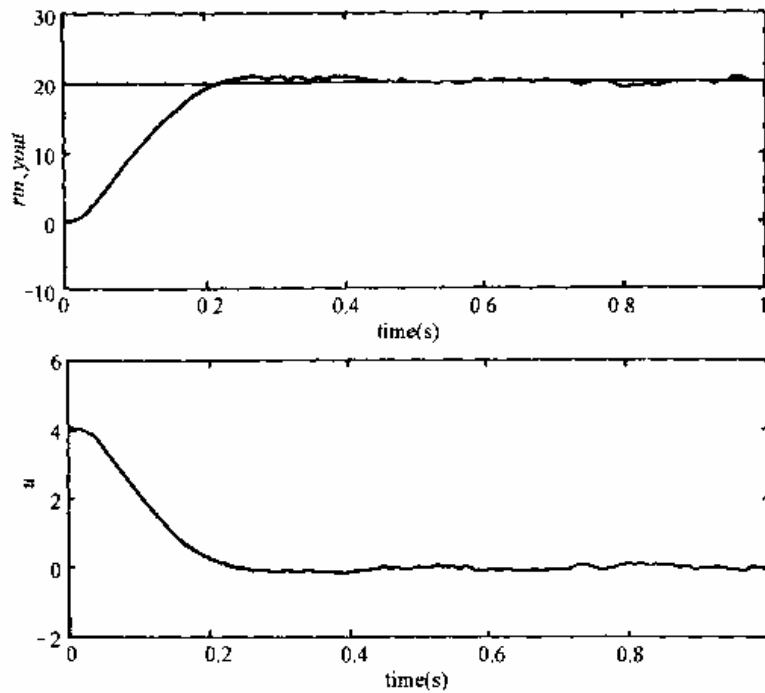


图 1-30 加入滤波器后 PID 控制阶跃响应 ($M = 3$)

仿真程序: chap1_11.m

```
%PID Controller with Partial differential
```

```
clear all;
```

```
close all;
```

```
ts=0.001;
```

```
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
```

```
dsys=c2d(sys,ts,'z');
```

```
[num,den]=tfdata(dsys,'v');
```



```

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
yy_1=0;
error_1=0;error_2=0;ei=0;

kp=0.20;ki=0.05;

sys1=tf([1],[0.04,1]); %Low Freq Signal Filter
dsys1=c2d(sys1,ts,'tustin');
[num1,den1]=tfdata(dsys1,'v');
f_1=0;

M=1;
for k=1:1:1000
time(k)=k*ts;

rin(k)=20; %Step Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+...
        num(3)*u_2+num(4)*u_3;

if M==1 %No disturbance signal
    error(k)=rin(k)-yout(k);
    filty(k)=yout(k);
end

D(k)=5.0*rand(1); %Disturbance signal
yyout(k)=yout(k)+D(k);

if M==2 %No filter
    filty(k)=yyout(k);
    error(k)=rin(k)-filty(k);
end

if M==3 %Using low frequency filter
    filty(k)=-den1(2)*f_1+num1(1)*(yyout(k)+yy_1);
    error(k)=rin(k)-filty(k);
end

%I separation

```

```

if abs(error(k))<=0.8
    ei=ei+error(k)*ts;
else
    ei=0;
end
u(k)=-kp*error(k)+ki*ei;

if u(k)>10      % Restricting the output of controller
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
% -----Return of PID parameters-----
rin_1=rin(k);
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

f_1=filty(k);
yy_1=yyout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,filty,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,D,'r');
xlabel('time(s)');ylabel('Disturbance signal');

```

1.3.10 不完全微分 PID 控制算法及仿真

在 PID 控制中，微分信号的引入可改善系统的动态特性，但也易引进高频干扰，在误差扰动突变时尤其显出微分项的不足。若在控制算法中加入低通滤波器，则可使系统性能得到改善。

克服上述缺点的方法之一是在 PID 算法中加入一个一阶惯性环节（低通滤波器）

$G_f(s) = \frac{1}{1+T_f s}$ ，可使系统性能得到改善。

不完全微分 PID 的结构如图 1-31 (a)、(b) 所示，其中图 (a) 是将低通滤波器直接加在微分环节上，图 (b) 是将低通滤波加在整个 PID 控制器之后。下面以图 (a) 为例进行仿真说明不完全微分 PID 如何改进了普通 PID 的性能。

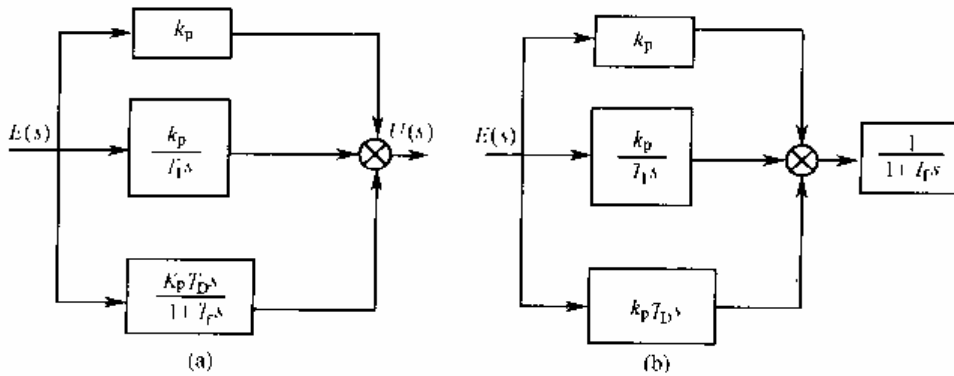


图 1-31 不完全微分算法结构图

对图 (a) 所示的不完全微分结构，其传递函数为

$$U(s) = (k_p + \frac{k_p}{T_i} \frac{1}{s} + \frac{k_p T_D s}{T_i s + 1}) E(s) = u_p(s) + u_i(s) + u_D(s) \quad (1.14)$$

将上式离散化为

$$u(k) = u_p(k) + u_i(k) + u_D(k) \quad (1.15)$$

现将 $u_D(k)$ 推导

$$u_D(s) = \frac{k_p T_D s}{T_f s + 1} E(s) \quad (1.16)$$

写成微分方程为

$$u_D(k) + T_f \frac{du_D(t)}{dt} = k_p T_D \frac{derror(t)}{dt}$$

将上式离散化为

$$u_D(k) + T_f \frac{u_D(k) - u_D(k-1)}{T_s} = k_p T_D \frac{error(k) - error(k-1)}{\Delta t} \quad (1.17)$$

经整理得

$$u_D(k) = \frac{T_f}{T_s + T_f} u_D(k-1) + k_p \frac{T_D}{T_s + T_f} (error(k) - error(k-1)) \quad (1.18)$$

令 $\alpha = \frac{T_f}{T_s + T_f}$ ，则 $\frac{T_s}{T_s + T_f} = 1 - \alpha$ ，显然有 $\alpha < 1$ ， $1 - \alpha < 1$ 成立，则可得不完全微分算

法

$$u_D(k) = K_D (1 - \alpha) (error(k) - error(k-1)) + \alpha u_D(k-1) \quad (1.19)$$

可见，不完全微分的 $u_D(k)$ 多了一项 $\alpha u_D(k-1)$ ，而原微分系数由 k_d 降至 $k_d(1 - \alpha)$ 。

以上各式中， T_s 为采样时间， $\Delta t = T_s$ ， k_p 为比例系数， T_i 和 T_D 分别为积分时间常数和微分时间常数， T_f 为滤波器系数。

仿真实例

采用第一种不完全微分算法，被控对象为一时滞系统传递函数

$$G(s) = \frac{e^{-80s}}{60s + 1}$$

在对象的输出端加幅值为 0.01 的随机信号。采样时间为 20ms。

低通滤波器为

$$Q(s) = \frac{1}{180s + 1}$$

取 $M=1$ ，采用具有不完全微分 PID 方法，其控制阶跃响应结果如图 1-32 所示。取 $M=2$ ，采用普通 PID 方法，阶跃响应结果如图 1-33 所示。由仿真结果可以看出，引入不完全微分后，能有效地克服普通 PID 的不足。尽管不完全微分 PID 控制算法比普通 PID 控制算法要复杂些，但由于其良好的控制特性，近年来越来越得到广泛的应用。

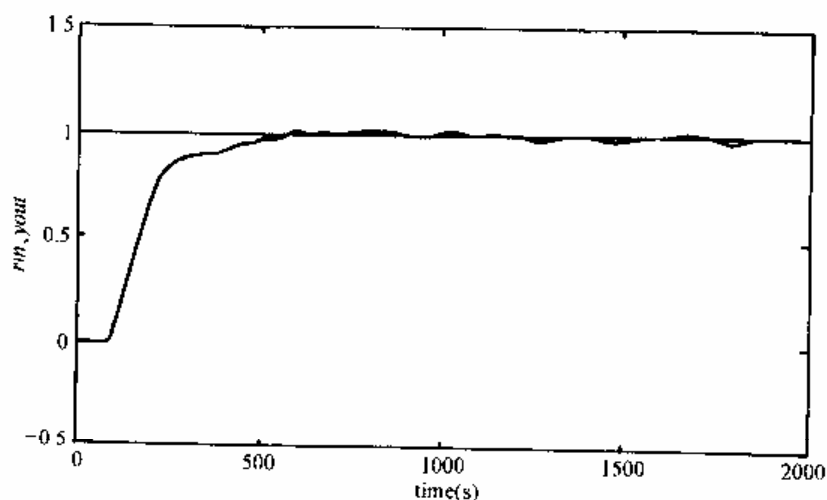


图 1-32 不完全微分控制阶跃响应 ($M=1$)

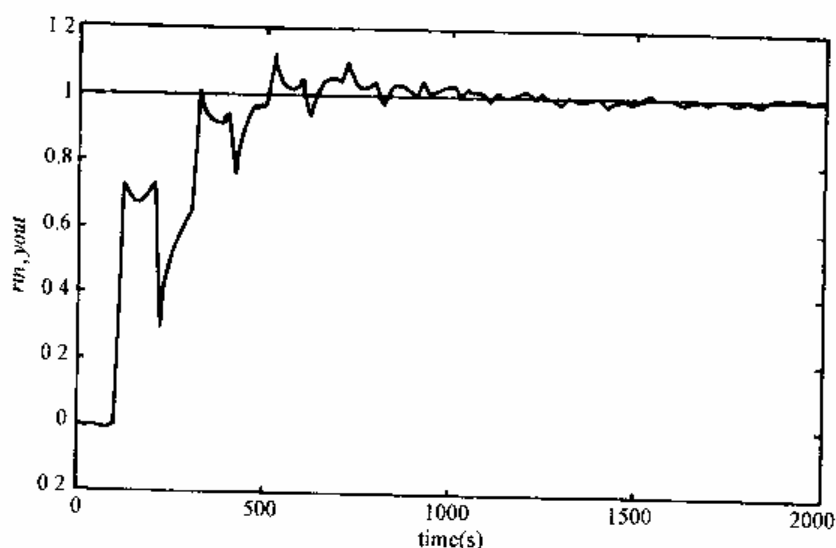


图 1-33 普通 PID 控制阶跃响应 ($M=2$)

仿真程序: chup1_12.m

```
%PID Controller with Partial differential
clear all;
close all;
```

```

ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
ud_1=0;
y_1=0;y_2=0;y_3=0;
error_1=0;
ei=0;

for k=1:1:100
time(k)=k*ts;

rin(k)=1.0;

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

D(k)=0.01*rands(1);
yout(k)=yout(k)+D(k);

error(k)=rin(k)-yout(k);

%PID Controller with partly differential
ei=ei+error(k)*ts;
kc=0.30;
ki=0.0055;
TD=140;

kd=kc*TD/ts;

Tf=180;
Q=tf([1],[Tf,1]); %Low Freq Signal Filter

M=2;
if M==1 %Using PID with Partial differential
    alfa=Tf/(ts+Tf);
    ud(k)=kd*(1-alfa)*(error(k)-error_1)+alfa*ud_1;

```

```

    u(k)=kc*error(k)+ud(k)+ki*ei;
    ud_1=ud(k);
elseif M==2 %Using Simple PID
    u(k)=kc*error(k)+kd*(error(k)-error_1)+ki*ei;
end

%Restricting the output of controller
if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');
figure(4);
bode(Q,'r');
dcgain(Q);

```

1.3.11 微分先行 PID 控制算法及仿真

微分先行 PID 控制的结构如图 1-34 所示,其特点是只对输出量 $yout(k)$ 进行微分,而对给定值 $rin(k)$ 不作微分。这样,在改变给定值时,输出不会改变,而被控量的变化通常是比较缓和的。这种输出量先行微分控制适用于给定值 $rin(k)$ 频繁升降的场合,可以避免给定值升降时所引起的系统振荡,从而明显地改善了系统的动态特性。

令微分部分的传递函数为

$$\frac{u_D(s)}{y(s)} = \frac{T_D s + 1}{\gamma T_D s + 1} \quad \gamma < 1 \quad (1.20)$$

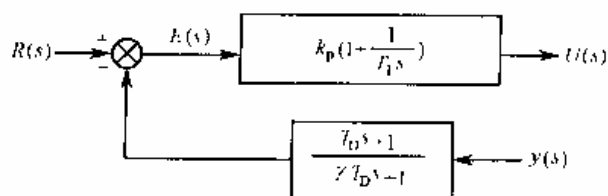


图 1-34 微分先行 ID 控制结构图

则

$$\gamma T_D \frac{du_D}{dt} + u_D = T_D \frac{dy}{dt} + y \quad (1.21)$$

由差分得

$$\begin{aligned} \frac{du_D}{dt} &\approx \frac{u_D(k) - u_D(k-1)}{T} \\ \frac{dy}{dt} &\approx \frac{y(k) - y(k-1)}{T} \\ \gamma T_D \frac{u_D(k) - u_D(k-1)}{T} + u_D(k) &= T_D \frac{y(k) - y(k-1)}{T} + y(k) \\ u_D(k) &= \left(\frac{\gamma T_D}{\gamma T_D + T} \right) u_D(k-1) + \left(\frac{T_D + T}{\gamma T_D + T} \right) y(k) - \left(\frac{T_D}{\gamma T_D + T} \right) y(k-1) \\ u_D(k) &= c_1 u_D(k-1) + c_2 y(k) - c_3 y(k-1) \end{aligned} \quad (1.22)$$

式中,

$$c_1 = \frac{\gamma T_D}{\gamma T_D + T}, \quad c_2 = \frac{T_D + T}{\gamma T_D + T}, \quad c_3 = \frac{T_D}{\gamma T_D + T} \quad (1.23)$$

仿真实例

设被控对象为一延迟对象

$$G(s) = \frac{e^{-80s}}{60s + 1}$$

采样时间为 20s, 延迟时间为 4 个采样时间, 即 80s。采用 PID 控制器进行阶跃响应。输入信号为带有高频下扰的方波信号: $rin(t) = 1.0 \text{sgn}(\sin(0.0005\pi t)) + 0.05 \sin(0.03\pi t)$ 。

取 $M = 1$, 采用微分先行 PID 控制方法, 其方波响应结果如图 1-35 和图 1-36 所示。取 $M = 2$, 采用普通 PID 方法, 其方波响应控制结果如图 1-37 和图 1-38 所示。由仿真结果可以看出, 对于给定值 $rin(k)$ 频繁升降的场合, 引入微分先行后, 可以避免给定值升降时所引起的系统振荡, 明显地改善了系统的动态特性。

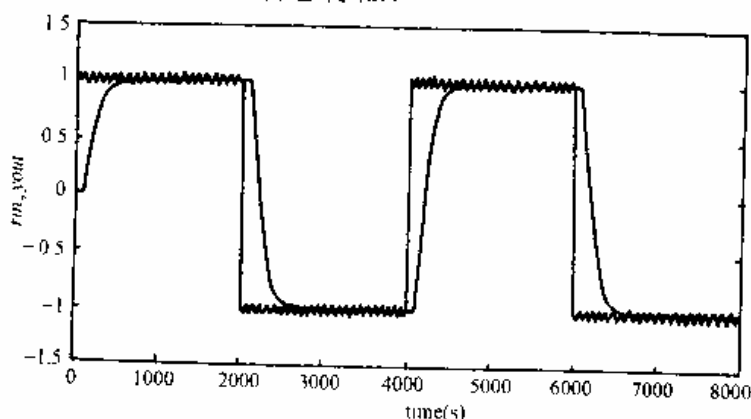


图 1-35 微分先行 PID 控制方波响应 ($M=1$)

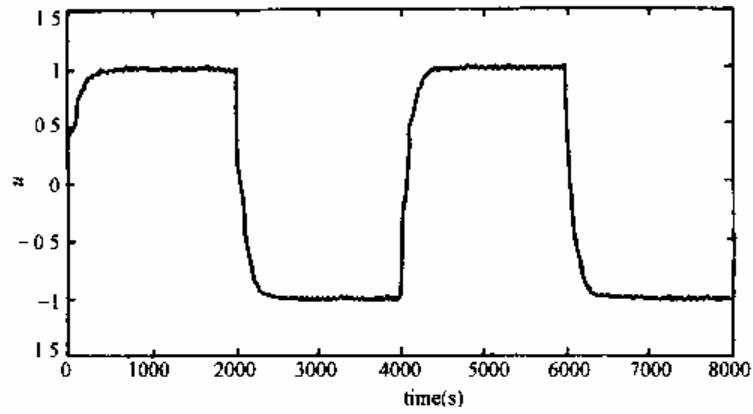


图 1-36 微分先行 PID 控制方波响应控制器输出 ($M=1$)

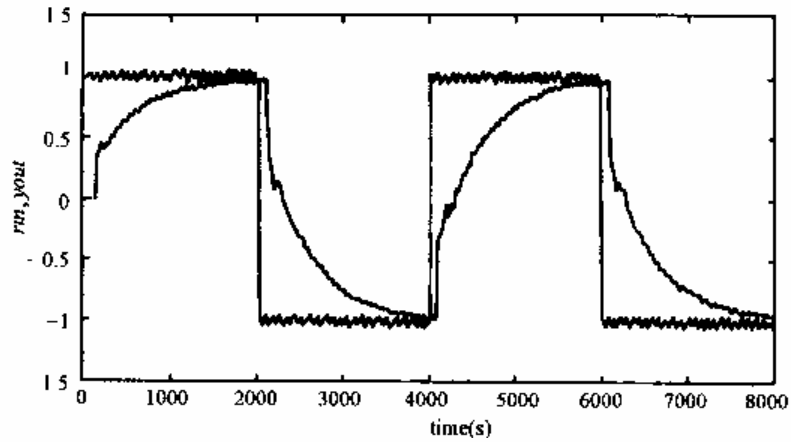


图 1-37 普通 PID 控制方波响应 ($M=2$)

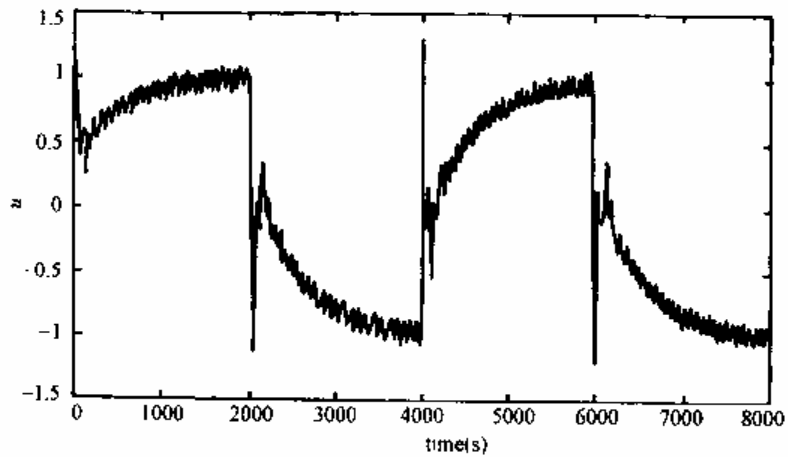


图 1-38 普通 PID 控制方波响应控制器输出 ($M=2$)

仿真程序: chap1_13.m

```
%PID Controller with differential in advance
```

```
clear all;
```

```
close all;
```

```
ts=20;
```



```

sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
ud_1=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;
ei=0;
for k=1:1:400
time(k)=k*ts;

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

kp=0.36;kd=14;ki=0.0021;

rin(k)=1.0*sign(sin(0.00025*2*pi*k*ts));
rin(k)=rin(k)+0.05*sin(0.03*pi*k*ts);

error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

gama=0.50;
Td=kd/kp;
Ti=0.5;

c1=gama*Td/(gama*Td+ts);
c2=(Td+ts)/(gama*Td+ts);
c3=Td/(gama*Td+ts);

M=2;
if M==1 %PID Control with differential in advance
ud(k)=c1*ud_1+c2*yout(k)-c3*y_1;
u(k)=kp*error(k)+ud(k)+ki*ei;
elseif M==2 %Simple PID Control
u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+ki*ei;
end

```

```

if u(k)>=110
    u(k)=110;
end
if u(k)<=-110
    u(k)=-110;
end
%Update parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin, 'r',time,yout, 'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u, 'r');
xlabel('time(s)');ylabel('u');

```

1.3.12 带死区的 PID 控制算法及仿真

在计算机控制系统中，某些系统为了避免控制作用过于频繁，消除由于频繁动作所引起的振荡，可采用带死区的 PID 控制算法，控制算式为

$$e(k) = \begin{cases} 0 & |e(k)| \leq |e_0| \\ e(k) & |e(k)| > |e_0| \end{cases} \quad (1.24)$$

式中， $e(k)$ 为位置跟踪偏差， e_0 是一个可调参数，其具体数值可根据实际控制对象由实验确定。若 e_0 值太小，会使控制动作过于频繁，达不到稳定被控对象的目的；若 e_0 太大，则系统将产生较大的滞后。

带死区的控制系统实际上是一个非线性系统，当 $|e(k)| \leq |e_0|$ 时，数字调节器输出为零；当 $|e(k)| > |e_0|$ 时，数字输出调节器有 PID 输出。带死区的 PID 控制算法流程图如图 1-39 所示。

仿真实例

被控对象为

$$G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms，对象输出上有一个幅值为 0.5 的正态分布的随机干扰信号。采用积分分离式 PID 控制算法进行阶跃响应，取 $\varepsilon = 0.20$ ，死区参数 $e_0 = 0.10$ ，采用低通滤波器对

对象输出信号进行滤波，滤波器为

$$Q(s) = \frac{1}{0.04s + 1}$$

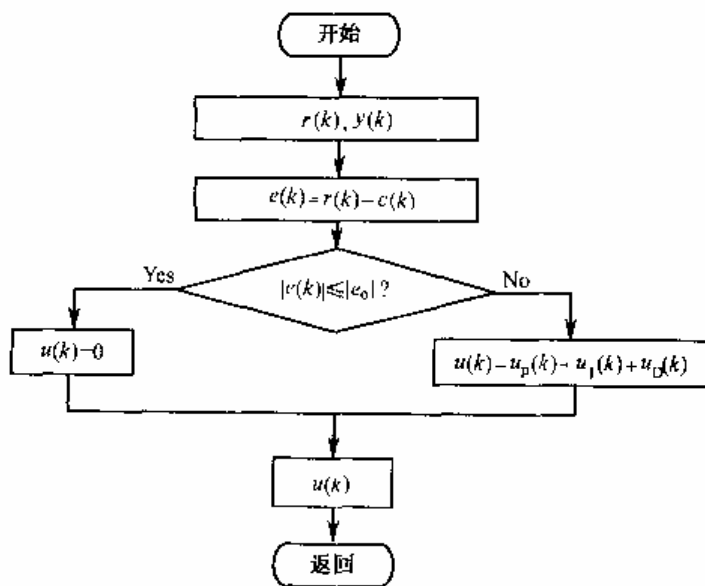


图 1-39 带死区的 PID 控制算法程序框图

取 $M = 1$ ，采用一般积分分离式 PID 控制方法，其控制结果如图 1-40 所示。取 $M = 2$ ，采用带死区的积分分离式 PID 控制方法，其控制结果如图 1-41 所示。由仿真结果可以看出，引入带死区 PID 控制后，控制器输出更加平稳。

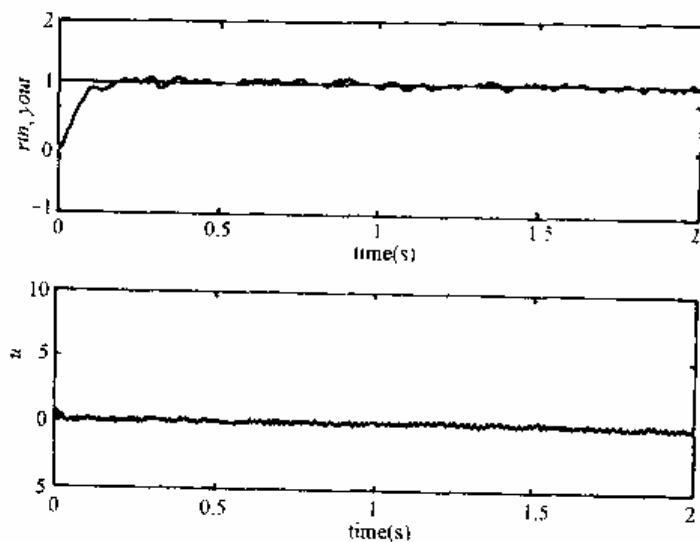


图 1-40 不带死区 PID 控制 ($M = 1$)

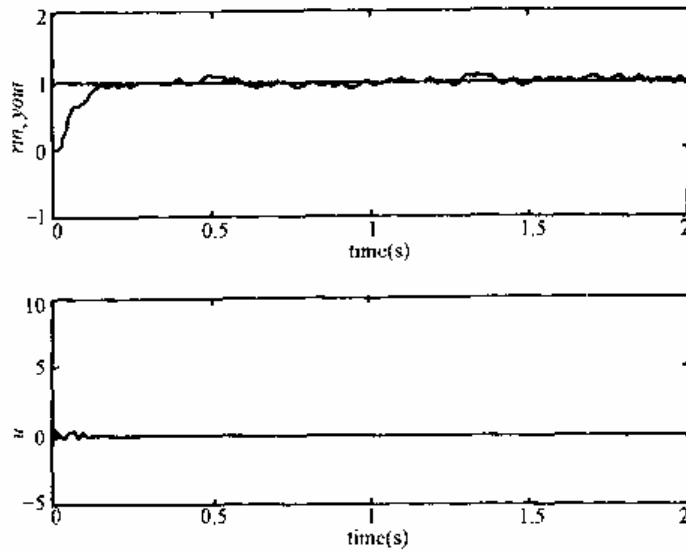


图 1-41 带死区 PID 控制 ($M = 2$)

仿真程序: chap1_14.m

%PID Controller with dead zone

clear all;

close all;

ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]);

dsys=c2d(sys,ts,'z');

[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;

y_1=0;y_2=0;y_3=0;

yy_1=0;

error_1=0;error_2=0;ei=0;

sys1=tf([1],[0.04,1]); %Low Freq Signal Filter

dsys1=c2d(sys1,ts,'tustin');

[num1,den1]=tfdata(dsys1,'v');

f_1=0;

for k=1:1:2000

time(k)=k*ts;

rin(k)=1; %Step Signal

%Linear model

yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+...

```

num(3)*u_2+num(4)*u_3;

D(k)=0.50*rands(1);    %Disturbance signal
yyout(k)=yout(k)+D(k);

%Low frequency filter
filty(k)=-den1(2)*f_1+num1(1)*(yyout(k)+yy_1);
error(k)=rin(k)-filty(k);

if abs(error(k))<=0.20
    ei=ei+error(k)*ts;
else
    ei=0;
end

kp=0.50;ki=0.10;kd=0.020;
u(k)=kp*error(k)+ki*ei+kd*(error(k)-error_1)/ts;

M=2;
if M==1
    u(k)=u(k);
elseif M==2 %Using Dead zone
    if abs(error(k))<=0.10
        u(k)=0;
    end
end

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

%-----Return of PID parameters-----
rin_1=rin(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

f_1=filty(k);
yy_1=yyout(k);

```

```

error_2=error_1;
error_1=error(k);
end
figure(1);
subplot(211);
plot(time,rin,'r',time,filty,'b');
xlabel('time(s)');ylabel('rin,yout');
subplot(212);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(2);
plot(time,D,'r');
xlabel('time(s)');ylabel('Disturbance signal');

```

1.3.13 基于前馈补偿的 PID 控制算法及仿真

在高精度伺服控制中，前馈控制可用来提高系统的跟踪性能。经典控制理论中的前馈控制设计是基于复合控制思想，当闭环系统为连续系统时，使前馈环节与闭环系统的传递函数之积为 1，从而实现输出完全复现输入。作者利用前馈控制的思想，针对 PID 控制设计了前馈补偿，以提高系统的跟踪性能，其结构如图 1-42 所示。

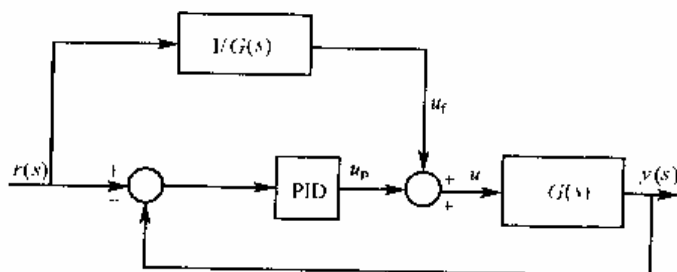


图 1-42 PID 前馈控制结构

设计前馈补偿控制器为

$$u_f(s) = r(s) \frac{1}{G(s)} \quad (1.25)$$

总控制输出为 PID 控制输出 + 前馈控制输出

$$u(t) = u_p(t) + u_f(t) \quad (1.26)$$

写成离散形式为

$$u(k) = u_p(k) + u_f(k) \quad (1.27)$$

仿真实例

被控对象为

$$G(s) = \frac{133}{s^2 + 25s}$$

输入信号为： $r(k) = 0.5 \sin(6\pi t)$ ，采样时间为 1ms。

$$u_f(t) = \frac{25}{133} \dot{r}(t) + \frac{1}{133} \ddot{r}(t)$$

写成离散形式为

$$u_f(k) = \frac{25}{133} \dot{r}(k) + \frac{1}{133} \ddot{r}(k)$$

只采用 PID 正弦跟踪控制方法的结果如图 1-43 和图 1-44 所示, 采用前馈 PID 控制方法的跟踪结果如图 1-45 和图 1-46 所示。可见通过前馈补偿可大大提高系统的跟踪性能。

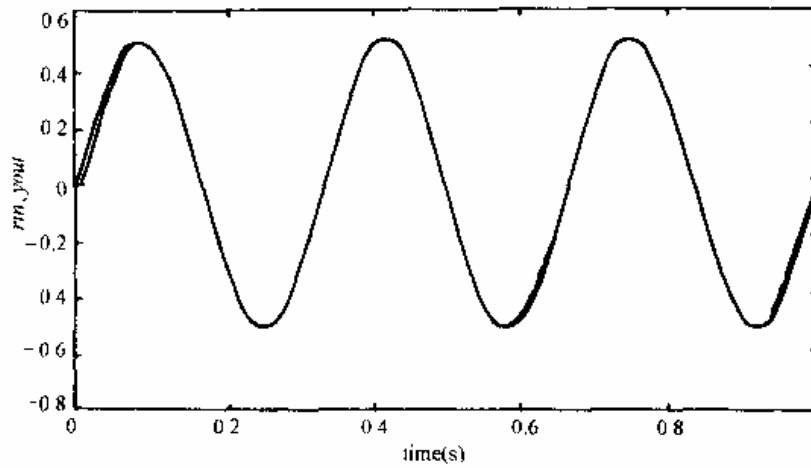


图 1-43 PID 正弦跟踪 ($M=1$)

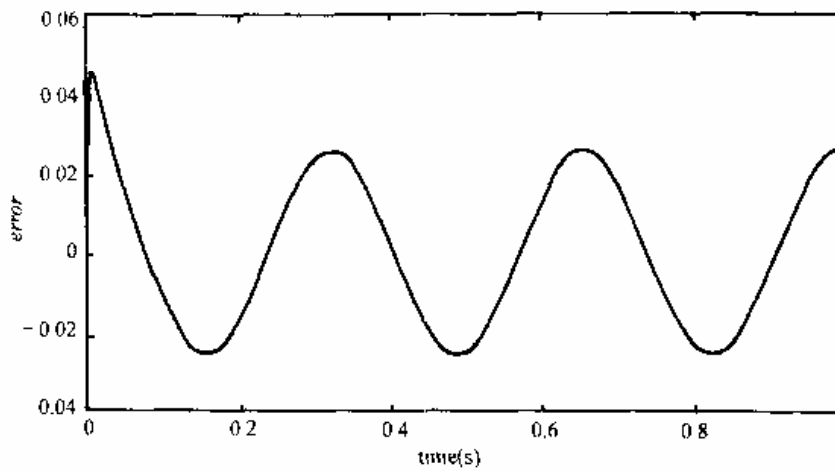


图 1-44 PID 正弦跟踪误差 ($M=1$)

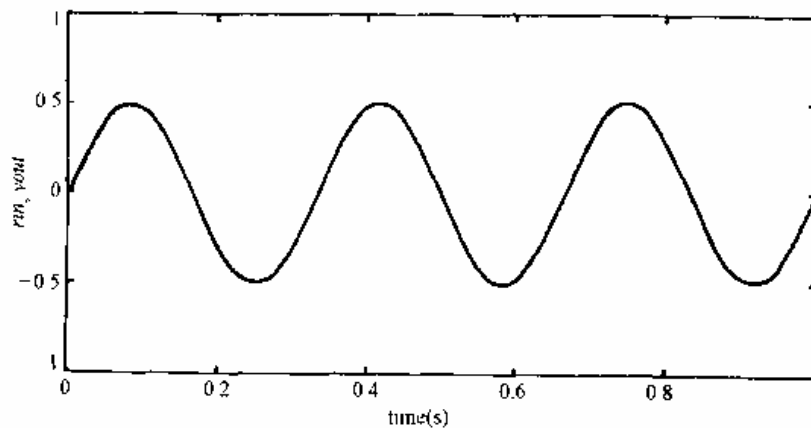


图 1-45 PID+前馈补偿正弦跟踪 ($M=2$)

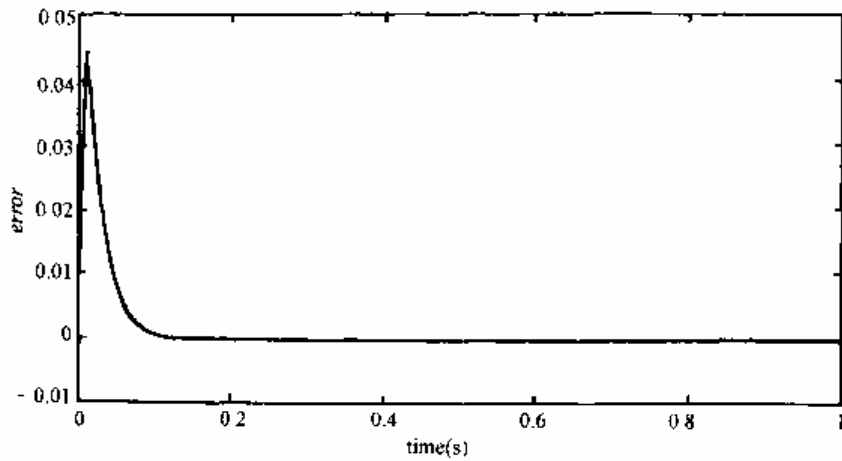


图 1-46 PID+前馈补偿正弦跟踪误差 ($M=2$)

仿真程序: chap1_15.m

```
%PID Feedforward Controller
```

```
clear all;
```

```
close all;
```

```
ts=0.001;
```

```
sys=tf(133,[1,25,0]);
```

```
dsys=c2d(sys,ts,'z');
```

```
[num,den]=tfdata(dsys,'v');
```

```
u_1=0;u_2=0;
```

```
y_1=0;y_2=0;
```

```
error_1=0;ei=0;
```

```
for k=1:1:1000
```

```
time(k)=k*ts;
```

```
A=0.5;F=3.0;
```

```
rin(k)=A*sin(F*2*pi*k*ts);
```

```
drin(k)=A*F*2*pi*cos(F*2*pi*k*ts);
```

```
ddrin(k)=-A*F*2*pi*F*2*pi*sin(F*2*pi*k*ts);
```

```
%Linear model
```

```
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
```

```
error(k)=rin(k)-yout(k);
```

```
ei=ei+error(k)*ts;
```



```

up(k)=80*error(k)+20*ei+2.0*(error(k)-error_1)/ts;

uf(k)=25/133*drin(k)+1/133*ddrin(k);

M=2;
if M==1      %Only using PID
    u(k)=up(k);
elseif M==2  %PID+Feedforward
    u(k)=up(k)+uf(k);
end

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,up,'k',time,uf,'b',time,u,'r');
xlabel('time(s)');ylabel('up,uf,u');

```

1.3.14 步进式 PID 控制算法及仿真

在较大阶跃响应时，很容易产生超调。采用步进式积分分离 PID 控制，该方法不直接对阶跃信号进行响应，而是使输入指令信号一步一步地逼近所要求的阶跃信号，可使对象运行平稳，适用于高精度伺服系统的位置跟踪。

仿真实例

被控对象为

$$G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms，输入指令信号为 $rd = 20$ 。采用本控制算法进行阶跃响应。其中 $M = 1$ 时为积分分离式 PID 控制，响应结果见图 1-47， $M = 2$ 时为步进式积分分离 PID 控制，响应结果及输入信号的变化见图 1-48 和图 1-49。

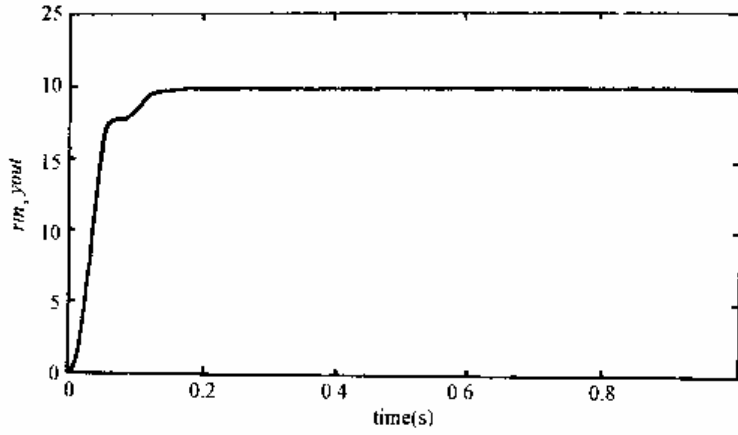


图 1-47 积分分离阶跃响应 ($M = 1$)

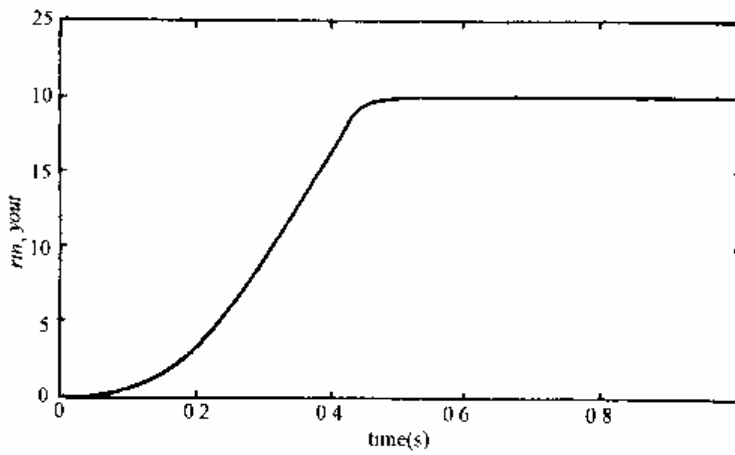


图 1-48 步进式积分分离阶跃响应 ($M = 2$)

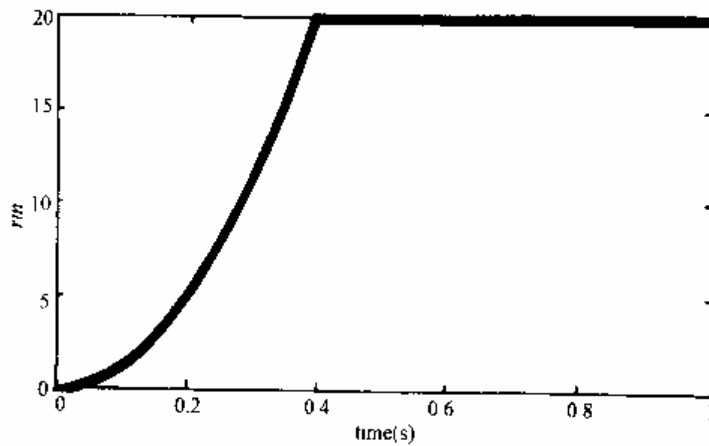


图 1-49 步进式阶跃信号的变化

在步进式 PID 控制的仿真中, 实际输入指令 $rin(k)$ 采用 0.25 的步长变化, 逐渐逼近输入指令信号 rd 。仿真结果表明, 采用积分分离式 PID 控制, 响应速度快, 但阶跃响应不平稳; 而采用步进式 PID 控制, 虽然响应速度慢, 但阶跃响应平稳, 具有很好的工程实用价值。

仿真程序: chap1_16.m

```
%PID Control with Gradual approaching input value
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;ei=0;

kp=0.50;ki=0.05;
rate=0.25;
rini=0.0;

for k=1:1:1000
time(k)=k*ts;
rd=20; %Step Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

M=2;
if M==1 %Using simple PID
rin(k)=rd;
error(k)=rin(k)-yout(k);
end
if M==2 %Using Gradual approaching input value
if rini<rd-0.25
rin(k)=rini+k*ts*rate;
elseif rini>rd+0.25
```

```

        rini=rini-k*ts*rate;
    else
        rini=rd;
    end
    rin(k)=rini;
    error(k)=rin(k)-yout(k);
end

%PID with I separation
if abs(error(k))<=0.8
    ei=ei+error(k)*ts;
else
    ei=0;
end
u(k)=kp*error(k)+ki*ei;

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
%--- -----Return of PID parameters-----
rin_1=rin(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rd,'b',time,yout,'r');
xlabel('time(s)');ylabel('rd,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,rin,'r');
xlabel('time(s)');ylabel('rin');

```

第 2 章 常用的 PID 控制系统

2.1 单回路 PID 控制系统

系统只有一个 PID 控制器，如图 2-1 所示。本书所述的大部分内容都是关于单回路 PID 控制系统的。

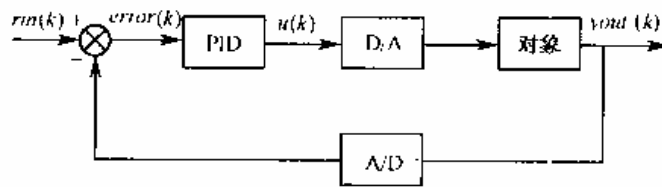


图 2-1 单回路 PID 控制系统

单回路 PID 控制系统的 MATLAB 仿真见第 1 章。

2.2 串级 PID 控制

2.2.1 串级 PID 控制原理

串级计算机控制系统的典型结构如图 2-2 所示，系统中有两个 PID 控制器， $G_{c2}(s)$ 称为副调节器传递函数，包围 $G_{c2}(s)$ 的内环称为副回路。 $G_{c1}(s)$ 称为主调节器传递函数，包围 $G_{c1}(s)$ 的外环称为主回路。主调节器的输出控制量 u_1 作为副回路的给定量 $R_2(s)$ 。

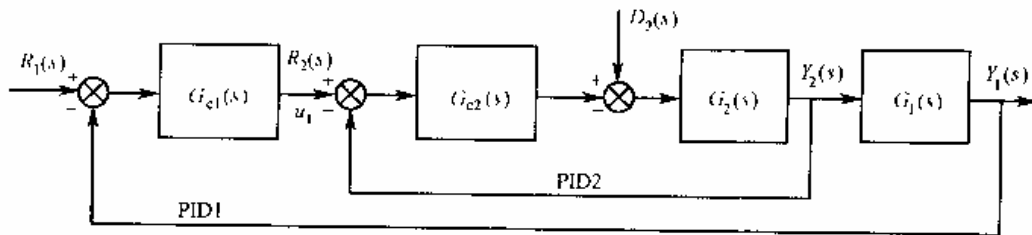


图 2-2 串级控制系统框图

串级控制系统的计算顺序是先主回路 (PID1)，后副回路 (PID2)。控制方式有两种：一种是异步采样控制，即主回路的采样控制周期 T_1 是副回路采样控制周期 T_2 的整数倍。这是因为一般串级控制系统中主控对象的响应速度慢、副控对象的响应速度快的缘故。另一种是同步采样控制，即主、副回路的采样控制周期相同。这时，应根据副回路选择采样周期，因为副回路的受控对象的响应速度较快。

串级控制的主要优点：

- (1) 将干扰加到副回路中，由副回路控制对其进行抑制；
- (2) 副回路中参数的变化，由副回路给予控制，对被控量 G_{c1} 的影响大为减弱；

(3) 副回路的惯性由副回路给予调节, 因而提高了整个系统的响应速度。

副回路是串级系统设计的关键。副回路设计的方式有很多种, 下面介绍按预期闭环特性设计副调节器的设计方法。

由副回路框图可得副回路闭环系统的传递函数为

$$\varphi_2(z) = \frac{Y_2(z)}{U_1(z)} = \frac{G_{c2}(z)G_2(z)}{1 + G_{c2}(z)G_2(z)} \quad (2.1)$$

可得副调节器控制律

$$G_{c2}(z) = \frac{\varphi_2(z)}{G_2(z)(1 - \varphi_2(z))} \quad (2.2)$$

一般选择

$$\varphi_2(z) = z^{-n} \quad (2.3)$$

式中, n 为 $G_2(z)$ 有理多项式分母最高次幂。

2.2.2 仿真程序及分析

仿真实例

设副对象特性为 $G_2(s) = \frac{1}{T_{02}s + 1}$, 主对象特性为 $G_1(s) = \frac{1}{T_{01}s + 1}$, $T_{01} = T_{02} = 10$, 采样时间为 2s, 外加干扰信号为一幅度为 0.01 的随机信号

$$d_2(k) = 0.01\text{rands}(1)$$

按预期闭环方法设计副调节器。由于副对象的传递函数为一阶, 故由式 (2.3) 得到副回路闭环系统传递函数

$$\varphi_2(z) = z^{-1}$$

主调节器采用 PI 控制, 取 $k_p = 1.2, k_i = 0.02$, 副调节器按控制律式 (2.2) 设计。针对阶跃信号的输入、输出跃阶响应结果如图 2-3 至 2-5 所示。

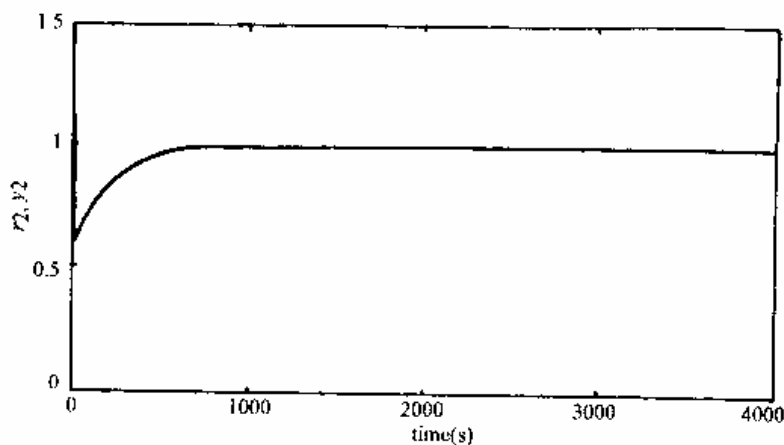


图 2-3 副回路输入、输出

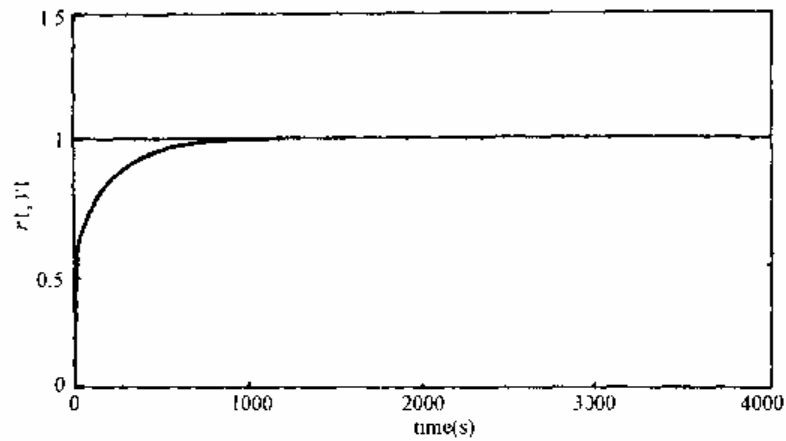


图 2-4 主回路阶跃响应

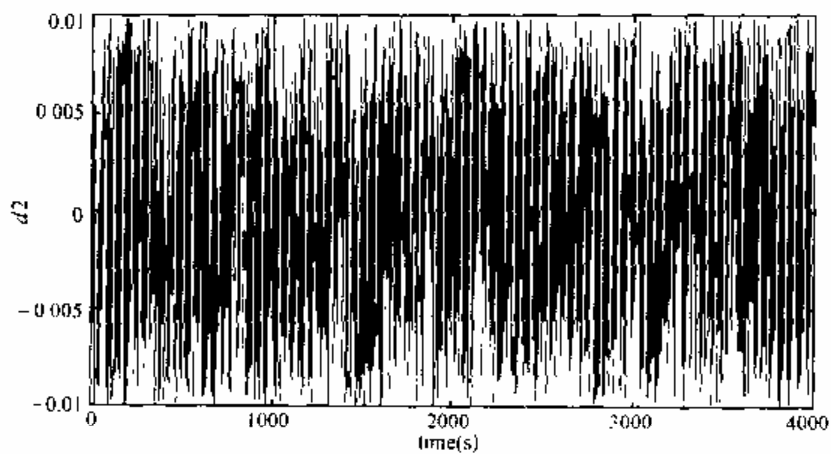


图 2-5 外加干扰信号

```

仿真程序: chap2_1.m
%Series System Control
clear all;
close all;

ts=2;
sys1=tf(1,[10,1]);
dsys1=c2d(sys1,ts,'z');
[num1,den1]=tfdata(dsys1,'v');

sys2=tf(1,[10,1]);
dsys2=c2d(sys2,ts,'z');
[num2,den2]=tfdata(dsys2,'v');

dph=1/zpk('z',ts);
Gc2=dph/(dsys2*(1-dph));
[nump,denp]=tfdata(Gc2,'v');

```

```

u1_1=0.0;u2_1=0.0;
y1_1=0;y2_1=0;
e2_1=0;ei=0;

for k=1:1:2000
time(k)=k*ts;

r1(k)=1;
%Linear model
y1(k)=-den1(2)*y1_1+num1(2)*y2_1; %Main plant

y2(k)=-den2(2)*y2_1+num2(2)*u2_1; %Assistant plant

error(k)=r1(k)-y1(k);
ei=ei+error(k);
u1(k)=1.2*error(k)+0.02*ei; %Main Controller

e2(k)=u1(k)-y2(k); %Assistant Controller
u2(k)=-denp(2)*u2_1+nump(1)*e2(k)+nump(2)*e2_1;

d2(k)=0.01*rands(1);
u2(k)=u2(k)+d2(k);

%-----Return of PID parameters-----
u1_1=u1(k);
u2_1=u2(k);

e2_1=e2(k);

y1_1=y1(k);
y2_1=y2(k);
end
figure(1); %Assistant Control
plot(time,u1,'b',time,y2,'r');
xlabel('time(s)');ylabel('u1,y2');

figure(2); %Main Control
plot(time,r1,'b',time,y1,'r');
xlabel('time(s)');ylabel('r1,y1');

```



```

figure(3);
plot(time,d2,'r');
xlabel('time(s)');ylabel('disturbance');

```

2.3 纯滞后系统的大林控制算法

2.3.1 大林控制算法原理

早在 1968 年, 美国 IBM 公司的大林 (Dahlin) 就提出了一种不同于常规 PID 控制规律的新型算法, 即大林算法。该算法的最大特点是将期望的闭环响应设计成一阶惯性加纯延迟, 然后反过来得到能满足这种闭环响应的控制器。

对于如图 2-6 所示的单回路控制系统, $G_c(z)$ 为数字控制器, $G_p(z)$ 为被控对象, 则闭环系统传递函数为

$$\phi(z) = \frac{Y(z)}{R(z)} = \frac{G_c(z)G_p(z)}{1 + G_c(z)G_p(z)} \quad (2.4)$$

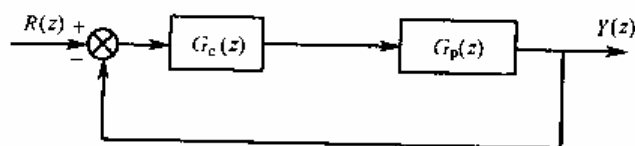


图 2-6 单回路控制系统框图

则

$$G_c(z) = \frac{U(z)}{E(z)} = \frac{1}{G_p(z)} \frac{\phi(z)}{1 - \phi(z)} \quad (2.5)$$

如果能事先设定系统的闭环响应 $\phi(z)$, 则可得控制器 $G_c(z)$ 。大林指出, 通常的期望闭环响应是一阶惯性加纯延迟形式, 其延迟时间等于对象的纯延迟时间 τ

$$\phi(s) = \frac{Y(s)}{R(s)} = \frac{e^{-\tau s}}{T_\phi s + 1} \quad (2.6)$$

式中, T_ϕ 为闭环系统的时间常数, 由此而得到的控制律称为大林算法。

2.3.2 仿真程序及分析

仿真实例

设被控对象为

$$G_p(s) = \frac{e^{-0.76s}}{0.4s + 1}$$

采样时间为 0.5s, 期望的闭环响应设计为

$$\phi(s) = \frac{Y(s)}{R(s)} = \frac{e^{-0.76s}}{0.15s + 1}$$

$M=1$ 时为采用大林控制算法; $M=2$ 时为采用普通 PID 控制算法。可见, 采用大林算法可取得很好的控制效果, 其阶跃响应结果如图 2-7 和图 2-8 所示。

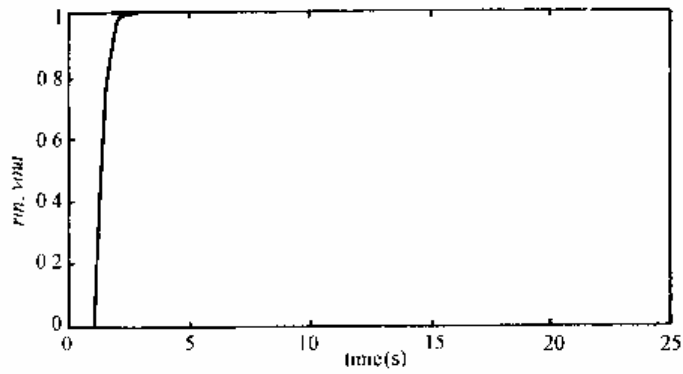


图 2-7 大林算法阶跃响应 ($M=1$)

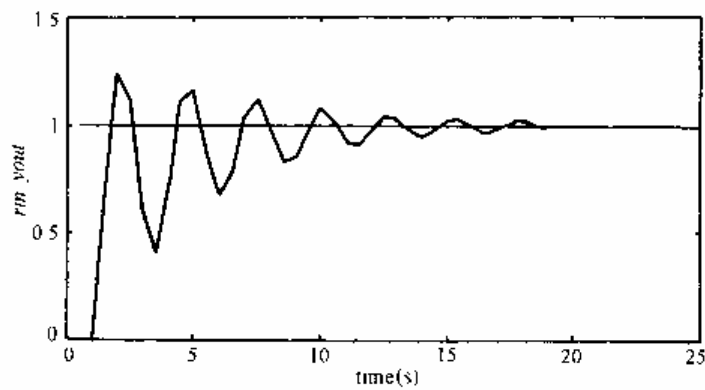


图 2-8 普通 PID 算法阶跃响应 ($M=2$)

仿真程序: chap2 2.m

%Delay Control with Dalin Algorithm

clear all;

close all;

ts=0.5;

%Plant

sys1=tf([1],[0.4,1],'inputdelay',0.76);

dsys1=c2d(sys1,ts,'zoh');

[num1,den1]=tfdata(dsys1,'v');

%Ideal closed loop

sys2=tf([1],[0.15,1],'inputdelay',0.76);

dsys2=c2d(sys2,ts,'zoh');

%Design Dalin controller

dsys=1/dsys1*dsys2/(1-dsys2);

[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;u_4=0.0;u_5=0.0;

```

y_1=0.0;

error_1=0.0;error_2=0.0;error_3=0.0;
ei=0;
for k=1:1:50
time(k)=k*ts;

rin(k)=1.0; %Tracing Step Signal

yout(k)=-den1(2)*y_1+num1(2)*u_2+num1(3)*u_3;
error(k)=rin(k)-yout(k);

M=1;
if M==1 %Using Dalin Method
u(k)=(num(1)*error(k)+num(2)*error_1+num(3)*error_2+num(4)*error_3...
-den(3)*u_1-den(4)*u_2-den(5)*u_3-den(6)*u_4-den(7)*u_5)/den(2);
elseif M==2 %Using PID Method
ei=ei+error(k)*ts;
u(k)=1.0*error(k)+0.10*(error(k)-error_1)/ts+0.50*ei;
end
%-----Return of dalin parameters-----
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);

error_3=error_2;error_2=error_1;error_1=error(k);
end
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');

```

2.4 纯滞后系统的 Smith 控制算法

在工业过程控制中，许多被控对象具有纯滞后的性质。Smith（史密斯）提出了一种纯滞后补偿模型，其原理为：与 PID 控制器并接一补偿环节，该补偿环节称为 Smith 预估器。

2.4.1 连续 Smith 预估控制

带有纯延迟的单回路控制系统如图 2-9 所示，其闭环传递函数为

$$\phi(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G_0(s)e^{-\tau s}}{1 + G_c(s)G_0(s)e^{-\tau s}} \quad (2.7)$$

其特征方程为

$$1 + G_c(s)G_0(s)e^{-\tau s} = 0 \quad (2.8)$$

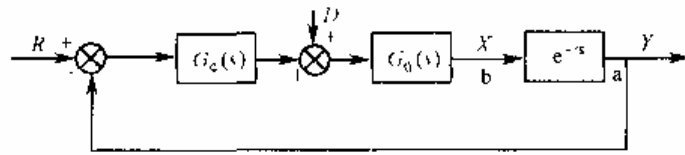


图 2-9 有纯延迟的单回路控制系统

可见，特征方程中出现了纯延迟环节，使系统稳定性降低，如果 τ 足够大，系统将不稳定，这就是大延迟过程难于控制的本质。而 $e^{-\tau s}$ 之所以在特征方程中出现，是由于反馈信号是从系统的 a 点引出来的，若能将反馈信号从 b 点引出，则把纯延迟环节移到控制回路的外边，如图 2-10 所示，经过 τ 的延迟时间后，被调量 Y 将重复 X 同样的变化。

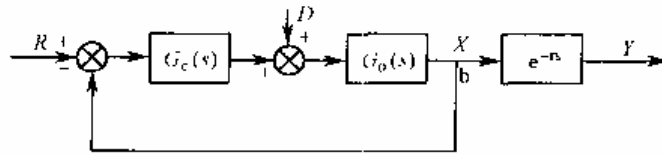


图 2-10 改进的有纯延迟的单回路控制系统

由于反馈信号 X 没有延迟，系统的响应会大大地改善。但在实际系统中，b 点或是不存在，或是受物理条件的限制，无法从 b 点引出反馈信号来。针对这种问题，Smith 提出采用人造模型的方法，构造如图 2-11 所示的控制系统。

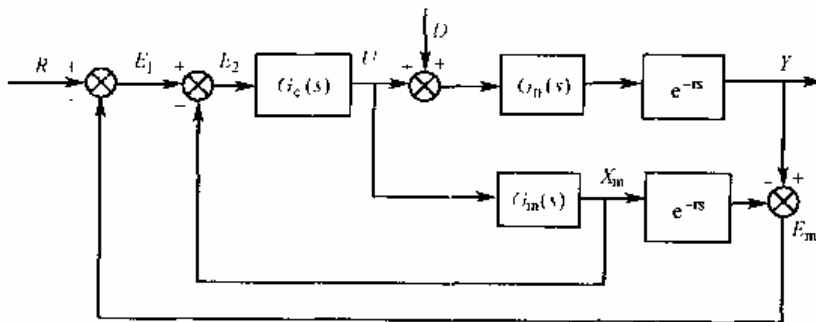


图 2-11 Smith 预估控制系统

如果模型是精确的，即 $G_0(s) = G_m(s)$, $\tau = \tau_m$ ，且不存在负荷扰动 ($D = 0$)，则 $Y = Y_m$ ， $E_m = Y - Y_m = 0$ ， $X = X_m$ ，则可以用 X_m 代替 X 作第一条反馈回路，实现将纯延迟环节移到控制回路的外边。如果模型是不精确的或是出现负荷扰动，则 X 就不等于 X_m ， $E_m = Y - Y_m \neq 0$ ，控制精度也就不能令人满意。为此，采用 E_m 实现第二条反馈回路。这就是 Smith 预估器的控制策略。

实际上预估模型不是并联在过程上，而是反向并联在控制器上的，因此，将图 2-11 变换可得到 Smith 预估控制系统等效图，如图 2-12 所示。

显然，Smith 控制方法的前提是必须确切地知道被控对象的数学模型，在此基础上才能建立精确的预估模型。

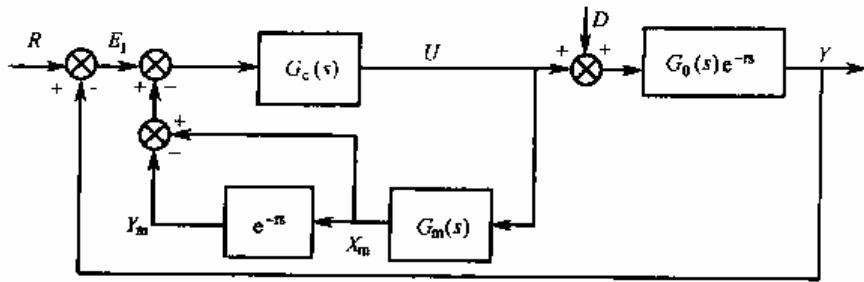


图 2-12 Smith 预估控制系统等效图

2.4.2 仿真程序及分析

仿真实例

被控对象为

$$G_p(s) = \frac{e^{-80s}}{60s+1}$$

采用 Smith 控制方法, 在 PI 控制中, 取 $k_p = 4.0, k_i = 0.022$, 假设预测模型精确, 阶跃指令信号取 100。Simulink 仿真程序及仿真结果如图 2-13 至图 2-15 所示, 仿真结果表明, Smith 控制方法具有很好控制效果。

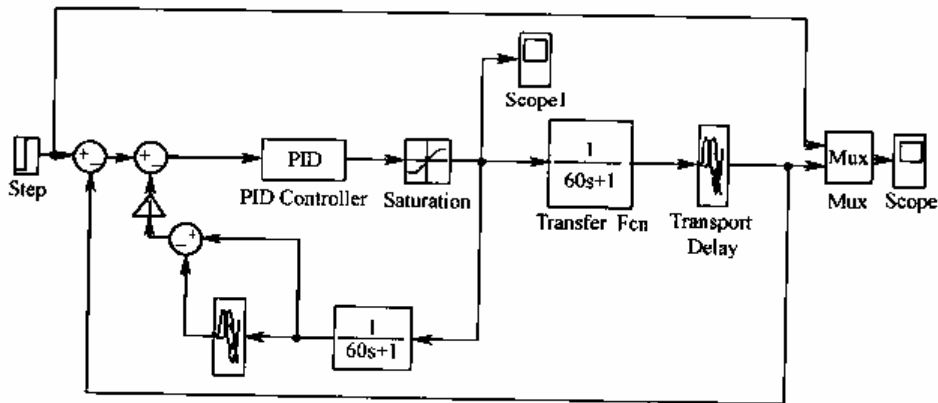


图 2-13 Simulink 仿真程序

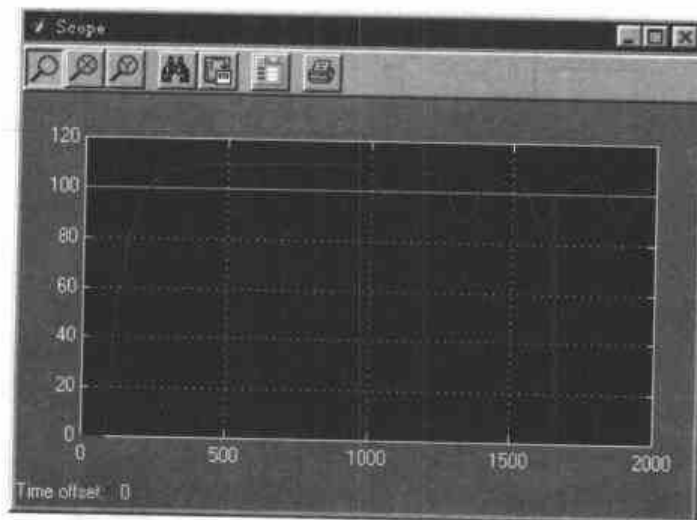


图 2-14 不用 Smith 补偿的阶跃响应

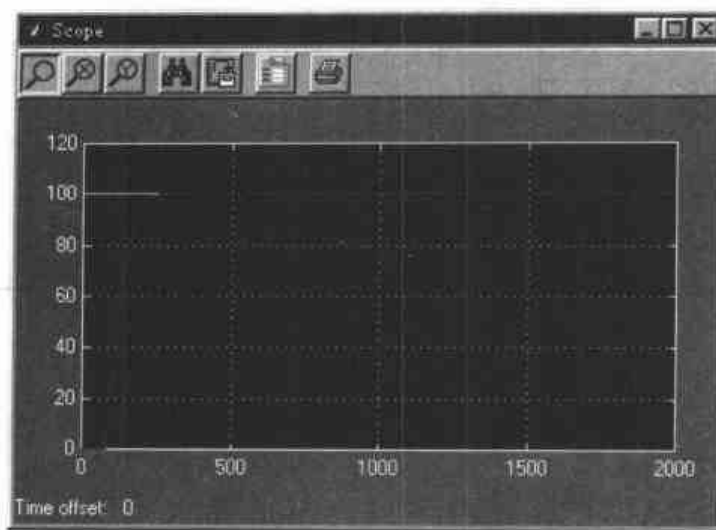


图 2-15 采用 Smith 补偿的阶跃响应

仿真程序: chap2_3.mdl, 如图 2-13 所示。

2.4.3 数字 Smith 预估控制

主要研究带有纯延迟的一阶过程在计算机控制时的史密斯预估控制算法的仿真。设被控对象的传递函数为

$$G_p(s) = \frac{k_p e^{-\alpha}}{T_p s + 1} = G_0(s) e^{-\alpha} \quad (2.9)$$

数字 Smith 预估控制系统的框图如图 2-16 所示。

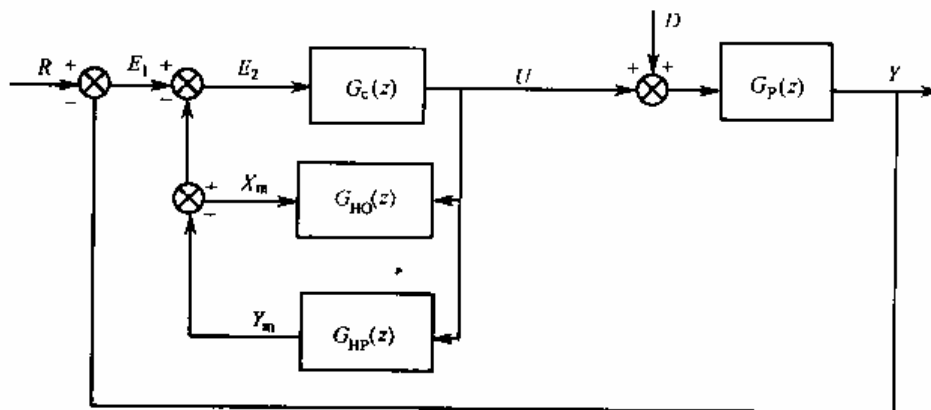


图 2-16 数字 Smith 预估控制系统框图

由图 2-16 可得

$$e_2(k) = e_1(k) - x_m(k) + y_m(k) = r(k) - y(k) - x_m(k) + y_m(k) \quad (2.10)$$

若模型是精确的, 则

$$y(k) = y_m(k) \quad (2.11)$$

$$e_2(k) = r(k) - x_m(k) \quad (2.12)$$

$e_2(k)$ 为数字控制器 $G_c(z)$ 的输入, $G_c(z)$ 一般采用 PI 控制算法。

2.4.4 仿真程序及分析

仿真实例

设被控对象为

$$G_p(s) = \frac{e^{-80s}}{60s+1}$$

采样时间为 20s，按 Smith 算法设计控制器。 S 代表指令信号的类型， $S=1$ 为阶跃响应， $S=2$ 为方波响应， M 代表三种情况下的仿真： $M=1$ 为模型不精确， $M=2$ 为模型精确， $M=3$ 为采用 PI 控制。取 $S=2$ ，针对 $M=1$ ， $M=2$ ， $M=3$ 三种情况进行仿真。在 PI 控制中， $k_p=0.50, k_i=0.010$ 。其响应结果如图 2-17 至图 2-19 所示。

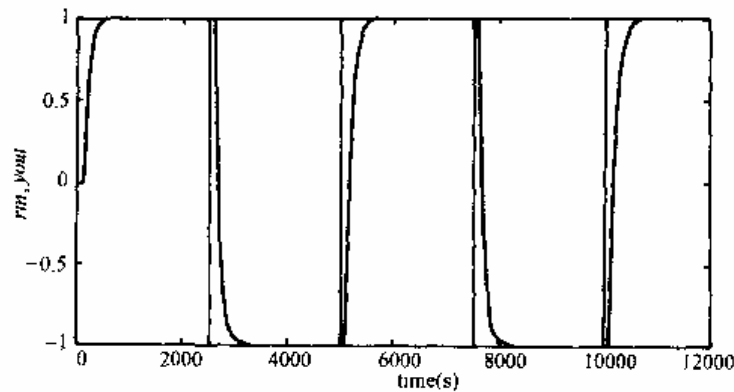


图 2-17 模型不精确时方波响应 ($M=1$)

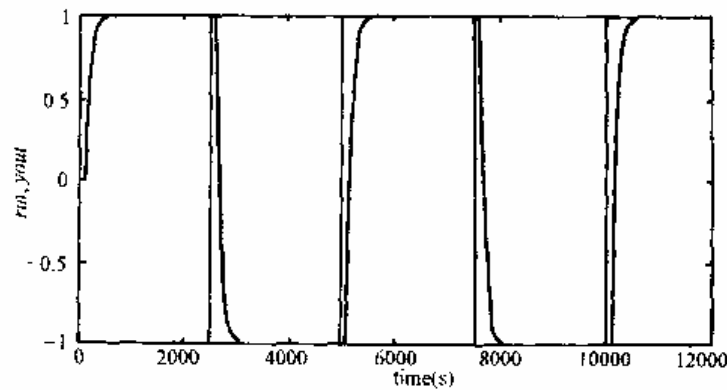


图 2-18 模型精确时方波响应 ($M=2$)

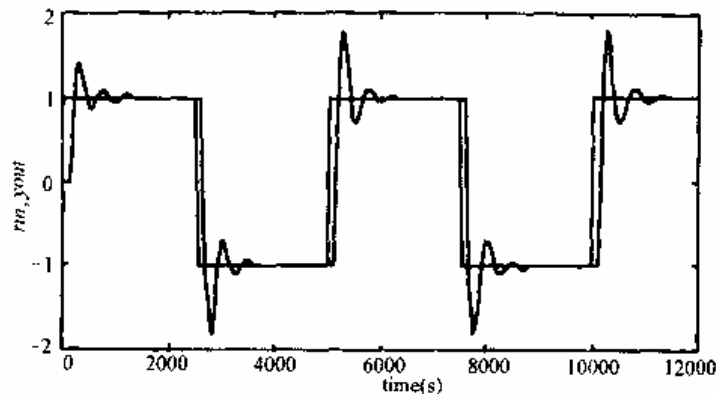


图 2-19 PI 控制时方波响应 ($M=3$)

仿真程序清单: chap2_4.m

```
%Big Delay PID Control with Smith Algorithm
```

```
clear all;close all;
```

```
Ts=20;
```

```
%Delay plant
```

```
kp=1;
```

```
Tp=60;
```

```
tol=80;
```

```
sys=tf([kp],[Tp,1],'inputdelay',tol);
```

```
dsys=c2d(sys,Ts,'zoh');
```

```
[num,den]=tfdata(dsys,'v');
```

```
M=1;
```

```
%Prediction model
```

```
if M==1 %No Precise Model: PI+Smith
```

```
    kp1=kp*1.10;
```

```
    Tp1=Tp*1.10;
```

```
    toll=tol*1.0;
```

```
elseif M==2|M==3 %Precise Model: PI+Smith
```

```
    kp1=kp;
```

```
    Tp1=Tp;
```

```
    toll=tol;
```

```
end
```

```
sys1=tf([kp1],[Tp1,1],'inputdelay',toll);
```

```
dsys1=c2d(sys1,Ts,'zoh');
```

```
[num1,den1]=tfdata(dsys1,'v');
```

```
u_1=0.0;u_2=0.0;u_3=0.0;u_4=0.0;u_5=0.0;
```

```
e1_1=0;
```

```
e2=0.0;
```

```
e2_1=0.0;
```

```
ei=0;
```

```
xm_1=0.0;
```

```
ym_1=0.0;
```

```
y_1=0.0;
```

```
for k=1:1:600
```



```

    time(k)=k*Ts;

S=2;
if S==1
    rin(k)=1.0;    %Tracing Step Signal
end
if S==2
    rin(k)=sign(sin(0.0002*2*pi*k*Ts)); %Tracing Square Wave Signal
end

%Prediction model
xm(k)=-den1(2)*xm_1+num1(2)*u_1;
ym(k)=-den1(2)*ym_1+num1(2)*u_5; %With Delay

yout(k)=-den(2)*y_1+num(2)*u_5;

if M==1    %No Precise Model: PI+Smith
    e1(k)=rin(k)-yout(k);
    e2(k)=e1(k)-xm(k)+ym(k);
    ei=ei+Ts*e2(k);
    u(k)=0.50*e2(k)+0.010*ei;
    e1_1=e1(k);
elseif M==2 %Precise Model: PI+Smith
    e2(k)=rin(k)-xm(k);
    ei=ei+Ts*e2(k);
    u(k)=0.50*e2(k)+0.010*ei;
    e2_1=e2(k);
elseif M==3 %Only PI
    e1(k)=rin(k)-yout(k);
    ei=ei+Ts*e1(k);
    u(k)=0.50*e1(k)+0.010*ei;
    e1_1=e1(k);
end

%-----Return of smith parameters-----
xm_1=xm(k);
ym_1=ym(k);

u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);

```

```
end  
plot(time,rin,'b',time,yout,'r');  
xlabel('time(s)');ylabel('rin,yout');
```

第3章 专家PID控制和模糊PID控制

3.1 专家PID控制

3.1.1 专家PID控制原理

专家控制 (Expert Control) 的实质是基于受控对象和控制规律的各种知识, 并以智能的方式利用这些知识来设计控制器。利用专家经验来设计PID参数便构成专家PID控制。

典型的二阶系统单位阶跃响应误差曲线如图3-1所示。对于典型的二阶系统阶跃响应过程作如下分析。

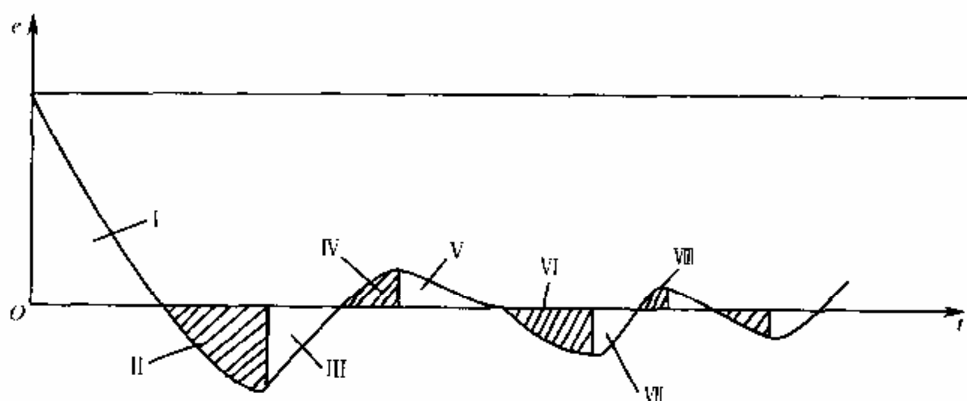


图3-1 典型二阶系统单位阶跃响应误差曲线

令 $e(k)$ 表示离散化的当前采样时刻的误差值, $e(k-1)$ 、 $e(k-2)$ 分别表示前一个和前两个采样时刻的误差值, 则有

$$\begin{aligned}\Delta e(k) &= e(k) - e(k-1) \\ \Delta e(k-1) &= e(k-1) - e(k-2)\end{aligned}\quad (3.1)$$

根据误差及其变化, 可设计专家PID控制器, 该控制器可分为以下五种情况进行设计:

(1) 当 $|e(k)| > M_1$ 时, 说明误差的绝对值已经很大。不论误差变化趋势如何, 都应考虑控制器的输出应按最大 (或最小) 输出, 以达到迅速调整误差, 使误差绝对值以最大速度减小。此时, 它相当于实施开环控制。

(2) 当 $e(k)\Delta e(k) > 0$ 时, 说明误差在朝误差绝对值增大方向变化, 或误差为某一常值, 未发生变化。此时, 如果 $|e(k)| \geq M_2$, 说明误差也较大, 可考虑由控制器实施较强的控制作用, 以达到扭转误差绝对值朝减小方向变化, 并迅速减小误差的绝对值, 控制器输出可为

$$u(k) = u(k-1) + k_1 \{k_p [e(k) - e(k-1)] + k_i e(k) + k_d [e(k) - 2e(k-1) + e(k-2)]\} \quad (3.2)$$

此时, 如果 $|e(k)| < M_2$, 说明尽管误差朝绝对值增大方向变化, 但误差绝对值本身并不

很大,可考虑控制器实施一般的控制作用,只要扭转误差的变化趋势,使其朝误差绝对值减小方向变化,控制器输出为

$$u(k) = u(k-1) + k_p [e(k) - e(k-1)] + k_i e(k) + k_d [e(k) - 2e(k-1) + e(k-2)] \quad (3.3)$$

(3) 当 $e(k)\Delta e(k) < 0$ 、 $\Delta e(k)\Delta e(k-1) > 0$ 或者 $e(k) = 0$ 时,说明误差的绝对值朝减小的方向变化,或者已经达到平衡状态。此时,可考虑采取保持控制器输出不变。

(4) 当 $e(k)\Delta e(k) < 0$ 、 $\Delta e(k)\Delta e(k-1) < 0$ 时,说明误差处于极值状态。如果此时误差的绝对值较大,即 $|e(k)| \geq M_2$,可考虑实施较强的控制作用

$$u(k) = u(k-1) + k_1 k_p e_m(k) \quad (3.4)$$

如果此时误差的绝对值较小,即 $|e(k)| < M_2$,可考虑实施较弱的控制作用

$$u(k) = u(k-1) + k_2 k_p e_m(k) \quad (3.5)$$

(5) 当 $|e(k)| \leq \varepsilon$ 时,说明误差的绝对值很小,此时加入积分,减少稳态误差。

式中, $e_m(k)$ ——误差 e 的第 k 个极值;

$u(k)$ ——第 k 次控制器的输出;

$u(k-1)$ ——第 $k-1$ 次控制器的输出;

k_1 ——增益放大系数, $k_1 > 1$;

k_2 ——抑制系数, $0 < k_2 < 1$;

M_1, M_2 ——设定的误差界限, $M_1 > M_2$;

k ——控制周期的序号(自然数);

ε ——任意小的正实数。

图 3-1 中, I、III、V、VII、... 区域,误差朝绝对值减小的方向变化。此时,可采取保持等待措施,相当于实施开环控制; II、IV、VI、VIII、... 区域,误差绝对值朝增大的方向变化。此时,可根据误差的大小分别实施较强或一般的控制作用,以抑制动态误差。

3.1.2 仿真程序及分析

仿真实例

求三阶传递函数的阶跃响应

$$G_p(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

其中对象采样时间为 1ms。

采用专家 PID 设计控制器。在仿真过程中, ε 取 0.001, 程序中的五条规则与控制算法的五种情况相对应, 其结果如图 3-2 至图 3-3 所示。

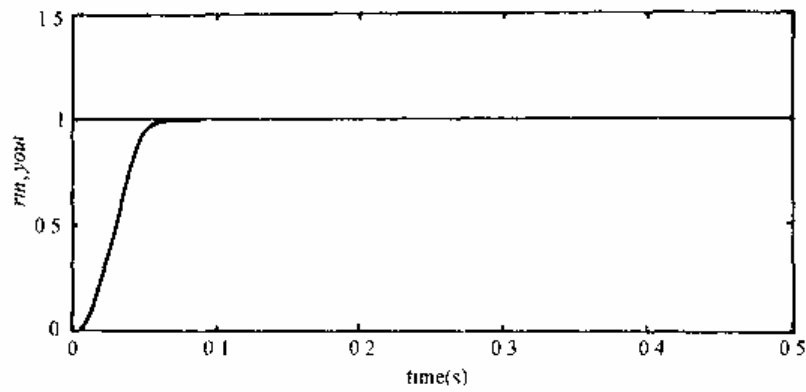


图 3-2 专家 PID 控制阶跃响应曲线

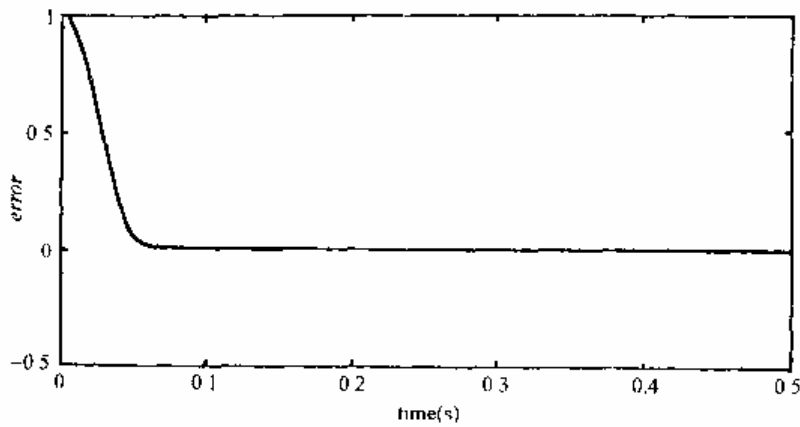


图 3-3 误差响应曲线

```

仿真程序: chap3_1.m
%Expert PID Controller
clear all;
close all;
ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';
x2_1=0;

kp=0.6;
ki=0.03;
kd=0.01;

```

```

error_i=0;
for k=1:1:500
time(k)=k*ts;

rin(k)=1.0;          %Tracing Jieyue Signal

u(k)=kp*x(1)+kd*x(2)+ki*x(3); %PID Controller

%Expert control rule
if abs(x(1))>0.8      %Rule1:Unclosed control firstly
    u(k)=0.45;
elseif abs(x(1))>0.40
    u(k)=0.40;
elseif abs(x(1))>0.20
    u(k)=0.12;
elseif abs(x(1))>0.01
    u(k)=0.10;
end

if x(1)*x(2)>0|(x(2)==0)      %Rule2
    if abs(x(1))>=0.05
        u(k)=u_1+2*kp*x(1);
    else
        u(k)=u_1+0.4*kp*x(1);
    end
end

if (x(1)*x(2)<0&x(2)*x2_1>0)|(x(1)==0) %Rule3
    u(k)=u(k);
end

if x(1)*x(2)<0&x(2)*x2_1<0 %Rule4
    if abs(x(1))>=0.05
        u(k)=u_1+2*kp*error_1;
    else
        u(k)=u_1+0.6*kp*error_1;
    end
end
end

```

```

if abs(x(1))<=0.001 %Rule5:Integration separation PI control
    u(k)=0.5*x(1)+0.010*x(3);
end

%Restricting the output of controller
if u(k)>=10
    u(k)=10;
end
if u(k)<= -10
    u(k)=-10;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(1)*u(k)+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);

% - - - - -Return of PID parameters-- - - - - - %
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k); % Calculating P
x2_1=x(2);
x(2)=(error(k)-error_1)/ts; % Calculating D
x(3)=x(3)+error(k)*ts; % Calculating I

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');

```

3.2 模糊自适应整定 PID 控制

3.2.1 模糊自适应整定 PID 控制原理

在工业生产过程中，许多被控对象随着负荷变化或干扰因素影响，其对象特性参数或

结构发生改变。自适应控制运用现代控制理论在线辨识对象特征参数，实时改变其控制策略，使控制系统品质指标保持在最佳范围内，但其控制效果的好坏取决于辨识模型的精确度，这对于复杂系统是非常困难的。因此，在工业生产过程中，大量采用的仍然是 PID 算法，PID 参数的整定方法很多，但大多数都以对象特性为基础。

随着计算机技术的发展，人们利用人工智能的方法将操作人员的调整经验作为知识存入计算机中，根据现场实际情况，计算机能自动调整 PID 参数，这样就出现了智能 PID 控制器。这种控制器把古典的 PID 控制与先进的专家系统相结合，实现系统的最佳控制。这种控制必须精确地确定对象模型，首先将操作人员（专家）长期实践积累的经验知识用控制规则模型化，然后运用推理便可对 PID 参数实现最佳调整。

由于操作者经验不易精确描述，控制过程中各种信号量以及评价指标不易定量表示，模糊理论为解决这一问题的有效途径，所以人们运用模糊数学的基本理论和方法，把规则的条件、操作用模糊集表示，并把这些模糊控制规则及有关信息(如评价指标、初始 PID 参数等)作为知识存入计算机知识库中，然后计算机根据控制系统的实际响应情况（即专家系统的输入条件），运用模糊推理，即可自动实现对 PID 参数的最佳调整，这就是模糊自适应 PID 控制。模糊自适应 PID 控制器目前有多种结构形式，但其工作原理基本一致。

自适应模糊 PID 控制器以误差 e 和误差变化 ec 作为输入，可以满足不同时刻的 e 和 ec 对 PID 参数自整定的要求。利用模糊控制规则在线对 PID 参数进行修改，便构成了自适应模糊 PID 控制器，其结构如图 3-4 所示。

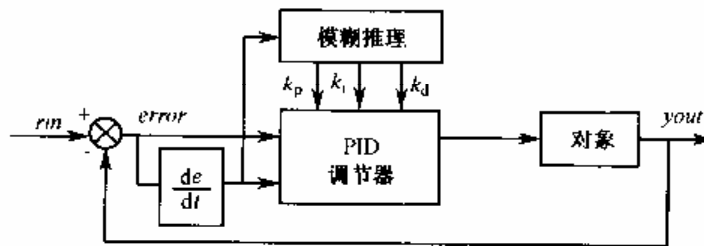


图 3-4 自适应模糊控制器结构

PID 参数模糊自整定是找出 PID 三个参数与 e 和 ec 之间的模糊关系，在运行中通过不断检测 e 和 ec ，根据模糊控制原理来对 3 个参数进行在线修改，以满足不同 e 和 ec 时对控制参数的不同要求，而使被控对象有良好的动、静态性能。

从系统的稳定性、响应速度、超调量和稳态精度等各方面来考虑， k_p, k_i, k_d 的作用如下：

(1) 比例系数 k_p 的作用是加快系统的响应速度，提高系统的调节精度。 k_p 越大，系统的响应速度越快，系统的调节精度越高，但易产生超调，甚至会导致系统不稳定。 k_p 取值过小，则会降低调节精度，使响应速度缓慢，从而延长调节时间，使系统静态、动态特性变坏

(2) 积分作用系数 k_i 的作用是消除系统的稳态误差。 k_i 越大，系统的静态误差消除越快，但 k_i 过大，在响应过程的初期会产生积分饱和现象，从而引起响应过程的较大超调。若 k_i 过小，将使系统静态误差难以消除，影响系统的调节精度。

(3) 微分作用系数 k_d 的作用是改善系统的动态特性，其作用主要是在响应过程中抑制

偏差向任何方向的变化，对偏差变化进行提前预报。但 k_d 过大，会使响应过程提前制动，从而延长调节时间，而且会降低系统的抗干扰性能。

PID 参数的整定必须考虑到在不同时刻三个参数的作用以及相互之间的互联关系。

在线实时模糊自整定 PID 控制器控制方案原理如图 3-4 所示。

模糊自整定 PID 是在 PID 算法的基础上，通过计算当前系统误差 e 和误差变化率 ec ，利用模糊规则进行模糊推理，查询模糊矩阵表进行参数调整。

模糊控制设计的核心是总结工程设计人员的技术知识和实际操作经验，建立合适的模糊规则表，得到针对 k_p ， k_i ， k_d 三个参数分别整定的模糊控制表。

(1) k_p 的模糊规则表（见表 3-1）

表 3-1 k_p 的模糊规则表

		ec						
		NB	NM	NS	ZO	PS	PM	PB
e	Δk_p							
	NB	PB	PB	PM	PM	PS	ZO	ZO
	NM	PB	PB	PM	PS	PS	ZO	NS
	NS	PM	PM	PM	PS	ZO	NS	NS
	ZO	PM	PM	PS	ZO	NS	NM	NM
	PS	PS	PS	ZO	NS	NS	NM	NM
	PM	PS	ZO	NS	NM	NM	NM	NB
	PB	ZO	ZO	NM	NM	NM	NB	NB

(2) k_i 的模糊规则表（见表 3-2）

表 3-2 k_i 的模糊规则表

		ec						
		NB	NM	NS	ZO	PS	PM	PB
e	Δk_i							
	NB	NB	NB	NM	NM	NS	ZO	ZO
	NM	NB	NB	NM	NS	NS	ZO	ZO
	NS	NB	NM	NS	NS	ZO	PS	PS
	ZO	NM	NM	NS	ZO	PS	PM	PM
	PS	NM	NS	ZO	PS	PS	PM	PB
	PM	ZO	ZO	PS	PS	PM	PB	PB
	PB	ZO	ZO	PS	PM	PM	PB	PB

(3) k_d 的模糊控制规则表（见表 3-3）

表 3-3 k_d 的模糊控制规则表

Δk_d	ec							
		NB	NM	NS	ZO	PS	PM	PB
e	NB	PS	NS	NB	NB	NB	NM	PS
	NM	PS	NS	NB	NM	NM	NS	ZO
	NS	ZO	NS	NM	NM	NS	NS	ZO
	ZO	ZO	NS	NS	NS	NS	NS	ZO
	PS	ZO	ZO	ZO	ZO	ZO	ZO	ZO
	PM	PB	NS	PS	PS	PS	PS	PB
	PB	PB	PM	PM	PM	PS	PS	PB

k_p , k_i , k_d 的模糊控制规则表建立好后, 可根据如下方法进行 k_p , k_i , k_d 的自适应校正。

将系统误差 e 和误差变化率 ec 变化范围定义为模糊集上的论域。

$$e, ec = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\} \quad (3.6)$$

其模糊子集为 $e, ec = \{NB, NM, NS, O, PS, PM, PB\}$, 子集中元素分别代表负大, 负中, 负小, 零, 正小, 正中, 正大。设 e, ec 和 k_p , k_i , k_d 均服从正态分布, 因此可得出各模糊子集的隶属度, 根据各模糊子集的隶属度赋值表和各参数模糊控制模型, 应用模糊合成推理设计 PID 参数的模糊矩阵表, 查出修正参数代入下式计算

$$\begin{aligned} k_p &= k_p' + \{e, ec\}_p \\ k_i &= k_i' + \{e, ec\}_i \\ k_d &= k_d' + \{e, ec\}_d \end{aligned} \quad (3.7)$$

在线运行过程中, 控制系统通过对模糊逻辑规则的结果处理、查表和运算, 完成对 PID 参数的在线自校正。其工作流程图如图 3-5 所示。

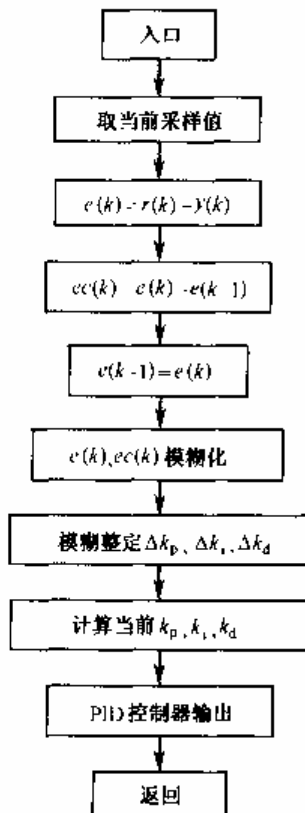


图 3-5 在线自校正工作流程图

3.2.2 仿真程序及分析

仿真实例

被控对象为

$$G_p(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms, 采用模糊 PID 控制进行阶跃响应, 在第 300 个采样时间时控制器输出加 1.0 的干扰, 相应的响应结果如图 3-6 至 3-11 所示。

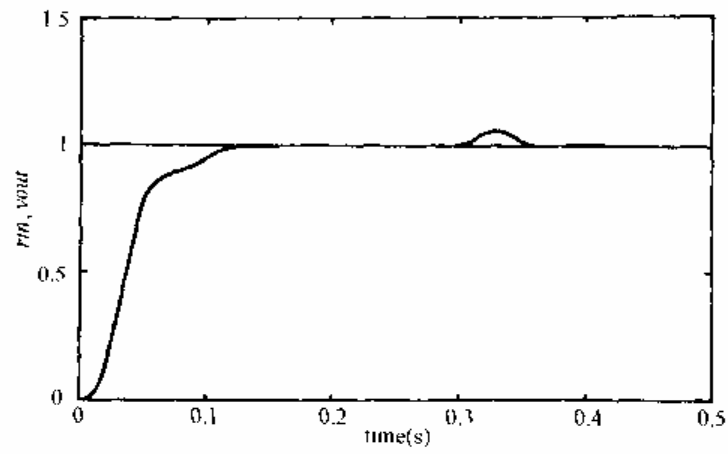


图 3-6 模糊 PID 控制阶跃响应

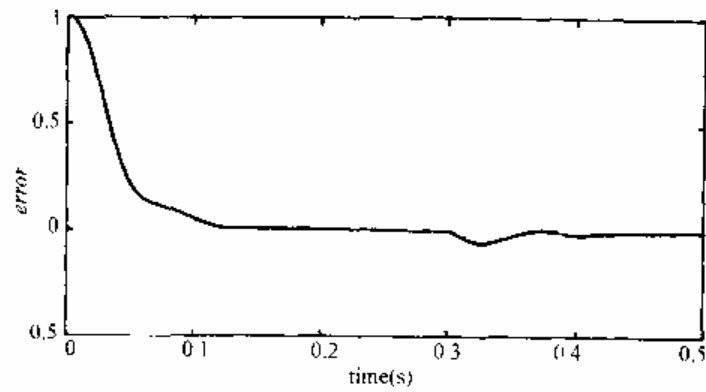


图 3-7 模糊 PID 控制误差响应

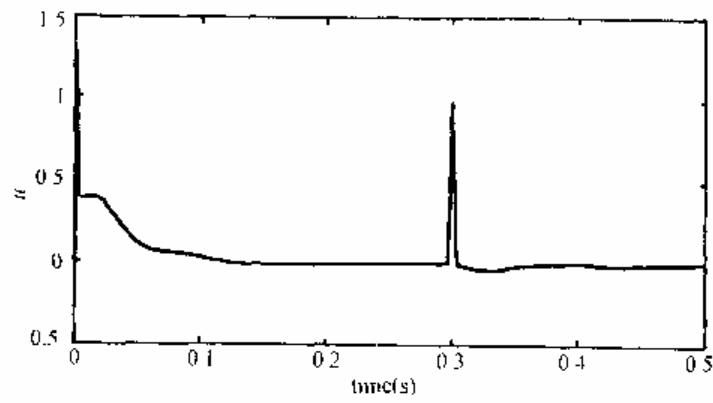


图 3-8 控制器输出 u

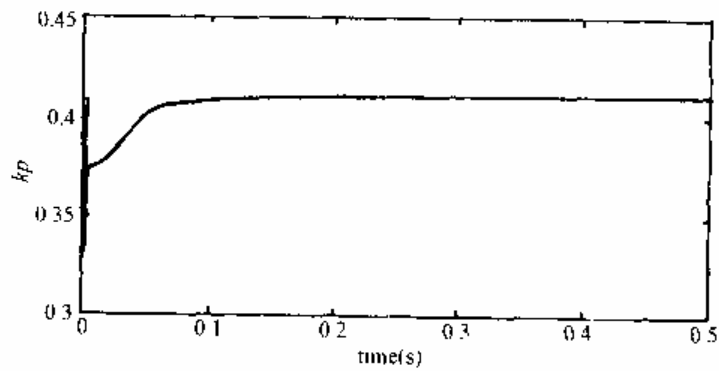


图 3-9 k_p 的自适应调整

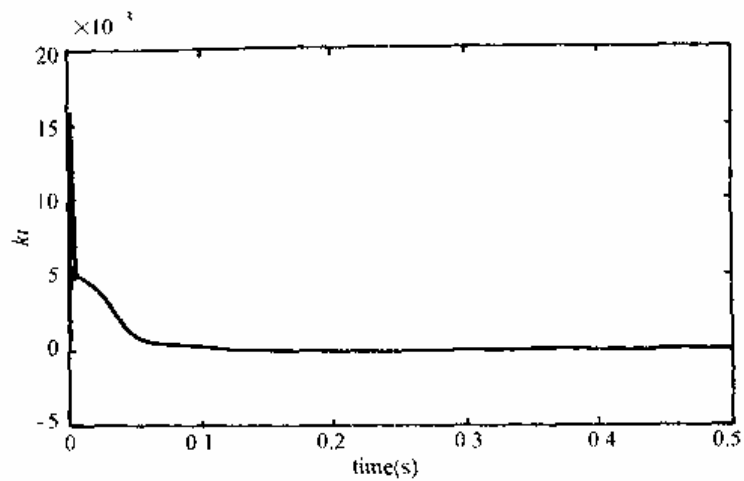


图 3-10 k_p 的自适应调整

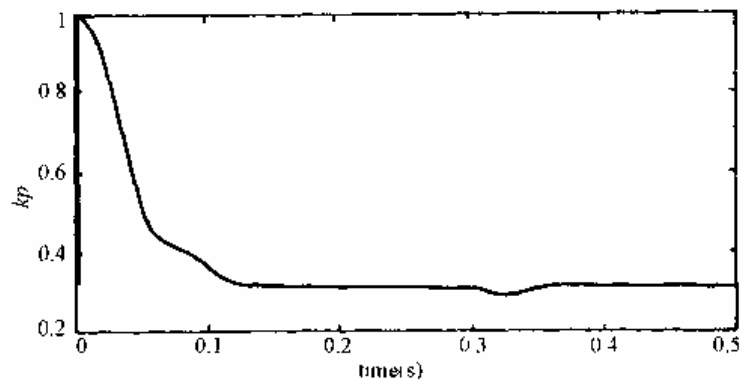


图 3-11 k_d 的自适应调整

仿真程序: chap3_2.m

```
%Fuzzy Tuning PID Control
```

```
clear all;
```

```
close all;
```

```
a=newfis('fuzzpid');
```

```
a=addvar(a,'input','c',[-3,3]);
```

```
%Parameter e
```

```
a=addmf(a,'input',1,'NB','zmf',[-3,-1]);
```

```
a=addmf(a,'input',1,'NM','trimf',[-3,-2,0]);
```

```
a=addmf(a,'input',1,'NS','trimf',[ -3,-1,1]);
```

```
a=addmf(a,'input',1,'Z','trimf',[-2,0,2]);
```

```
a=addmf(a,'input',1,'PS','trimf',[ 1,1,3]);
```

```
a=addmf(a,'input',1,'PM','trimf',[0,2,3]);
```

```
a=addmf(a,'input',1,'PB','smf',[1,3]);
```

```
a=addvar(a,'input','ec',[-3,3]);
```

```
%Parameter ec
```

```
a=addmf(a,'input',2,'NB','zmf',[-3,-1]);
```

```

a=addmf(a,'input',2,'NM','trimf',[-3,-2,0]);
a=addmf(a,'input',2,'NS','trimf',[-3,-1,1]);
a=addmf(a,'input',2,'Z','trimf',[ 2,0,2]);
a=addmf(a,'input',2,'PS','trimf',[-1,1,3]);
a=addmf(a,'input',2,'PM','trimf',[0,2,3]);
a=addmf(a,'input',2,'PB','smf',[1,3]);

a=addvar(a,'output','kp',[ 0.3,0.3]); %Parameter kp
a=addmf(a,'output',1,'NB','zmf',[-0.3,-0.1]);
a=addmf(a,'output',1,'NM','trimf',[-0.3,-0.2,0]);
a=addmf(a,'output',1,'NS','trimf',[-0.3, 0.1,0.1]);
a=addmf(a,'output',1,'Z','trimf',[-0.2,0,0.2]);
a=addmf(a,'output',1,'PS','trimf',[ 0.1,0.1,0.3]);
a=addmf(a,'output',1,'PM','trimf',[0,0.2,0.3]);
a=addmf(a,'output',1,'PB','smf',[0.1,0.3]);

a=addvar(a,'output','ki',[-0.06,0.06]); %Parameter ki
a=addmf(a,'output',2,'NB','zmf',[-0.06,-0.02]);
a=addmf(a,'output',2,'NM','trimf',[-0.06,-0.04,0]);
a=addmf(a,'output',2,'NS','trimf',[-0.06,-0.02,0.02]);
a=addmf(a,'output',2,'Z','trimf',[-0.04,0,0.04]);
a=addmf(a,'output',2,'PS','trimf',[-0.02,0.02,0.06]);
a=addmf(a,'output',2,'PM','trimf',[0,0.04,0.06]);
a=addmf(a,'output',2,'PB','smf',[0.02,0.06]);

a=addvar(a,'output','kd',[-3,3]); %Parameter kp
a=addmf(a,'output',3,'NB','zmf',[ 3, 1]);
a=addmf(a,'output',3,'NM','trimf',[ 3,-2,0]);
a=addmf(a,'output',3,'NS','trimf',[ 3,-1,1]);
a=addmf(a,'output',3,'Z','trimf',[-2,0,2]);
a=addmf(a,'output',3,'PS','trimf',[ 1,1,3]);
a=addmf(a,'output',3,'PM','trimf',[0,2,3]);
a=addmf(a,'output',3,'PB','smf',[1,3]);

rulelist=[1 1 7 1 5 1 1;
          1 2 7 1 3 1 1;
          1 3 6 2 1 1 1;
          1 4 6 2 1 1 1;
          1 5 5 3 1 1 1;
          1 6 4 4 2 1 1;

```

1 7 4 4 5 1 1;

2 1 7 1 5 1 1;

2 2 7 1 3 1 1;

2 3 6 2 1 1 1;

2 4 5 3 2 1 1;

2 5 5 3 2 1 1;

2 6 4 4 3 1 1;

2 7 3 4 4 1 1;

3 1 6 1 4 1 1;

3 2 6 2 3 1 1;

3 3 6 3 2 1 1;

3 4 5 3 2 1 1;

3 5 4 4 3 1 1;

3 6 3 5 3 1 1;

3 7 3 5 4 1 1;

4 1 6 2 4 1 1;

4 2 6 2 3 1 1;

4 3 5 3 3 1 1;

4 4 4 4 3 1 1;

4 5 3 5 3 1 1;

4 6 2 6 3 1 1;

4 7 2 6 4 1 1;

5 1 5 2 4 1 1;

5 2 5 3 4 1 1;

5 3 4 4 4 1 1;

5 4 3 5 4 1 1;

5 5 3 5 4 1 1;

5 6 2 6 4 1 1;

5 7 2 7 4 1 1;

6 1 5 4 7 1 1;

6 2 4 4 5 1 1;

6 3 3 5 5 1 1;

6 4 2 5 5 1 1;

6 5 2 6 5 1 1;

6 6 2 7 5 1 1;

```
6 7 1 7 7 1 1;
```

```
7 1 4 4 7 1 1;
```

```
7 2 4 4 6 1 1;
```

```
7 3 2 5 6 1 1;
```

```
7 4 2 6 6 1 1;
```

```
7 5 2 6 5 1 1;
```

```
7 6 1 7 5 1 1;
```

```
7 7 1 7 7 1 1];
```

```
a=addrule(a,rulelist);
```

```
a=setfis(a,'DefuzzMethod','mom');
```

```
writefis(a,'fuzzpid');
```

```
a=readfis('fuzzpid');
```

```
%PID Controller
```

```
ts=0.001;
```

```
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
```

```
dsys=c2d(sys,ts,'tustin');
```

```
[num,den]=tfdata(dsys,'v');
```

```
u_1=0.0;u_2=0.0;u_3=0.0;
```

```
y_1=0;y_2=0;y_3=0;
```

```
x=[0,0,0]';
```

```
error_1=0;
```

```
e_1=0.0;
```

```
ec_1=0.0;
```

```
kp0=0.40;
```

```
kd0=1.0;
```

```
ki0=0.0;
```

```
for k=1:1:500
```

```
time(k)=k*ts;
```

```
rin(k)=1;
```

```

%Using fuzzy inference to tuning PID
k_pid=evalfis([e_1,ec_1],a);
kp(k)=kp0+k_pid(1);
ki(k)=ki0+k_pid(2);
kd(k)=kd0+k_pid(3);
u(k)=kp(k)*x(1)+kd(k)*x(2)+ki(k)*x(3);

if k==300    % Adding disturbance(1.0v at time 0.3s)
    u(k)=u(k)+1.0;
end
if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(1)*u(k)+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Return of PID parameters%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u_3=u_2;
u_2=u_1;
u_1=u(k);

y_3=y_2;
y_2=y_1;
y_1=yout(k);

x(1)=error(k);           % Calculating P
x(2)=error(k)-error_1;   % Calculating D
x(3)=x(3)+error(k);     % Calculating I

e_1=x(1);
ec_1=x(2);

error_2=error_1;
error_1=error(k);
end
showrule(a)

```



```

figure(1);plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
figure(5);plot(time,ki,'r');
xlabel('time(s)');ylabel('ki');
figure(6);plot(time,kd,'r');
xlabel('time(s)');ylabel('kd');
figure(7);plotmf(a,'input',1);
figure(8);plotmf(a,'input',2);
figure(9);plotmf(a,'output',1);
figure(10);plotmf(a,'output',2);
figure(11);plotmf(a,'output',3);
plotfis(a);
fuzzy fuzzpid.fis

```

通过仿真，还可以得到以下几个结果：

(1) 在 MATLAB 下运行 `plotmf(a,'input', 1)` 可得到模糊系统第一个输入 e 的隶属函数，同理可得到 de ， k_p ， k_i ， k_d 的隶属函数，如图 3-12 至 3-16 所示。

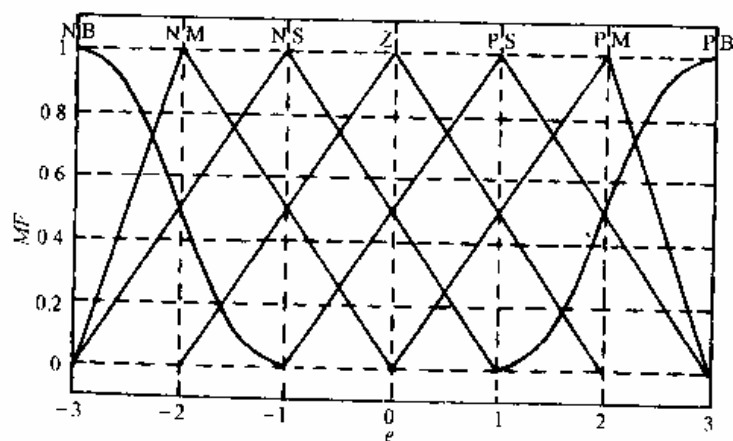


图 3-12 误差的隶属函数

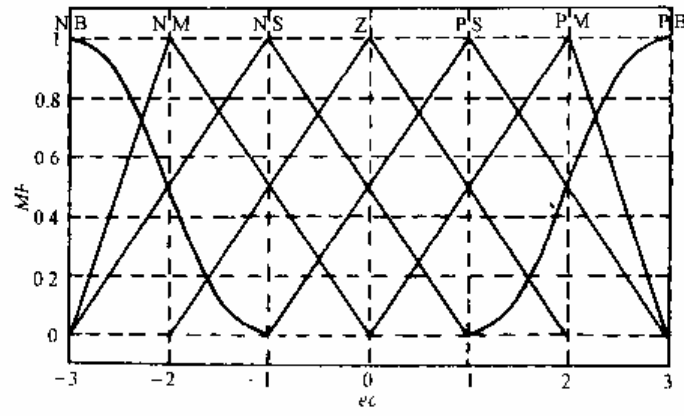


图 3-13 误差变化率的隶属函数

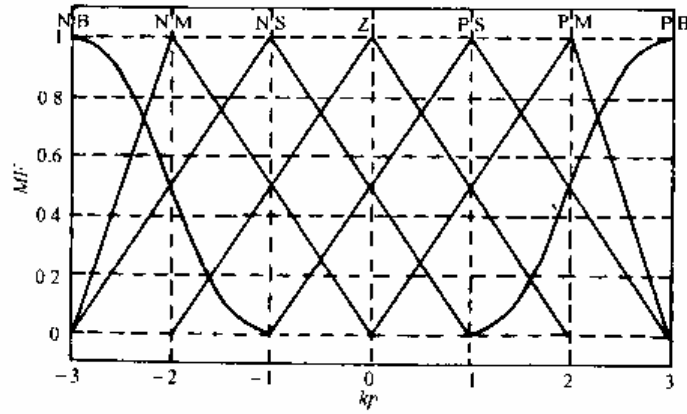


图 3-14 k_p 的隶属函数

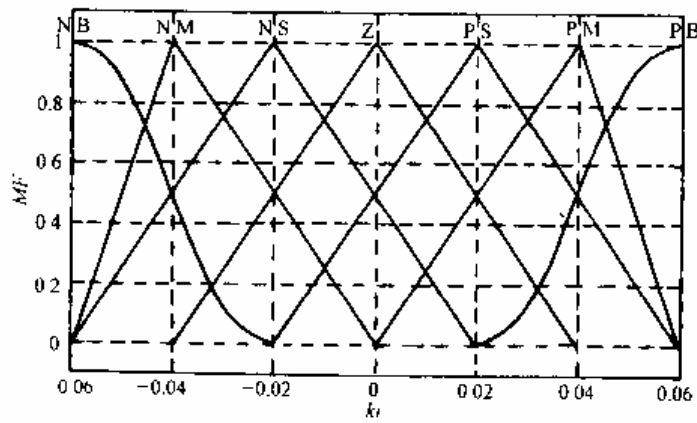


图 3-15 k_i 的隶属函数

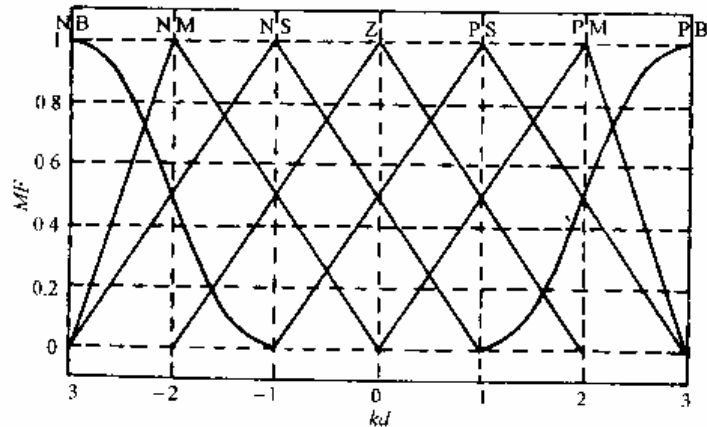


图 3-16 k_d 的隶属函数

(2) 在 MATLAB 下运行 `plotfis (a)` 可观察模糊控制系统的构成, 如图 3-17 所示。

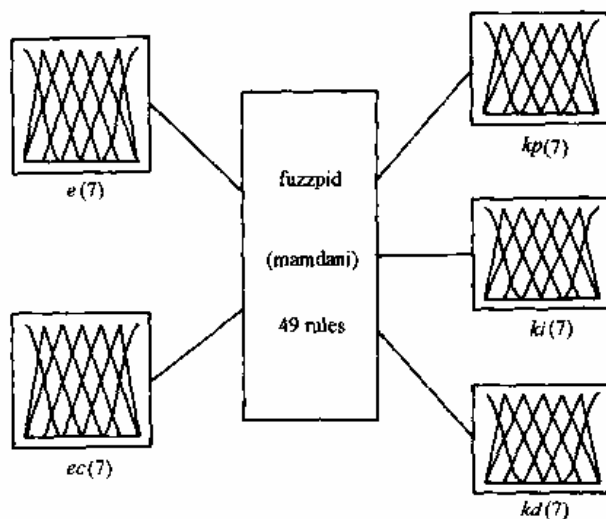


图 3-17 模糊 PID 控制系统构成

(3) 在 MATLAB 下运行 `fuzzy fuzzpid.fis` 可进入 MATLAB 动态仿真工具箱动态仿真环境, 如图 3-18 所示。

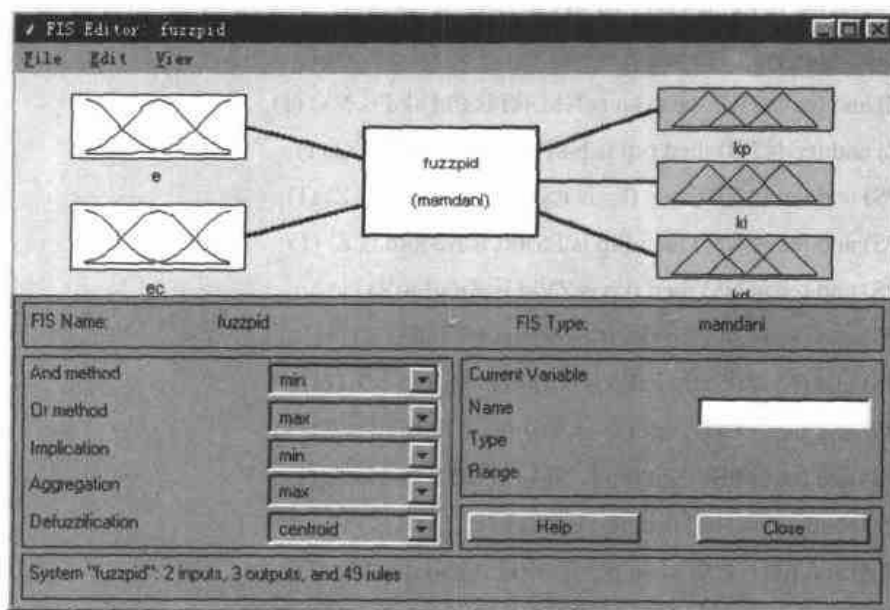


图 3-18 模糊 PID 动态仿真环境

(4) MATLAB 下运行 `showrule(a)`, 可得到以下 49 条模糊规则:

1. If (e is NB) and (ec is NB) then (kp is PB)(ki is NB)(kd is PS) (1)
2. If (e is NB) and (ec is NM) then (kp is PB)(ki is NB)(kd is NS) (1)
3. If (e is NB) and (ec is NS) then (kp is PM)(ki is NM)(kd is NB) (1)
4. If (e is NB) and (ec is Z) then (kp is PM)(ki is NM)(kd is NB) (1)
5. If (e is NB) and (ec is PS) then (kp is PS)(ki is NS)(kd is NB) (1)
6. If (e is NB) and (ec is PM) then (kp is Z)(ki is Z)(kd is NM) (1)
7. If (e is NB) and (ec is PB) then (kp is Z)(ki is Z)(kd is PS) (1)
8. If (e is NM) and (ec is NB) then (kp is PB)(ki is NB)(kd is PS) (1)

9. If (e is NM) and (ec is NM) then (kp is PB)(ki is NB)(kd is NS) (1)
10. If (e is NM) and (ec is NS) then (kp is PM)(ki is NM)(kd is NB) (1)
11. If (e is NM) and (ec is Z) then (kp is PS)(ki is NS)(kd is NM) (1)
12. If (e is NM) and (ec is PS) then (kp is PS)(ki is NS)(kd is NM) (1)
13. If (e is NM) and (ec is PM) then (kp is Z)(ki is Z)(kd is NS) (1)
14. If (e is NM) and (ec is PB) then (kp is NS)(ki is Z)(kd is Z) (1)
15. If (e is NS) and (ec is NB) then (kp is PM)(ki is NB)(kd is Z) (1)
16. If (e is NS) and (ec is NM) then (kp is PM)(ki is NM)(kd is NS) (1)
17. If (e is NS) and (ec is NS) then (kp is PM)(ki is NS)(kd is NM) (1)
18. If (e is NS) and (ec is Z) then (kp is PS)(ki is NS)(kd is NM) (1)
19. If (e is NS) and (ec is PS) then (kp is Z)(ki is Z)(kd is NS) (1)
20. If (e is NS) and (ec is PM) then (kp is NS)(ki is PS)(kd is NS) (1)
21. If (e is NS) and (ec is PB) then (kp is NS)(ki is PS)(kd is Z) (1)
22. If (e is Z) and (ec is NB) then (kp is PM)(ki is NM)(kd is Z) (1)
23. If (e is Z) and (ec is NM) then (kp is PM)(ki is NM)(kd is NS) (1)
24. If (e is Z) and (ec is NS) then (kp is PS)(ki is NS)(kd is NS) (1)
25. If (e is Z) and (ec is Z) then (kp is Z)(ki is Z)(kd is NS) (1)
26. If (e is Z) and (ec is PS) then (kp is NS)(ki is PS)(kd is NS) (1)
27. If (e is Z) and (ec is PM) then (kp is NM)(ki is PM)(kd is NS) (1)
28. If (e is Z) and (ec is PB) then (kp is NM)(ki is PM)(kd is Z) (1)
29. If (e is PS) and (ec is NB) then (kp is PS)(ki is NM)(kd is Z) (1)
30. If (e is PS) and (ec is NM) then (kp is PS)(ki is NS)(kd is Z) (1)
31. If (e is PS) and (ec is NS) then (kp is Z)(ki is Z)(kd is Z) (1)
32. If (e is PS) and (ec is Z) then (kp is NS)(ki is PS)(kd is Z) (1)
33. If (e is PS) and (ec is PS) then (kp is NS)(ki is PS)(kd is Z) (1)
34. If (e is PS) and (ec is PM) then (kp is NM)(ki is PM)(kd is Z) (1)
35. If (e is PS) and (ec is PB) then (kp is NM)(ki is PB)(kd is Z) (1)
36. If (e is PM) and (ec is NB) then (kp is PS)(ki is Z)(kd is PB) (1)
37. If (e is PM) and (ec is NM) then (kp is Z)(ki is Z)(kd is PS) (1)
38. If (e is PM) and (ec is NS) then (kp is NS)(ki is PS)(kd is PS) (1)
39. If (e is PM) and (ec is Z) then (kp is NM)(ki is PS)(kd is PS) (1)
40. If (e is PM) and (ec is PS) then (kp is NM)(ki is PM)(kd is PS) (1)
41. If (e is PM) and (ec is PM) then (kp is NM)(ki is PB)(kd is PS) (1)
42. If (e is PM) and (ec is PB) then (kp is NB)(ki is PB)(kd is PB) (1)
43. If (e is PB) and (ec is NB) then (kp is Z)(ki is Z)(kd is PB) (1)
44. If (e is PB) and (ec is NM) then (kp is Z)(ki is Z)(kd is PM) (1)
45. If (e is PB) and (ec is NS) then (kp is NM)(ki is PS)(kd is PM) (1)
46. If (e is PB) and (ec is Z) then (kp is NM)(ki is PM)(kd is PM) (1)
47. If (e is PB) and (ec is PS) then (kp is NM)(ki is PM)(kd is PS) (1)
48. If (e is PB) and (ec is PM) then (kp is NB)(ki is PB)(kd is PS) (1)

49. If (e is PB) and (ec is PB) then (kp is NB)(ki is PB)(kd is PB) (1)

3.3 模糊免疫 PID 控制算法

3.3.1 模糊免疫 PID 控制算法原理

常规增量式 PID 控制器离散形式如下

$$\begin{aligned} u(k) &= u(k-1) + k_p(e(k) - e(k-1)) + k_i e(k) + k_d(e(k) - 2e(k-1) + e(k-2)) \\ &= u(k-1) + k_p((e(k) - e(k-1)) + \frac{k_i}{k_p} e(k) + \frac{k_d}{k_p}(e(k) - 2e(k-1) + e(k-2))) \end{aligned} \quad (3.8)$$

式中, k_p, k_i, k_d 分别为比例、积分和微分系数

P 控制器的控制算法为

$$u(k) = k_p e(k) \quad (3.9)$$

免疫 PID 控制器是借鉴生物系统的免疫机理而设计出的一种非线性控制器。免疫是生物体的一种特性生理反应。生物的免疫系统对于外来侵犯的抗原,可产生相应的抗体来抵御。抗原和抗体结合后,会产生一系列的反应,通过吞噬作用或产生特殊酶的作用而毁坏抗原。生物的免疫系统由淋巴细胞和抗体分子组成,淋巴细胞又由胸腺产生的 T 细胞(分别为辅助细胞 T_H 和抑制细胞 T_S)和骨髓产生的 B 细胞组成。当抗原侵入机体并经周围细胞消化后,将信息传递给 T 细胞,即传递给 T_H 细胞和 T_S 细胞,然后刺激 B 细胞。B 细胞产生抗体以消除抗原。当抗原较多时,机体内的 T_H 细胞也较多,而 T_S 细胞却较少,从而会产生较多的 B 细胞。随着抗原的减少,体内 T_S 细胞增多,它抑制了 T_H 细胞的产生,则 B 细胞也随着减少。经过一段时间间隔后,免疫反馈系统便趋于平衡。抑制机理和主反馈机理之间的相互协作,是通过免疫反馈机理对抗原的快速反应和稳定免疫系统完成的。

免疫系统虽然十分复杂,但其抗御抗原的自适应能力却是十分明显的。生物信息系统的这些智能行为,为科学和工程领域提供了各种理论参考和技术方法。基于上述免疫反馈原理,提出了免疫 PID 控制器:假设第 k 代的抗原数量为 $\varepsilon(k)$,由抗原刺激的 T_H 细胞的输出为 $T_H(k)$, T_S 细胞对 B 细胞的影响为 $T_S(k)$,则 B 细胞接收的总刺激为

$$S(k) = T_H(k) - T_S(k) \quad (3.10)$$

式中, $T_H(k) = k_1 \varepsilon(k), T_S(k) = k_2 f(\Delta S(k)) \varepsilon(k)$

若以抗原的数量 $\varepsilon(k)$ 作为偏差 $e(k)$, B 细胞接收的总刺激 $S(k)$ 作为控制输入 $u(k)$,则有如下的反馈控制规律

$$u(k) = K(1 - \eta f(u(k), \Delta u(k))) e(k) = k_{pi} e(k) \quad (3.11)$$

式中, $k_{pi} = K(1 - \eta f(u(k), \Delta u(k)))$, $K = k_1$ 为控制反应速度, $\eta = \frac{k_2}{k_1}$ 为控制稳定效果, $f(\cdot)$ 为一选定的非线性函数。

利用模糊控制器可逼近非线性函数 $f(\cdot)$:每个输入变量被二个模糊集模糊化,分别是“正”(P)和“负”(N);输出变量被三个模糊集模糊化,分别是“正”(P)、零“Z”和负(N)。以上隶属度函数都定义在整个 $(+\infty, -\infty)$ 区间。模糊控制器可采用以下四条规则:

- (1) If u is P and Δu is P then $f(u, \Delta u)$ is N (1)
- (2) If u is P and Δu is N then $f(u, \Delta u)$ is Z (1)

(3) If u is N and Δu is P then $f(u, \Delta u)$ is Z (1)

(4) If u is N and Δu is N then $f(u, \Delta u)$ is P (1)

各规则中, 使用 Zadeh 的模糊逻辑 AND 操作, 并采用常用的 mom 反模糊化方法得到模糊控制器的输出 $f(\cdot)$ 。

基于免疫反馈原理的控制器实际上就是一个非线性 P 控制器, 其比例系数 $k_{p1} = K(1 - \eta f(u(k), \Delta u(k)))$ 随控制器输出的变化而变化, 其中 K 为增益, 则免疫 PID 控制器的输出为

$$\begin{aligned} u(k) &= u(k-1) + k_{p1}((e(k) - e(k-1)) + \frac{k_i}{k_p} e(k) + \frac{k_d}{k_p} (e(k) - 2e(k-1) + e(k-2))) \\ &= u(k-1) + k_{p1}((e(k) - e(k-1)) + k_i' e(k) + k_d' (e(k) - 2e(k-1) + e(k-2))) \end{aligned} \quad (3.12)$$

3.3.2 仿真程序及分析

仿真实例

设被控对象为一延迟系统

$$G_p(s) = \frac{1}{60s+1} e^{-80s}$$

采样时间为 20s, 采用免疫 PID 控制器式 (3.12), 取 $K = 0.30$, $\eta = 0.80$, $k_i' = 0.30, k_d' = 0.30$ 。输入的指令信号为 $1.0 \operatorname{sgn}(\sin(3\pi t))$, 仿真时间为 1000 个采样点。为了测试控制器的鲁棒性, 在第 500 个采样时间时加入一个干扰, 免疫 PID 控制方波跟踪结果如图 3-19 和图 3-20 所示。仿真结果表明, 免疫 PID 控制具有很好的控制效果和较高的鲁棒性。非线性函数 $f(\cdot)$ 为模糊控制的输出, 模糊控制输入、输出隶属函数如图 3-21 至图 3-23 所示。

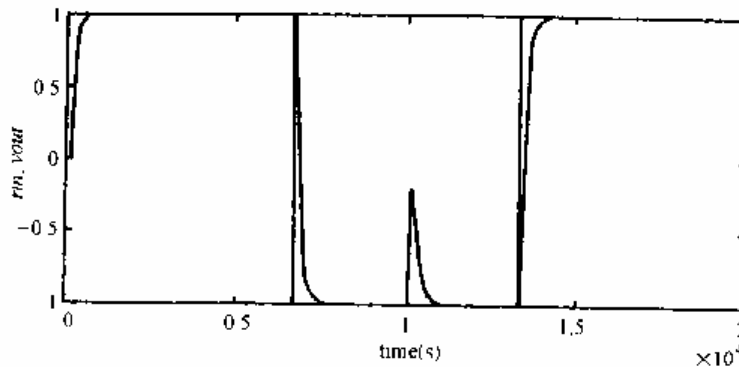


图 3-19 免疫 PID 控制方波跟踪

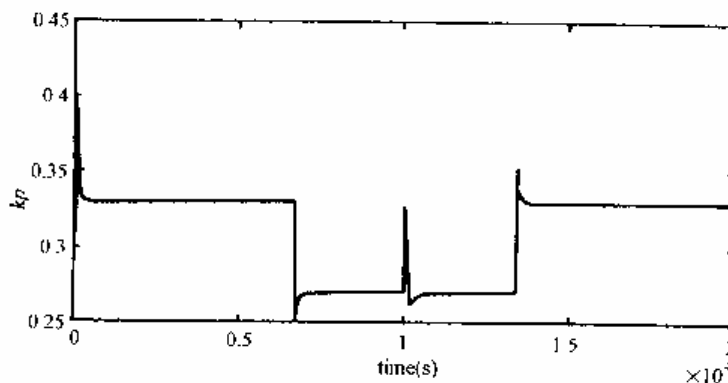


图 3-20 免疫 PID 控制 k_{p1} 的变化

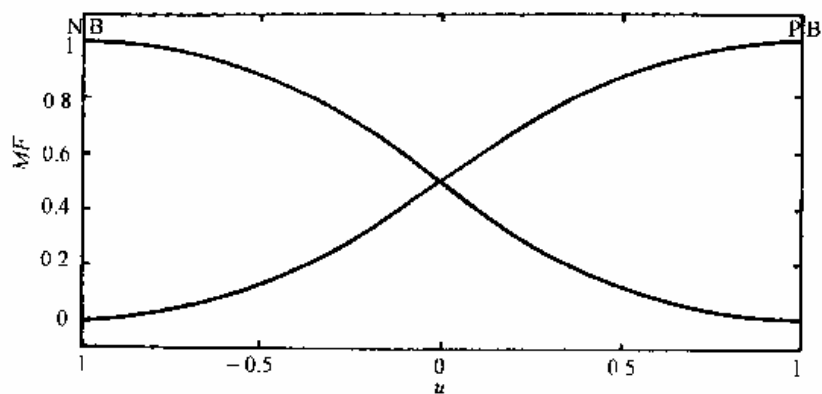


图 3-21 免疫 PID 控制 u 隶属函数

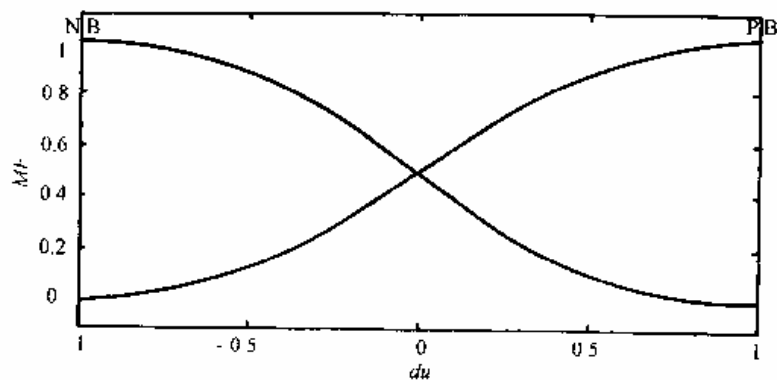


图 3-22 免疫 PID 控制 du 隶属函数

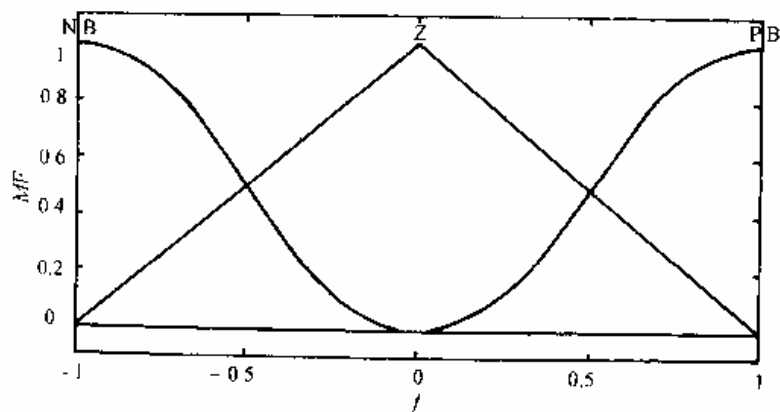


图 3-23 免疫 PID 控制 $f(\cdot)$ 隶属函数

仿真程序: chap3_3.m

```
%Fuzzy Immune PID Control
```

```
clear all;
```

```
close all;
```

```
a=newfis('fuzz_ljk');
```

```
f1=1.0;
```

```
a=addvar(a,'input','u',[-f1*1,f1*1]);
```

```
%Parameter e
```

```

a=addmf(a,'input',1,'NB','zmf',[-f1*1,f1*1]);
a=addmf(a,'input',1,'PB','smf',[-f1*1,f1*1]);

f2=1.0;
a=addvar(a,'input','du',[-f2*1,f2*1]);           %Parameter ec
a=addmf(a,'input',2,'NB','zmf',[-f2*1,f2*1]);
a=addmf(a,'input',2,'PB','smf',[-f2*1,f2*1]);

f3=1.0;
a=addvar(a,'output','f',[-f3*1,f3*1]);           %Parameter u
a=addmf(a,'output',1,'NB','zmf',[-f3*1,0]);
a=addmf(a,'output',1,'Z','trimf',[-f3*1,0,f3*1]);
a=addmf(a,'output',1,'PB','smf',[0,f3*1]);

rulelist=[2 2 1 1 1;    % Edit rule base
          2 1 2 1 1;
          1 2 2 1 1;
          1 1 3 1 1;];

a=addrule(a,rulelist);
%showrule(a)           % Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','mom'); % Defuzzy
writefis(a1,'ljk');      % save to fuzzy file "ljk.fis" which can be
                        % simulated with fuzzy tool
a2=readfis('ljk');
%plotfis(a2);

%%%%%%%%%% Using Fuzzy Controller%%%%%%%%%%
ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;
e_1=0;e_2=0;
for k=1:1:1000
time(k)=k*ts;

```



```

rin(k)=1.0*sign(sin(3*pi*k*0.001));

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

e(k)=rin(k)-yout(k);
ec(k)=e(k)-e_1;

f(k)=evalfis([u_1 u_1-u_2],a2);

K=0.30;
xite=0.80;
Kp(k)=K*(1-xite*f(k));

u(k)=u_1+Kp(k)*((e(k)-e_1)+0.3*e(k)+0.3*(e(k)-2*e_1+e_2));

if k==500
    u(k)=u(k)+1.0;
end

%Return of parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);
e_2=e_1;
e_1=e(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,e,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);
plot(time,Kp,'r');
xlabel('time(s)');ylabel('Kp');
figure(5);
plotmf(a,'input',1);

```

```
figure(6);  
plotmf(a, 'input', 2);  
figure(7);  
plotmf(a, 'output', 1);
```

第4章 神经PID控制

4.1 基于单神经网络的PID智能控制

由具有自学习和自适应能力的单神经元构成单神经元自适应智能PID控制器，不但结构简单，而且能适应环境变化，有较强的鲁棒性。

4.1.1 几种典型的学习规则

(1) 无监督 Hebb 学习规则

Hebb 学习是一类相关学习，其基本思想是：如果两个神经元同时被激活，则它们之间的联接强度的增强与它们激励的乘积成正比，以 o_i 表示神经元 i 的激活值， o_j 表示神经元 j 的激活值， w_{ij} 表示神经元 i 和神经元 j 的联接权值，则 Hebb 学习规则可表示

$$\Delta w_{ij}(k) = \eta o_j(k) o_i(k) \quad (4.1)$$

式中， η 为学习速率。

(2) 有监督的 Delta 学习规则

在 Hebb 学习规则中，引入教师信号，即将 o_j 换成希望输出 d_j 与实际输出 o_j 之差，就构成有监督学习的 Delta 学习规则

$$\Delta w_{ij}(k) = \eta (d_j(k) - o_j(k)) o_i(k) \quad (4.2)$$

(3) 有监督的 Hebb 学习规则

将无监督的 Hebb 学习规则和有监督的 Delta 学习两者结合起来就构成有监督的 Hebb 学习规则

$$\Delta w_{ij}(k) = \eta (d_j(k) - o_j(k)) o_i(k) o_j(k) \quad (4.3)$$

4.1.2 单神经元自适应PID控制

单神经元自适应PID控制的结构如图4-1所示。

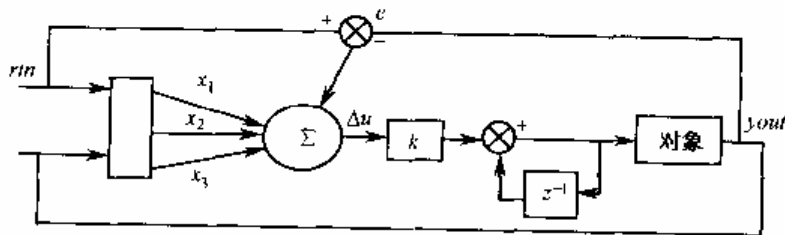


图4-1 单神经元自适应PID控制结构

单神经元自适应控制器是通过调整加权系数来实现自适应、自组织功能，权系数的调整是按有监督的 Hebb 学习规则实现的。控制算法及学习算法为

$$u(k) = u(k-1) + K \sum_{i=1}^3 w_i'(k) x_i(k) \quad (4.4)$$

$$w_i'(k) = w_i(k) / \sum_{i=1}^3 |w_i(k)| \quad (4.5)$$

$$\begin{aligned} w_1(k) &= w_1(k-1) + \eta_I z(k) u(k) x_1(k) \\ w_2(k) &= w_2(k-1) + \eta_P z(k) u(k) x_2(k) \\ w_3(k) &= w_3(k-1) + \eta_D z(k) u(k) x_3(k) \end{aligned} \quad (4.6)$$

式中, $x_1(k) = e(k)$;

$$x_2(k) = e(k) - e(k-1);$$

$$x_3(k) = \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2);$$

η_I, η_P, η_D 分别为积分、比例、微分的学习速率, K 为神经元的比例系数, $K > 0$ 。

对积分 I、比例 P 和微分 D 分别采用了不同的学习速率 η_I, η_P, η_D , 以便对不同的权系数分别进行调整。

K 值的选择非常重要。 K 越大, 则快速性越好, 但超调量大, 甚至可能使系统不稳定。当被控对象时延增大时, K 值必须减少, 以保证系统稳定。 K 值选择过小, 会使系统的快速性变差。

4.1.3 改进的单神经元自适应 PID 控制

在大量的实际应用中, 通过实践表明, PID 参数的在线学习修正主要与 $e(k)$ 和 $\Delta e(k)$ 有关。基于此可将单神经元自适应 PID 控制算法中的加权系数学习修正部分进行修改, 即将其中的 $x_i(k)$ 改为 $e(k) + \Delta e(k)$, 改进后的算法如下

$$u(k) = u(k-1) + K \sum_{i=1}^3 w_i(k) x_i(k) \quad (4.7)$$

$$w_i(k) = w_j(k) / \sum_{j=1}^3 |w_j(k)|$$

$$w_1(k) = w_1(k-1) + \eta_I z(k) u(k) (e(k) + \Delta e(k))$$

$$w_2(k) = w_2(k-1) + \eta_P z(k) u(k) (e(k) + \Delta e(k))$$

$$w_3(k) = w_3(k-1) + \eta_D z(k) u(k) (e(k) + \Delta e(k))$$

式中, $\Delta e(k) = e(k) - e(k-1)$, $z(k) = e(k)$

采用上述改进算法后, 权系数的在线修正就不完全是根据神经网络学习原理, 而是参考实际经验制定的。

4.1.4 仿真程序及分析

仿真实例

被控对象为

$$y(k) = 0.368y(k-1) + 0.26y(k-2) + 0.10u(k-1) + 0.632u(k-2)$$

输入指令为一方波信号: $rin(k) = 0.5 \text{sgn}(\sin(4\pi t))$, 采样时间为 1ms, 分别采用四种控制律进行单神经元 PID 控制, 即无监督的 Hebb 学习规则; 有监督的 Delta 学习规则; 有监

督的 Hebb 学习规则；改进的 Hebb 学习规则，跟踪结果如图 4-2 至 4-5 所示。

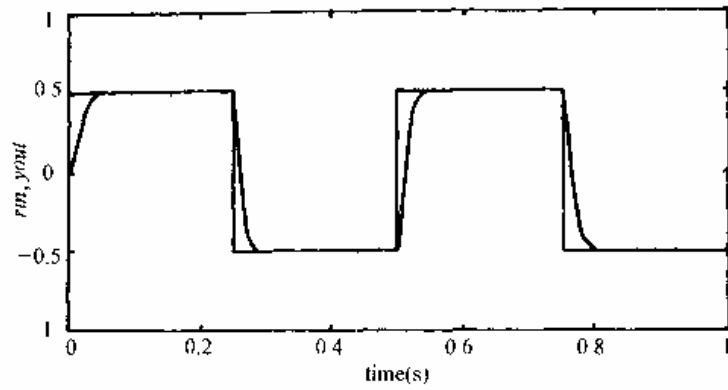


图 4-2 基于无监督 Hebb 学习规则的位置跟踪 ($M = 1$)

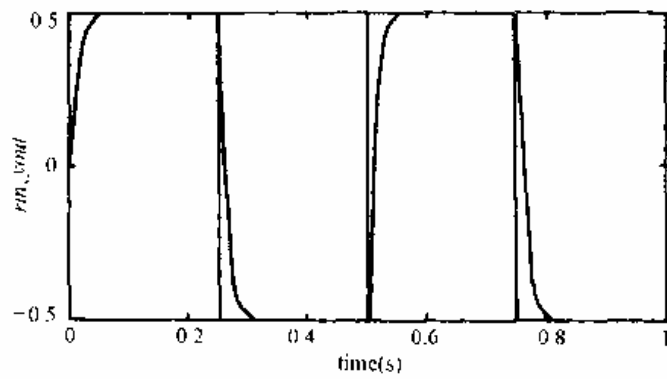


图 4-3 基于有监督的 Delta 学习规则的位置跟踪 ($M = 2$)

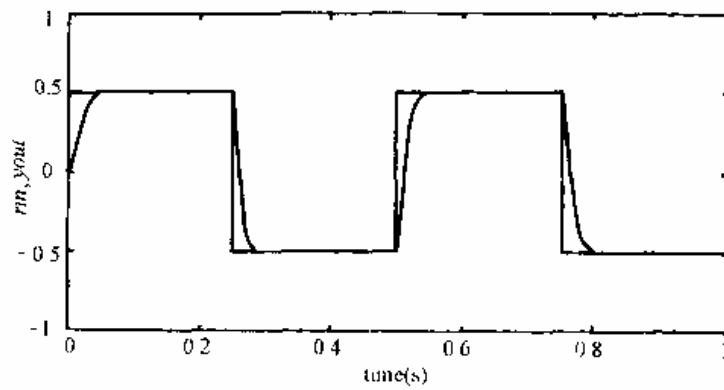


图 4-4 基于有监督 Hebb 学习规则的位置跟踪 ($M = 3$)

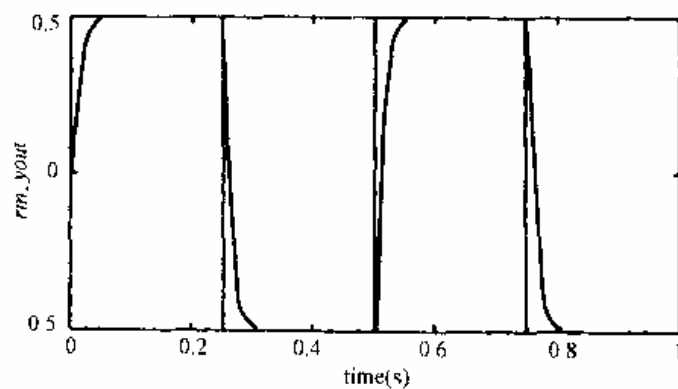


图 4-5 基于改进学习规则的位置跟踪 ($M = 4$)

```

仿真程序: chap4_1.m
%Single Neural Adaptive PID Controller
clear all;
close all;

x=[0,0,0]';

xiteP=0.40;
xiteI=0.35;
xiteD=0.40;

%Initilizing kp,ki and kd
wkp_1=0.10;
wki_1=0.10;
wkd_1=0.10;
%wkp_1=rand;
%wki_1=rand;
%wkd_1=rand;

error_1=0;
error_2=0;
y_1=0;y_2=0;y_3=0;
u_1=0;u_2=0;u_3=0;

ts=0.001;
for k=1:1:1000
    time(k)=k*ts;
    rin(k)=0.5*sign(sin(2*2*pi*k*ts));
    yout(k)=0.368*y_1+0.26*y_2+0.1*u_1-0.632*u_2;
    error(k)=rin(k)-yout(k);

%Adjusting Weight Value by hebb learning algorithm
M=4;
if M==1 %No Supervised Heb learning algorithm
    wkp(k)=wkp_1+xiteP*u_1*x(1); %P
    wki(k)=wki_1+xiteI*u_1*x(2); %I
    wkd(k)=wkd_1+xiteD*u_1*x(3); %D
    K=0.06;
elseif M==2 %Supervised Delta learning algorithm
    wkp(k)=wkp_1+xiteP*error(k)*u_1; %P

```

```

wki(k)=wki_1+xiteI*error(k)*u_1; %I
wkd(k)=wkd_1+xiteD*error(k)*u_1; %D
K=0.12;
elseif M==3 %Supervised Heb learning algorithm
wkp(k)=wkp_1+xiteP*error(k)*u_1*x(1); %P
wki(k)=wki_1+xiteI*error(k)*u_1*x(2); %I
wkd(k)=wkd_1+xiteD*error(k)*u_1*x(3); %D
K=0.12;
elseif M==4 %Improved Heb learning algorithm
wkp(k)=wkp_1+xiteP*error(k)*u_1*(2*error(k)-error_1);
wki(k)=wki_1+xiteI*error(k)*u_1*(2*error(k)-error_1);
wkd(k)=wkd_1+xiteD*error(k)*u_1*(2*error(k)-error_1);
K=0.12;
end

x(1)=error(k)-error_1; %P
x(2)=error(k); %I
x(3)=error(k)-2*error_1+error_2; %D

wadd(k)=abs(wkp(k))+abs(wki(k))+abs(wkd(k));
w11(k)=wkp(k)/wadd(k);
w22(k)=wki(k)/wadd(k);
w33(k)=wkd(k)/wadd(k);
w=[w11(k),w22(k),w33(k)];

u(k)=u_1+K*w*x; %Control law

if u(k)>10
u(k)=10;
end
if u(k)<-10
u(k)=-10;
end

error_2=error_1;
error_1=error(k);

u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

```

```

wkp_1=wkp(k);
wkd_1=wkd(k);
wki_1=wki(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');

```

4.1.5 基于二次型性能指标学习算法的单神经元自适应 PID 控制

在最优控制理论中，采用二次型性能指标来计算控制律可以得到所期望的优化效果。在神经元学习算法中，也可借助最优控制中二次型性能指标的思想，在加权系数的调整中引入二次型性能指标，使输出误差和控制增量加权平方和为最小来调整加权系数，从而间接实现对输出误差和控制增量加权的约束控制。

设性能指标为

$$E(k) = \frac{1}{2} (P(rin(k) - yout(k))^2 + Q\Delta^2 u(k)) \quad (4.8)$$

式中， P, Q 分别为输出误差和控制增量的加权系数， $r(k)$ 和 $y(k)$ 为 k 时刻的参考输入和输出。

神经元的输出为

$$u(k) = u(k-1) + K \sum_{i=1}^3 w'_i(k) x_i(k) \quad (4.9)$$

$$w'_i(k) = w_i(k) / \sum_{i=1}^3 |w_i(k)| \quad (i=1,2,3)$$

$$w_1(k) = w_1(k-1) + \eta_1 K (Pb_0 z(k) x_1(k) - QK \sum_{i=1}^3 (w_i(k) x_i(k)) x_1(k)) \quad (4.10)$$

$$w_2(k) = w_2(k-1) + \eta_2 K (Pb_0 z(k) x_2(k) - QK \sum_{i=1}^3 (w_i(k) x_i(k)) x_2(k))$$

$$w_3(k) = w_3(k-1) + \eta_3 K (Pb_0 z(k) x_3(k) - QK \sum_{i=1}^3 (w_i(k) x_i(k)) x_3(k))$$

式中，

$$\begin{aligned}
 x_1(k) &= e(k) \\
 x_2(k) &= e(k) - e(k-1) \\
 x_3(k) &= \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2) \\
 z(k) &= e(k)
 \end{aligned}
 \tag{4.11}$$

4.1.6 仿真程序及分析

仿真实例

设被控对象过程模型为

$$yout(k) = 0.368yout(k-1) + 0.264yout(k-2) + u(k-d) + 0.632u(k-d-1) + \xi(k)$$

应用最优二次型性能指标学习算法进行仿真研究。 $\xi(k)$ 为在 100 个采样时间的外加干扰, $\xi(100) = 0.10$, 输入为阶跃响应信号 $rin(k) = 1.0$ 。启动时采用开环控制, 取 $u = 0.1726, K = 0.02, P = 2, Q = 1, d = 6$, 比例、积分、微分三部分加权系数学习速率分别取 $\eta_p = 4, \eta_v = 120, \eta_d = 159, w_1(0) = 0.34, w_2(0) = 0.32, w_3(0) = 0.33$, 神经元自适应 PID 跟踪及中权值变化结果如图 4-6 和图 4-7 所示。

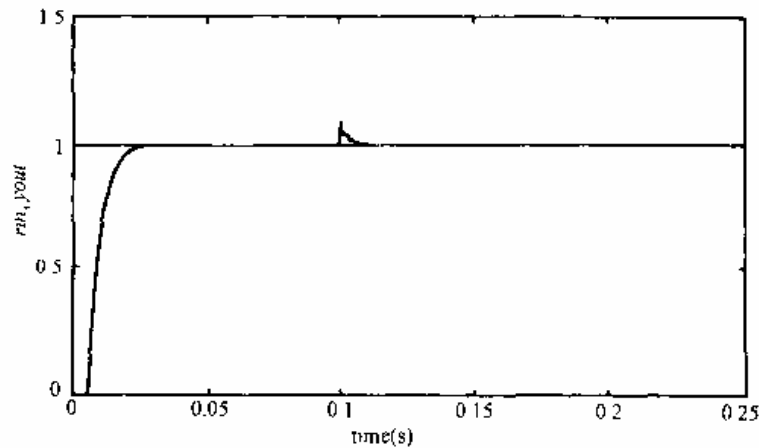


图 4-6 二次型性能指标学习单神经元自适应 PID 位置跟踪

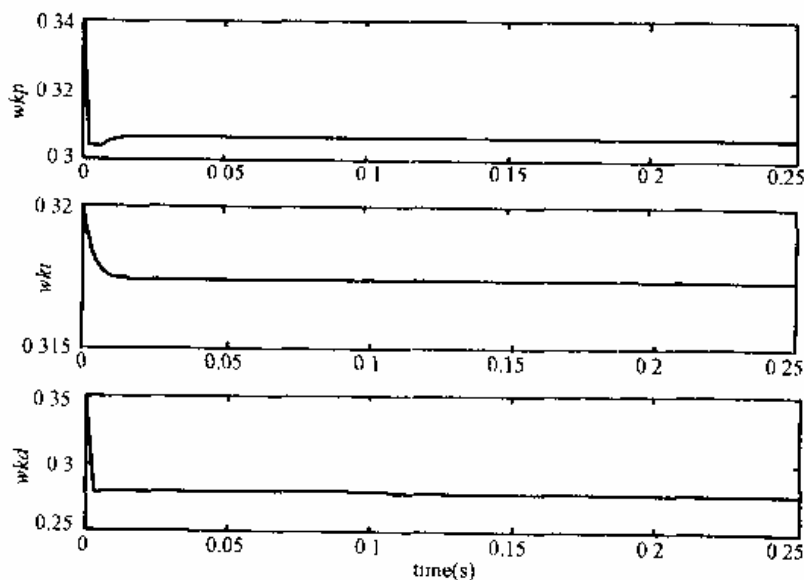


图 4-7 单神经元 PID 控制过程中权值变化

```

仿真程序: chap4_2.m
%Single Neural Net PID Controller based on Second Type Learning Algorithm
clear all;
close all;

xc = [0,0,0]';

K=0.02;P=2;Q=1;d=6;

xiteP=120;
xiteI=4;
xiteD=159;

%(nitializing kp,ki and kd
wkp_1=rand;
wki_1=rand;
wkd_1=rand;

wkp_1=0.34;
wki_1=0.32;
wkd_1=0.33;

error_1=0;error_2=0;
y_1=0;y_2=0;
u_1=0.1726;u_2=0;u_3=0;u_4=0;u_5=0;u_6=0;u_7=0;

ts=0.001;
for k=1:1:250
    time(k)=k*ts;
    rin(k)=1.0;           %Tracing Step Signal

ym(k)=0;
if k==100
    ym(k)=0.10; %Disturbance
end
yout(k)=0.368*y_1+0.26*y_2+u_6+0.632*u_7+ym(k);
error(k)=rin(k)-yout(k);

wx=[wkp_1,wkd_1,wki_1];
wx=wx*xc;

```

```

b0=yout(1);
K=0.0175;
wkp(k)=wkp_1+xiteP*K*[P*b0*error(k)*xc(1)-Q*K*wx*xc(1)];
wki(k)=wki_1+xiteI*K*[P*b0*error(k)*xc(2)-Q*K*wx*xc(2)];
wkd(k)=wkd_1+xiteD*K*[P*b0*error(k)*xc(3)-Q*K*wx*xc(3)];

xc(1)=error(k)-error_1;           %P
xc(2)=error(k);                   %I
xc(3)=error(k)-2*error_1+error_2; %D

wadd(k)=abs(wkp(k))+abs(wki(k))+abs(wkd(k));
w11(k)=wkp(k)/wadd(k);
w22(k)=wki(k)/wadd(k);
w33(k)=wkd(k)/wadd(k);
w=[w11(k),w22(k),w33(k)];

u(k)=u_1+K*w*xc;    % Control law

if u(k)>10
    u(k)=10;
end
if u(k)<-10
    u(k)=-10;
end

error_2=error_1;
error_1=error(k);

u_7=u_6;u_6=u_5;u_5=u_4;u_4=u_3;
u_3=u_2;u_2=u_1;u_1=u(k);

wkp_1=wkp(k);
wkd_1=wkd(k);
wki_1=wki(k);

y_2=y_1;y_1=yout(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');

```

```

xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('Time(s)');ylabel('u');
figure(3);
subplot(311);
plot(time,wkp,'r');
xlabel('time(s)');ylabel('wkp');
subplot(312);
plot(time,wki,'r');
xlabel('time(s)');ylabel('wki');
subplot(313);
plot(time,wkd,'r');
xlabel('time(s)');ylabel('wkd');

```

4.2 基于 BP 神经网络整定的 PID 控制

4.2.1 基于 BP 神经网络的 PID 整定原理

PID 控制要取得较好的控制效果,就必须通过调整好比例、积分和微分三种控制作用,形成控制量中既相互配合又相互制约的关系,这种关系不一定是简单的“线性组合”,从变化无穷的非线性组合中可以找出最佳的关系。神经网络所具有的任意非线性表达能力,可以通过对系统性能的学习来实现具有最佳组合的 PID 控制。采用 BP 网络,可以建立参数 k_p , k_i , k_d 自学习的 PID 控制器。

基于 BP (Back Propagation) 网络的 PID 控制系统结构如图 4-8 所示,控制器由两部分构成:

(1) 经典的 PID 控制器,直接对被控对象进行闭环控制,并且三个参数 k_p , k_i , k_d 为在线调整方式;

(2) 神经网络,根据系统的运行状态,调节 PID 控制器的参数,以期达到某种性能指标的最优化,使输出层神经元的输出状态对应于 PID 控制器的三个可调参数 k_p , k_i , k_d 通过神经网络的自学习、加权系数调整,使神经网络输出对应于某种最优控制律下的 PID 控制器参数。

经典增量式数字 PID 的控制算法为

$$u(k) = u(k-1) + k_p(\text{error}(k) - \text{error}(k-1)) + k_i \text{error}(k) + k_d(\text{error}(k) - 2\text{error}(k-1) + \text{error}(k-2)) \quad (4.12)$$

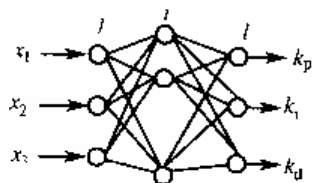


图 4-8 BP 网络结构

式中, k_p , k_i , k_d 分别为比例、积分、微分系数。

采用三层 BP 网络,其结构如图 4-8 所示。

网络输入层的输入为

$$O_j^{(1)} = x(j) \quad j = 1, 2, \dots, M \quad (4.13)$$

式中，输入变量的个数 M 取决于被控系统的复杂程度。

网络隐含层的输入、输出为

$$net_i^{(2)}(k) = \sum_{j=0}^M w_{ij}^{(2)} O_j^{(1)} \quad (4.14)$$

$$O_i^{(2)}(k) = f(net_i^{(2)}(k)) \quad (i=1, \dots, Q)$$

式中， $w_{ij}^{(2)}$ - 隐含层加权系数；上角标 (1)、(2)、(3) 分别代表输入层、隐含层和输出层。

隐层神经元的活化函数取正负对称的 Sigmoid 函数

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4.15)$$

网络输出层的输入输出为

$$net_l^{(3)}(k) = \sum_{i=0}^Q w_{li}^{(3)} O_i^{(2)}(k)$$

$$O_l^{(3)}(k) = g(net_l^{(3)}(k)) \quad (l=1,2,3)$$

$$O_1^{(3)}(k) = k_p \quad (4.16)$$

$$O_2^{(3)}(k) = k_i$$

$$O_3^{(3)}(k) = k_d$$

输出层输出节点分别对应三个可调参数 k_p 、 k_i 、 k_d 。由于 k_p 、 k_i 、 k_d 不能为负值，所以输出层神经元的活化函数取非负的 Sigmoid 函数

$$g(x) = \frac{1}{2}(1 + \tanh(x)) = \frac{e^x}{e^x + e^{-x}} \quad (4.17)$$

取性能指标函数为

$$E(k) = \frac{1}{2} (rin(k) - yout(k))^2 \quad (4.18)$$

按照梯度下降法修正网络的权系数，即按 $E(k)$ 对加权系数的负梯度方向搜索调整，并附加一使搜索快速收敛全局极小的惯性项

$$\Delta w_{li}^{(3)}(k) = -\eta \frac{\partial E(k)}{\partial w_{li}^{(3)}} + \alpha \Delta w_{li}^{(3)}(k-1) \quad (4.19)$$

式中， η 为学习速率； α 为惯性系数。

$$\frac{\partial E(k)}{\partial w_{li}^{(3)}} = \frac{\partial E(k)}{\partial y(k)} \cdot \frac{\partial y(k)}{\partial u(k)} \cdot \frac{\partial u(k)}{\partial O_l^{(3)}(k)} \cdot \frac{\partial O_l^{(3)}(k)}{\partial net_l^{(3)}(k)} \cdot \frac{\partial net_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)} \quad (4.20)$$

$$\frac{\partial net_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)} = O_i^{(2)}(k) \quad (4.21)$$

由于 $\frac{\partial y(k)}{\partial u(k)}$ 未知，所以近似用符号函数 $\text{sgn}(\frac{\partial y(k)}{\partial u(k)})$ 取代，由此带来计算不精确的影响可以通过调整学习速率 η 来补偿。

由式 (4.12) 和式 (4.16)，可求得

$$\frac{\partial u(k)}{\partial O_l^{(3)}(k)} = error(k) - error(k-1) \quad (4.22)$$

$$\frac{\partial u(k)}{\partial O_2^{(3)}(k)} = error(k) \quad (4.23)$$

$$\frac{\partial u(k)}{\partial O_3^{(3)}(k)} = error(k) - 2error(k-1) + error(k-2) \quad (4.24)$$

上述分析可得网络输出层权的学习算法为

$$\Delta w_{ii}^{(3)}(k) = \alpha \Delta w_{ii}^{(3)}(k-1) + \eta \delta_i^{(3)} O_i^{(2)}(k) \quad (4.25)$$

$$\delta_i^{(3)} = error(k) \operatorname{sgn}\left(\frac{\partial y(k)}{\partial u(k)}\right) \frac{\partial u(k)}{\partial O_i^{(3)}(k)} g'(net_i^{(3)}(k)) \quad (i=1,2,3) \quad (4.26)$$

同理可得隐含层加权系数的学习算法

$$\Delta w_{ij}^{(2)}(k) = \alpha \Delta w_{ij}^{(2)}(k-1) + \eta \delta_i^{(2)} O_j^{(1)}(k) \quad (4.27)$$

$$\delta_i^{(2)} = f'(net_i^{(2)}(k)) \sum_{l=1}^3 \delta_l^{(3)} w_{il}^{(3)}(k) \quad (i=1,2,\dots,Q) \quad (4.28)$$

$$\text{式中, } g'(\bullet) = g(x)(1-g(x)), \quad f'(\bullet) = (1-f^2(x))/2 \quad (4.29)$$

基于 BP 网络的 PID 控制器结构如图 4-9 所示, 该控制器控制算法归纳如下:

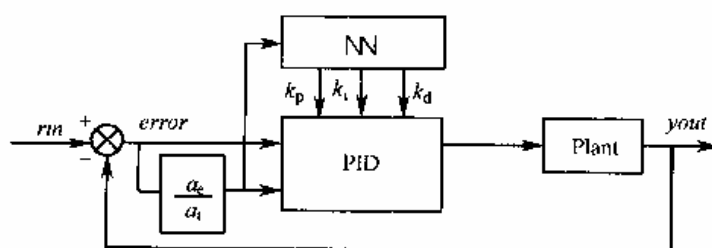


图 4-9 基于 BP 网络的 PID 控制器结构

(1) 确定 BP 网络的结构, 即确定输入层节点数 M 和隐含层节点数 Q , 并给出各层加权系数的初值 $w_{ij}^1(0)$ 和 $w_{ii}^2(0)$, 选定学习速率 η 和惯性系数 α , 此时 $k=1$;

(2) 采样得到 $rin(k)$ 和 $yout(k)$, 计算该时刻误差 $error(k) = rin(k) - yout(k)$;

(3) 计算神经网络 NN 各层神经元的输入、输出, NN 输出层的输出即为 PID 控制器的三个可调参数 k_p, k_i, k_d ;

(4) 根据式 (4.12) 计算 PID 控制器的输出 $u(k)$;

(5) 进行神经网络学习, 在线调整加权系数 $w_{ij}^1(k)$ 和 $w_{ii}^2(k)$, 实现 PID 控制参数的自适应调整;

(6) 置 $k=k+1$, 返回到 (1)。

4.2.2 仿真程序及分析

仿真实例

设被控对象的近似数学模型为

$$yout(k) = \frac{a(k)yout(k-1)}{1+yout^2(k-1)} + u(k-1)$$

式中, 系数 $a(k)$ 是慢时变的, $a(k) = 1.2(1 - 0.8e^{-0.1k})$ 。

神经网络的结构选 4-5-3，学习速率 $\eta = 0.28$ 和惯性系数 $\alpha = 0.04$ ，加权系数初始值取区间 $[-0.5, 0.5]$ 上的随机数。输入指令信号分为两种：

(1) $rin(k) = 1.0$;

(2) $rin(k) = \sin(2\pi t)$ 。取 $S = 1$ 时为阶跃跟踪， $S = 2$ 时为正弦跟踪，初始权值取随机值，运行稳定后用稳定权值代替随机值。其跟踪结果和相应的曲线如图 4-10 至图 4-14 所示。

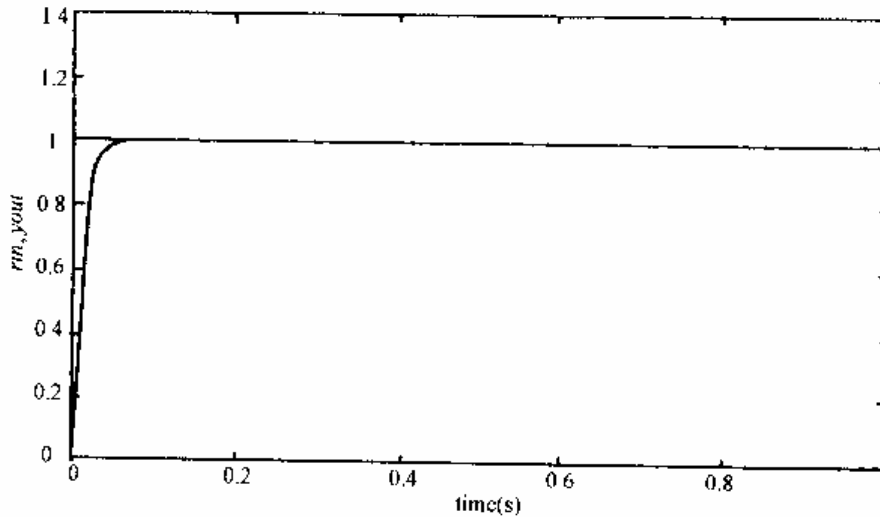


图 4-10 阶跃响应曲线 ($S = 1$)

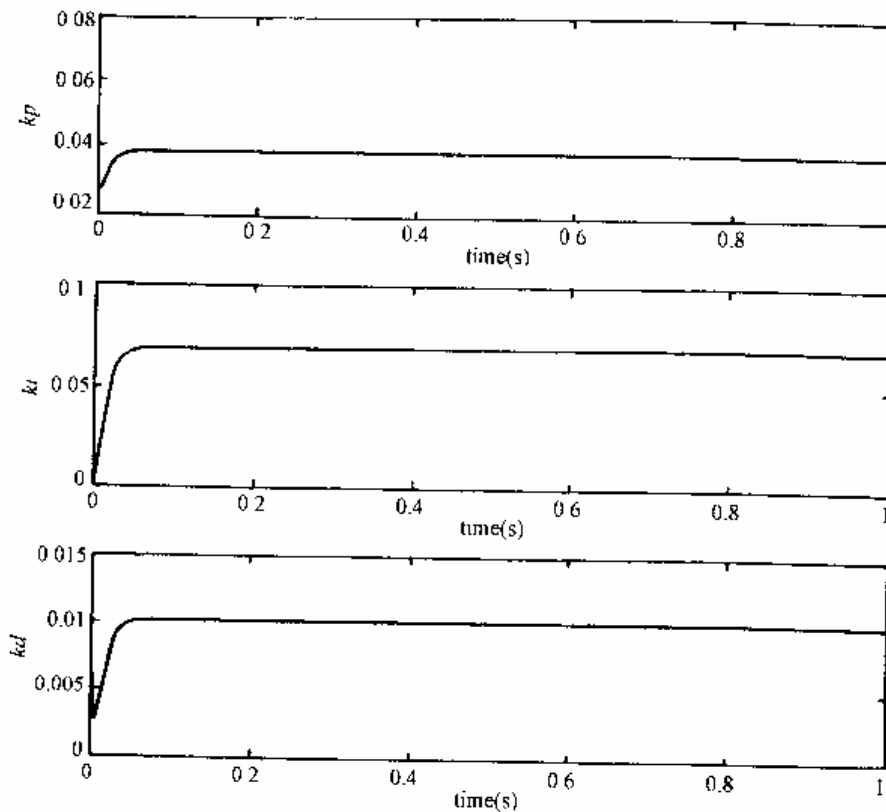


图 4-11 参数自适应整定曲线

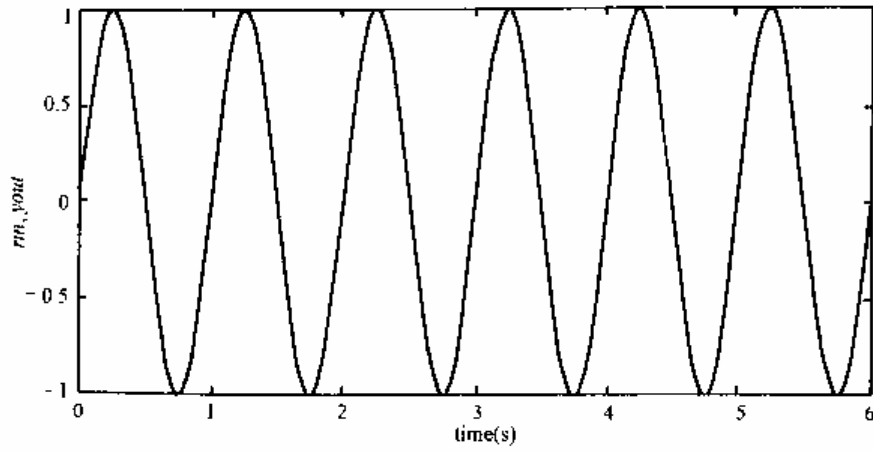


图 4-12 正弦跟踪曲线 ($S=2$)

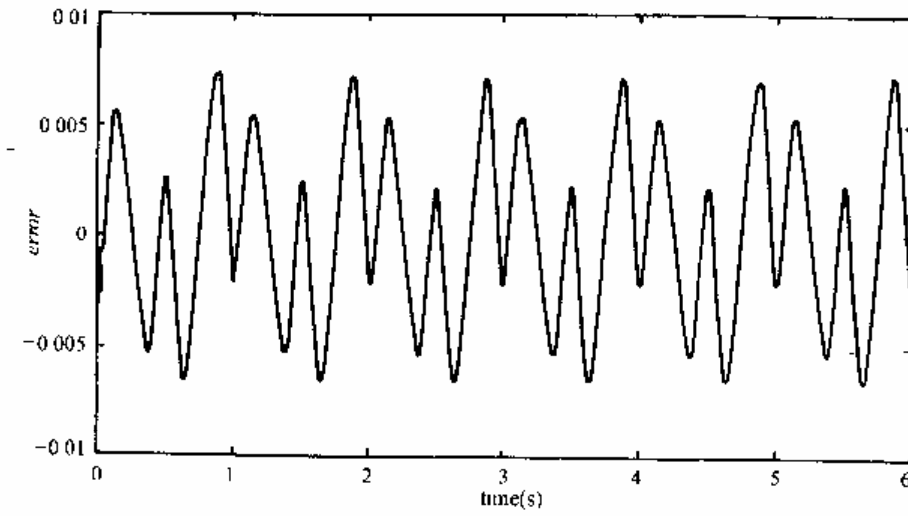


图 4-13 跟踪误差曲线

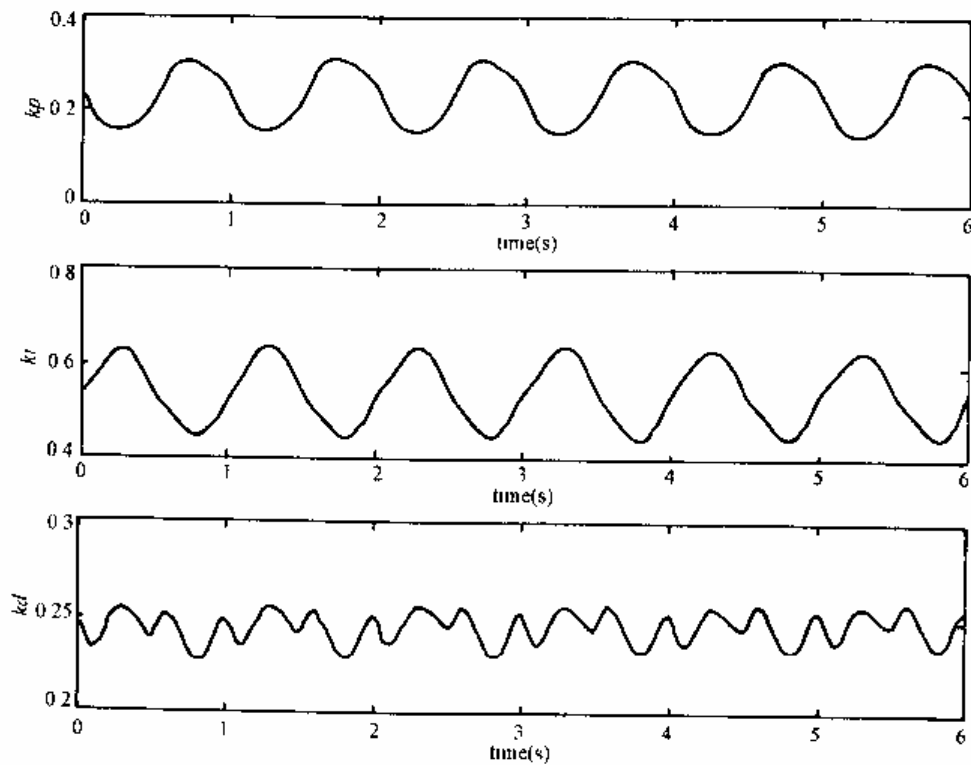


图 4-14 参数自适应整定曲线

仿真程序: chap4_3.m

%BP based PID Control

clear all;

close all;

xite=0.25;

alfa=0.05;

S=2; %Signal type

IN=4;H=5;Out=3; %NN Structure

if S==1 %Step Signal

```
wi=[-0.6394  -0.2696  -0.3756  -0.7023;
     -0.8603  -0.2013  -0.5024  -0.2596;
     -1.0749   0.5543  -1.6820  -0.5437;
     -0.3625   0.0724  -0.6463  -0.2859;
     0.1425   0.0279  -0.5406  -0.7660];
```

%wi=0.50*rands(H,IN);

wi_1=wi;wi_2=wi;wi_3=wi;

```
wo=[0.7576  0.2616  0.5820  -0.1416  -0.1325;
     -0.1146  0.2949  0.8352  0.2205  0.4508;
     0.7201  0.4566  0.7672  0.4962  0.3632];
```

%wo=0.50*rands(Out,H);

wo_1=wo;wo_2=wo;wo_3=wo;

end

if S==2 %Sine Signal

```
wi=[-0.2846   0.2193  -0.5097  -1.0668;
     -0.7484  -0.1210  -0.4708   0.0988;
     -0.7176   0.8297  -1.6000   0.2049;
     -0.0858   0.1925  -0.6346   0.0347;
     0.4358   0.2369   0.4564  -0.1324];
```

%wi=0.50*rands(H,IN);

wi_1=wi;wi_2=wi;wi_3=wi;

```
wo=[1.0438   0.5478   0.8682   0.1446   0.1537;
     0.1716   0.5811   1.1214   0.5067   0.7370;
     1.0063   0.7428   1.0534   0.7824   0.6494];
```

%wo=0.50*rands(Out,H);

wo_1=wo;wo_2=wo;wo_3=wo;

end

```

x=[0,0,0];
u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;

Oh=zeros(H,1);    %Output from NN middle layer
l=Oh;             %Input to NN middle layer
error_2=0;
error_1=0;

ts=0.001;
for k=1:1:6000
time(k)=k*ts;

if S==1
    rin(k)=1.0;
elseif S==.2
    rin(k)=sin(1*2*pi*k*ts);
end

%Unlinear model
a(k)=1.2*(1-0.8*exp(-0.1*k));
yout(k)=a(k)*y_1/(1+y_1^2)+u_1;

error(k)=rin(k)-yout(k);

xi=[rin(k),yout(k),error(k),1];

x(1)=error(k)-error_1;
x(2)=error(k);
x(3)=error(k)-2*error_1+error_2;

epid=[x(1);x(2);x(3)];
f=xi*wi';
for j=1:1:H
    Oh(j)=(exp(l(j))-exp(-l(j)))/(exp(l(j))+exp(-l(j))); %Middle Layer
end
K=wo*Oh;          %Output Layer
for l=1:1:Out
    K(l)=exp(K(l))/(exp(K(l))+exp(-K(l)));          %Getting kp,ki,kd

```

```

end
kp(k)=K(1);ki(k)=K(2);kd(k)=K(3);
Kpid=[kp(k),ki(k),kd(k)];

du(k)=Kpid*epid;
u(k)=u_1+du(k);
if u(k)>=10      % Restricting the output of controller
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

dyu(k)=sign((yout(k)-y_1)/(u(k)-u_1+0.0000001));

%Output layer
for j=1:1:Out
    dK(j)=2/(exp(K(j))+exp(-K(j)))^2;
end
for l=1:1:Out
    delta3(l)=error(k)*dyu(k)*epid(l)*dK(l);
end

for l=1:1:Out
    for i=1:1:H
        d_wo=xite*delta3(l)*Oh(i)+alfa*(wo_1-wo_2);
    end
end
wo=wo_1+d_wo+alfa*(wo_1-wo_2);
%Hidden layer
for i=1:1:H
    dO(i)=4/(exp(I(i))+exp(-I(i)))^2;
end
segma=delta3*wo;
for i=1:1:H
    delta2(i)=dO(i)*segma(i);
end

d_wi=xite*delta2'*xi;
wi=wi_1+d_wi+alfa*(wi_1-wi_2);

```

```

%Parameters Update
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_2-y_1;y_1=yout(k);

wo_3=wo_2;
wo_2=wo_1;
wo_1=wo;

wi_3=wi_2;
wi_2=wi_1;
wi_1=wi;

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout ');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error ');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);
subplot(311);
plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'g');
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'b');
xlabel('time(s)');ylabel('kd');

```

4.3 基于 RBF 神经网络整定的 PID 控制

4.3.1 RBF 神经网络模型

径向基函数(RBF-Radial Basis Function)神经网络是由 J.Moody 和 C.Darken 在 20 世纪 80

年代末提出的一种神经网络,它是具有单隐层的三层前馈网络。由于它模拟了人脑中局部调整、相互覆盖接收域(或称感受野-Receptive Field)的神经网络结构,因此,RBF网络是一种局部逼近网络,已证明它能以任意精度逼近任意连续函数。

(1) 网络结构

RBF网络是一种三层前向网络,由输入到输出的映射是非线性的,而隐含层空间到输出空间的映射是线性的,从而大大加快了学习速度并避免局部极小问题。

RBF网络结构如图4-15所示。

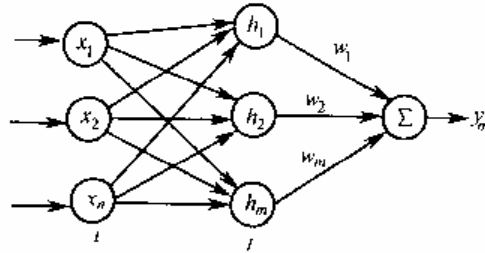


图4-15 RBF神经网络结构

(2) 被控对象 Jacobian 信息的辨识算法

在RBF网络结构中, $\mathbf{X}=[x_1, x_2, \dots, x_n]^T$ 为网络的输入向量。设RBF网络的径向基向量 $\mathbf{H}=[h_1, h_2, \dots, h_j, \dots, h_m]^T$,其中 h_j 为高斯基函数

$$h_j = \exp\left(-\frac{\|\mathbf{X} - \mathbf{C}_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \dots, m \quad (4.30)$$

网络的第 j 个结点的中心矢量为 $\mathbf{C}_j = [c_{j1}, c_{j2}, \dots, c_{jn}]^T$,其中, $i = 1, 2, \dots, n$
设网络的基宽向量为

$$\mathbf{B} = [b_1, b_2, \dots, b_m]^T$$

b_j 为节点 j 的基宽度参数,且为大于零的数。网络的权向量为

$$\mathbf{W} = [w_1, w_2, \dots, w_m]^T \quad (4.31)$$

辨识网络的输出为

$$y_m(k) = w_1 h_1 + w_2 h_2 + \dots + w_m h_m \quad (4.32)$$

辨识器的性能指标函数为

$$J_i = \frac{1}{2} (\text{yout}(k) - y_m(k))^2 \quad (4.33)$$

根据梯度下降法,输出权、节点中心及节点基宽参数的迭代算法如下

$$w_j(k) = w_j(k-1) + \eta(\text{yout}(k) - y_m(k))h_j + \alpha(w_j(k-1) - w_j(k-2)) \quad (4.34)$$

$$\Delta b_j = (\text{yout}(k) - y_m(k))w_j h_j \frac{\|\mathbf{X} - \mathbf{C}_j\|^2}{b_j^3} \quad (4.35)$$

$$b_j(k) = b_j(k-1) + \eta \Delta b_j + \alpha(b_j(k-1) - b_j(k-2)) \quad (4.36)$$

$$\Delta c_{ji} = (\text{yout}(k) - y_m(k))w_j \frac{x_{ji} - c_{ji}}{b_j^2} \quad (4.37)$$

$$c_{ji}(k) = c_{ji}(k-1) + \eta \Delta c_{ji} + \alpha(c_{ji}(k-1) - c_{ji}(k-2)) \quad (4.38)$$

式中, η 为学习速率, α 为动量因子。

Jacobian 阵(即为对象的输出对控制输入的灵敏度信息)算法为

$$\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial y_m(k)}{\partial u(k)} = \sum_{j=1}^m w_j h_j \frac{c_{ji} - x_1}{b_j^2} \quad (4.39)$$

式中, $x_1 = u(k)$

4.3.2 RBF 网络 PID 整定原理

采用增量式 PID 控制器, 控制误差为

$$error(k) = rin(k) - yout(k)$$

PID 三项输入为

$$\begin{aligned} xc(1) &= error(k) - error(k-1) \\ xc(2) &= error(k) \\ xc(3) &= error(k) - 2error(k-1) + error(k-2) \end{aligned} \quad (4.40)$$

控制算法为

$$u(k) = u(k-1) + k_p(error(k) - error(k-1)) + k_i error(k) + k_d(error(k) - 2error(k-1) + error(k-2)) \quad (4.41)$$

神经网络整定指标为

$$E(k) = \frac{1}{2} error(k)^2 \quad (4.42)$$

k_p, k_i, k_d 的调整采用梯度下降法

$$\Delta k_p = -\eta \frac{\partial E}{\partial k_p} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial k_p} = \eta error(k) \frac{\partial y}{\partial u} xc(1) \quad (4.43)$$

$$\Delta k_i = -\eta \frac{\partial E}{\partial k_i} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial k_i} = \eta error(k) \frac{\partial y}{\partial u} xc(2) \quad (4.44)$$

$$\Delta k_d = -\eta \frac{\partial E}{\partial k_d} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial k_d} = \eta error(k) \frac{\partial y}{\partial u} xc(3) \quad (4.45)$$

式中, $\frac{\partial y}{\partial u}$ 为被控对象的 Jacobian 信息, 可通过神经网络的辨识而得。

RBF 整定 PID 控制系统的结构如图 4-16 所示。

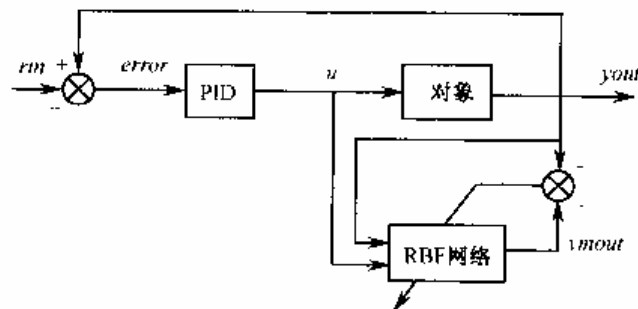


图 4-16 RBF 网络整定 PID 控制框图

4.3.3 仿真程序及分析

仿真实例

被控对象为

$$yout(k) = \frac{-0.1yout(k-1) + u(k-1)}{1 + yout(k-1)^2}$$

输入指令信号为 $rin(t) = 1.0 \operatorname{sgn}(\sin(2\pi t))$ ，RBF 网络结构选为 3-6-1，网络辨识的三个输入为： $u(k)$ ， $yout(k)$ ， $yout(k-1)$ 。 $M=1$ 时为 RBF 整定的 PID 控制，其结果如图 4-17 至图 4-19 所示， $M=2$ 时为未加整定的 PID 控制，其结果如图 4-20 所示。

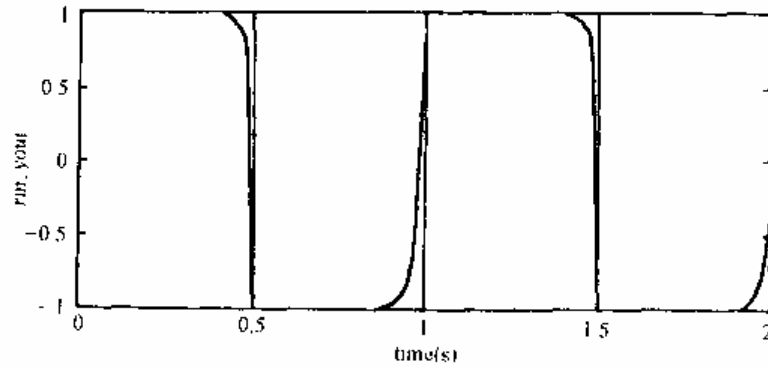


图 4-17 RBF 整定 PID 控制方波响应 ($M=1$)

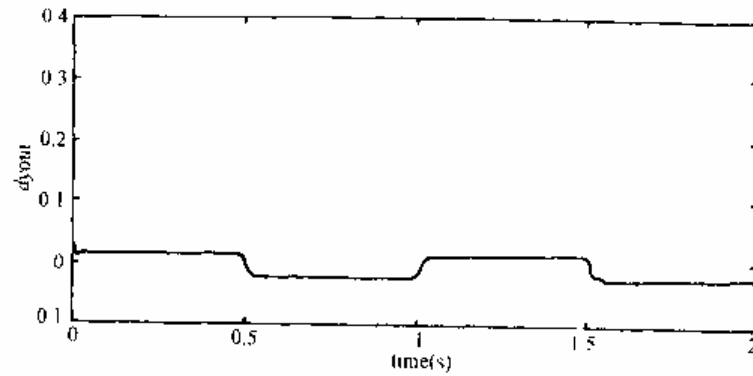


图 4-18 对象 Jacobian 信息的辨识结果

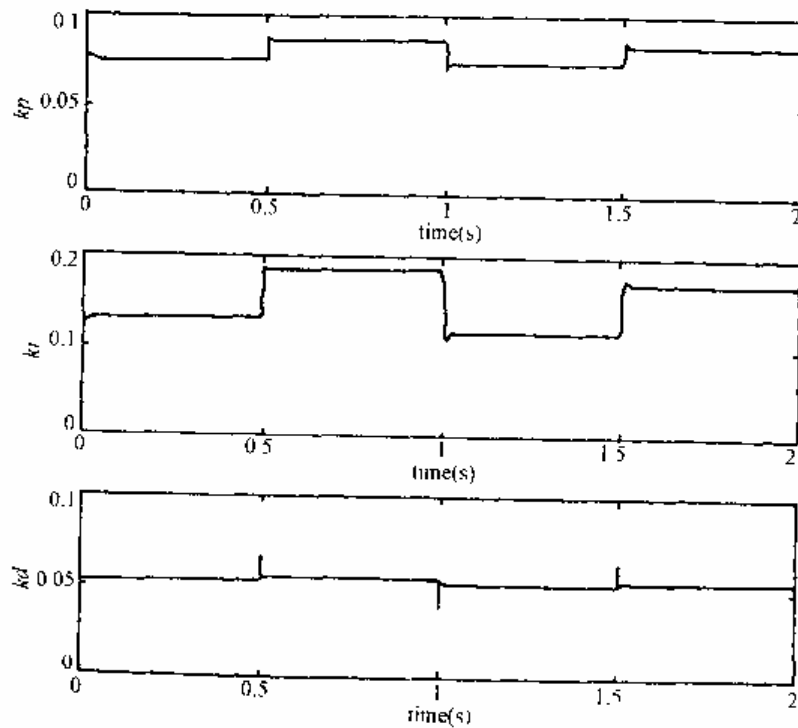


图 4-19 参数自适应整定曲线

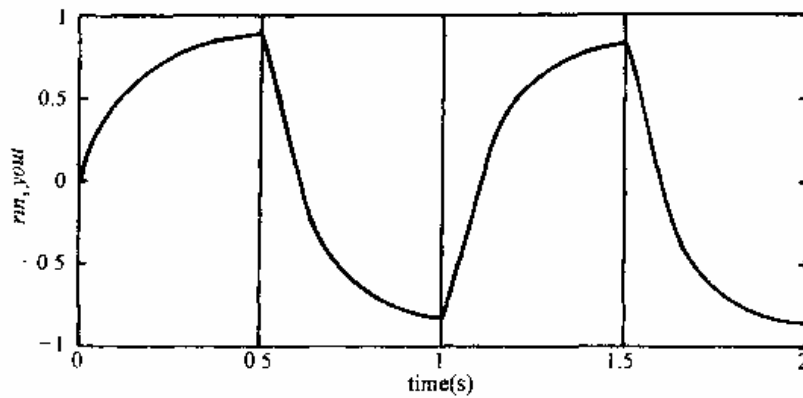


图 4-20 未整定的 PID 控制方波响应 ($M = 2$)

仿真程序: chap4_4.m

```
%Adaptive PID control based on RBF Identification
clear all;
close all;

xite=0.25;
alfa=0.05;
belte=0.01;
x=[0,0,0]';

ci=30*ones(3,6);
bi=40*ones(6,1);
w=10*ones(6,1);

h=[0,0,0,0,0,0]';

ci_1=ci;ci_3=ci_1;ci_2=ci_1;
bi_1=bi;bi_2=bi_1;bi_3=bi_2;
w_1=w;w_2=w_1;w_3=w_1;

u_1=0;y_1=0;
xc=[0,0,0]';
error_1=0;error_2=0;error=0;

%kp=rand(1);
%ki=rand(1);
%kd=rand(1);
kp=0.03;
ki=0.01;
kd=0.03;
```



```

kp_1=kp;
kd_1=kd;
ki_1=ki;

xitekp=0.20;
xitekd=0.20;
xiteki=0.20;

ts=0.001;
for k=1:1:2000
    time(k)=k*ts;
    rin(k)=1.0*sign(sin(2*pi*k*ts));
    yout(k)=(-0.1*y_1+u_1)/(1+y_1^2); %Nonlinear plant

    for j=1:1:6
        h(j)=exp(-norm(x-ci_1(:,j))^2/(2*bi_1(j)*bi_1(j)));
    end

    ymout(k)=w_1'*h;

    d_w=0*w; % Defining matrix number of d_w equal to that of w
    for j=1:1:6
        d_w(j)=xite*(yout(k)-ymout(k))*h(j);
    end
    w=w_1+d_w+alfa*(w_1-w_2)+belte*(w_2-w_3);

    d_bi=0*bi;
    for j=1:1:6
        d_bi(j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(bi_1(j)^-3)*norm(x-
ci_1(:,j))^2;
    end
    bi=bi_1+d_bi+alfa*(bi_1-bi_2)+belte*(bi_2-bi_3);
    for j=1:1:6
        for i=1:1:3
            d_ci(i,j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(x(i)-
ci_1(i,j))*(bi_1(j)^-2);
        end
    end
    ci=ci_1+d_ci+alfa*(ci_1-ci_2)+belte*(ci_2-ci_3);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yu=0;
for j=1:1:6
    yu=yu+w(j)*h(j)*( x(1)+ci(1,j))/bi(j)^2;
end
dyout(k)=yu;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Start of Control system%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
error(k)=rin(k)-yout(k);
kp(k)=kp_1+xitekp*error(k)*dyout(k)*xc(1);
kd(k)=kd_1+xitek*error(k)*dyout(k)*xc(2);
ki(k)=ki_1+xiteki*error(k)*dyout(k)*xc(3);
if kp(k)<0
    kp(k)=0;
end
if kd(k)<0
    kd(k)=0;
end
if ki(k)<0
    ki(k)=0;
end

M=1;
if M==2 %Only PID Control
    kp(k)=0.03;
    ki(k)=0.01;
    kd(k)=0.03;
end
u(k)=u_1+kp(k)*xc(1)+kd(k)*xc(2)+ki(k)*xc(3);

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
%Return of parameters
x(1)=u(k);
x(2)=yout(k);
x(3)=y_1;

```

```

u_1=u(k);
y_1=yout(k);

w_3=w_2;
w_2=w_1;
w_1=w;

xc(1)=error(k)-error_1;           %Calculating P
xc(2)=error(k)-2*error_1+error_2; %Calculating D
xc(3)=error(k);                   %Calculating T

error_2=error_1;
error_1=error(k);

kp_1=kp(k);
kd_1=kd(k);
ki_1=ki(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,yout,'r',time,ymout,'b');
xlabel('time(s)');ylabel('yout,ymout');
figure(3);
plot(time,dyout);
xlabel('time(s)');ylabel('dyout');
figure(4);
subplot(311);
plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'r');
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'r');
xlabel('time(s)');ylabel('kd');

```

4.4 基于 RBF 神经网络辨识的单神经元 PID 模型参考自适应控制

4.4.1 神经网络模型参考自适应控制原理

图 4-21 所示为单神经元 PID 模型参考自适应控制系统框图，单神经元 PID 作为控制器 NNC，RBF 网络作为辨识器 NNI，实现对被控对象的 Jacobian 信息辨识。

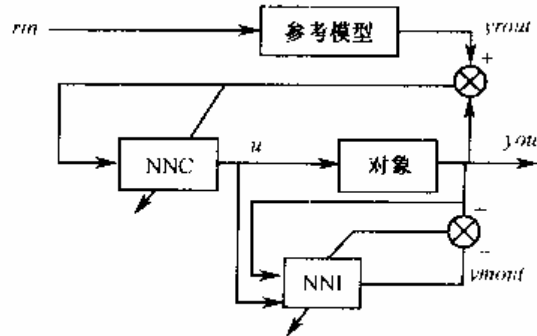


图 4-21 单神经元 PID 模型参考自适应控制系统

单神经元网络的输入为

$$\begin{aligned} xc_1(k) &= error(k) \\ xc_2(k) &= (error(k) - error(k-1))/T \\ xc_3(k) &= \sum_{k=1}^k error(k)T \end{aligned} \quad (4.46)$$

神经网络的输出即为控制器的输出

$$u(k) = \sum_{i=1}^3 wc_i(k)xc_i(k) \quad (4.47)$$

神经网络控制器的学习算法采用 delta 学习规则

$$E = \frac{1}{2} error(k)^2 = \frac{1}{2} (yrout(k) - yout(k))^2 \quad (4.48)$$

$$\Delta wc_i(k) = -\eta \frac{\partial E}{\partial wc_i} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial u} \frac{\partial u}{\partial wc_i} = \eta error(k) \frac{\partial y}{\partial u} xc_i(k) \quad (4.49)$$

式中， $\frac{\partial y}{\partial u}$ 为被控对象的 Jacobian 信息，通过 RBF 神经网络的辨识而得。

4.4.2 仿真程序及分析

仿真实例

针对二阶传递函数进行单神经元 PID 模型参考自适应控制，被控对象为

$$G_p(s) = \frac{1}{0.003 s^2 + 0.067 s}$$

采样时间为 1ms， $S=1$ 时，输入指令信号为正弦信号； $S=2$ 时，输入指令信号为参考模型。取 $S=2$ ，参数辨识和 Jacobian 信息辨识以及正弦位置跟踪结果如图 4-22 至图 4-24 所示。

参考模型指令信号为

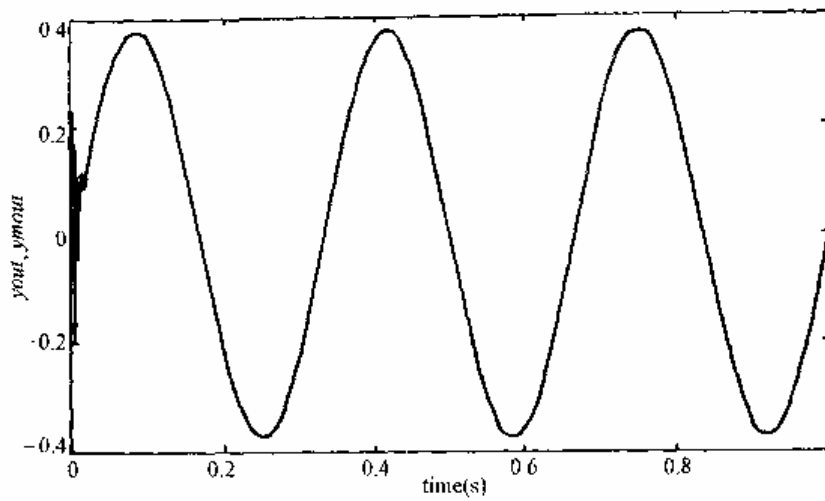


图 4-22 参考模型辨识结果

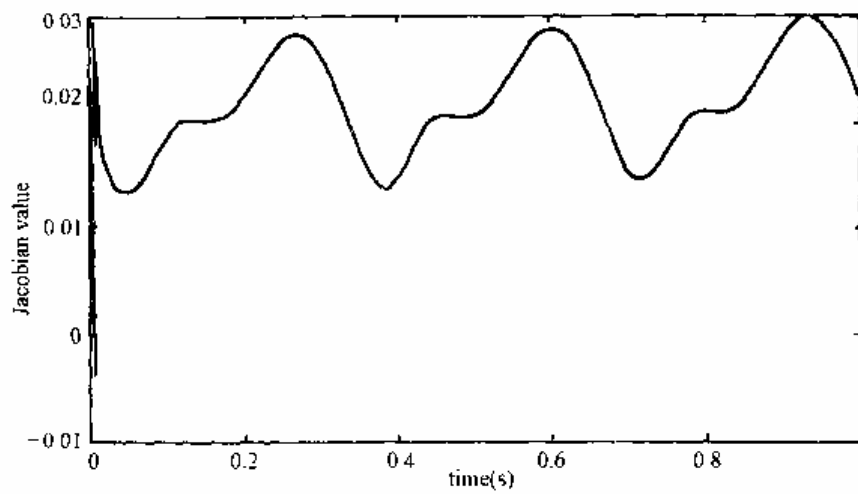


图 4-23 Jacobian 信息辨识结果

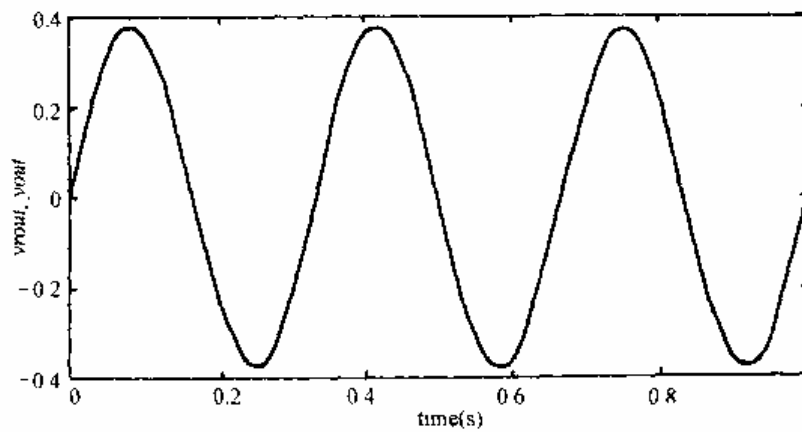


图 4-24 正弦位置跟踪

$$rin(k) = 0.50\sin(0.006\pi k)$$

$$yrou(k) = 0.2yrou(k-1) + 0.6rin(k)$$

仿真程序: chap4_5.m

%Single Neural Net PID Controller based on RBF Identification

```

clear all;close all;

Jp=0.0030;bp=0.067;
ts=0.001;

Gp=tf([1],[Jp,bp,0]);
Gpz=c2d(Gp,ts,'z');
[num,den]=tfdata(Gpz,'v');

h=zeros(6,1);
%w=rands(6,1);
w=[-0.5646;
    0.3937;
   -0.5556;
    0.3981;
    0.4495;
    0.2565];
w_1=w;w_2=w;w_3=w;

xite=0.40;
alfa=0.05;
belte=0.01;

x=[0,0,0]';

%c=0.1*ones(3,6);
%b=0.1*ones(6,1);
c=[-3.1829    0.5211    7.1754   11.6631    3.6992  -10.9150;
   -3.8909    2.3999    5.1730    8.5871  -11.3737   -7.0179;
   -4.2018    2.6742    5.1828    8.5238   -1.8936   -6.1845];
b=[ 5.3074;
    1.4771;
   26.4114;
   22.1716;
   52.9082;
    5.6906];
c_1=c;c_2=c_1;c_3=c_2;
b_1=b;b_2=b_1;b_3=b_2;

xc=[0,0,0]';

```

```

xitec=0.60;

kp=80;
kd=5;
ki=50;

wc=[kp,kd,ki];
wc_1=wc;wc_2=wc;wc_3=wc;

error_1=0;error_2=0;
y_1=0;y_2=0;
u_1=0;u_2=0;

ei=0;
c_size=size(c);

for k=1:1:1000
    time(k)=k*ts;
    rin(k)=0.50*sin(3*2*pi*k*ts);

S=2;
if S==1
    yout(k)=1.0*rin(k);
end
if S==2
    yout(k)=0.2*y_1+0.6*rin(k); %Reference Model
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

for j=1:1:c_size(2),
    h(j)=exp(-norm(x-c_1(:,j))^2/(2*b_1(j)*b_1(j)));
end
ymout(k)=w_1'*h;

id=abs(yout(k)-ymout(k));
if id>0.0001,
%-----Adjusting RBF parameters-----%
    d_w=0*w; % Defining matrix number of d_w equal to that of w

```

```

    for j=1:1:6
        d_w(j)=xite*(yout(k)-ymout(k))*h(j);
    end
    w=w_1+d_w+alfa*(w_1-w_2)+belte*(w_2-w_3);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    d_b=0*b;
    for j=1:1:6
        d_b(j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(b_1(j)^-3)*norm(x-
c_1(:,j))^2;
    end
    b=b_1+ d_b+alfa*(b_1-b_2)+belte*(b_2-b_3);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for j=1:1:6
        for i=1:1:3
            d_c(i,j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(x(i)-
c_1(i,j))*(b_1(j)^-2);
        end
    end
    c=c_1+d_c+alfa*(c_1-c_2)+belte*(c_2-c_3);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculating Jacobian %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dyu=0;
for j=1:1:c_size(2)
    dyu=dyu+w(j)*h(j)*(-x(1)+c(1,j))/b(j)^2;
end
dyout(k)=dyu;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Parameters Return%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
error(k)=yrou(k)-yout(k);
xc(1)=error(k);
xc(2)=(error(k)-error_1)/ts;
ei=ei+error(k)*ts;
xc(3)=ei;

u(k)=wc*xc; %Control law
if u(k)>10,
    u(k)=10;
end
if u(k)<-10,
    u(k)=-10;
end

```



```

d_wc=0*wc;      % Defining matrix number of d_w equal to that of w
for j=1:1:3
    d_wc(j)=xitec*error(k)*xc(j)*dyout(k);
end
wc=wc_1+d_wc+alfa*(wc_1-wc_2)+belte*(wc_2-wc_3);

error_2=error_1;error_1=error(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

x(3)=y_2;
x(2)=y_1;
x(1)=u_1;

w_3=w_2;w_2=w_1;w_1=w;
c_3=c_2;c_2=c_1;c_1=c;
b_3=b_2;b_2=b_1;b_1=b;
wc_3=wc_2;wc_2=wc_1;wc_1=wc;
end
figure(1);
plot(time,yout,'r',time,ymout,'b');
xlabel('time(s)');ylabel('yout,ymout');
figure(2);
plot(time,dyout,'r');
xlabel('time(s)');ylabel('Jacobian value');
figure(3);
plot(time,yrou,'b',time,yout,'r');
xlabel('time(s)');ylabel('yrou,yout');
figure(4);
plot(time,yrou-yout,'r');
xlabel('time(s)');ylabel('control error');

```

4.5 基于 CMAC（神经网络）与 PID 的并行控制

4.5.1 CMAC 概述

小脑模型神经网络（CMAC-Cerebellar Model Articulation Controller）是一种表达复杂非线性函数的表格查询型自适应神经网络，该网络可通过学习算法改变表格的内容，具有信息分类存储的能力。

CMAC 把系统的输入状态作为一个指针，把相关信息分布式地存入一组存储单元。它本质上是一种用于映射复杂非线性函数的查表技术。具体作法是将输入空间分成许多分块，每个分块指定一个实际存储器位置；每个分块学习到的信息分布式地存储到相邻分块的位置上；存储单元数通常比所考虑问题的最大可能输入空间的分块数少得多，故实现的是多对一的映射，即多个分块映射到同样一个存储器地址上。

CMAC 已被公认为是一类联想记忆神经网络的重要组成部分，它能够学习任意多维非线性映射。CMAC 算法可有效地用于非线性函数逼近、动态建模、控制系统设计等。CMAC 较其他神经网络的优越性体现在：

- (1) 它是基于局部学习的神经网络，它把信息存储在局部结构上，使每次修正的权很少，在保证函数非线性逼近性能的前提下，学习速度快，适合于实时控制；
- (2) 具有一定的泛化能力，即所谓相近输入产生相近输出，不同输入给出不同输出；
- (3) 连续（模拟）输入、输出能力；
- (4) 寻址编程方式，在利用串行计算机仿真时，它可使回响速度加快；
- (5) 作为非线性逼近器，它对学习数据出现的次序不敏感。

由于 CMAC 所具有的上述优越性能，使它比一般神经网络具有更好的非线性逼近能力，更适用于复杂动态环境下的非线性实时控制。

CMAC 的基本思想在于：在输入空间中给出一个状态，从存储单元中找到对应于该状态的地址，将这些存储单元中的内容求和得到 CMAC 的输出；将此响应值与期望输出值进行比较，并根据学习算法修改这些已激活的存储单元的内容。

CMAC 的设计方法分为以下三步：

(1) 量化（概念映射）

在输入层对 N 维输入空间进行划分，每一个输入都降落到 N 维网格基的一个超立方体单元内。中间层由若干个判断区间构成，对任意一个输入只有少数几个区间的输出为非零值，非零值区间的个数为范化参数 c ，它规定了网络内部影响网络输出的区域大小。

(2) 地址映射（实际映射）

采用除余数法，将输入样本映射至概念存储器的地址，除以一个数，得到的余数作为实际存储器的地址值。即将概念存储器中的 c 个单元映射至实际存储器的 c 个地址。

(3) CMAC 的函数计算（CMAC 输出）

将输入映射至实际存储器的 c 个单元，每个单元中存放着相应权值，CMAC 的输出为 c 个实际存储器单元加权之和。

CMAC 的结构如图 4-25 所示。

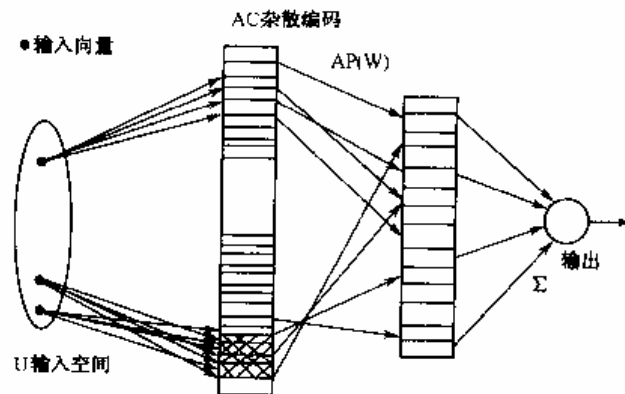


图 4-25 CMAC 结构

4.5.2 CMAC 与 PID 复合控制算法

CMAC 一开始就被应用于机器人控制中,目前有多种控制形式,如 CMAC 直接逆运动控制、CMAC 前馈控制、CMAC 反馈控制等,本书采用的是 CMAC 前馈控制。CMAC 与 PID 复合控制结构如图 4-26 所示,该系统通过 CAMC 和 PID 的复合控制实现前馈反馈控制,其特点为:

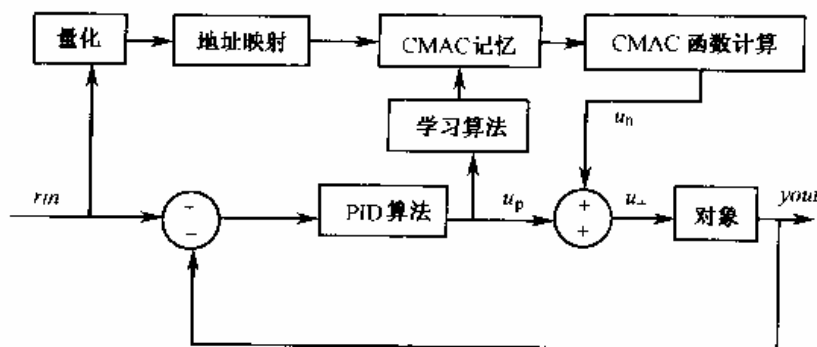


图 4-26 CMAC 与 PID 复合控制结构图

- (1) 小模型神经控制器实现前馈控制,实现被控对象的逆动态模型;
- (2) 常规控制器实现反馈控制,保证系统的稳定性,且抑制扰动。

CMAC 采用有导师的学习算法。每一控制周期结束时,计算出相应的 CMAC 输出 $u_n(k)$,并与总控制输入 $u(k)$ 相比较,修正权重,进入学习过程。学习的目的是使总控制输入与 CMAC 的输出之差最小。经过 CMAC 的学习,使系统的总控制输出由 CMAC 产生。而常规控制器采用传统的 PD 算法而不用 PID 控制算法,使 CMAC 的学习仅仅依赖于误差的当时测量值及变化值。

该系统的控制算法为

$$u_n(k) = \sum_{i=1}^c w_i a_i \quad (4.50)$$

$$u(k) = u_n(k) + u_p(k) \quad (4.51)$$

式中, a_i 为二进制选择向量, c 为 CMAC 网络的泛化参数, $u_n(k)$ 为 CMAC 产生相应的输出, $u_p(k)$ 为常规控制器 PID 产生的输出。

每一控制周期结束时,CMAC 输出 $u_n(k)$ 与总控制输出 $u(k)$ 相比较,修正权重,进入学习过程。学习的目的是使总控制输入与 CMAC 的输出之差最小,即使系统的总控制输出主要由 CMAC 控制器产生。

CMAC 的调整指标为

$$E(k) = \frac{1}{2} (u(k) - u_n(k))^2 \cdot \frac{a_i}{c} \quad (4.52)$$

$$\Delta w(k) = \eta \frac{u(k) - u_n(k)}{c} a_i = \eta \frac{u_p(k)}{c} a_i \quad (4.53)$$

$$w(k) = w(k-1) + \Delta w(k) + \alpha (w(k) - w(k-1)) \quad (4.54)$$

式中, η 为网络学习速率, $\eta \in (0,1)$, α 为惯性量, $\alpha \in (0,1)$ 。

当系统开始运行时,置 $w = 0$,此时 $u_n = 0$, $u = u_p$,系统由常规控制器进行控制。通

过 CMAC 的学习, 使 PID 产生的输出控制量 $u_p(k)$ 逐渐为零, CMAC 产生的输出控制量 $u_n(k)$ 逐渐逼近控制器总输出 $u(k)$ 。

4.5.3 仿真程序及分析

仿真实例

被控对象采用二阶传递函数

$$G_p(s) = \frac{1770}{s^2 + 60s + 1770}$$

PID 控制参数取 $k_p = 25, k_i = 0, k_d = 0.28$, 采样时间为 1ms。取输入信号为正弦叠加信号, 其跟踪结果如图 4-27 和图 4-28 所示, 其中图 4-28 分别给出了 CMAC 控制器、PD 控制器和总控制器的输出。

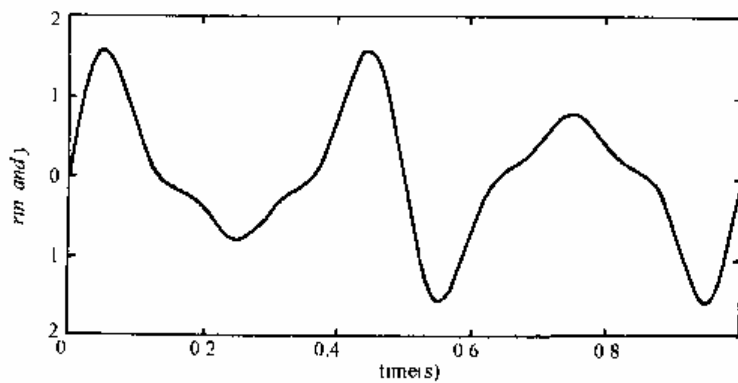


图 4-27 正弦叠加信号跟踪

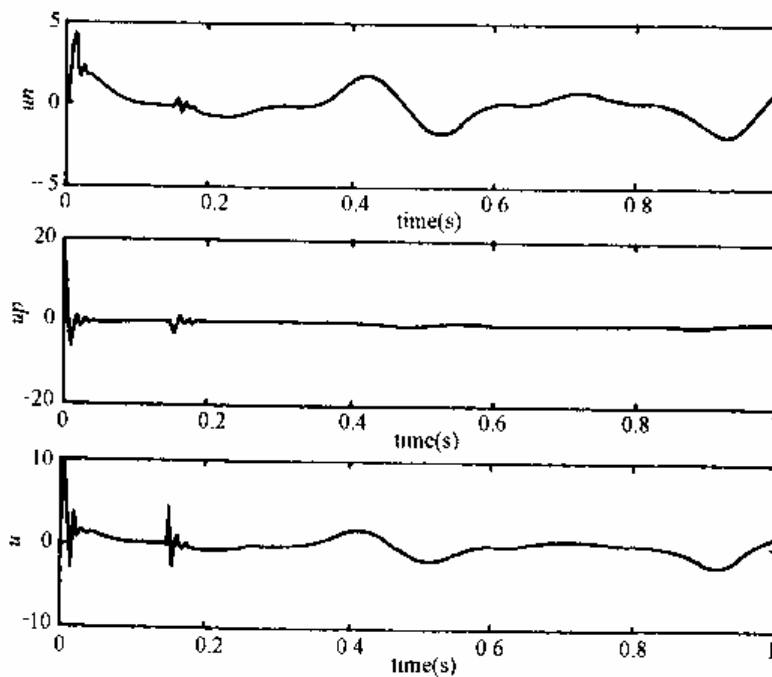


图 4-28 各个控制器的输出

仿真程序: chap4_6.m

%CMAC and PID Concurrent Control

```

clear all;
close all;

ts=0.001;
sys=tf(1770,[1,60,1770]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

alfa=0.04;

N=100;C=5;

w=zeros(N+C,1);
w_1=w;w_2=w;d_w=w;

y_1=0;y_2=0;y_3=0;
u_1=0.0;u_2=0.0;u_3=0.0;

x=[0,0,0]';
error_1=0;

s=4;      %Selecting Signal
if s==1    %Sine Signal
    A=0.50;
    Smin= -A;
    Smax=A;
    xite=0.10;
    kp=25;
    ki=0.0;
    kd=0.28;

elseif s==2    %Square Wave Signal
    A=0.50;
    Smin=-A;
    Smax=A;
    xite=0.10;
    kp=25;
    ki=0.0;
    kd=0.28;

elseif s==3    %Step Signal

```

```

    Smin=0;
    Smax=1.0;
    xite=0.10;
    kp=25;
    ki=0.0;
    kd=0.28;
end
if s==4      %Random Signal
    A=1.80;
    Smin=-A;
    Smax=A;
    xite=0.10;
    kp=20;
    ki=0.0;
    kd=0.28;
end

%Coding Input Value
dvi=(Smax-Smin)/(N-1);

for i=1:1:C      %C size
    v(i)=Smin;
end
for i=C+1:1:C+N  %N size
    v(i)=v(i-1)+dvi;
end
for i=N+C+1:1:N+2*C  %C size
    v(i)=Smax;
end

for k=1:1:1000
    time(k)=k*ts;

if s==1
    rin(k)=A*sin(4*2*pi*k*ts);      %Sin Signal
elseif s==2
    rin(k)=A*sign(sin(2*2*pi*k*ts));  %Square Signal
elseif s==3
    rin(k)=1;                        %Step Signal
elseif s==4

```

```

    rin(k)-1.0*sin(2*pi*3*k*ts)+0.5*sin(2*pi*5*k*ts)+0.3*sin(2*pi*7*k*ts);
end

for i=1:1:N+C
if rin(k)>=v(i)&rin(k)<=-v(i+C)
    a(i)-1;
else
    a(i)-0;
end
end

yout(k)=-den(2)*y_1 den(3)*y_2+num(2)*u_1+num(3)*u_2;

error(k)=rin(k)-yout(k);

%CMAC Neural Network Controller
un(k)=a*w;

%PID Controller
up(k)=-kp*x(1)+kd*x(2)+ki*x(3);

MM=2;
if MM==1 %Only Using PID Control
    u(k)=up(k);
elseif MM=-2 %Total control output
    u(k)=up(k)+un(k);
end

if k==150 %Disturbance
    u(k)=u(k)+5.0;
end

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end

%Update NN Weight

```

```

if s==1|s==4
    d_w=xite*(u(k)-un(k))/C;    %Sin Signal
elseif s==2|s==3
    d_w=a'*xite*(u(k)-un(k))/C;    %Step Signal and Square Signal
end

w=w_1+ d_w+alfa*(w_1-w_2);

%Parameters Update
w_3=w_2;w_2=w_1;w_1=w;
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

x(1)=error(k);                % Calculating P
x(2)=(error(k)-error_1)/ts;    % Calculating D
x(3)=x(3)+error(k)*ts;        % Calculating I

error_2=error_1;error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('(rin and y)');
figure(2);
subplot(311);
plot(time,un);
xlabel('time(s)');ylabel('un');
subplot(312);
plot(time,up);
xlabel('time(s)');ylabel('up');
subplot(313);
plot(time,u);
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');

```

CMAC 控制算法虽然是由 PD 控制器的输出训练的，但并不是 PD 控制器的简单复制。加入 PD 控制器是为了评判 CMAC 控制器的性能，增强系统的稳定性，抑制扰动。PD 单独控制时，增益 k_p 的值在很大程度上决定着控制效果，而采用 PD+CMAC 控制时控制效果不依赖于 k_p 的值， k_p 的值只需在一个合理的范围内即可。

通过仿真结果可以看出,开始的时候主要是常规 PD 控制器起作用,经过对常规控制器的输出的不断学习,逐渐由小脑模型的输出起控制作用。小脑模型的加入使得控制效果比单独的 PID 控制效果要好很多,当方波(阶跃)输入时,大大减小了超调,加快了控制响应速度,充分的体现了小脑模型的特点,即输出误差小、实时性好、鲁棒性强等。

在加入干扰的情况下,会发现由于 CMAC 的加入使得干扰作用下的控制系统很快地恢复稳定状态。CMAC+PD 并行控制在一定程度上克服了常规控制器所不能避免的一些弊端,使控制效果得到提高。

4.6 CMAC 与 PID 并行控制的 Simulink 仿真

4.6.1 Simulink 仿真方法

采用 Simulink 仿真实现 CAMC 和 PID 的并行控制。通过 Simulink 仿真,可以使得控制结构看来更加清晰。

采用调用 M 函数的形式来编写 CMAC 神经网络控制器和 PID 控制器,指令信号类型是在 M 函数中实现的。通过选择 S 值不同的输入信号来实现阶跃、方波和正弦信号的跟踪。

4.6.2 仿真程序及分析

仿真实例

被控对象采用二阶传递函数

$$G_p(z) = \frac{0.0008674z + 0.0008503}{z^2 - 1.94z + 0.9418}$$

控制系统由 Simulink 程序来实现,控制系统中的 CMAC 和 PID 控制器由两个 M 函数来实现,即 chap4_7f1.m 和 chap4_7f2.m。

在 PID 控制 M 函数中采用了 persistent 命令实现动态参数的保持,采用 global 命令实现参数的共享,采用时钟信号 Clock 实现参数的初始化。S 为输入指令信号的类型,其中 S=1 时,输入指令信号为阶跃信号; S=2 时,输入指令信号为正弦信号; S=3 时,输入指令信号为方波信号。

在 CMAC 控制 M 函数中利用时钟 Clock 功能实现神经网络权值的初始化,CMAC 神经网络参数取 $N=300$, $C=5$, $\eta=0.10$ 。

Simulink 仿真时间取 1s,仿真结果如下:

(1) 阶跃响应:在 Simulink 程序中取阶跃响应输入,在 M 函数中取 $S=1$,PID 参数取 $k_p=0.40, k_i=0, k_d=0.28$ 。阶跃响应如图 4-29 所示。

(2) 正弦响应:在 Simulink 程序中取正弦输入信号,幅值为 0.50,频率为 4Hz。在 M 函数中取 $S=2$,PID 参数取 $k_p=40, k_i=0, k_d=0.28$ 。正弦响应如图 4-30 所示。

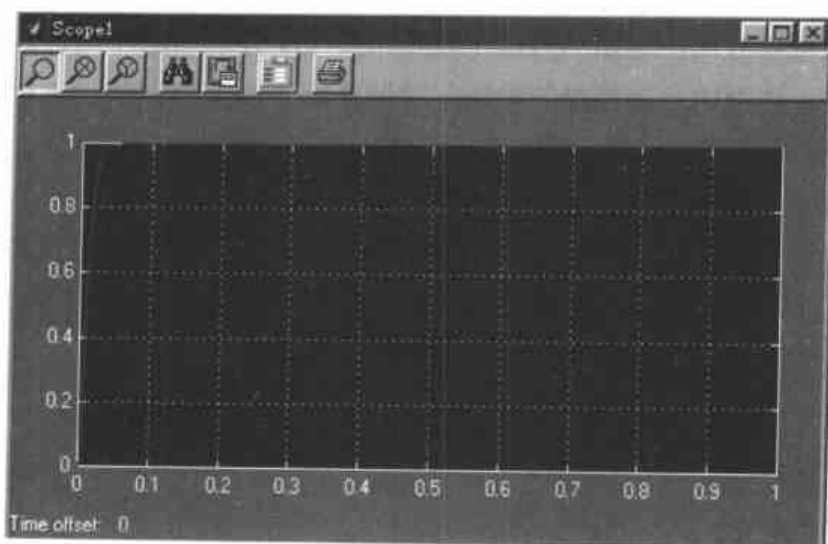


图 4-29 阶跃响应

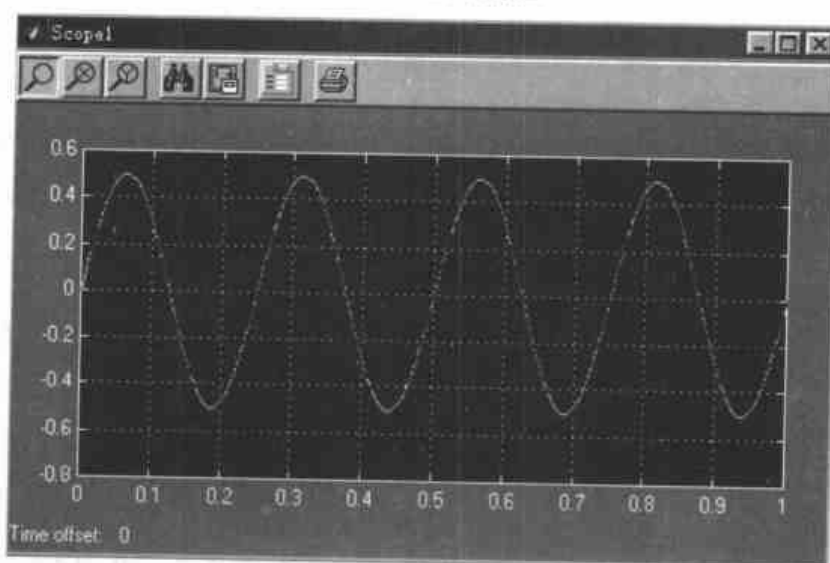


图 4-30 正弦响应

(3) 方波响应: 在 Simulink 程序中取方波响应输入, 幅值为 0.50, 频率为 4Hz。在 M 函数中取 $S=3$, PID 参数取 $k_p=0.01, k_i=0, k_d=0.28$, 方波响应如图 4-31 所示。

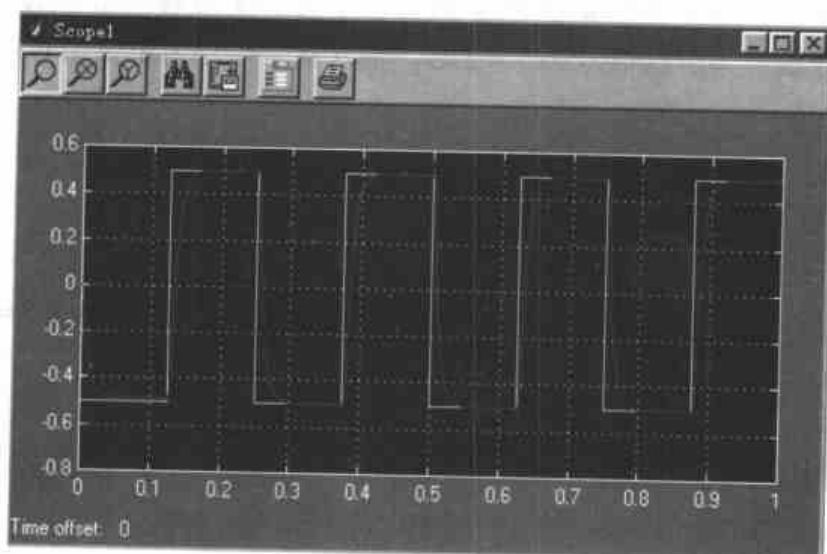


图 4-31 方波响应

仿真程序:

仿真程序 1: Simulink 程序 chap4_7.mdl, 如图 4-32 所示。

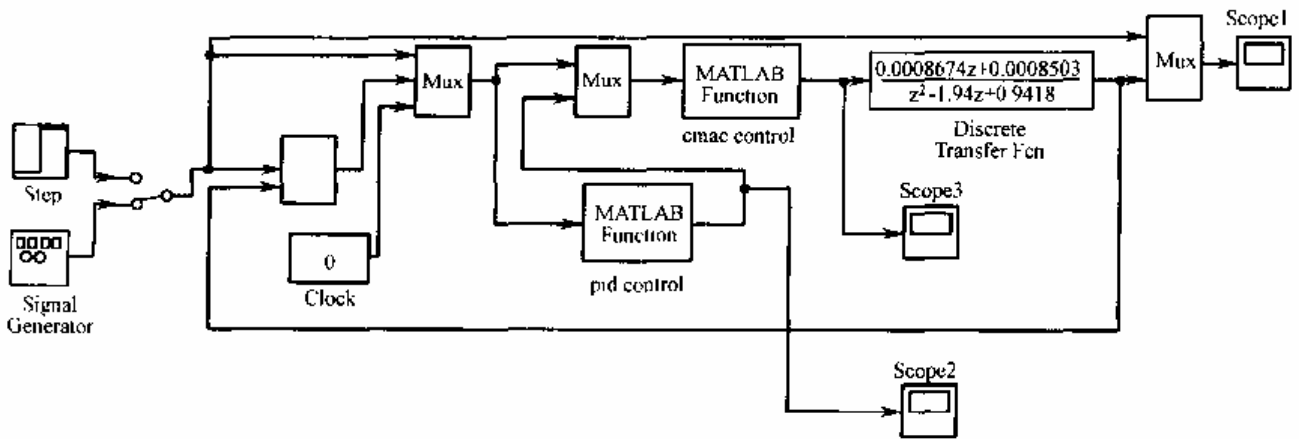


图 4-32 Simulink 控制仿真结构图

仿真程序 2: 用于 PID 控制的 M 函数 chap4_7f1.m

```
function [u]=chap4_7f1(u1,u2,u3)
global s
persistent error1 error_1

ts=0.001;
if u3~=0
    error1=0;
    error_1=0;
end

s=3           %Selecting Signal Type
if s==1      %Step Signal
    kp=0.4;
    ki=0.0;
    kd=0.28;
elseif s==2  %Sine Signal
    kp=40;
    ki=0.0;
    kd=0.28;
elseif s==3  %Square Wave Signal
    kp=0;
    ki=0.0;
    kd=0.28;
end
error=u2;
error1d=(error-error_1)/ts;
```

```

errori=errori+error*ts;

u=kp*error+kd*errord+ki*errori;
error_1=error;

```

仿真程序 3: 用于 CMAC 控制的 M 函数 chap4_7f2.m

```

function [u]=chap4_7f2(u1,u2,u3,u4)
global s
persistent w x1 x2 x3 w_1 w_2 w_3

N=300;
C=5;

if u3==0
    w=zeros(N+C,1);
    w_1=w;
    w_2=w;
    d_w=w;
end

alfa=0.04;

if s==1    %Step Signal
    Smin=0;
    Smax=1;
elseif s==2    %Sine Signal
    Smin=-0.5;
    Smax=0.5;
elseif s==3    %Square Wave Signal
    Smin= 0.5;
    Smax=0.5;
end

xite=0.1;

dvi=(Smax-Smin)/(N-1);

for i=1:1:C    %C size
    v(i)=Smin;
end

```

```

for i=C+1:1:C+N      %N size
    v(i)=v(i-1)+dvi;
end
for i=N+C+1:1:N+2*C  %C size
    v(i)=Smax;
end

rin=u1;
for i=1:1:N+C
if rin>=v(i)&rin<=v(i+C)
    a(i)=1;
else
    a(i)=0;
end
end

errork=u2;

un=a*w_1;
up=u4;
u=up+un;           %Total control output

if u>=10
    u=10;
end
if u<=-10
    u=-10;
end
if s==1|s==3
    d_w=a'*xite*up/C;
elseif s==2
    d_w=xite*up/C;   %For Sine Signal
end

w-w_1+d_w+alfa*(w_1-w_2);
w_3=w_2;
w_2=w_1;
w_1=w;

```

第5章 基于遗传算法整定的PID控制

5.1 遗传算法的基本原理

遗传算法简称 GA (Genetic Algorithms) 是 1962 年由美国 Michigan 大学的 Holland 教授提出的模拟自然界遗传机制和生物进化论而成的一种并行随机搜索最优化方法。它将“优胜劣汰，适者生存”的生物进化原理引入优化参数形成的编码串联群体中，按所选择的适配值函数并通过遗传中的复制、交叉及变异对个体进行筛选，使适配值高的个体被保留下来，组成新的群体，新的群体既继承了上一代的信息，又优于上一代。这样周而复始，群体中个体适应度不断提高，直到满足一定的条件。其算法简单，可并行处理，能得到全局最优解。

遗传算法的主要特点：

- (1) 遗传算法是对参数的编码进行操作，而非对参数本身；
- (2) 遗传算法是从许多点开始并行操作，而非局限于一点；
- (3) 遗传算法通过目标函数来计算适配值，而不需要其他推导，从而对问题的依赖性较小；
- (4) 遗传算法的寻优规则是由概率决定的，而非确定性的；
- (5) 遗传算法在解空间进行高效启发式搜索，而非盲目地穷举或完全随机搜索；
- (6) 遗传算法对于待寻优的函数基本无限制，它既不要求函数连续，也不要求函数可微，既可以是数学解析式所表示的显函数，又可以是映射矩阵甚至是神经网络的隐函数，因而应用范围较广；
- (7) 遗传算法具有并行计算的特点，因而可通过大规模并行计算来提高计算速度；
- (8) 遗传算法更适合大规模复杂问题的优化；
- (9) 遗传算法计算简单，功能强。

遗传算法的基本操作为：

(1) 复制 (Reproduction Operator)

复制是从一个旧种群中选择生命力强的个体位串产生新种群的过程。根据位串的适配值拷贝，也就是指具有高适配值的位串更有可能在下一代中产生一个或多个子孙。它模仿了自然现象，应用了达尔文的适者生存理论。复制操作可以通过随机方法来实现。若用计算机程序来实现，可考虑首先产生 0~1 之间均匀分布的随机数，若某串的复制概率为 40%，则当产生的随机数在 0~0.40 之间时，该串被复制，否则被淘汰。此外，还可以通过计算方法实现，其中较典型的几种方法为适应度比例法、期望值法、排位次法等。适应度比例法较常用。

(2) 交叉 (Crossover Operator)

复制操作能从旧种群中选择出优秀者，但不能创造新的染色体。而交叉模拟了生物进化过程中的繁殖现象，通过两个染色体的交换组合，来产生新的优良品种。它的过程为：在匹配池中任选两个染色体，随机选择一点或多点交换点位置；交换双亲染色体交换点右边的部分，即可得到两个新的染色体数字串。交换体现了自然界中信息交换的思想。交叉有一点

交叉、多点交叉、还有一致交叉、顺序交叉和周期交叉。一点交叉是最基本的方法，应用较广。它是指染色体切断点有一处，例：

A:101100 1110 → 101100 0101

B:001010 0101 → 001010 1110

(3) 变异(Mutation Operator)

变异运算用来模拟生物在自然的遗传环境中由于各种偶然因素引起的基因突变，它以很小的概率随机地改变遗传基因（表示染色体的符号串的某一位）的值。在染色体以二进制编码的系统中，它随机地将染色体的某一个基因由 1 变为 0，或由 0 变为 1。若只有选择和交叉，而没有变异，则无法在初始基因组合以外的空间进行搜索，使进化过程在早期就陷入局部解而进入终止过程，从而影响解的质量。为了在尽可能大的空间中获得质量较高的优化解，必须采用变异操作。

5.2 遗传算法的优化设计

5.2.1 遗传算法的构成要素

(1) 染色体编码方法：基本遗传算法使用固定长度的二进制符号来表示群体中的个体，其等位基因是由二值符号集 {0,1} 所组成。初始个体的基因值可用均匀分布的随机值来生成，如， $x = 100111001000101101$ 就可表示一个个体，该个体的染色体长度是 $n = 18$ 。

(2) 个体适应度评价：基本遗传算法与个体适应度成正比的概率来决定当前群体中每个个体遗传到下一代群体中的概率多少。为正确计算这个概率，要求所有个体的适应度必须为正数或零。因此，必须先确定由目标函数值到个体适应度之间的转换规则。

(3) 遗传算子：基本遗传算法使用下述三种遗传算子：

- ① 选择运算使用比例选择算子；
- ② 交叉运算使用单点交叉算子；
- ③ 变异运算使用基本位变异算子或均匀变异算子。

(4) 基本遗传算法的运行参数：有下述 4 个运行参数需要提前设定：

M ：群体大小，即群体中所含个体的数量，一般取为 20 ~ 100；

G ：遗传算法的终止进化代数，一般取为 100 ~ 500；

P_c ：交叉概率，一般取为 0.4 ~ 0.99；

P_m ：变异概率，一般取为 0.0001 ~ 0.1。

5.2.2 遗传算法的应用步骤

对于一个需要进行优化的实际问题，一般可按下述步骤构造遗传算法：

第一步：确定决策变量及各种约束条件，即确定出个体的表现型 X 和问题的解空间；

第二步：建立优化模型，即确定出目标函数的类型及数学描述形式或量化方法；

第三步：确定表示可行解的染色体编码方法，即确定出个体的基因型 x 及遗传算法的搜索空间；

第四步：确定解码方法，即确定出由个体基因型 x 到个体表现型 X 的对应关系或转换方法；

第五步：确定个体适应度的量化评价方法，即确定出由目标函数值 $f(x)$ 到个体适应度 $F(x)$ 的转换规则；

第六步：设计遗传算子，即确定选择运算、交叉运算、变异运算等遗传算子的具体操作方法。

第七步：确定遗传算法的有关运行参数，即 M, G, P_c, P_m 等参数。

5.3 遗传算法求函数极大值

5.3.1 遗传算法求函数极大值实例

利用遗传算法求 Rosenbrock 函数的极大值

$$\begin{cases} f_2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \\ -2.048 \leq x_i \leq 2.048 \quad (i=1,2) \end{cases}$$

该函数有两个局部极大点，分别是 $f(2.048, -2.048) = 3897.7342$ 和 $f(-2.048, -2.048) = 3905.9262$ 其中后者为全局最大点。

求解该问题遗传算法的构造过程：

- (1) 确定决策变量和约束条件；
- (2) 建立优化模型；

(3) 确定编码方法：用长度为 10 位的二进制编码串来分别表示两个决策变量 x_1, x_2 。10 位二进制编码串可以表示从 0 到 1023 之间的 1024 个不同的数，故将 x_1, x_2 的定义域离散化为 1023 个均等的区域，包括两个端点在内共有 1024 个不同的离散点。从离散点 -2.048 到离散点 2.048，依次让它们分别对应于从 0000000000(0) 到 1111111111(1023) 之间的二进制编码。再将分别表示 x_1, x_2 的两个 10 位长的二进制编码串连接在一起，组成一个 20 位长的二进制编码串，它就构成了这个函数优化问题的染色体编码方法。使用这种编码方法，解空间和遗传算法的搜索空间就具有一一对应的关系。例如： $x: 0000110111 1101110001$ 就表示一个个体的基因型，其中前 10 位表示 x_1 ，后 10 位表示 x_2 。

(4) 确定解码方法：解码时需要将 20 位长的二进制编码串切断为两个 10 位长的二进制编码串，然后分别将它们转换为对应的十进制整数代码，分别记为 y_1 和 y_2 。依据个体编码方法和对定义域的离散化方法可知，将代码 y_i 转换为变量 x_i 的解码公式为

$$x_i = 4.096 \times \frac{y_i}{1023} - 2.048 \quad (i=1,2)$$

例如，对个体 $x: 0000110111 1101110001$ ，它由两个代码所组成

$$y_1 = 55, y_2 = 881$$

上述两个代码经过解码后，可得到两个实际的值

$$x_1 = -1.828, x_2 = 1.476$$

(5) 确定个体评价方法：由于 Rosenbrock 函数的值域总是非负的，并且优化目标是求函数的最大值，故可将个体的适应度直接取为对应的目标函数值，即

$$F(x) = f(x_1, x_2)$$

选个体适应度的倒数作为目标函数

$$J(x) = \frac{1}{F(x)}$$

(6) 设计遗传算子：选择运算使用比例选择算子，交叉运算使用单点交叉算子，变异运算使用基本位变异算子。

(7) 确定遗传算法的运行参数：群体大小 $M = 80$ ，终止进化代数 $G = 100$ ，交叉概率 $P_c = 0.60$ ，变异概率 $P_m = 0.10$ 。

上述七个步骤构成了用于求 Rosenbrock 函数极大值的优化计算基本遗传算法。

采用上述方法进行仿真，经过 100 步迭代，最佳样本为

$$\text{BestS} = [0000000000000000000000]$$

即当 $x_1 = -2.0480, x_2 = -2.0480$ 时，Rosenbrock 函数具有极大值，极大值为 3905.9。

遗传算法的优化过程中目标函数 J 和适应度函数 F 的变化过程如图 5-1 和 5-2 所示，由仿真结果可知，随着进化过程的进行，群体中适应度较低的一些个体被逐渐淘汰掉，而适应度较高的一些个体会越来越多，并且它们都集中在所求问题的最优点附近，从而搜索到问题的最优解。

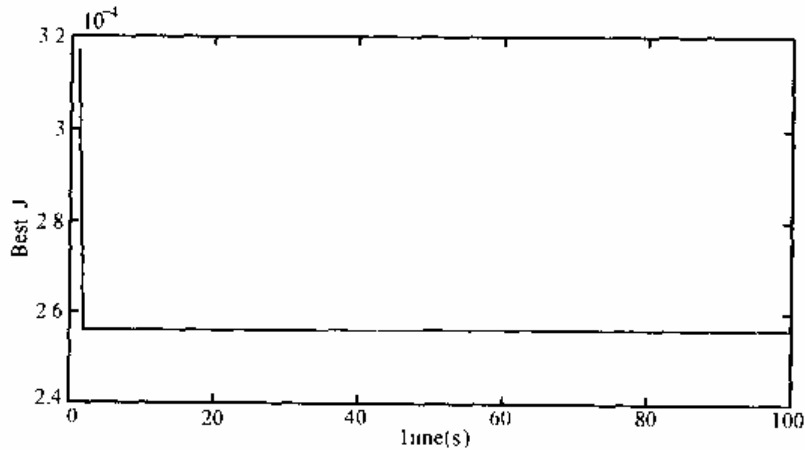


图 5-1 目标函数 J 的优化过程

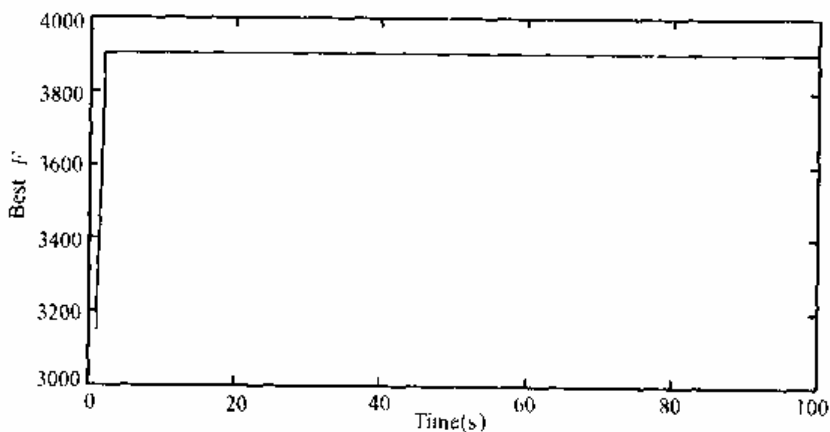


图 5-2 适应度 F 的优化过程

5.3.2 仿真程序

仿真程序: chap5_1.m

```

%Generic Algorithm for function f(x1,x2) optimun
clear all;
close all;

%Parameters
Size=80;
G=100;
CodeL=10;

umax=2.048;
umin=-2.048;

E=round(rand(Size,2*CodeL)); %Initial Code

%Main Program
for k=1:1:G
time(k)=k;

for s=1:1:Size
m=E(s,:);
y1=0;y2=0;

%Uncoding
m1=m(1:1:CodeL);
for i=1:1:CodeL,
    y1=y1+m1(i)*2^(i-1);
end
x1=(umax-umin)*y1/1023+umin;
m2=m(CodeL+1:1:2*CodeL);
for i=1:1:CodeL
    y2=y2+m2(i)*2^(i-1);
end
x2=(umax-umin)*y2/1023+umin;

F(s)=100*(x1^2-x2)^2+(1-x1)^2;
end

Ji=1./F;
%***** Step 1 : Evaluate BestJ *****
BestJ(k)=min(Ji);

```

```

fi=F; %Fitness Function
[Oderfi,Indexfi]=sort(fi); %Arranging fi small to bigger
Bestfi=Oderfi(Size); %Let Bestfi=max(fi)
BestS=E(Indexfi(Size),:); %Let BestS=E(m), m is the Indexfi belong to
max(fi)
bfi(k)=Bestfi;

%***** Step 2 : Select and Reproduct Operation*****
fi_sum=sum(fi);
fi_Size=(Oderfi/fi_sum)*Size;

fi_S=floor(fi_Size); %Selecting Bigger fi value

kk=1;
for i=1:1:Size
    for j=1:1:fi_S(i) %Select and Reproduce
        TempE(kk,:)=E(Indexfi(i),:);
        kk=kk+1; %kk is used to reproduce
    end
end

%***** Step 3 : Crossover Operation *****
pc=0.60;
n=ceil(20*rand);
for i=1:2:(Size-1)
    temp=rand;
    if pc>temp %Crossover Condition
        for j=n:1:20
            TempE(i,j)=E(i+1,j);
            TempE(i+1,j)=E(i,j);
        end
    end
end

TempE(Size,:)=BestS;
E=TempE;

%***** Step 4: Mutation Operation *****
%pm=0.001;
%pm=0.001-[1:1:Size]*(0.001)/Size; %Bigger fi, smaller Pm

```

```

%pm=0.0;    %No mutation
pm=0.1;    %Big mutation

for i=1:1:Size
    for j=1:1:2*CodeL
        temp=rand;
        if pm>temp                %Mutation Condition
            if TempE(i,j)==0
                TempE(i,j)=1;
            else
                TempE(i,j)=0;
            end
        end
    end
end

%Guarantee TempPop(30,:) is the code belong to the best individual(max(fi))
TempE(Size,:)=BestS;
E=TempE;
end

Max_Value=Bestfi
BestS
x1
x2
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(time,bfi);
xlabel('times');ylabel('Best F');

```

5.4 基于遗传算法的 PID 整定

PID 控制是工业过程控制中应用最广的策略之一，因此 PID 控制器参数的优化成为人们关注的问题，它直接影响控制效果的好坏，并和系统的安全、经济运行有着密不可分的关系。目前 PID 参数的优化方法有很多，如间接寻优法，梯度法，爬山法等，而在热工系统中单纯形法、专家整定法则应用较广。虽然这些方法都具有良好的寻优特性，但却存在着一些弊端，单纯形法对初值比较敏感，容易陷入局部最优化解，造成寻优失败。专家整定法则需要太多的经验，不同的目标函数对应不同的经验，而整理知识库则是一项长时间的工程。因此

我们选取了遗传算法来进行参数寻优，该方法是一种不需要任何初始信息并可以寻求全局最优解的、高效的优化组合方法。

采用遗传算法进行 PID 三个系数的整定，具有以下优点：

(1) 与单纯形法相比，遗传算法同样具有良好的寻优特性，且它克服了单纯形法参数初值的敏感性。在初始条件选择不当的情况下，遗传算法在不需给出调节器初始参数的情况下，仍能寻找到合适的参数，使控制目标满足要求。同时单纯形法难以解决多值函数问题以及在多参数寻优（如串级系统）中，容易造成寻优失败或时间过长，而遗传算法的特性决定了它能很好地克服以上问题。

(2) 与专家整定法相比，它具有操作方便、速度快的优点，不需要复杂的规则，只通过字串进行简单地复制、交叉、变异，便可达到寻优。避免了专家整定法中前期大量的知识库整理工作及大量的仿真实验。

(3) 遗传算法是从许多点开始并行操作，在解空间进行高效启发式搜索，克服了从单点出发的弊端以及搜索的盲目性，从而使寻优速度更快，避免了过早陷入局部最优解。

(4) 遗传算法不仅适用于单目标寻优，而且也适用于多目标寻优。根据不同的控制系统，针对一个或多个目标，遗传算法均能在规定的范围内寻找到合适参数。

遗传算法作为一种全局优化算法，得到了越来越广泛的应用。近年来，遗传算法在控制上的应用日益增多。

5.4.1 基于遗传算法的 PID 整定原理

(1) 参数的确定及表示

首先确定参数范围，该范围一般是由用户给定，然后由精度的要求，对其进行编码。选取二进制字符串来表示每一个参数，并建立与参数间的关系。再把二进制串连接起来就组成一个长的二进制字符串，该字符串为遗传算法可以操作的对象。

(2) 选取初始种群

因为需要编程来实现各过程，所以采用计算机随机产生初始种群。针对二进制编码而言，先产生 0~1 之间均分布的随机数，然后规定产生的随机数 0~0.5 之间代表 0, 0.5~1 之间代表 1。此外，考虑到计算的复杂程度来规定种群的大小。

(3) 适配函数的确定

一般的寻优方法在约束条件下可以求得满足条件的一组参数，在设计中是从该组参数中寻找一个最好的。衡量一个控制系统的指标有三个方面的，即稳定性、准确性和快速性。而上升时间反映了系统的快速性，上升时间越短，控制进行得就越快，系统品质也就越好。

如果单纯追求系统的动态特性，得到的参数很可能使控制信号过大，在实际应用中会因系统中固有的饱和特性而导致系统不稳定，为了防止控制能量过大，在目标函数中加入控制量。因此为了使控制效果更好，我们给出了控制量、误差和上升时间作为约束条件。因为适应函数同目标函数相关，所以目标函数确定后，直接将其作为适配函数进行参数寻优。最优的控制参数也就是在满足约束条件下使 $f(x)$ 最大时， x 所对应的控制器参数。

(4) 遗传算法的操作

首先利用适应度比例法进行复制。即通过适配函数求得适配值，进而求每个串对应的复制概率。复制概率与每代字符串的个数的乘积为该串在下一代中应复制的个数。复制概率大

的在下一代中将有较多的子孙，相反则会被淘汰。

其次进行单点交叉，交叉概率为 P_c 。从复制后的成员里以 P_c 的概率选取字符串组成匹配池，而后对匹配池的成员随机匹配，交叉的位置也是随机确定的。

最后以概率 P_m 进行变异。假如每代有 15 个字符串，每个字符串 12 位，则共有 $15 \times 12 = 180$ 个串位，期望的变异串位数为 $180 \times 0.01 = 2$ (位)，即每代中有两个串位要由 1 变为 0 或由 0 变为 1。

初始种群通过复制、交叉及变异得到了新一代种群，该代种群经解码后代入适配函数，观察是否满足结束条件，若不满足，则重复以上操作直到满足为止。

结束条件由具体问题所定，只要各目标参数在规定范围内，则终止计算。

以上操作过程可以用图 5-3 来表示。

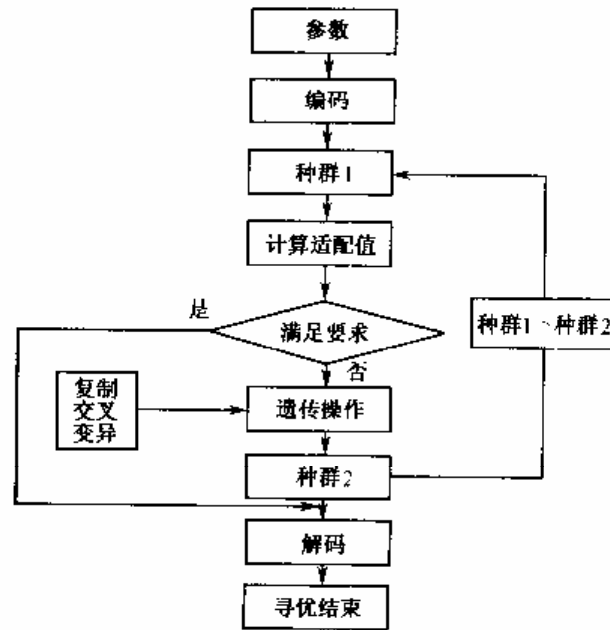


图 5-3 遗传算法流程图

利用遗传算法优化 k_p, k_i, k_d 的具体步骤如下：

- (1) 确定每个参数的大致范围和编码长度，进行编码；
- (2) 随机产生 n 个个体构成初始种群 $P(0)$ ；
- (3) 将种群中各个体解码成对应的参数值，用此参数求代价函数值 J 及适应函数值 f ，取 $f = \frac{1}{J}$ ；
- (4) 应用复制、交叉和变异算子对种群 $P(t)$ 进行操作，产生下一代种群 $P(t+1)$ ；
- (5) 重复步骤 (3) 和 (4)，直至参数收敛或达到预定的指标。

5.4.2 基于实数编码遗传算法的 PID 整定

被控对象为二阶传递函数

$$G(s) = \frac{400}{s^2 + 50s}$$

采样时间为 1ms，输入指令为一阶跃信号。

为获取满意的过渡过程动态特性，采用误差绝对值时间积分性能指标作为参数选择的最小目标函数。为了防止控制能量过大，在目标函数中加入控制输入的平方项。选用下式作为参数选取的最优指标

$$J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t)) dt + w_3 \cdot t_u$$

式中， $e(t)$ 为系统误差， $u(t)$ 为控制器输出， t_u 为上升时间， w_1, w_2, w_3 为权值。

为了避免超调，采用了惩罚功能，即一旦产生超调，将超调量作为最优指标的一项，此时最优指标为

$$\text{if } e(t) < 0 \quad J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t) + w_4 |e(t)|) dt + w_3 \cdot t_u$$

式中， w_3 为权值，且 $w_4 \gg w_1$ 。

遗传算法中使用的样本个数为 30，交叉概率和变异概率分别为： $P_c = 0.9, P_m = 0.033$ 。参数 k_p 的取值范围为 $[0, 20]$ ， k_i, k_d 的取值范围为 $[0, 1]$ ，取 $w_1 = 0.999, w_2 = 0.001, w_4 = 100, w_3 = 2.0$ 。采用实数编码方式，经过 100 代进化，获得的优化参数如下：PID 整定结果为 $k_p = 19.0823, k_d = 0.2434, k_i = 0.0089$ ，性能指标 $J = 23.9936$ 。代价函数 J 的优化过程和采用整定后的 PID 控制阶跃响应如图 5-4 和图 5-5 所示。

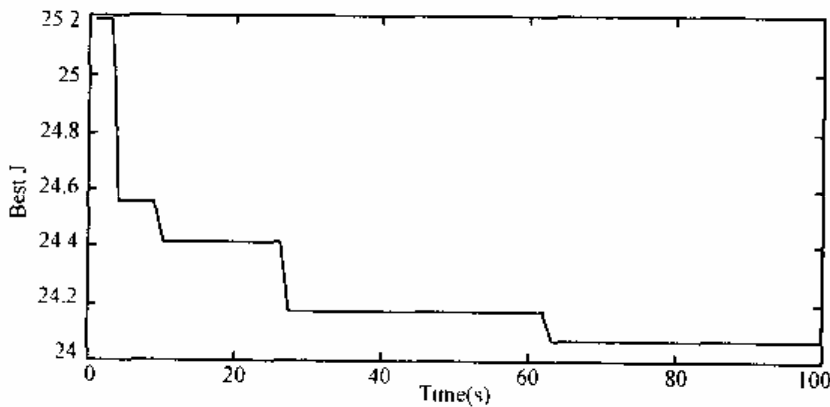


图 5-4 代价函数值 J 的优化过程

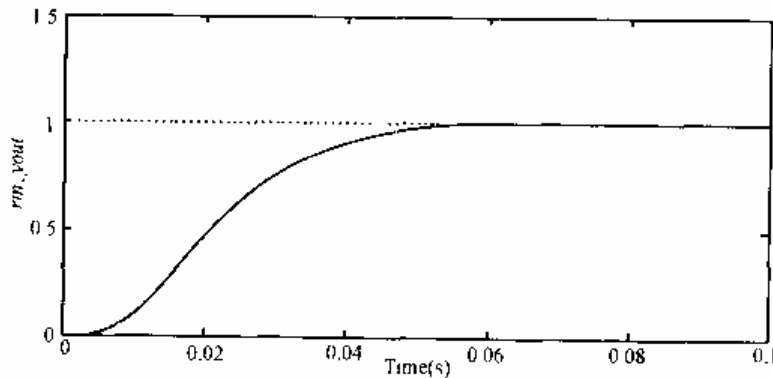


图 5-5 整定后的 PID 阶跃响应

在应用遗传算法时，为了避免参数选取范围过大，可以先按经验选取一组参数，然后再在这组参数的周围利用遗传算法进行设计，从而大大减少初始寻优的盲目性，节约计算量。

5.4.3 仿真程序

仿真程序:

主程序: chap5_2.m

```
%GA(Generic Algorithm) Program to optimize PTD Parameters
clear all;
close all;
global rin yout timef

Size=30;
Code1=3;

MinX(1)=zeros(1);
MaxX(1)=20*ones(1);
MinX(2)=zeros(1);
MaxX(2)=1.0*ones(1);
MinX(3)=zeros(1);
MaxX(3)=1.0*ones(1);

Kpid(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
Kpid(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);
Kpid(:,3)=MinX(3)+(MaxX(3)-MinX(3))*rand(Size,1);

G=100;
BsJ=0;

%***** Start Running *****
for kg=1:1:G
    time(kg)=kg;

%***** Step 1 : Evaluate BestJ *****
for i=1:1:Size
    Kpidi=Kpid(i,:);

    [Kpidi,BsJ]=chap5_2f(Kpidi,BsJ);

    BsJi(i)=BsJ;
end
```



```

[OderJi, IndexJi]=sort(BsJi);
BestJ(kg)=OderJi(1);
BJ=BestJ(kg);
Ji=BsJi+1e-10;    %Avoiding deviding zero

    fi=1./Ji;
% Cm=max(Ji);
% fi=Cm-Ji;

    [Oderfi, Indexfi]=sort(fi);    %Arranging fi small to bigger
    Bestfi=Oderfi(Size);    %Let Bestfi=max(fi)
    BestS=Kpid(Indexfi(Size),:); %Let BestS=E(m), m is the Indexfi belong to
max(fi)

    kg
    BJ
    BestS
%***** Step 2 : Select and Reproduct Operation*****
    fi_sum=sum(fi);
    fi_Size=(Oderfi/fi_sum)*Size;

    fi_S=floor(fi_Size);    % Selecting Bigger fi value
    r=Size-sum(fi_S);

    Rest=fi_Size-fi_S;
    [RestValue, Index]=sort(Rest);

    for i=Size:-1:Size-r+1
        fi_S(Index(i))=fi_S(Index(i))+1;    % Adding rest to equal Size
    end

    k=1;
    for i=Size:-1:1    % Select the Sizeth and Reproduce firstly
        for j=1:1:fi_S(i)
            TempE(k,:)=Kpid(Indexfi(i),:);    % Select and Reproduce
            k=k+1;    % k is used to reproduce
        end
    end

%***** Step 3 : Crossover Operation *****

```

```

Pc=0.90;
for i=1:2:(Size 1)
    temp=rand;
    if Pc>temp                %Crossover Condition
        alfa=rand;
        TempE(1,:)=alfa*Kpid(i+1,:)+(1-alfa)*Kpid(i,:);
        TempE(i+1,:)=alfa*Kpid(i,:)+(1-alfa)*Kpid(i+1,:);
    end
end
TempE(Size,:)=BestS;
Kpid=TempE;

%***** Step 4: Mutation Operation *****
Pm=0.10-[1:1:Size]*(0.01)/Size;    %Bigger fi,smaller Pm
Pm_rand=rand(Size,CodeL);
Mean=(MaxX + MinX)/2;
Dif=(MaxX-MinX);

for i=1:1:Size
    for j=1:1:CodeL
        if Pm(i)>Pm_rand(i,j)    %Mutation Condition
            TempE(i,j)=Mean(j)+Dif(j)*(rand-0.5);
        end
    end
end
%Guarantee TempE(Size,:) belong to the best individual
TempE(Size,:)=BestS;
Kpid=TempE;
end
Bestfi
BestLS
Best_J=BestJ(G)
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,rin,'r',timef,yout,'b');
xlabel('Time(s)');ylabel('rin,yout');

```

程序: chap5_2f.m

```

function [Kpidi,BsJ]=pid_gaf(Kpidi,BsJ)
global rin yout timef

ts=0.001;
sys=tf(400,[1,50,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

rin=1.0;
u_1=0.0;u_2=0.0;
y_1=0.0;y_2=0.0;
x=[0,0,0]';
B=0;
error_1=0;
tu=1;
s=0;
P=100;

for k=1:1:P
    timef(k)=k*ts;
    r(k)=rin;

    u(k)=Kpidi(1)*x(1)+Kpidi(2)*x(2)+Kpidi(3)*x(3);

    if u(k)>=10
        u(k)=10;
    end
    if u(k)<=-10
        u(k)=-10;
    end

    yout(k)=den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
    error(k)=r(k)-yout(k);
%----- Return of PID parameters -----
    u_2=u_1;u_1=u(k);
    y_2=y_1;y_1=yout(k);

    x(1)=error(k);           % Calculating P
    x(2)=(error(k)-error_1)/ts; % Calculating D
    x(3)=x(3)+error(k)*ts;   % Calculating I
end

```

```

    error_2=error_1;
    error_1=error(k);
if s==0
    if yout(k)>0.95&yout(k)<1.05
        tu=timef(k);
        s=1;
    end
end
end

for i=1:1:P
    J1(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
    B=B+J1(i);
    if i>1
        erry(i)-yout(i)-yout(i-1);
        if erry(i)<0
            B=B+100*abs(erry(i));
        end
    end
end
BsJ=B+0.2*tu*10;

```

5.4.4 基于二进制编码遗传算法的 PID 整定

被控对象为二阶传递函数

$$G(s) = \frac{400}{s^2 + 50s}$$

采样时间为 1ms，输入指令为阶跃信号。

采用二进制编码方式，用长度为 10 位的二进制编码串来分别表示三个决策变量 k_p, k_i, k_d 。最优指标的选取同十进制编码遗传算法的 PID 整定。遗传算法中使用的样本个数为 $\text{Size} = 30$ ，交叉概率和变异概率分别为： $P_c = 0.60, P_m = 0.001 - [1:1:\text{Size}] \times 0.001/\text{Size}$ 。

参数 k_p 的取值范围为 $[0, 20]$ ， k_i, k_d 的取值范围为 $[0, 1]$ ， w_1, w_2, w_3, w_4 的取值同十进制编码遗传算法的 PID 整定。经过 100 代进化，获得的优化参数如下：

最优个体为 $\text{BestS} = [01011011111011000100010000100]$ 。PID 优化参数为： $k_p = 16.1290, k_d = 0.2209, k_i = 0.2209$ ，性能指标 $J = 24.9812$ ，整定过程中代价函数 J 的变化如图 5-6 所示。采用整定后的二进制遗传算法优化 PID 阶跃响应如图 5-7 所示。

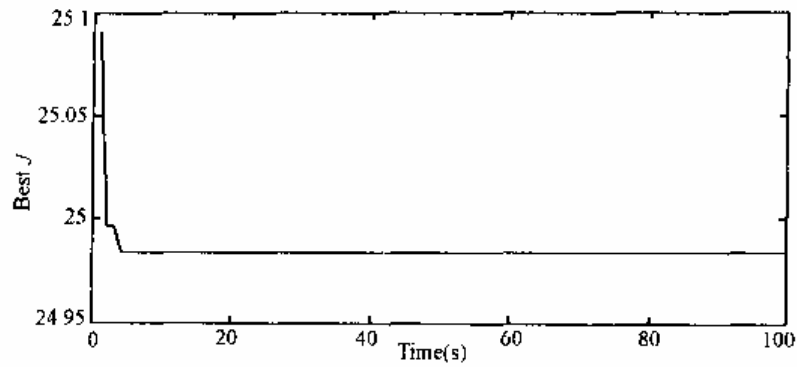


图 5-6 代价函数值 J 的优化过程

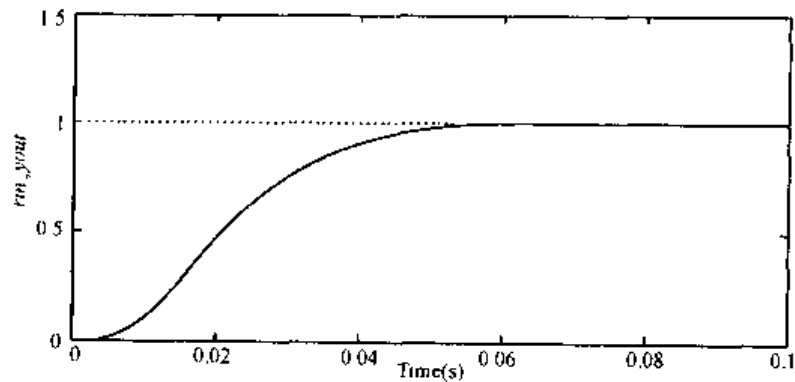


图 5-7 二进制遗传算法优化 PID 阶跃响应

5.4.5 仿真程序

仿真程序:

主程序: chap5_3.m

```
%GA(Generic Algorithm) Program to optimize Parameters of PID
clear all;
close all;
global rin yout timef

G=100;
Size=30;
CodeL=10;

MinX(1)=zeros(1);
MaxX(1)=20*ones(1);
MinX(2)=zeros(1);
MaxX(2)=1.0*ones(1);
MinX(3)=zeros(1);
MaxX(3)=1.0*ones(1);
```

```

E=round(rand(Size,3*CodeL)); %Initial Code!

BsJ=0;

for kg=1:1:G
    Lime(kg)=kg;

    for s=1:1:Size
        m=E(s,:);
        y1=0;y2=0;y3=0;

        m1=m(1:1:CodeL);
        for i=1:1:CodeL
            y1=y1+m1(i)*2^(i-1);
        end
        Kpid(s,1)=(MaxX(1)-MinX(1))*y1/1023+MinX(1);

        m2=m(CodeL+1:1:2*CodeL);
        for i=1:1:CodeL
            y2=y2+m2(i)*2^(i-1);
        end
        Kpid(s,2)=(MaxX(2)-MinX(2))*y2/1023+MinX(2);

        m3=m(CodeL+1:1:2*CodeL);
        for i=1:1:CodeL
            y3=y3+m3(i)*2^(i-1);
        end
        Kpid(s,3)=(MaxX(3)-MinX(3))*y3/1023+MinX(3);

        %***** Step 1 : Evaluate BestJ *****
        Kpidi=Kpid(s,:);

        [Kpidi,BsJ]=chap5_2f(Kpidi,BsJ);

        BsJi(s)=BsJ;
    end

    [OderJi,IndexJi]=sort(BsJi);
    BestJ(kg)=OderJi(1);
    BJ=BestJ(kg);

```

```

Ji=BsJi+1e-10;

    fi=1./Ji;
% Cm=max(Ji);
% fi=Cm-Ji;      %Avoiding deviding zero

    [Oderfi,Indexfi]=sort(fi);      %Arranging fi small to bigger
% Bestfi=Oderfi(Size);      %Let Bestfi=max(fi)
% BestS=Kpid(Indexfi(Size),:); %Let BestS=E(m), m is the Indexfi belong
to max(fi)

Bestfi=Oderfi(Size);      % Let Bestfi=max(fi)
BestS=E(Indexfi(Size),:); % Let BestS=E(m), m is the Indexfi belong to max(fi)

kg
BJ
BestS;

%***** Step 2 : Select and Reproduct Operation*****
    fi_sum=sum(fi);
    fi_Size=(Oderfi/fi_sum)*Size;

    fi_S=floor(fi_Size);      %Selecting Bigger fi value

    kk=1;
    for i=1:1:Size
        for j=1:1:fi_S(i)      %Select and Reproduce
            TempE(kk,:)=E(Indexfi(i),:);
            kk=kk+1;          %kk is used to reproduce
        end
    end

%***** Step 3 : Crossover Operation *****
pc=0.60;
n=ceil(20*rand);
for i=1:2:(Size-1)
    temp=rand;
    if pc>temp      %Crossover Condition
        for j=n:1:20
            TempE(i,j)=E(i+1,j);
        end
    end
end

```

```

        TempE(i+1,j)=E(i,j);
    end
end
end
TempE(Size,:)=BestS;
E=TempE;

%***** Step 4: Mutation Operation *****
%pm=0.001;
pm=0.001-[1:1:Size]*(0.001)/Size; %Bigger fi, smaller pm
%pm=0.0; %No mutation
%pm=0.1; %Big mutation

for i=1:1:Size
    for j=1:1:2*CodeL
        temp=rand;
        if pm>temp %Mutation Condition
            if TempE(i,j)==0
                TempE(i,j)=1;
            else
                TempE(i,j)=0;
            end
        end
    end
end
end
%Guarantee TempE(Size,:) belong to the best individual
TempE(Size,:)=BestS;
E=TempE;
%*****

end

Bestfi
BestS
Kpidi
Best_J=BestJ(G)
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,rin,'r',timef,yout,'b');

```



```
xlabel('Time(s)');ylabel('rin,yout');
```

子程序: chap5_2f.m (同前)

5.5 基于遗传算法摩擦模型参数辨识的 PID 控制

5.5.1 仿真实例

被控对象为二阶传递函数

$$G(s) = \frac{400}{s^2 + 50s}$$

设外加在控制器输出上的干扰为一等效摩擦

当 $F = 1$ 时为库仑摩擦, 摩擦模型为

$$F_f(t) = 0.8 \operatorname{sgn}(\dot{\theta}(t))$$

当 $F = 2$ 时为库仑摩擦+粘性摩擦, 摩擦模型为

$$F_f(t) = \operatorname{sgn}(\dot{\theta}(t))(0.30|\dot{\theta}(t)| + 1.50)$$

采样时间为 1ms, 取 $S = 2$, 使输入指令为阶跃信号。

为获取满意的过渡过程动态特性, 采用误差绝对值时间积分性能指标作为参数选择的最小目标函数。为了防止控制能量过大, 在目标函数中加入控制输入的平方项。选用下式作为参数选取的最优指标

$$J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t)) dt$$

式中, $e(t)$ 为系统误差, w_1 和 w_2 为权值。

为了避免超调, 采用了惩罚功能, 即一旦产生超调, 将超调量作为最优指标的一项, 此时最优指标为

$$\text{if } e(t) < 0 \quad J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t) + w_3 |e(t)|) dt$$

式中, w_3 为权值, 且 $w_3 \gg w_1$ 。

在应用遗传算法时, 为了避免参数选取范围过大, 可以先按经验选取一组参数, 然后再在这组参数的周围利用遗传算法进行设计, 从而大大减少初始寻优的盲目性, 节约计算量。

采用实数编码方式, 遗传算法中使用的样本个数为 30, 交叉概率和变异概率分别为 $P_c = 0.9$, $P_m = 0.10 - [1:1:\text{Size}] \times 0.01/\text{Size}$, $w_1 = 0.999$, $w_2 = 0.001$, $w_3 = 10$ 。

通过取 $kx = [0,0]$ 使摩擦补偿 $F_{fc} = 0$, 得到无摩擦补偿情况下阶跃响应如图 5-8 所示。

取 $F = 2$, 运行程序 chap5_4.m, 采用遗传算法对摩擦模型进行辨识。待辨识参数采用实数编码法, 取 $kx = [0.3, 1.5]$, 辨识参数 kx_1 和 kx_2 的范围选为 $[0, 2.0]$, 取进化代数为 50。经过优化获得的最优样本和最优指标为: $\text{BestS} = [0.3523, 1.2257]$, $\text{Best}_J = 25.5439$ 。摩擦参数辨识结果为 $kx_1 = 0.3523, kx_2 = 1.2257$, 采用摩擦补偿后的 PID 控制阶跃响应如图 5-9 所示, 代价函数值 J 的优化过程如图 5-10 所示。

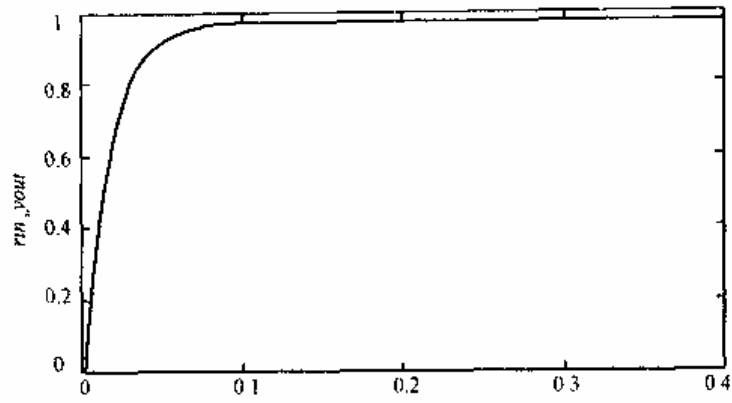


图 5-8 无摩擦补偿的阶跃响应

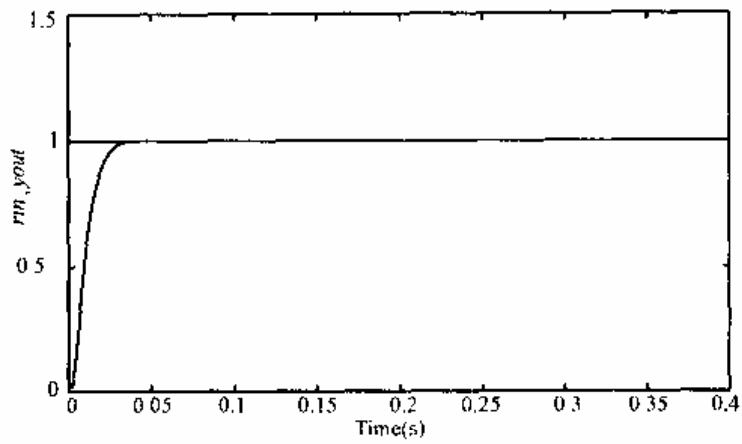


图 5-9 采用摩擦补偿的阶跃响应

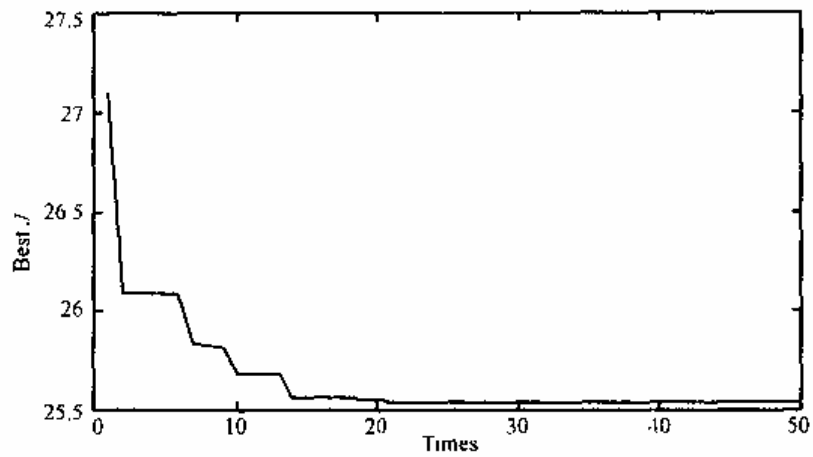


图 5-10 代价函数值 J 的优化过程

5.5.2 仿真程序

仿真程序: 主程序为遗传算法程序, 子程序为带有摩擦模型的 PID 控制程序。

主程序: chap5_4.m

```

%GA(Generic Algorithm) Program to predict Parameters of Friction
clear all;
close all;
global rin yout timeF F

Size=30;
F=2;
if F==1
    CodeL=1;
    MinX=zeros(CodeL,1);
    MaxX=1.0*ones(CodeL,1);
end
if F==2
    CodeL=2;
    MinX=zeros(CodeL,1);
    MaxX=2.0*ones(CodeL,1);
end

for i=1:1:CodeL
    kxi(:,i)=MinX(i)+(MaxX(i)-MinX(i))*rand(Size,1);
end

G=50;
BsJ=0;

for kg=1:1:G
    time(kg)=kg;

    %***** Step 1:Evaluate BestJ *****
    for i=1:1:Size
        kx=kxi(i,:);

        [kx,BsJ]=chap5_4f(kx,BsJ);

        BsJi(i)=BsJ;
    end

    [OderJi, IndexJi]=sort(BsJi);
    BestJ(kg)=OderJi(1);
    BJ=BestJ(kg);

```

```

J1=BsJ1+1e-10;

    fi=1./Ji;
% Cm=max(Ji);
% fi=Cm-Ji;           %Avoiding deviding zero

    [Oderfi,Indexfi]=sort(fi);   %Arranging fi small to bigger
    Bestfi=Oderfi(Size);        %Let Bestfi=max(fi)
%Let BestS=E(m), m is the Indexfi belong to max(fi)
    BestS=kxi(Indexfi(Size),:);

kg
BJ
BestS
kx
%***** Step 2 : Select and Reproduct Operation*****
    fi_sum=sum(fi);
    fi_Size=(Oderfi/fi_sum)*Size;

    fi_S=floor(fi_Size);        %Selecting Bigger fi value
    r=Size-sum(fi_S);

    Rest=fi_Size-fi_S;
    [RestValue,Index]=sort(Rest);

    for i=Size:-1:Size-r+1
        fi_S(Index(i))=fi_S(Index(i))+1; %Adding rest to equal Size
    end

    k=1;
    for i=Size:-1:1            %Select the Sizeth and Reproduce first!
        for j=1:1:fi_S(i)      %Notice: If i=1:1:Size then k plus meaningless
            TempE(k,:)=kxi(Indexfi(i),:); %Select and Reproduce
            k=k+1;             %k is used to reproduce
        end
    end

%***** Step 3 : Crossover Operation *****
    Pc=0.90;
    for i=1:2:(Size-1)

```

```

        temp=rand;
    if Pc>temp                %Crossover Condition
        alfa=rand;
        TempE(i,:)=alfa*kxi(i+1,:)+(1-alfa)*kxi(j,:);
        TempE(i+1,:)=alfa*kxi(i,:)+(1-alfa)*kxi(i+1,:);
    end
end
TempE(Size,:)=BestS;
kxi=TempE;

%***** Step 4: Mutation Operation *****
Pm=0.10-[1:1:Size]*(0.01)/Size;        %Bigger fi, smaller Pm
Pm_rand=rand(Size,CodeL);
Mean=(MaxX + MinX)/2;
Dif=(MaxX-MinX);

    for i=1:1:Size
        for j=1:1:CodeL
            if Pm(i)>Pm_rand(i,j)        %Mutation Condition
                TempE(i,j)=Mean(j)+Dif(j)*(rand-0.5);
            end
        end
    end

%Guarantee TempE(Size,:) belong to the best individual
TempE(Size,:)=BestS;
kxi=TempE;
%*****
end
Bestfi
BestS
Best_J=BestJ(G)

figure(1);
plot(timef,rin,'b',timef,yout,'r');
xlabel('Time(s)');ylabel('rin,yout');
figure(2);
plot(time,BestJ,'r');
xlabel('Times');ylabel('Best J');

```

子程序: chap5_4f.m

```

function [kx,BsJ]=pid_fm_gaf(kx,BsJ)
global rin yout timef F

a=50;b=400;
ts=0.001;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;
y_1=0;y_2=0;
e_1=0;
B=0;

G=400;
for k=1:1:G
    timef(k)=k*ts;
    S=2;
    if S==1
        fre=5;
        AA=0.5;
        rin(k)=AA*sin(2*pi*fre*k*ts);
    end
    if S==2
        rin(k)=1;
    end

    yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
    error(k)=rin(k)-yout(k);
    derror(k)=(error(k)-e_1)/ts;

    u(k)=50*error(k)+0.50*derror(k);

    speed(k)=(yout(k)-y_1)/ts;

    if F==1 % Disturbance Signal: Coulomb Friction
        Ff(k)=0.8*sign(speed(k));
    end
    if F==2 % Disturbance Signal: Coulomb & Viscous Friction
        Ff(k)=sign(speed(k))*(0.30*abs(speed(k))+1.50);
    end
end

```

```

end

%kx=[0,0];      %No GA Identification
%kx=[0.3,1.5]; %Idea Identification

u(k)=u(k)-Ff(k);

if F==1
    Ffc(k)=kx*sign(speed(k)); %Friction Estimation
end
if F==2 %Friction Estimation
    Ffc(k)=sign(speed(k))*(kx(1)*abs(speed(k))+kx(2));
end

u(k)=u(k)+Ffc(k);

if u(k)>110
    u(k)=110;
end
if u(k)<-110
    u(k)=-110;
end

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
e_1=error(k);
end
for i=1:1:G
    Ji(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
    B=B+Ji(i);
    if error(i)<0 %Punishment
        B=B+10*abs(error(i));
    end
end
BsJ=B;

```

第 6 章 先进 PID 多变量解耦控制

6.1 PID 多变量解耦控制

6.1.1 PID 解耦控制原理

通过 PID 控制, 可实现对多变量系统的解耦控制, 图 6-1 给出了一个二变量 PID 解耦控制系统框图, 该系统由两个 PID 控制器构成, 控制算法为

$$\begin{aligned}
 u_1(k) &= k_{p1}error_1(k) + k_{d1} \frac{error_1(k) - error_1(k-1)}{T} + k_{i1} \sum_{i=1}^k error_1(i)T \\
 u_2(k) &= k_{p2}error_2(k) + k_{d2} \frac{error_2(k) - error_2(k-1)}{T} + k_{i2} \sum_{i=1}^k error_2(i)T
 \end{aligned}
 \tag{6.1}$$

式中, T 为采样时间, $error_1(k) = r_1(k) - y_1(k)$, $error_2(k) = r_2(k) - y_2(k)$ 。

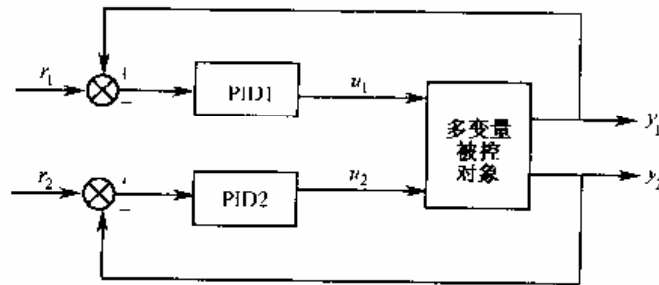


图 6-1 二变量 PID 解耦控制

6.1.2 仿真程序及分析

仿真实例

设有耦合二变量耦合被控对象

$$\begin{aligned}
 y_1(k) &= 1.0/(1 + y_1(k-1))^2(0.8y_1(k-1) + u_1(k-2) + 0.2u_2(k-3)) \\
 y_2(k) &= 1.0/(1 + y_2(k-1))^2(0.9y_2(k-1) + 0.3u_1(k-3) + u_2(k-2))
 \end{aligned}$$

设采样时间 $T = 1s$ 。给定输入为单位阶跃输入, 即

$$\begin{aligned}
 \mathbf{R}_1 &= \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
 \mathbf{R}_2 &= \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
 \end{aligned}$$

当输入指令为 \mathbf{R}_1 时的解耦响应曲线如图 6-2 所示, 当输入指令为 \mathbf{R}_2 时的解耦响应曲线如图 6-3 所示。

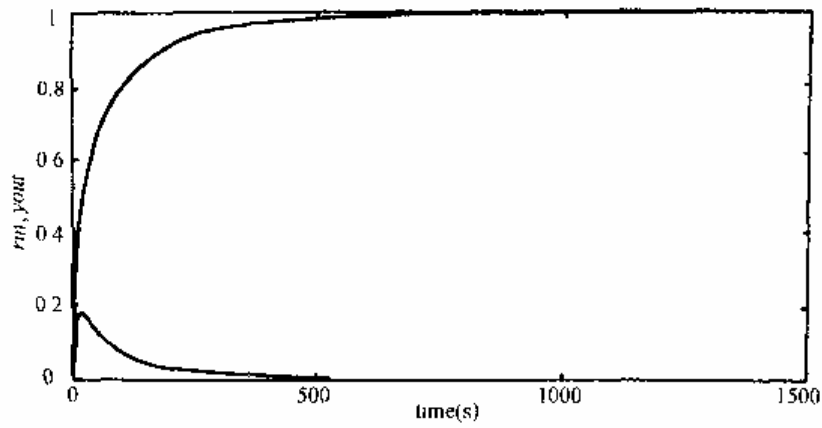


图 6-2 解耦响应曲线($R=[1;0]$)

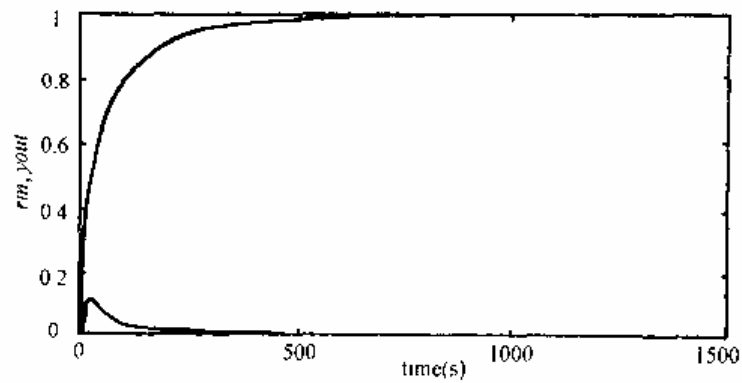


图 6-3 解耦响应曲线($R=[0;1]$)

仿真程序: chap6_1.m

```
%PID Controller for coupling plant
clear all;
close all;

u1_1=0.0;u1_2=0.0;u1_3=0.0;u1_4=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;u2_4=0.0;

y1_1=0;y2_1=0;

x1=[0;0];x2=[0;0];x3=[0;0];

kp=0.020;
ki=0.050;
kd=0.0001;

error_1=[0;0];
ts=1;
for k=1:1:1500
```

```

time(k)=k*ts;

%Step Signal
R=[1;0];
%R=[0;1];

%PID Decouple Controller
u1(k)=kp*x1(1)+kd*x2(1)+ki*x3(1);
u2(k)=-kp*x1(2)+kd*x2(2)+ki*x3(2);
u=[u1(k),u2(k)];

if u1(k)>-10
    u1(k)=10;
end
if u2(k)>=10
    u2(k)=10;
end
if u1(k)<=-10
    u1(k)=-10;
end
if u2(k)<=-10
    u2(k)=-10;
end

%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

error1(k)=R(1)-yout1(k);
error2(k)=R(2)-yout2(k);
error=[error1(k);error2(k)];

%----- ----Return of PID parameters -----%
u1_4=u1_3;u1_3=u1_2;u1_2=u1_1;u1_1=u(1);

u2_4=u2_3;u2_3=u2_2;u2_2=u2_1;u2_1=u(2);

y1_1=yout1(k);y2_1=yout2(k);

x1=error;          %Calculating P

```

```

x2=(error-error_1)/ts; %Calculating D
x3=x3+error*ts; %Calculating I

error_1=error;
end
hold on;
plot(time,R(1),'b',time,yout1,'r');
plot(time,R(2),'r',time,yout2,'b');
xlabel('time(s)');ylabel('rin,yout');

```

6.2 单神经元 PID 解耦控制

6.2.1 单神经元 PID 解耦控制原理

通过单神经元 PID 控制, 可较好地实现对多变量系统的解耦控制, 图 6-4 给出了二变量单神经元 PID 解耦控制系统框图, 该系统由两个单神经元 PID 控制器构成。

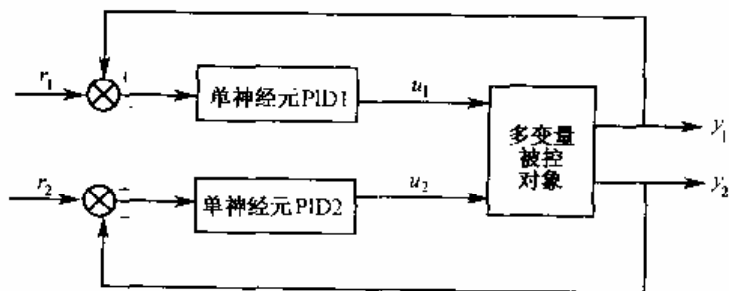


图 6-4 二变量单神经元 PID 解耦控制

单神经元自适应控制器是通过调整加权系数来实现自适应、自组织功能, 权系数的调整是按有监督的 Hebb 学习规则实现的。以第一个单神经元 PID 控制器为例, 控制算法及学习算法为

$$u_1(k) = u_1(k-1) + k_1 \sum_{i=1}^3 w_i'(k) x_i(k) \quad (6.2)$$

$$w_i'(k) = w_i(k) / \sum_{i=1}^3 |w_i(k)|$$

$$\begin{aligned} w_1(k) &= w_1(k-1) + \eta_I z(k) u(k) x_1(k) \\ w_2(k) &= w_2(k-1) + \eta_P z(k) u(k) x_2(k) \\ w_3(k) &= w_3(k-1) + \eta_D z(k) u(k) x_3(k) \end{aligned} \quad (6.3)$$

式中, $x_1(k) = e(k)$;

$$x_2(k) = e(k) - e(k-1);$$

$$x_3(k) = \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2);$$

η_I, η_P, η_D 分别为积分、比例、微分的学习速率, k_1 为神经元的比例系数, $k_1 > 0$ 。

对积分、比例和微分分别采用了不同的学习速率 η_I, η_P, η_D , 以便对不同的权系数分别

进行调整。

k_1 值的选择非常重要。 k_1 越大, 则快速性越好, 但超调量大, 甚至可能使系统不稳定。当被控对象时延增大时, k_1 值必须减少, 以保证系统稳定。 k_1 值选择过小, 会使系统的快速性变差。

6.2.2 仿真程序及分析

仿真实例

仍以 PID 解耦控制中二变量耦合被控对象为例

$$y_1(k) = 1.0 / (1 + y_1(k-1))^2 (0.8y_1(k-1) + u_1(k-2) + 0.2u_2(k-3))$$

$$y_2(k) = 1.0 / (1 + y_2(k-1))^2 (0.9y_2(k-1) + 0.3u_1(k-3) + u_2(k-2))$$

设采样时间 $T = 1s$ 。给定输入为单位阶跃输入, 即

$$R_1 = \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

采用单神经元 PID 解耦控制, 取 $k_1 = k_2 = 0.16$ 。 k_2 为第二个神经元的比例系数。当输入指令为 R_1 时的解耦响应曲线如图 6-5 所示, 当输入指令为 R_2 时的解耦响应曲线如图 6-6 所示。

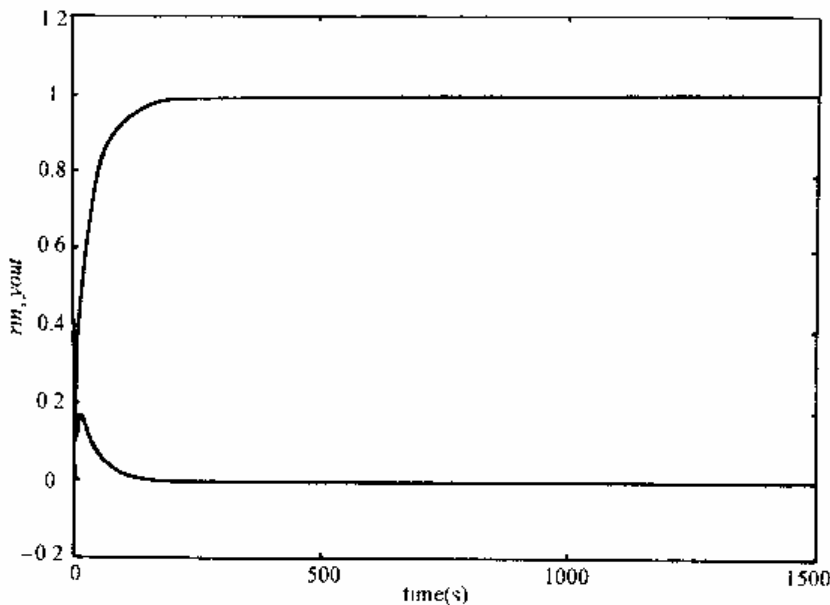


图 6-5 解耦响应曲线 ($R = [1; 0]$)

与 PID 解耦控制相比, 单神经元 PID 解耦控制具有响应速度快, 自适应能力强, 抗干扰能力强等优点。

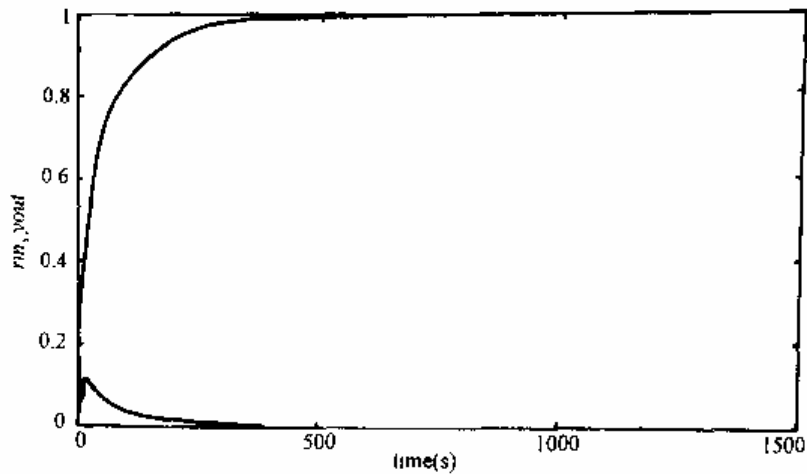


图 6-6 解耦响应曲线 ($R = [0;1]$)

仿真程序: chap6_2.m

```
%Single Neural Net PID Decouple Controller based on Hebb Learning
%Algorithm to adjust kp,ki,kd
clear all;
close all;

xc1=[0,0,0]';
xc2=[0,0,0]';

xiteP=0.40;
xiteI=0.40;
xiteD=0.40;

%Initilizing kp,ki and kd
%Radom Value
%wkp1_1=rand;wki1_1=rand;wkd1_1=rand;
%wkp2_1=rand;wki2_1=rand;wkd2_1=rand;

wkp1_1=0.3150;wki1_1=1.1615;wkd1_1=1.4948;
wkp2_1=0.2067;wki2_1=0.6365;wkd2_1=0.4996;

error1_1=0;error1_2=0;
error2_1=0;error2_2=0;

u1_1=0.0;u1_2=0.0;u1_3=0.0;u1_4=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;u2_4=0.0;

y1_1=0;y2_1=0;
```

```

ts=1;
for k=1:1:1500
time(k)=k*ts;

%Step Signal
%R=[1;0];
R=[0;1];

%----- Calculating practical output -----%
%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

error1(k)=R(1)-yout1(k);
error2(k)=R(2)-yout2(k);

%For Variable1
%Adjusting NNC Weight Value by adopting hebb learning algorithm
wkp1(k)=wkp1_1+xiteP*error1(k)*u1_1*xc1(1); %P
wki1(k)=wki1_1+xiteI*error1(k)*u1_1*xc1(2); %I
wkd1(k)=wkd1_1+xiteD*error1(k)*u1_1*xc1(3); %D
xc1(1)=error1(k)-error1_1; %P
xc1(2)=error1(k); %I
xc1(3)=(error1(k)-2*error1_1+error1_2); %D

wadd1(k)=abs(wkp1(k))+abs(wki1(k))+abs(wkd1(k));
w111(k)=wkp1(k)/wadd1(k);
w122(k)=wki1(k)/wadd1(k);
w133(k)=wkd1(k)/wadd1(k);
w1=[w111(k),w122(k),w133(k)];
k1=0.16;
u1(k)=u1_1+k1*w1*xc1;

%For Variable2
%Adjusting NNC Weight Value by adopting hebb learning algorithm
wkp2(k)=wkp2_1+xiteP*error2(k)*u2_1*xc2(1); %P
wki2(k)=wki2_1+xiteI*error2(k)*u2_1*xc2(2); %I
wkd2(k)=wkd2_1+xiteD*error2(k)*u2_1*xc2(3); %D
xc2(1)=error2(k)-error2_1; %P

```

```

xc2(2)=error2(k); %I
xc2(3)=(error2(k)-2*error2_1+error2_2); %D

wadd2(k)=abs(wkp2(k))+abs(wki2(k))+abs(wkd2(k));
w211(k)=wkp2(k)/wadd2(k);
w222(k)=wki2(k)/wadd2(k);
w233(k)=wkd2(k)/wadd2(k);
w2=[w211(k),w222(k),w233(k)];
k2=0.16;
u2(k)=u2_1+k2*w1*xc2;

%-- -----Return of PID parameters-----%
%For Variable1
error1_2=error1_1;
error1_1=error1(k);
wkp1_1=wkp1(k);
wkd1_1=wkd1(k);
wki1_1=wki1(k);

u1_4=u1_3;
u1_3=u1_2;
u1_2=u1_1;
u1_1=u1(k);

y1_1=yout1(k);

%For Variable2
error2_2=error2_1;
error2_1=error2(k);
wkp2_1=wkp2(k);
wkd2_1=wkd2(k);
wki2_1=wki2(k);

u2_4=u2_3;
u2_3=u2_2;
u2_2=u2_1;
u2_1=u2(k);

y2_1=yout2(k);
end

```

```

hold on;
plot(time,R(1),'b',time,yout1,'r');
plot(time,R(2),'r',time,yout2,'b');
xlabel('time(s)');ylabel('rnn,yout');

```

6.3 基于 DRNN 神经网络整定的 PID 解耦控制

6.3.1 基于 DRNN 神经网络参数自学习 PID 解耦控制原理

双输入、双输出多变量自整定 PID 控制器如图 6-7 所示, 其中 NN1 和 NN2 为神经网络, 用于控制器 u_1 和 u_2 的 PID 参数为 k_p, k_i, k_d 。 r_1 和 r_2 为系统输入指令, y_1 和 y_2 为系统输出值。

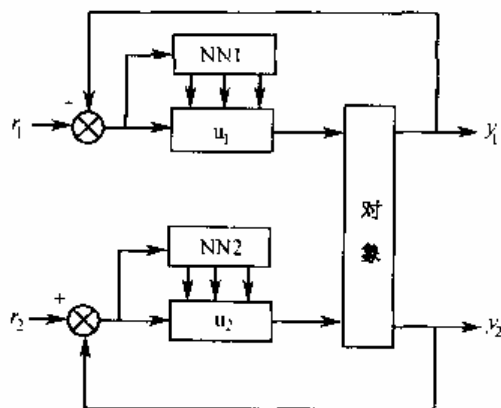


图 6-7 多变量自整定 PID 控制器

以控制器 u_1 为例, 控制算法如下

$$u_1(k) = k_{p1}(k)x_1(k) + k_{i1}(k)x_2(k) + k_{d1}(k)x_3(k) \quad (6.4)$$

$$error_1(k) = r_1(k) - y_1(k), \quad \text{且}$$

$$x_1(k) = error_1(k)$$

$$x_2(k) = \sum_{i=1}^k (error_1(i) \times T) \quad (6.5)$$

$$x_3(k) = \frac{error_1(k) - error_1(k-1)}{T}$$

式中, T 为采样时间。PID 三项系数 $k_{p1}(k), k_{i1}(k), k_{d1}(k)$ 采用 DRNN 神经网络进行整定。

定义如下的指标

$$E_1(k) = \frac{1}{2}(r_1(k) - y_1(k))^2 \quad (6.6)$$

$$\begin{aligned}
 k_{p1}(k) &= k_{p1}(k-1) - \eta_p \frac{\partial E_1}{\partial k_{p1}} = k_{p1}(k-1) + \eta_p (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} \frac{\partial u_1}{\partial k_{p1}} \\
 &= k_{p1}(k-1) + \eta_p (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} x_1(k)
 \end{aligned} \quad (6.7)$$

$$\begin{aligned}
 k_{i1}(k) &= k_{i1}(k-1) - \eta_i \frac{\partial E_1}{\partial k_{i1}} = k_{i1}(k-1) + \eta_i (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} \frac{\partial u_1}{\partial k_{i1}} \\
 &= k_{i1}(k-1) + \eta_i (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} x_2(k)
 \end{aligned} \tag{6.8}$$

$$\begin{aligned}
 k_{d1}(k) &= k_{d1}(k-1) - \eta_d \frac{\partial E_1}{\partial k_{d1}} = k_{d1}(k-1) + \eta_d (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} \frac{\partial u_1}{\partial k_{d1}} \\
 &= k_{d1}(k-1) + \eta_d (r_1(k) - y_1(k)) \frac{\partial y_1}{\partial u_1} x_3(k)
 \end{aligned} \tag{6.9}$$

式中, $\frac{\partial y_1}{\partial u_1}$ 为对象的 Jacobian 信息, 该信息可以由 DRNN 网络进行辨识。

6.3.2 DRNN 神经网络的 Jacobian 信息辨识

DRNN (Diagonal Recurrent Neural Network) 神经网络是一种回归神经网络, 网络结构共有三层, 隐层为回归层。DRNN 神经网络的结构如图 6-8 所示。

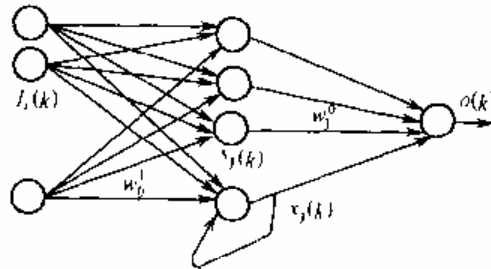


图 6-8 DRNN 神经网络结构

在 DRNN 神经网络中, 设 $I = [I_1, I_2, \dots, I_n]$ 为网络输入向量, $I_i(k)$ 为输入层第 i 个神经元的输入, 网络回归层第 j 个神经元的输出为 $X_j(k)$, $S_j(k)$ 为第 j 个回归神经元输入总和, $f(\cdot)$ 为 S 函数, $O(k)$ 为 DRNN 网络的输出。

DRNN 神经网络的算法为

$$O(k) = \sum_j W_j^0 X_j(k), \quad X_j(k) = f(S_j(k)), \quad S_j(k) = W_j^D X_j(k-1) + \sum_i W_{ij}^1 I_i(k) \tag{6.10}$$

式中, W^D 和 W^0 为网络回归层和输出层的权值向量, W^1 为网络输入层的权值向量。

图 6-9 中, k 为网络的迭代步骤, $u(k)$ 和 $y(k)$ 为辨识器的输入。DRNN 为网络辨识器。 $y(k)$ 为被控对象实际输出, $ym(k)$ 为 DRNN 的输出。将系统输出 $y(k)$ 及输入 $u(k)$ 的值作为辨识器 DRNN 的输入, 将系统输出 $y(k)$ 与网络输出的误差作为辨识器的调整信号。

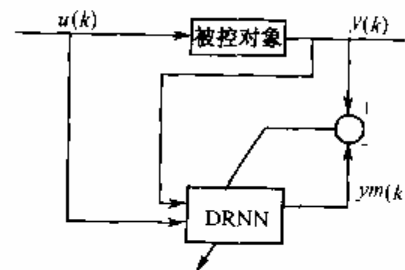


图 6-9 DRNN 神经网络辨识

网络输出层的输出为

$$ym(k) = O(k) = \sum_j w_j^0 X_j(k) \tag{6.11}$$

网络回归层的输出为

$$X_j(k) = f(S_j(k)) \tag{6.12}$$

网络回归层的输入为

$$S_j(k) = w_j^o X_j(k-1) + \sum_i (w_{ij}^1 I_i(k)) \quad (6.13)$$

辨识误差为

$$em(k) = y(k) - ym(k) \quad (6.14)$$

辨识指标取

$$Em(k) = \frac{1}{2} em(k)^2 \quad (6.15)$$

学习算法采用梯度下降法

$$\Delta w_j^o(k) = -\frac{\partial Em(k)}{\partial w_j^o} = em(k) \frac{\partial ym}{\partial w_j^o} = em(k) X_j(k) \quad (6.16)$$

$$w_j^o(k) = w_j^o(k-1) + \eta_o \Delta w_j^o(k) + \alpha (w_j^o(k-1) - w_j^o(k-2)) \quad (6.17)$$

$$\Delta w_{ij}^1(k) = -\frac{\partial Em(k)}{\partial w_{ij}^1} = em(k) \frac{\partial ym}{\partial w_{ij}^1} = em(k) \frac{\partial ym}{\partial X_j} \frac{\partial X_j}{\partial w_{ij}^1} = em(k) w_j^o Q_{ij}(k) \quad (6.18)$$

$$w_{ij}^1(k) = w_{ij}^1(k-1) + \eta_1 \Delta w_{ij}^1(k) + \alpha (w_{ij}^1(k-1) - w_{ij}^1(k-2)) \quad (6.19)$$

$$\Delta w_j^D(k) = -\frac{\partial E(k)}{\partial w_j^D} = em(k) \frac{\partial ym}{\partial X_j} \frac{\partial X_j}{\partial w_j^D} = em(k) w_j^o P_j(k) \quad (6.20)$$

$$w_j^D(k) = w_j^D(k-1) + \eta_D \Delta w_j^D(k) + \alpha (w_j^D(k-1) - w_j^D(k-2)) \quad (6.21)$$

其中回归层神经元取双 S 函数为

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \quad (6.22)$$

$$P_j(k) = \frac{\partial X_j}{\partial w_j^D} = f'(S_j) X_j(k-1) \quad (6.23)$$

$$Q_{ij}(k) = \frac{\partial X_j}{\partial w_{ij}^1} = f'(S_j) I_i(k) \quad (6.24)$$

式中, η_1 、 η_D 、 η_o 分别为输入层、回归层和输出层的学习速率, α 为惯性系数。

对象的 Jacobian 信息 $\frac{\partial y}{\partial u}$ 为

$$\frac{\partial y}{\partial u} = \frac{\partial ym}{\partial u} = \sum_j w_j^o f'(S_j) w_{ij}^1 \quad (6.25)$$

6.3.3 仿真程序及分析

仿真实例

仍以 PID 解耦控制中二变量耦合被控对象为例

$$y_1(k) = 1.0 / (1 + y_1(k-1))^2 (0.8 y_1(k-1) + u_1(k-2) + 0.2 u_2(k-3))$$

$$y_2(k) = 1.0 / (1 + y_2(k-1))^2 (0.9 y_2(k-1) + 0.3 u_1(k-3) + u_2(k-2))$$

设采样时间为 1s。给定输入为单位阶跃输入, 即

$$R_1 = \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{R}_2 = \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

采用基于 DRNN 神经网络整定的 PID 解耦控制, 网络结构取 3-7-1, 网络输入为 $I = \{u(k-1), y(k), 1.0\}$, $\eta_o = 0.40$, $\eta_D = 0.40$, $\eta_I = 0.40$, $\alpha = 0.04$, 权值取 $[-1, +1]$ 范围内的随机值。

当输入指令为 \mathbf{R}_1 时的 Jacobian 信息及解耦响应结果如图 6-10、6-11 所示, k_{p1}, k_{i1}, k_{d1} 和 k_{p2}, k_{i2}, k_{d2} 整定结果如图 6-12 和 6-13 所示。当输入指令为 \mathbf{R}_2 时的 Jacobian 信息及阶跃响应结果如图 6-14 和 6-15 所示, k_{p1}, k_{i1}, k_{d1} 和 k_{p2}, k_{i2}, k_{d2} 整定结果如图 6-16 和 6-17 所示。

与 PID 解耦控制相比, 基于 DRNN 神经网络整定的 PID 解耦控制具有响应速度快, 自适应能力强, 抗干扰能力强等优点。

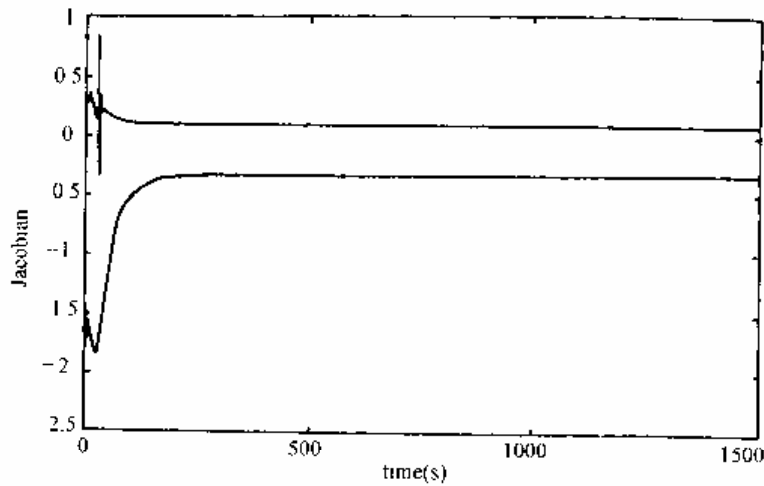


图 6-10 Jacobian 信息 ($\mathbf{R} = [1;0]$)

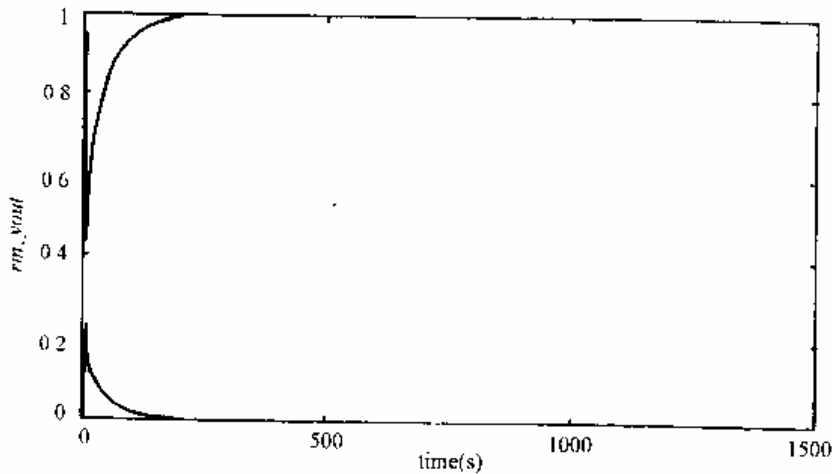


图 6-11 DRNN 解耦响应 ($\mathbf{R} = [1;0]$)

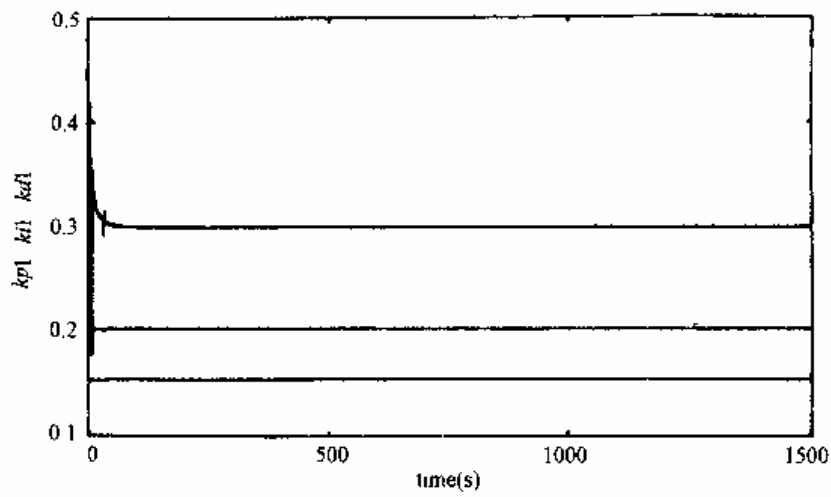


图 6-12 k_{p1}, k_{i1}, k_{d1} 的整定结果

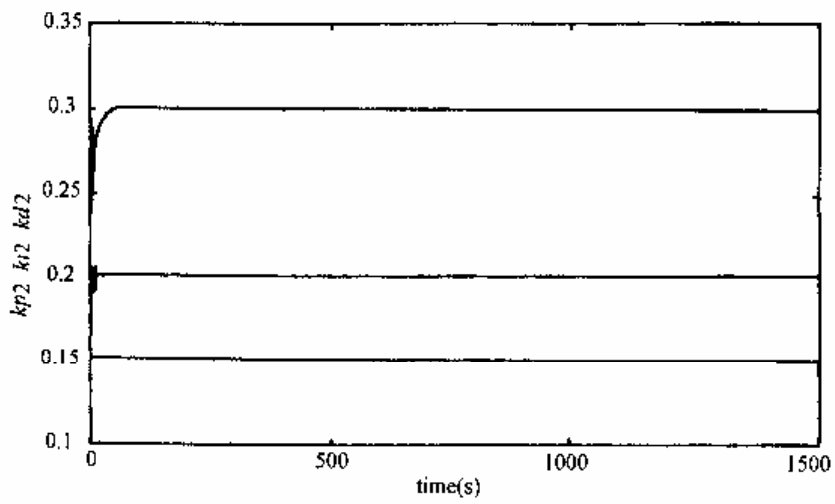


图 6-13 k_{p2}, k_{i2}, k_{d2} 的整定结果

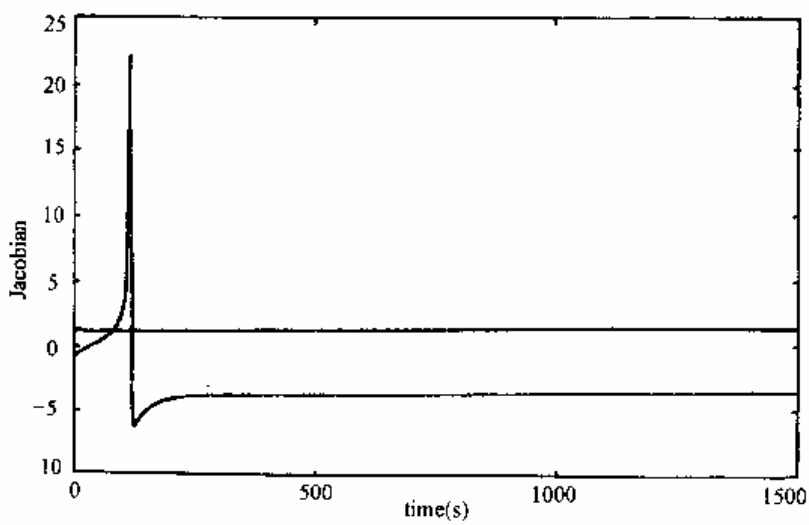


图 6-14 Jacobian 信息 ($R = [0;1]$)

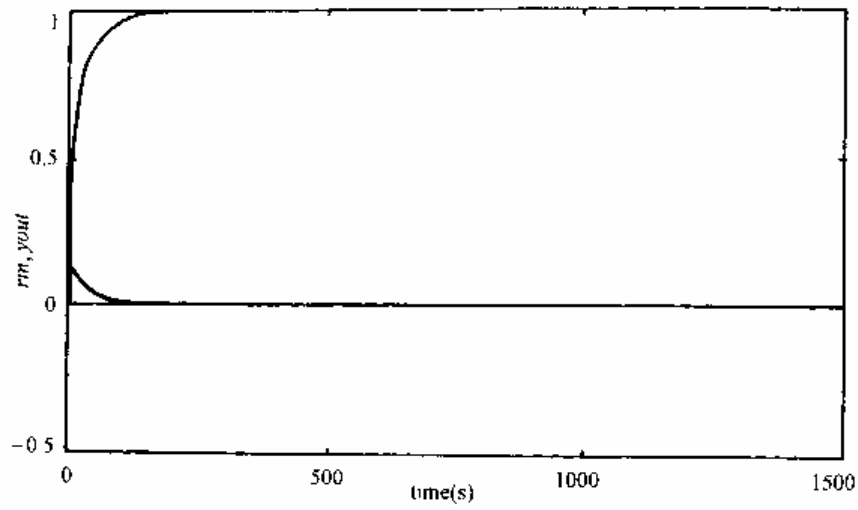


图 6-15 DRNN 解耦响应($R=[0;1]$)

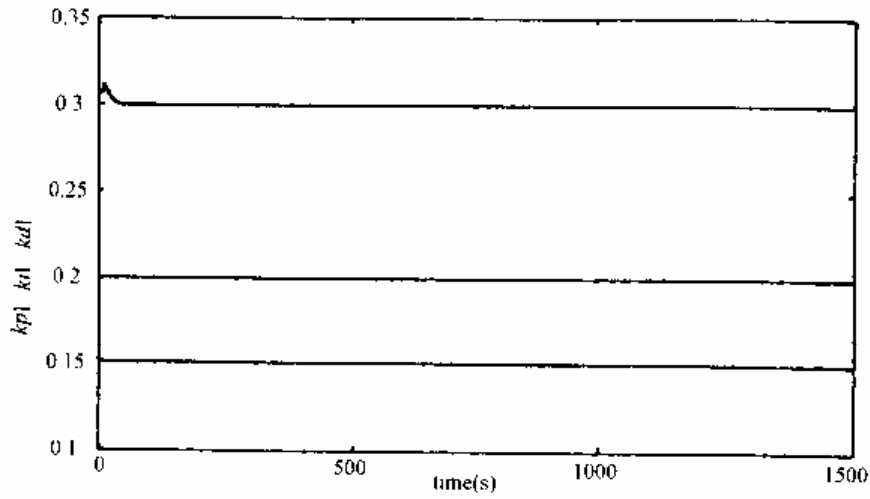


图 6-16 k_{p1}, k_{i1}, k_{d1} 的整定结果

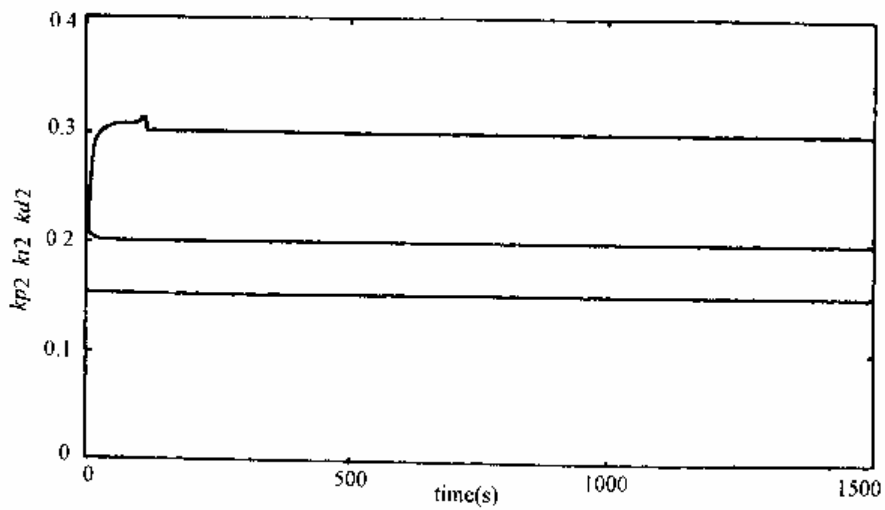


图 6-17 k_{p2}, k_{i2}, k_{d2} 的整定结果

仿真程序: chap6_3.m

%DRNN Tuning PID Controller for coupling plant

```

clear all;
close all;

u1_1=0.0;u1_2=0.0;u1_3=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;
y1_1=0;y2_1=0;

wd1=rand(7,1);
w01=rand(7,1);
wi1=rand(3,7);
wd1_1=wd1;w01_1=w01;wi1_1=wi1;

x1=zeros(7,1);
x1_1=x1;

wd2=rand(7,1);
w02=rand(7,1);
wi2=rand(3,7);
wd2_1=wd2;w02_1=w02;wi2_1=wi2;

xitei1=0.40;xited1=0.40;xiteo1=0.40;
xitei2=0.40;xited2=0.40;xiteo2=0.40;
alfa1=0.04;alfa2=0.04;

x2=zeros(7,1);
x2_1=x2;

error1_1=0;error1_2=0;
error2_1=0;error2_2=0;

kp1=0.30;ki1=0.150;kd1=0.2;
kp2=0.30;ki2=0.150;kd2=0.2;

kp1_1=kp1;kd1_1=kd1;ki1_1=ki1;
kp2_1=kp2;kd2_1=kd2;ki2_1=ki2;

xitekpl=0.5;xitekdl=0.3;xiteki1=0.001;
xitekpl2=0.5;xitekdl2=0.3;xiteki2=0.0001;

ei1=0;ei2=0;

```

```

ts=1;
for k=1:1:1500
time(k)=k*ts;

%Step Signal
R=[1;0];
%R=[0;1];

%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

In1=[u1_1,yout1(k),1]';
for j=1:1:7
    si(j)=In1'*wil(:,j)+wd1(j)*x1(j);
end

for j=1:1:7
    x1(j)=(1-exp(-si(j)))/(1+exp(-si(j)));
end

Pi=0*x1;
for j=1:1:7
    Pi(j)=w01(j)*(1+x1(j))*(1-x1(j))*x1_1(j);
end

Qi=0*wil;
for j=1:1:7
    for i=1:1:3
        Qi(i,j)=w01(j)*(1+x1(j))*(1-x1(j))*In1(i);
    end
end

ym=0;
for j=1:1:7
    ym=ym+x1(j)*w01(j);
end
ymout1(k)=ym;

w01=w01+xite01*(yout1(k)-ymout1(k))*x1+alfa1*(w01-w01_1);

```

```

wdl=wdl+xited1*(yout1(k)-ymout1(k))*Pi+alfal*(wdl-wdl_1);
for j=1:1:7
    if abs(wdl(j))>1
        wdl(j)=0.5*sign(wdl(j));
    end
end
wil=wil+xiteil*(yout1(k)-ymout1(k))*Qi+alfal*(wil-wil_1);

yu=0;
for j=1:1:7
    yu=yu+wol(j)*wil(1,j)*(1+x1(j))*(1-x1(j));
end
dyout1(k)=yu;

error1(k)=R(1) yout1(k);

xc1(1)=error1(k);           %Calculating P
xc1(2)=error1(k)-error1_1; %Calculating D
eil=eil+error1(k)*ts;
xc1(3)=eil;                %Calculating I

kpl(k)=kpl_1+xitekpl*error1(k)*xc1(1);
kd1(k)=kd1_1+xitekdl*error1(k)*xc1(2);
kil(k)=kil_1+xitekil*error1(k)*xc1(3);

kpl(k)=kpl_1+xitekpl*error1(k)*xc1(1)*dyout1(k);
kd1(k)=kd1_1+xitekdl*error1(k)*xc1(2)*dyout1(k);
kil(k)=kil_1+xitekil*error1(k)*xc1(3)*dyout1(k);

if kpl(k)<0
    kpl(k)=0;
end
if kd1(k)<0
    kd1(k)=0;
end
if kil(k)<0
    kil(k)=0;
end

u1(k)=kpl(k)*xc1(1)+kd1(k)*xc1(2)+kil(k)*xc1(3);

```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ln2=[u2_1,yout2(k),1]';

for j=1:1:7
    si(j)=ln1'*wi2(:,j)+wd2(j)*x2(j);
end

for j=1:1:7
    x2(j)=(1-exp(-si(j)))/(1+exp(-si(j)));
end

Pi=0*x2;
for j=1:1:7
    Pi(j)=wo2(j)*(1+x2(j))*(1-x2(j))*x2_1(j);
end

Qi=0*wi2;
for j=1:1:7
    for i=1:1:3
        Qi(i,j)=wo2(j)*(1+x2(j))*(1-x2(j))*ln2(i);
    end
end

ym=0;
for j=1:1:7
    ym=ym+x2(j)*wo2(j);
end
ymout2(k)=ym;

wo2=wo2+xiteo2*(yout2(k)-ymout2(k))*x2+alfa2*(wo2-wo2_1);
wd2=wd2+xited2*(yout2(k)-ymout2(k))*Pi+alfa2*(wd2-wd2_1);
for j=1:1:7
    if abs(wd2(j))>1
        wd2(j)=0.5*sign(wd2(j));
    end
end
wi2=wi2+xitei2*(yout2(k)-ymout2(k))*Qi+alfa2*(wi2-wi2_1);

yu=0;

```

```

for j=1:1:7
    yu-yu+wo2(j)*wi2(1,j)*(1+x2(j))*(1-x2(j));
end
dyout2(k)-yu;

error2(k)=R(2) yout2(k);

xc2(1)=error2(k);           %Calculating P
xc2(2)=error2(k) error2_1;  %Calculating D
ei2=ei2+error2(k)*ts;
xc2(3)=ei2;                 %Calculating I

kp2(k)=kp2_1+xitekp2*error2(k)*xc2(1)*dyout2(k);
kd2(k)=kd2_1+xitek2*error2(k)*xc2(2)*dyout2(k);
ki2(k)=ki2_1+xiteki2*error2(k)*xc2(3)*dyout2(k);

if kp2(k)<0
    kp2(k)=0;
end
if kd2(k)<0
    kd2(k)=0;
end
if ki2(k)<0
    ki2(k)=0;
end

u2(k)=kp2(k)*xc2(1)+kd2(k)*xc2(2)+ki2(k)*xc2(3);

%Return of PID parameters
error1_2=error1_1;
error1_1=error1(k);

error2_2=error2_1;
error2_1=error2(k);

wd1_1=wd1;wo1_1=wo1;wi1_1=wi1;
wd2_1=wd2;wo2_1=wo2;wi2_1=wi2;

u1_4=u1_3;u1_3=u1_2;u1_2=u1_1;u1_1=u1(k);
u2_4=u2_3;u2_3=u2_2;u2_2=u2_1;u2_1=u2(k);

```

```

y1_1=yout1(k);
y2_1=yout2(k);
end
figure(1);
plot(time,dyout1,'r');
xlabel('time(s)');ylabel('Jacobian');
hold on;
plot(time,dyout2,'b');
xlabel('time(s)');ylabel('Jacobian');
figure(2);
plot(time,R(1),'b',time,yout1,'r');
hold on;
plot(time,R(2),'r',time,yout2,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(3);
plot(time,kp1,'r',time,ki1,'b',time,kd1,'k');
xlabel('time(s)');ylabel('kp1 ki1 kd1');
figure(4);
plot(time,kp2,'r',time,ki2,'b',time,kd2,'k');
xlabel('time(s)');ylabel('kp2 ki2 kd2');

```

第 7 章 几种先进 PID 控制方法

7.1 基于干扰观测器的 PID 控制

7.1.1 干扰观测器设计原理

干扰观测器的基本思想是将外部力矩干扰及模型参数变化造成的实际对象与名义模型输出的差异等效到控制输入端，即观测出等效干扰。在控制中引入等效的补偿，实现对干扰完全抑制。其基本结构如图 7-1 所示。

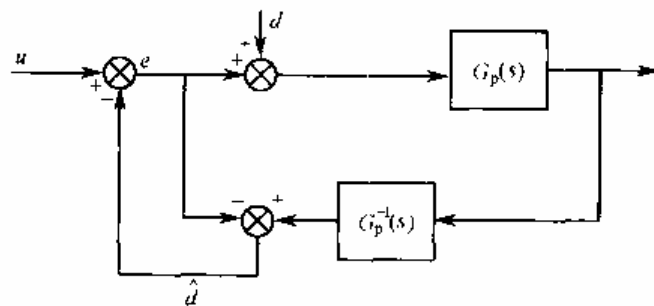


图 7-1 干扰观测器的基本结构

图中的 $G_p(s)$ 为对象的传递函数， d 为等效干扰， \hat{d} 为观测的干扰， u 为控制输入。由此图可求出等效干扰的估计值 \hat{d} 为

$$\hat{d} = (e + d) \cdot G_p(s) \cdot G_p^{-1}(s) - e = d \quad (7.1)$$

对于实际的物理系统，其实现存在如下问题：

- (1) 通常情况下， $G_p(s)$ 的相对阶不为零，其逆在物理上不可实现；
- (2) 对象 $G_p(s)$ 的精确数学模型无法得到；
- (3) 考虑到测量噪声的影响，该方法的控制性能将下降。

解决上述问题的惟一方法是在 \hat{d} 的后面串入低通滤波器 $Q(s)$ ，并用名义模型 $G_n(s)$ 的逆 $G_n^{-1}(s)$ 来代替 $G_p(s)$ ，从而得到如图 7-2 所示的干扰观测器框图，其中虚线部分为干扰观测器。

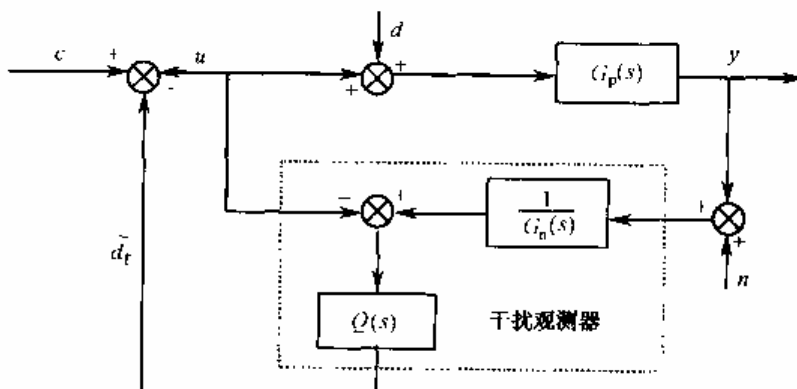


图 7-2 干扰观测器原理框图

控制器的输出为

$$u = c - \tilde{d}_f + d \quad (7.2)$$

式中, c 为 PID 控制器的输出, \tilde{d}_f 为干扰 d 的估计值。

由图 7-2 可得

$$G_{CY}(s) = \frac{Y(s)}{U(s)} = \frac{G_p(s)G_n(s)}{G_n(s) + Q(s)(G_p(s) - G_n(s))} \quad (7.3)$$

$$G_{DY}(s) = \frac{Y(s)}{D(s)} = \frac{G_p(s)G_n(s)(1 - Q(s))}{G_n(s) + Q(s)(G_p(s) - G_n(s))} \quad (7.4)$$

$$G_{NY}(s) = \frac{Y(s)}{N(s)} = \frac{G_p(s)Q(s)}{G_n(s) + Q(s)(G_p(s) - G_n(s))} \quad (7.5)$$

设低通滤波器 $Q(s)$ 的频带为 f_q 。则当 $f \leq f_q$ 时, $Q \approx 1$, $G_{CY} \approx G_n(s)$, $G_{DY} \approx 0$, $G_{NY} \approx 1$ 。当 $f > f_q$ 时, $Q \approx 0$, $G_{CY}(s) \approx G_p(s)$, $G_{DY}(s) \approx G_p(s)$, $G_{NY}(s) \approx 0$ 。通过低通滤波器 $Q(s)$ 的设计可较好地抵抗外加干扰。

由上面分析可见, $Q(s)$ 的设计是干扰观测器设计中的一个重要环节。首先, 为使 $Q(s)G_n^{-1}(s)$ 正则, $Q(s)$ 的相对阶应不小于 $G_n(s)$ 的相对阶; 其次, $Q(s)$ 带宽的设计应是在干扰观测器的鲁棒稳定性和干扰抑制能力之间的折中。

设 $G_p(s)$ 的名义模型为 $G_n(s)$, 则不确定对象的集合可以用乘积摄动来描述, 即

$$G_p(s) = G_n(s)(1 + \Delta(s)) \quad (7.6)$$

式中, $\Delta(s)$ 为可变的传递函数。

如图 7-3 为转台伺服系统某框的实测频率特性 $G_p(s)$ 与名义模型 $G_n(s)$ 频率特性, 由图可见, 当频率增加时, 对象的不确定性增大, $|\Delta(j\omega)|$ 表现为频率 ω 的增函数。

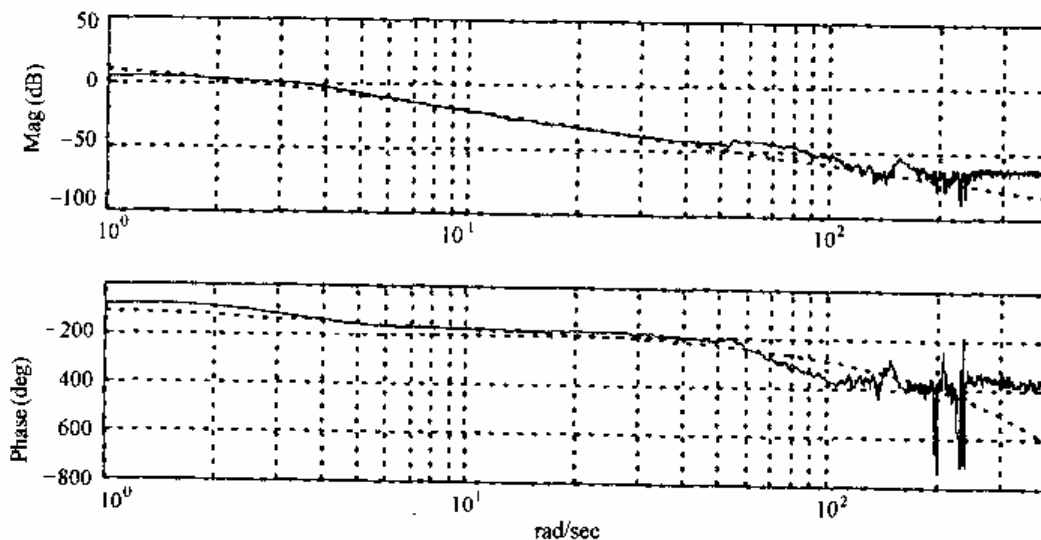


图 7-3 实测被控对象 $G_p(s)$ 与名义模型 $G_n(s)$ 频率特性

由鲁棒稳定性定理, 干扰观测器 $Q(s)$ 鲁棒稳定的充分条件是

$$\|\Delta(s)Q(s)\|_{\infty} \leq 1 \quad (7.7)$$

式(7.7)是 $Q(s)$ 设计的基础, 通过 $Q(s)$ 的设计, 可实现鲁棒性要求。

忽略非建模动态和不确定性的影响, $G_n(s)$ 可描述为

$$G_n(s) = \frac{1}{s(J_n s + b_n)} \quad (7.8)$$

式中, J_n 为等效惯性力矩, b_n 为等效阻尼系数。

采用如下形式的低通滤波器

$$Q(s) = \frac{3\tau + 1}{\tau^3 s^3 + 3\tau^2 \tau s^2 + 3\tau s + 1} \quad (7.9)$$

由 $\Delta(s) = \frac{G_p(s)}{G_n(s)} - 1$ 可得 $\Delta(s)$ 的频率特性, 它表明了实际对象频率特性对名义模型的摄

动, $\Delta(s)$ 和不同带宽的 $Q(s)$ 的幅频特性见图 7-4, 可见当 $Q(s) = Q_2(s)$ 时鲁棒稳定性可以得到满足, 并且外界干扰可以得到很好的抑制。因此 $Q(s) = Q_2(s)$ 为理想的低通滤波器, 此时 $\tau = 0.001$ 。

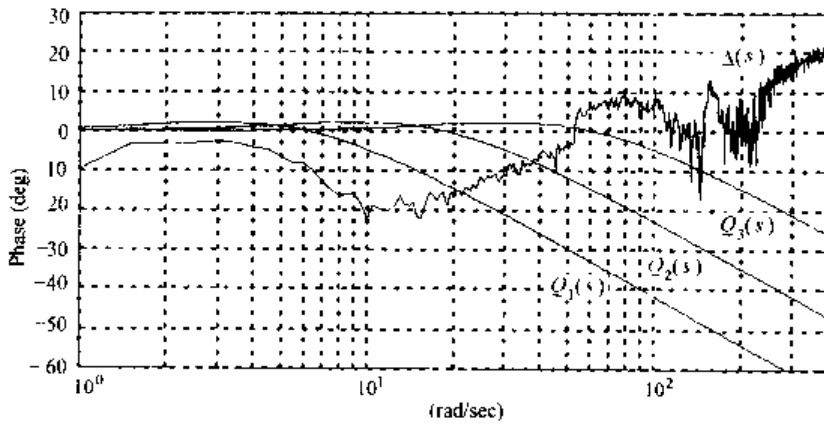


图 7-4 $\Delta(s)$ 和 $Q(s)$ 的幅频特性

7.1.2 连续系统的控制仿真

仿真实例

设实际的被控对象为

$$G_p(s) = \frac{1}{0.003s^2 + 0.067s}$$

名义模型取

$$G_n(s) = \frac{1}{0.0033s^2 + 0.0673s}$$

取指令信号为: $r(t) = 1.0 \sin(2\pi t)$, 干扰信号为: $d(t) = 3 \sin(5\pi t)$, PID 控制器中取 $k_p = 5.0, k_i = 0, k_d = 0.50$ 。 $Q(s)$ 按(7.9)式进行设计, 并取 $\tau = 0.001$ 。干扰观测器的 Simulink 仿真程序见图 7-5 所示, 先运行参数初始化程序 chap7_1f.m, 对干扰信号的观测结果如图 7-6 所示, 分别对加入干扰观测器和不加入干扰观测器两种情况进行仿真, 其正弦跟踪如图 7-7 和图 7-8 所示。

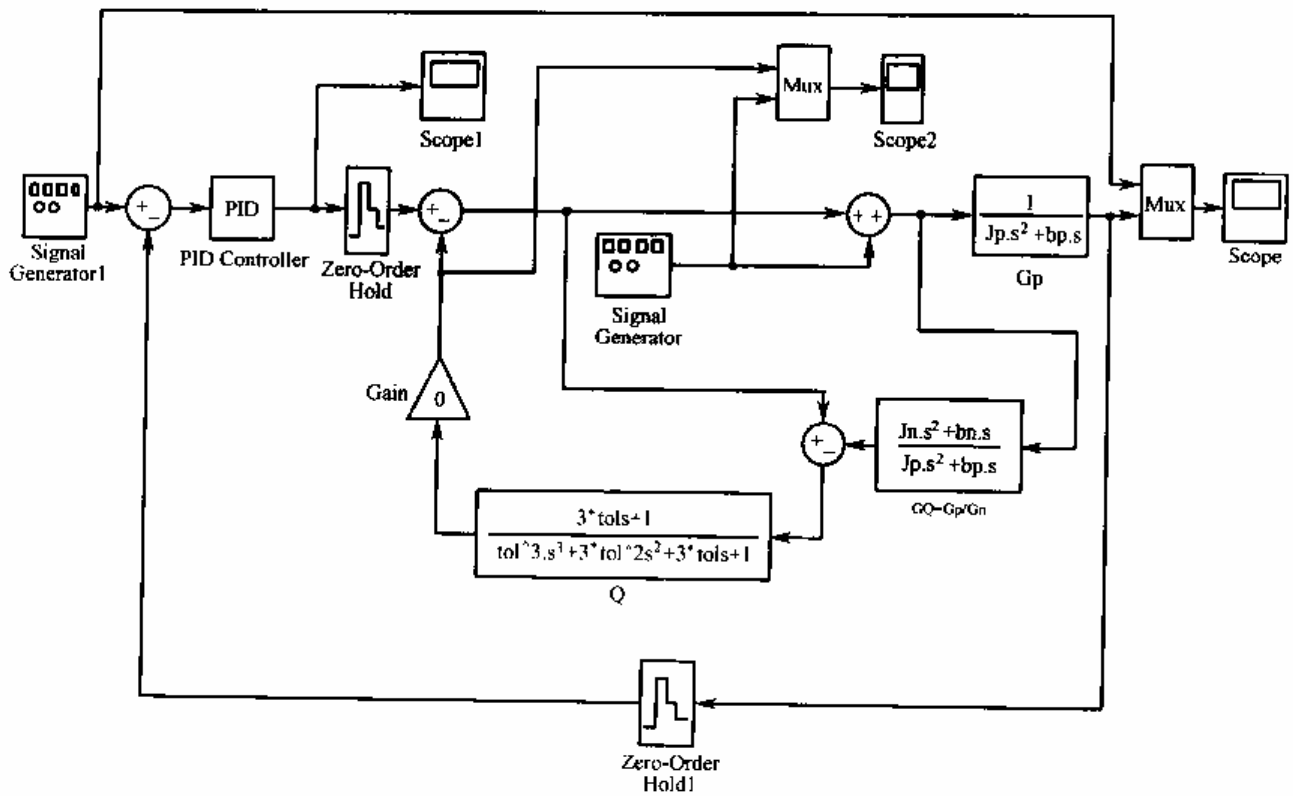


图 7-5 干扰观测器 Simulink 仿真

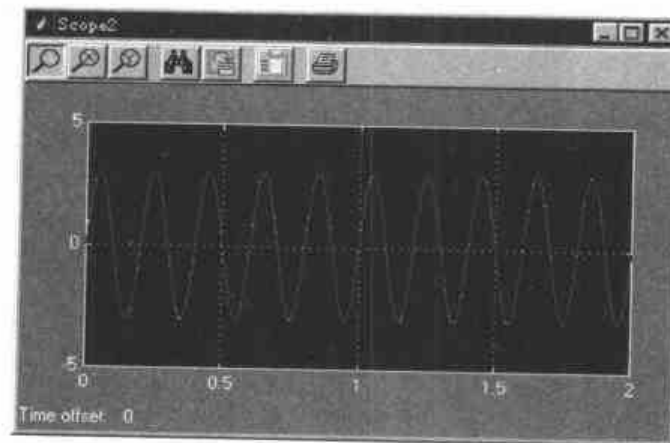


图 7-6 干扰信号的观测结果

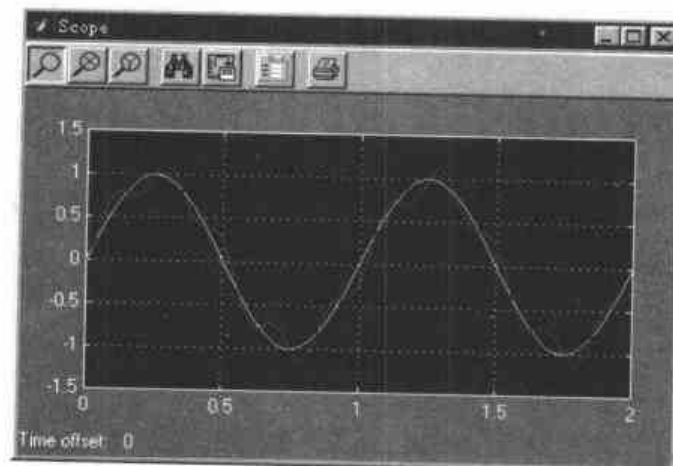


图 7-7 无干扰观测器时的正弦跟踪

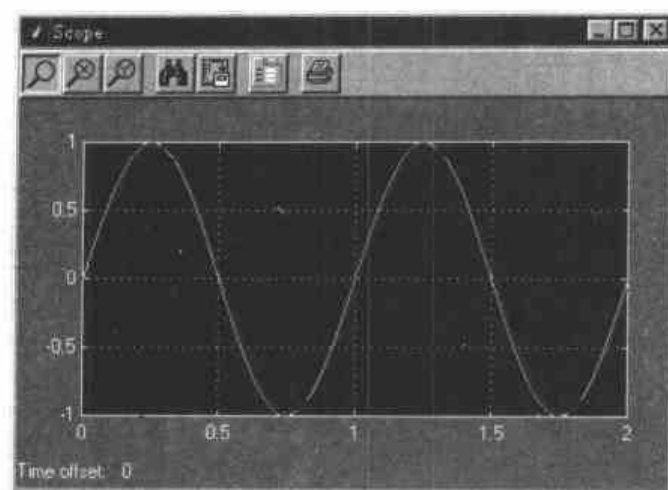


图 7-8 有干扰观测器时的正弦跟踪

仿真程序:

参数初始化程序: chap7_1f.m

```
clear all;
close all;
Jp=0.0030;bp=0.067;
Jn=0.0033;bn=0.0673;
Gp=tf([1],[Jp,bp,0]);
Gn=tf([1],[Jn,bn,0]);

tol=0.001;
Q=tf([3*tol,1],[tol^3,3*tol^2,3*tol,1]);
bode(Q);
dcgain(Q)
OD1=1/(1-Q);
OD2=Q/Gn;
```

仿真主程序: chap7_1.mdl, 见图 7-5 所示。

7.1.3 离散系统的控制仿真

由连续干扰观测器可得到离散干扰观测器的结构, 如图 7-9 所示, $Q(z^{-1})$ 为低通滤波器, 则

$$G_{CY}(z^{-1}) = \frac{G_p(z^{-1})G_n(z^{-1})}{G_n(z^{-1}) + Q(z^{-1})(G_p(z^{-1}) - G_n(z^{-1}))} \quad (7.10)$$

$$G_{DY}(z^{-1}) = \frac{G_p(z^{-1})G_n(z^{-1})(1 - Q(z^{-1}))}{G_n(z^{-1}) + Q(z^{-1})(G_p(z^{-1})(z^{-1}) - G_n(z^{-1}))} \quad (7.11)$$

$$G_{NY}(z^{-1}) = \frac{G_p(z^{-1})Q(z^{-1})}{G_n(z^{-1}) + Q(z^{-1})(G_p(z^{-1}) - G_n(z^{-1}))} \quad (7.12)$$

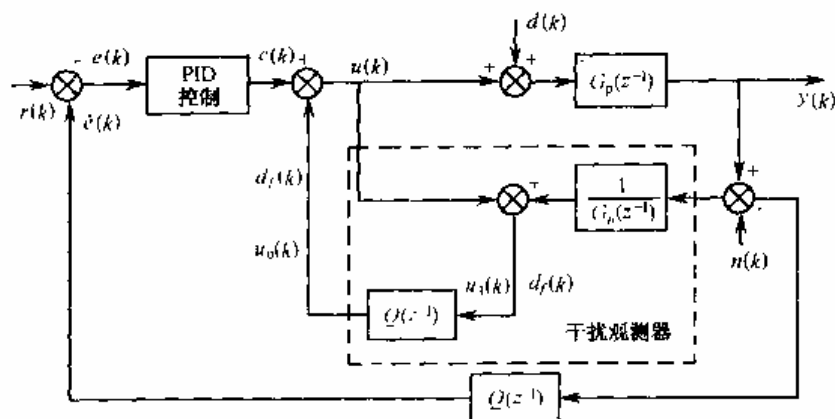


图 7-9 离散系统干扰观测器

设 $Q(z^{-1})$ 为理想的低通滤波器，即在低频段，当 $f \leq f_q$ 时， $Q(z^{-1}) \approx 1$ ；在高频段，当 $f \geq f_q$ 时， $Q(z^{-1}) \approx 0$ 。

在低频段时，有 $G_{CY}(z^{-1}) \approx G_n(z^{-1})$, $G_{DY}(z^{-1}) \approx 0$, $G_{NY}(z^{-1}) \approx 1$ ，说明干扰观测器对于低频干扰具有很好的抑制能力，但对低频测量噪声非常敏感。在高频段时，有 $G_{CY}(z^{-1}) \approx G_p(z^{-1})$, $G_{DY}(z^{-1}) \approx G_p(z^{-1})$, $G_{NY}(z^{-1}) \approx 0$ ，说明干扰观测器对于高频段测量噪声具有很好的抑制能力，但对于干扰却没有抑制作用。

正确地选择 $Q(z^{-1})$ 可实现对干扰 $d(k)$ 和测量噪声 $n(k)$ 的完全抑制。

仿真实例

设实际的被控对象为

$$G_p(s) = \frac{1}{0.003s^2 + 0.067s}$$

名义模型取

$$G_n(s) = \frac{1}{0.00305s^2 + 0.0671s}$$

采样时间为 0.001s。假设干扰信号为 $d(k) = 50\sin(10\pi t)$ ， $n(k)$ 为幅值 0.001 的随机信号，指令信号为正弦信号： $r(k) = 0.50\sin(6\pi t)$ ，在 PD 控制中选取 $k_p = 15.0$, $k_d = 5.0$ 。

$Q(s)$ 按(7.9)式进行设计，并取 $\tau = 0.001$ 。图 7-10 为 $Q(s)$ 滤波前、后信号，图 7-11 为干扰 d 及其干扰观测器的观测结果 \tilde{d}_f ，图 7-12 为不加干扰观测器时的正弦跟踪 ($M = 1$)，图 7-13 为加入干扰观测器时的正弦跟踪 ($M = 2$)。

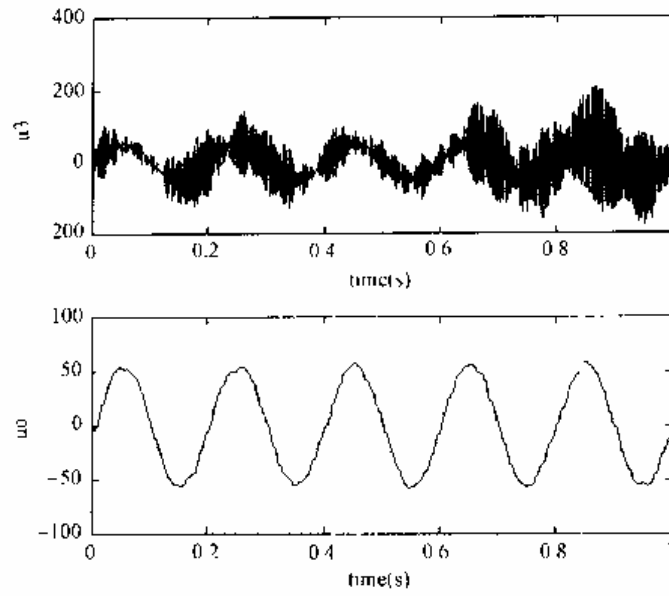


图 7-10 低通滤波器滤波前、后信号

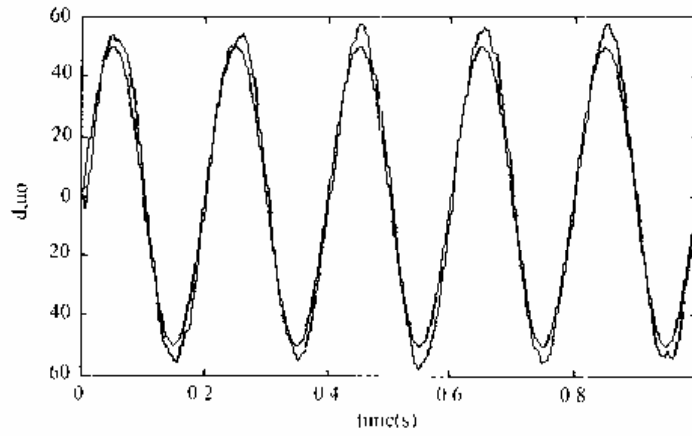


图 7-11 干扰 d 及其观测结果 \tilde{d}_1

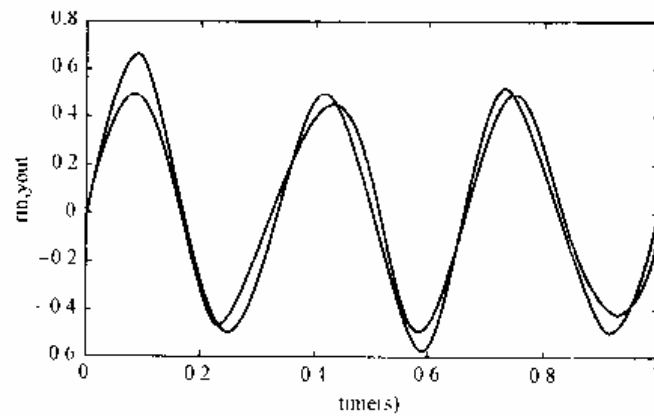


图 7-12 无干扰观测器时的位置跟踪 ($M=1$)

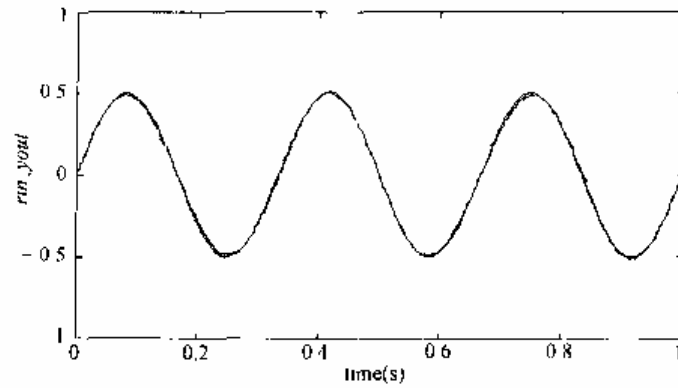


图 7-13 加入干扰观测器时的位置跟踪 ($M = 2$)

仿真程序: chap7_2.m

```
%PID Control based on Disturbance Observer
clear all;
close all;

Jp=0.0075;bp=0.1880;
Jn=Jp;bn=bp;

ts=0.001;

Gp=tf([1],[Jp,bp,0]); %Plant
Gpz=c2d(Gp,ts,'z');
[num,den]=tfdata(Gpz,'v');

Gn=tf([1],[Jn,bn,0]); %Nominal model
Gnz=c2d(Gn,ts,'z');
[num1,den1]=tfdata(Gnz,'v');

tol=0.0065;
Q=tf([3*tol,1],[tol^3,3*tol^2,3*tol,1]); %Low Pass Filter
Qz=c2d(Q,ts,'tustin');
[numq,denq]=tfdata(Qz,'v');

uu_1=0;
u2_1=0;
uo_1=0;uo_2=0;uo_3=0;
u3_1=0;u3_2=0;u3_3=0;

u_1=0.0;u_2=0.0;
y_1=0;y_2=0;
```

```

x=[0,0,0]';
error_1=0;

for k=1:1:1000
time(k)=k*ts;

rin(k)=0.5*sin(3*2*pi*k*ts); % Tracing Sine high frequency Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

n(k)=0.001*rands(1); % Measure noise
yout(k)=yout(k)+n(k); % Disturbance n(k)

error(k)=rin(k)-yout(k);

x(1)=error(k); % Calculating P
x(2)=(error(k)-error_1)/ts; % Calculating D
x(3)=x(3)+error(k)*ts; % Calculating I

kp=50;ki=0;kd=15.0; % Tracing Sine velocity

c(k)=kp*x(1)+kd*x(2)+ki*x(3); % PID Controller

u1(k)=uu_1; %Mn=1, one time delay

u2(k)=1/num1(2)*(-num1(3)*u2_1+yout(k)+den1(2)*y_1+den1(3)*y_2);

u3(k)=u2(k)-u1(k);

uo(k)=-denq(2)*uo_1-denq(3)*uo_2-
denq(4)*uo_3+numq(1)*u3(k)+numq(2)*u3_1+numq(3)*u3_2+numq(4)*u3_3;

Q=1;
if Q==0 %Not using Q(s)
    uo(k)=u3(k);
end

M=2;

```

```

if M==1    %Using Observer
    uu(k)=c(k)-uo(k);
end
if M==2    %No Observer
    uu(k)=c(k);
end

d(k)=50*sin(5*2*pi*k*ts);
u(k)=uu(k)+d(k);    % Disturbance d(k)

if u(k)>=110    % Restricting the output of controller
    u(k)=110;
end
if u(k)<=-110
    u(k)=-110;
end

uu_1=uu(k);
u2_1=u2(k);

uo_3=uo_2;uo_2=uo_1;uo_1=uo(k);
u3_3=u3_2;u3_2=u3_1;u3_1=u3(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

error_1=error(k);
end
figure(1);
subplot(211);
plot(time,u3,'k');
xlabel('time(s)');ylabel('u3');
subplot(212);
plot(time,uo,'k');
xlabel('time(s)');ylabel('uo');
figure(2);
plot(time,d,'k',time,uo,'k');
xlabel('time(s)');ylabel('d,uo');
figure(3);
plot(time,rin,'k',time,yout,'k');

```

```
xlabel('time(s)');ylabel('rin,yout');
```

7.2 非线性系统的 PID 鲁棒控制

7.2.1 基于 NCD 优化的非线性优化 PID 控制

MATLAB 不但有用于动态系统仿真的 Simulink 工具箱, 还有一个专用于非线性控制系统优化设计的工具箱 NCD(Nonlinear Control Design)。借助于 NCD 工具箱, 可以实现系统参数的优化设计。

在 MATLAB NCD 工具箱中提供有一个专门作系统优化设计的 NCD Blockset (非线性控制系统设计模块组), 利用该模块组, 系统的优化设计可以自动实现。

仿真实例

被控对象传递函数为

$$G(s) = \frac{1.5}{50s^2 + a_2s + a_1s + 1}$$

式中, $a_2 = 43, a_1 = 3$ 。

系统包含饱和环节和速度限制环节 ± 0.8 两个非线性环节。系统含有不确定因素: a_2 在 40 ~ 50 之间变化, a_1 在 $(0.5 \sim 2.0) \times 3$ 之间变化。采用 PID 控制器, PID 的优化指标为:

- (1) 最大超调量不大于 20%;
- (2) 上升时间不大于 10s;
- (3) 调整时间不大于 30s;
- (4) 系统具有鲁棒性。

仿真程序包括两个部分: Simulink 程序及初始化的 M 函数程序。采用 MATLAB 中的非线性系统设计工具箱 NCD 可实现 PID 控制器的优化。

仿真的关键是 NCD 功能的使用。在 Simulink 环境中双击 NCD Output 模块, 弹出 NCD Blockset 约束窗口, 通过 Options 菜单和 Parameters 菜单实现该功能的使用。具体说明如下:

(1) Options 菜单的使用

a. 通过 Step Response 命令定义阶跃响应性能指标:

Setting time:	调整时间, 选 30s
Rise time:	上升时间, 选 10s
Persent setting:	稳态误差百分数, 取 5
Persent overshoot:	超调量百分数, 取 20
Persent undershoot:	振荡负幅值百分数, 取 1
Step time:	启动时间, 取 0
Final time:	终止时间, 取 100
Initial output:	初始值, 取 0
Final output:	最终值, 取 1

b. 通过 Time range 命令, 设置优化时间, 取 0 ~ 100s

c. 通过选择 Y-Axis 命令, 设置阶跃响应范围, 取 [0, 1.5]

(2) Optimization 菜单的使用

a. 选择 Parameters 项, 定义调整变量及有关参数

输入: 待调整优化变量 k_p, k_i, k_d 及它们的上下限, 可取为:

$$\text{下限: } \frac{k_p}{5} \quad \frac{k_i}{5} \quad \frac{k_d}{5}$$

$$\text{上限: } 5k_p \quad 5k_i \quad 5k_d$$

变量允差: 0.001 约束允差: 0.001

b. 选择 Uncertainty 项, 定义不确定变量及有关参数

输入: 不确定变量 a_1, a_2 的上下限, 可取为:

$$\text{下限: } 1.5 \quad 40$$

$$\text{上限: } 6.0 \quad 50$$

c. 选择 Start 命令, 进行调整变量的优化, 直到阶跃响应指标达到要求为止。优化时 NCD 约束窗口不断显示阶跃响应的优化过程, MATLAB COMMAND 窗口也不断显示有关信息。

每次优化结果保存在 pid_ncd.mat 文件中, 以供下一次调用。阶跃响应性能限制可以直接由鼠标在 NCD Blockset 约束窗口设置。

仿真时首先运行初始化程序 chap7_3f.m, 然后运行 Simulink 主程序 chap7_3.mdl (见图 7-14)。优化参数如下: $k_p = 1.9193, k_i = 0.0978, k_d = 8.1678$, 响应曲线如图 7-15 所示。

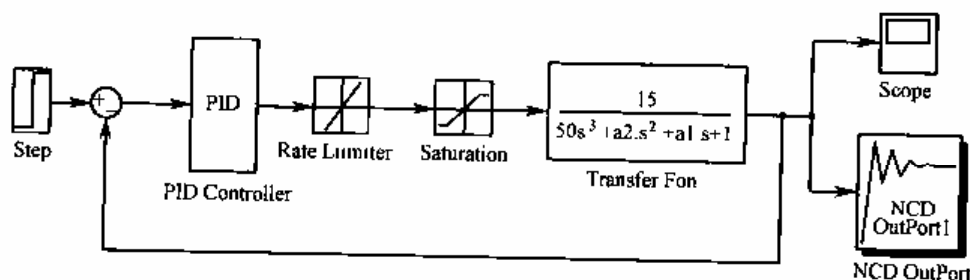


图 7-14 非线性系统 Simulink 主程序

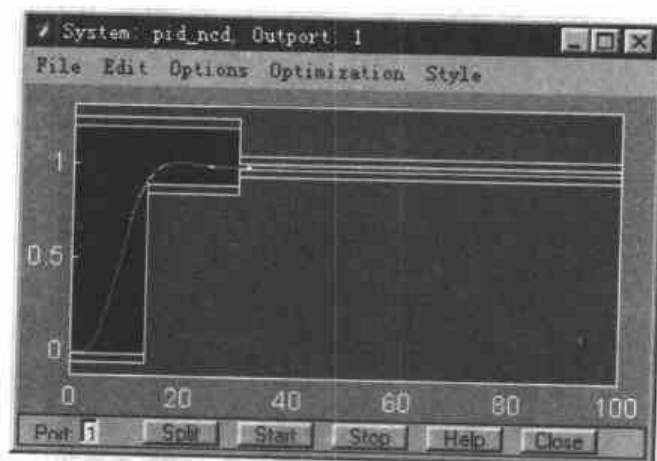


图 7-15 NCD 阶跃响应曲线面

(1) Simulink 初始化程序: chap7_3f.m

```
clear all;
```

```
close all;
```

```
%load pid_ncd.mat;
```

```
kp=0.63;
```

```

ki=0.0504;
kd=1.9688;
a2=43;
a1=3;

```

(2) Simulink 主程序: chap7_3.mdl, 如图 7-14 所示。

7.2.2 基于 NCD 与优化函数结合的非线性优化 PID 控制

采用 MATLAB 非线性控制系统设计工具箱 NCD 并结合优化工具箱提供的各类函数可实现 PID 的整定。

利用 MATLAB 非线性最小平方函数 lsqnonlin(), 按照最小平方指标 $J = \int e^2 dt$ 进行 PID 参数寻优, 从而实现 PID 的整定。

仿真实例

被控对象为

$$G(s) = \frac{50s + 50}{s^3 + s^2 + s}$$

运行主程序 chap7_4.m, 优化结果为 $k_p = 4.4124$, $k_i = 0.8809$, $k_d = 1.9377$, 采用优化参数运行 Simulink 程序 chap7_4f2.mdl, 如图 7-16 所示, 优化后的阶跃响应如图 7-17 所示。

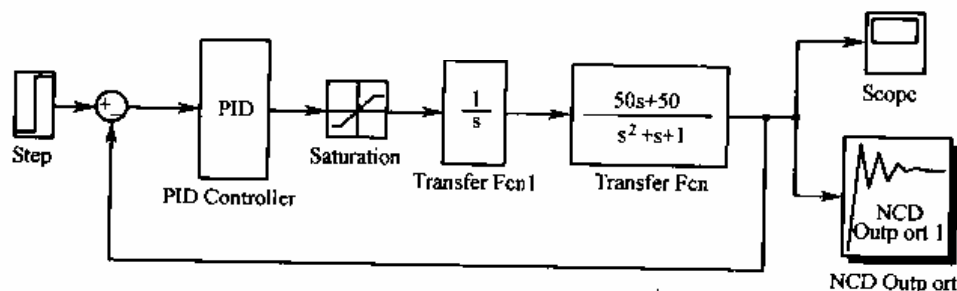


图 7-16 Simulink 程序

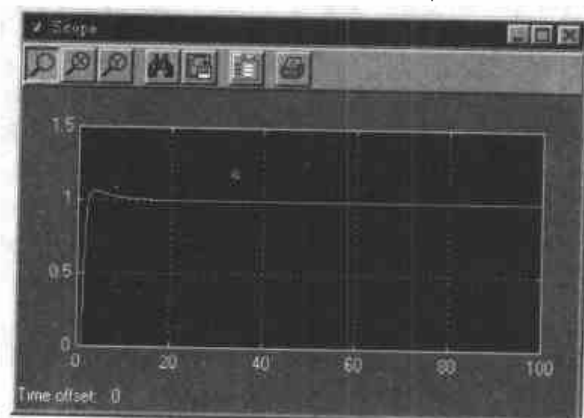


图 7-17 优化后的阶跃响应

仿真程序: 分为主程序、M 函数子程序和 Simulink 子程序三个部分。

主程序: chap7_4.m

```
clear all;
```

```
close all;
```



```

nl_pid0=[0 0 0];
options=[1 0.01 0.01];
nl_pid=lsqnonlin('chap7_4f1',nl_pid0,options)

```

M 函数子程序: chap7_4f1.m

```

function f=pid_ncd_pg_eq(nl_pid)
assignin('base','kp',nl_pid(1));
assignin('base','ki',nl_pid(2));
assignin('base','kd',nl_pid(3));
opt=simset('solver','ode5');
[tout,xout,yout]=sim('chap7_4f2',[0 10],opt);
f=yout-1;

```

Simulink 子程序: chap7_4f2.mdl, 如图 7-16 所示。

7.3 一类非线性 PID 控制器设计

7.3.1 非线性控制器设计原理

设图 7-18 是一般的系统阶跃响应曲线, 采用该曲线可以分析非线性 PID 控制器增益参数的构造思想。

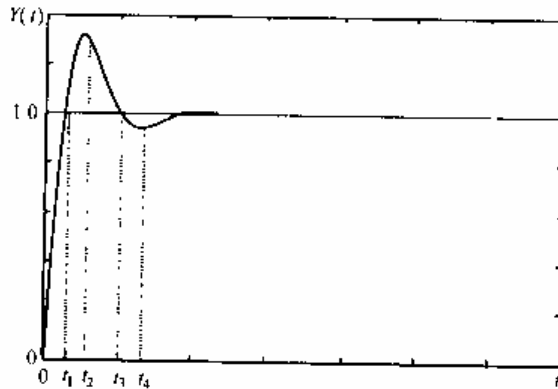


图 7-18 一般系统阶跃响应曲线

非线性控制器设计原理如下:

(1) 比例增益参数 k_p : 在响应时间 $0 \leq t \leq t_1$ 段, 为保证系统有较快的响应速度, 比例增益参数 k_p 在初始时应较大, 同时为了减小超调量, 希望误差 e_p 逐渐减小时, 比例增益也随之减小; 在 $t_1 \leq t \leq t_2$ 段, 为了增大反向控制作用, 减小超调, 期望 k_p 逐渐增大; 在 $t_2 \leq t \leq t_3$ 段, 为了使系统尽快回到稳定点, 并不再产生大的惯性, 期望 k_p 逐渐减小; 在 $t_3 \leq t \leq t_4$ 段, 期望 k_p 逐渐增大, 作用与 $t_1 \leq t \leq t_2$ 段相同。显然, 按上述变化规律, k_p 随误差 e_p 变化的大致形状如图 7-19(a)所示, 根据该图可以构造如下非线性函数

$$k_p(e_p(t)) = a_p + b_p(1 - \operatorname{sech}(c_p e_p(t))) \quad (7.13)$$

式中, a_p, b_p, c_p 为正实常数。当误差 $e_p \rightarrow \pm\infty$ 时, k_p 取最大值为 $a_p + b_p$; 当 $e_p = 0$ 时, k_p 取最小值为 a_p ; b_p 为 k_p 的变化区间, 调整 c_p 的大小可调整 k_p 变化的速率。

(2) 微分增益参数 k_d : 在响应时间 $0 \leq t \leq t_1$ 段, 微分增益参数 k_d 应由小逐渐增大, 这样可以保证在不影响响应速度的前提下, 抑制超调的产生; 在 $t_1 \leq t \leq t_2$ 段, 继续增大 k_d , 从而增大反向控制作用, 减小超调量。在 t_2 时刻, 减小微分增益参数 k_d , 并在随后的 $t_2 \leq t \leq t_4$ 段再次逐渐增大 k_d , 抑制超调的产生。根据 k_d 的变化要求, 在构造 k_d 的非线性函数时应考虑到误差变化速率 e_v 的符号。 k_d 的变化形状如图 7-19(b)所示, 所构造的非线性函数为

$$k_d(e_p(t)) = a_d + b_d / (1 + c_d \exp(d_d \cdot e_p(t))) \quad (7.14)$$

式中, $e_v = \dot{e}_p$ 为误差变化速率, a_d, b_d, c_d, d_d 为止实常数, a_d 为 k_d 的最小值, $a_d + b_d$ 为 k_d 的最大值, 当 $e_p = 0$ 时, $k_d = a_d + b_d / (1 + c_d)$, 调整 d_d 的大小可调整 k_d 的变化速率。

(3) 积分增益参数 k_i : 当误差信号较大时, 希望积分增益不要太大, 以防止响应产生震荡, 有利于减小超调量; 而当误差较小时, 希望积分增益增大, 以消除系统的稳态误差。根据积分增益的希望变化特性, 积分增益参数 k_i 的变化形状如图 7-19(c)所示, 其非线性函数可表示为

$$k_i(e_p(t)) = a_i \operatorname{sech}(c_i e_i(t)) \quad (7.15)$$

式中, $k_i(e_p(t)) = a_i \operatorname{sech}(c_i e_i(t))$ 为正实常数, k_i 的取值范围为 $(0, a_i)$, 当 $e_p = 0$ 时, k_i 取最大值。 c_i 的取值决定了 k_i 的变化快慢程度。

非线性 PID 调节器的控制输入为

$$u(t) = k_p(e_p(t))e_p(t) + k_i(e_p(t)) \int_0^t e_p(t) dt + k_d(e_p(t), e_v(t)) \frac{de_p(t)}{dt} \quad (7.16)$$

由上述分析可知, 如果非线性函数中的各项参数选择适当的话。能够使控制系统既达到响应快, 又无超调现象。另外, 由于非线性 PID 调节器中的增益参数能够随控制误差而变化, 因而其抗干扰能力也较常规线性 PID 控制强。 k_p, k_i, k_d 变化的示意图如图 7-19 所示。

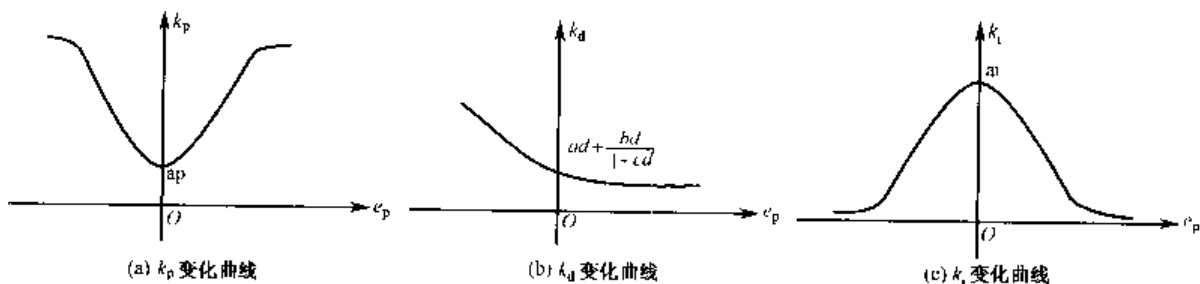


图 7-19 非线性增益调节参数变化曲线

7.3.2 仿真程序及分析

仿真实例

求二阶传递函数的阶跃响应

$$G_p(s) = \frac{133}{s^2 + 25s}$$

采用离散 PID 进行仿真, 采样时间为 1ms。

取 $S=1$, 针对阶跃进行仿真, 阶跃响应如图 7-20 所示, 其中 k_p, k_i, k_d 随偏差 ($M=1$) 的变化曲线如图 7-21 所示; k_p, k_i, k_d 随时间 ($M=2$) 的变化曲线如图 7-22 所示; 取 $S=2$, 针对正弦信号进行仿真, 正弦响应如图 7-23 所示; 其中 k_p, k_i, k_d 随偏差 ($M=1$) 的变化曲线如

图 7-24 所示, k_p, k_i, k_d 随时间 ($M = 2$) 的变化曲线如图 7-25 所示。

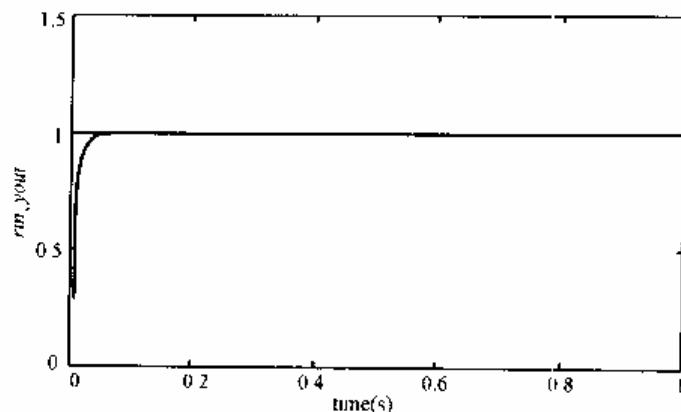
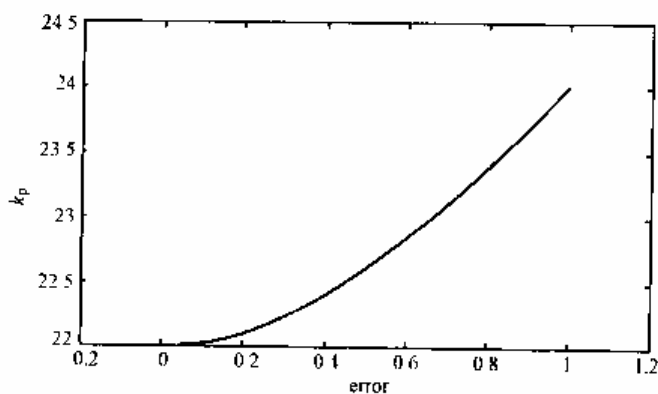
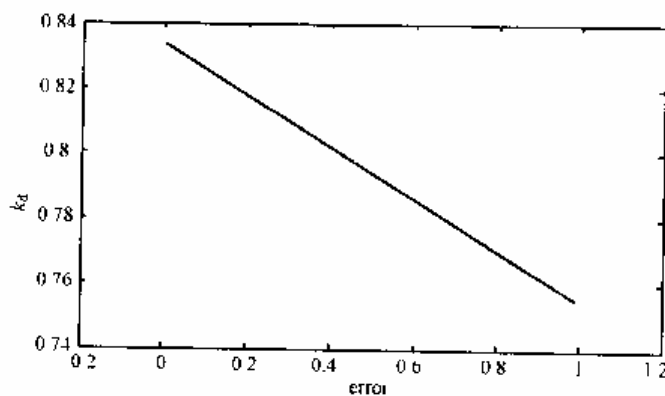


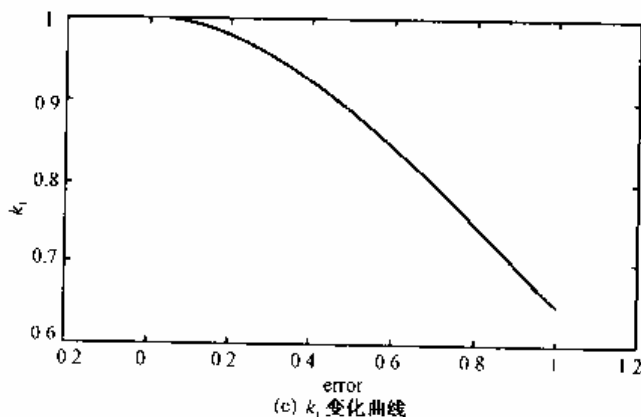
图 7-20 阶跃响应 ($S = 1$)



(a) k_p 变化曲线

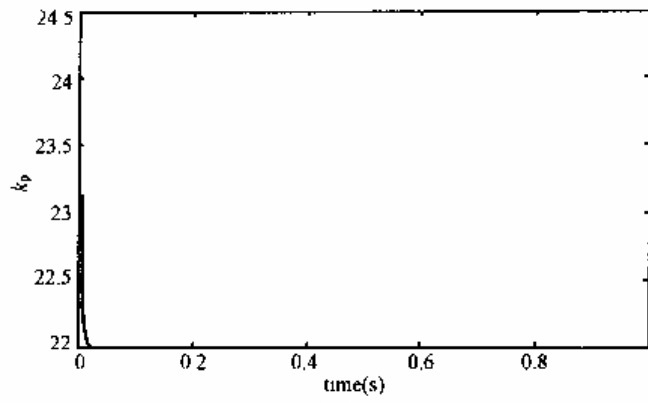


(b) k_d 变化曲线

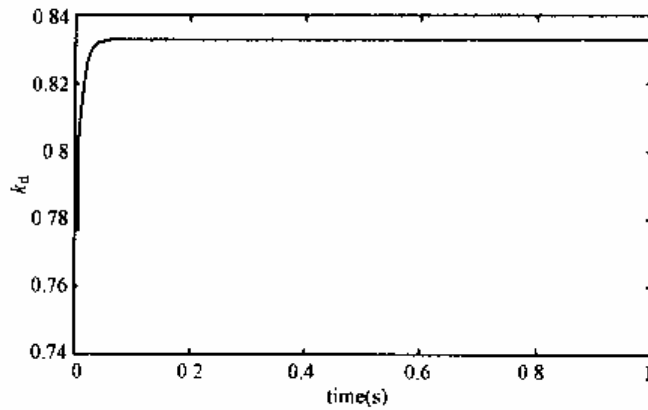


(c) k_i 变化曲线

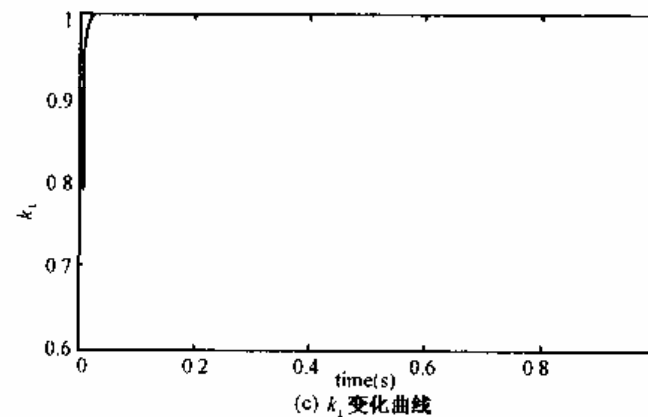
图 7-21 k_p, k_i, k_d 随偏差的变化曲线 ($M = 1$)



(a) k_p 变化曲线



(b) k_d 变化曲线



(c) k_i 变化曲线

图 7-22 k_p, k_i, k_d 随时间的变化曲线 ($M = 2$)

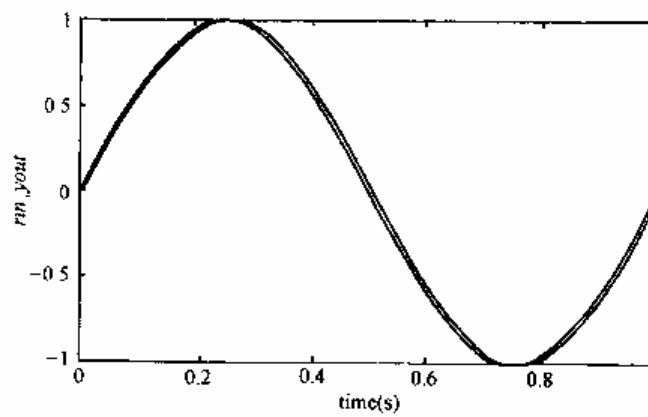
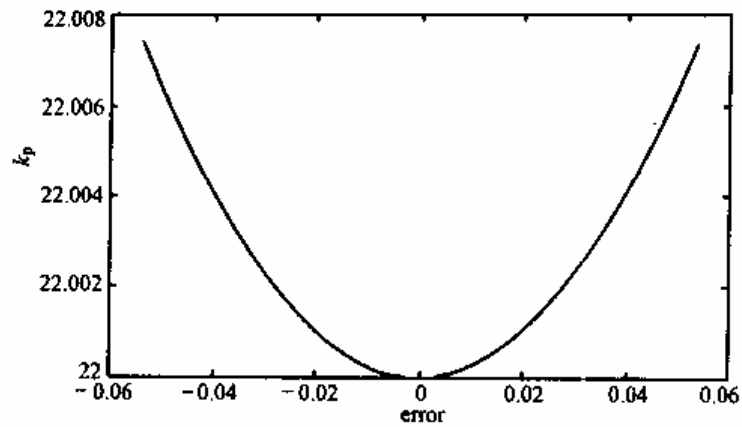
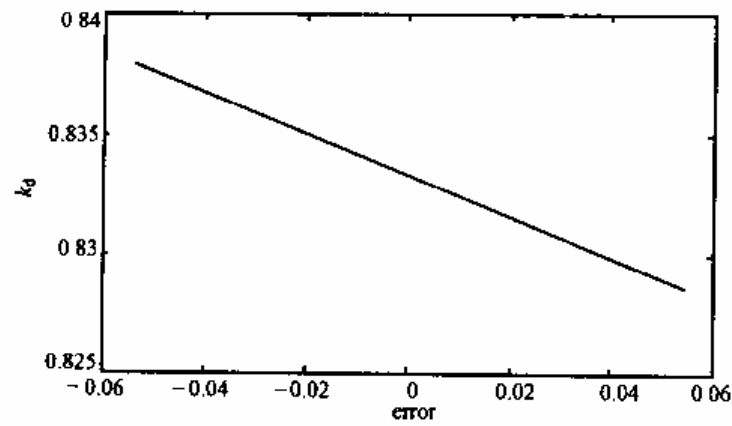


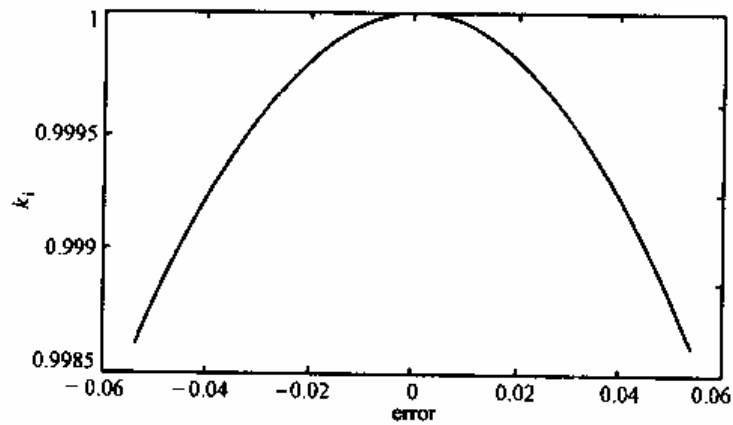
图 7-23 正弦响应 ($S = 2$)



(a) k_p 变化曲线

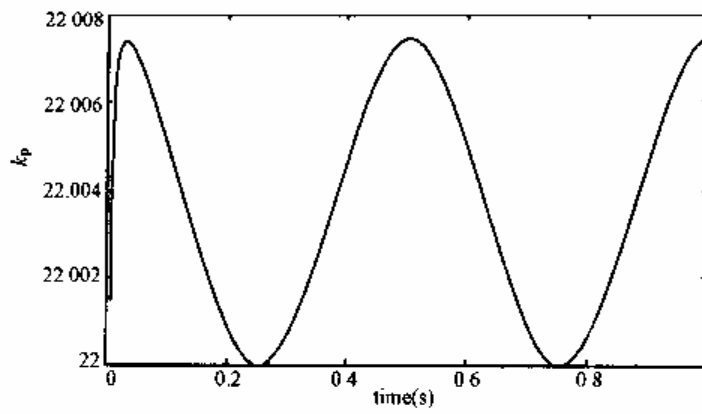


(b) k_d 变化曲线

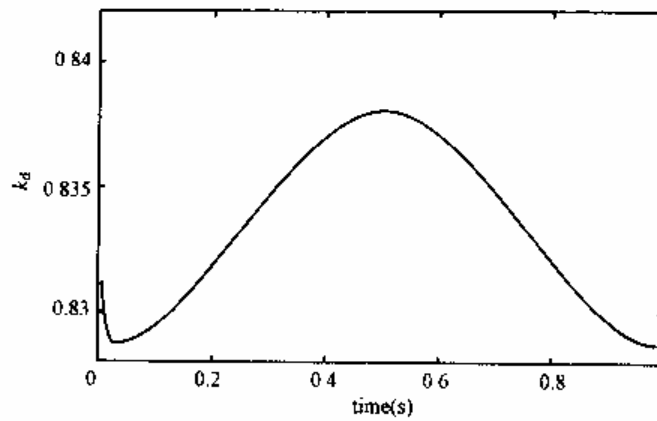


(c) k_i 变化曲线

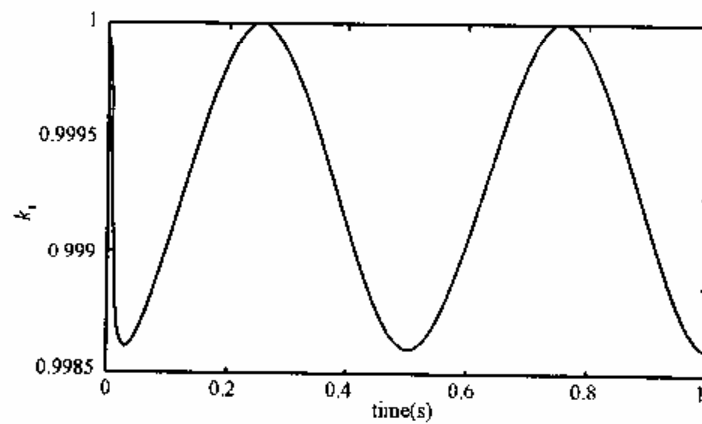
图 7-24 k_p, k_i, k_d 随误差的变化曲线 ($M=1$)



(a) k_p 变化曲线



(b) k_d 变化曲线



(c) k_i 变化曲线

图 7-25 k_p, k_i, k_d 随时间的变化曲线 ($M=2$)

从仿真结果可以看出, k_p, k_i, k_d 的变化规律符合 PID 控制的原理, 取得很好的仿真效果。

仿真程序: chap7_5.m

```
%Nonlinear PID Control for a servo system
```

```
clear all;
```

```
close all;
```

```
ts=0.001;
```

```
J=1/133;
```

```

q=25/133;
sys=tf(1,[J,q,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

r_1=0;r_2=0;
u_1=0;u_2=0;
y_1=0;y_2=0;
error_1=0;
ei=0;
for k=1:1:1000
time(k)=k*ts;

S=2;
if S==1 %Step Signal
rin(k)=1.0;
elseif S==2 %Sine Signal
rin(k)=1.0*sin(1*2*pi*k*ts);
end

yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=rin(k)-yout(k);
derror(k)=(error(k)-error_1)/ts;

ap=22;bp=8.0;cp=0.8;
kp(k)=ap+bp*(1-sech(cp*error(k)));

ad=0.5;bd=2.5;cd=6.5;dd=0.30;
kd(k)=ad+bd/(1+cd*exp(dd*error(k)));

ai=1;ci=1;
ki(k)=ai*sech(ci*error(k));

ei=ei+error(k)*ts;
u(k)=kp(k)*error(k)+kd(k)*derror(k)+ki(k)*ei;

%Update Parameters
r_2=r_1;r_1=rin(k);
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

```

```

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');ylabel('error');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,derror,'r');
xlabel('time(s)');ylabel('derror');

M=1;
if M==1
    figure(4);
    plot(error,kp,'r');xlabel('error');ylabel('kp');
    figure(5);
    plot(error,kd,'r');xlabel('error');ylabel('kd');
    ad+bd/(1+cd)
    figure(6);
    plot(error,ki,'r');xlabel('error');ylabel('ki');
elseif M==2
    figure(4);
    plot(time,kp,'r');xlabel('time(s)');ylabel('kp');
    figure(5);
    plot(time,kd,'r');xlabel('time(s)');ylabel('kd');
    figure(6);
    plot(time,ki,'r');xlabel('time(s)');ylabel('ki');
end
end

```

7.4 基于重复控制补偿的高精度 PID 控制

7.4.1 重复控制原理

重复控制是日本的 Inoue 于 1981 年首先提出来的,用于伺服系统重复轨迹的高精度控制。重复控制之所以能提高系统跟踪精度,其原理来源于内模原理。

加到被控对象的输入信号除偏差信号外,还叠加了一个“过去的控制偏差”,该偏差是上一周期该时刻的控制偏差。把上一次运行时的偏差反映到现在,和“现在的偏差”一起加到被控对象进行控制,这种控制方式,偏差重复被使用,称为重复控制。经过几个周期的重复控制之后可以大大提高系统的跟踪精度,改善系统品质。这种控制方法不仅适用于跟踪周

周期性输入信号，也可以抑制周期性干扰。

在重复控制中，一般期望重复控制作用在高频段的增益减小。为此，在重复控制中经常加入低通滤波器 $Q(s)$ 。本控制方法取

$$Q(s) = \frac{1}{1 + T_q s} \quad (7.17)$$

式中， $T_q > 0$ 为滤波器的时间常数。

重复控制信号是周期性信号，其基本构造如图 7-26 所示。应用重复控制不仅可以提高系统的跟踪精度，还可以提高系统的鲁棒性。

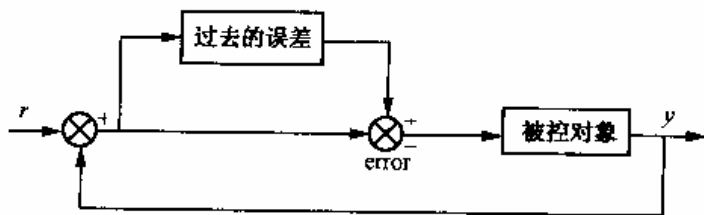


图 7-26 重复控制系统原理图

7.4.2 基于重复控制补偿的 PID 控制

基于重复控制补偿的 PID 控制系统框图如图 7-27 所示，其中 $u_p(k)$ 为 PID 控制的输出， $u_e(k)$ 为重复控制的输出。

控制算法为

$$u(k) = u_p(k) + u_e(k) \quad (7.18)$$

式中， $Q(s)$ 为低通滤波器。

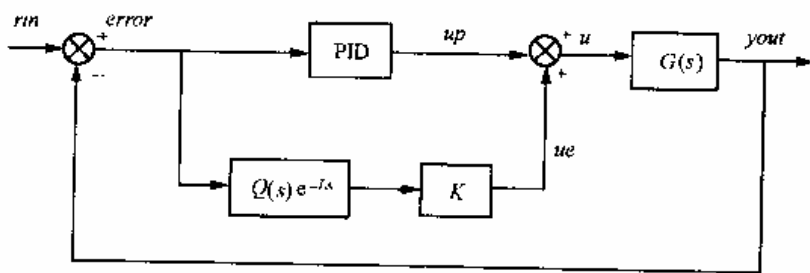


图 7-27 基于重复控制补偿的 PID 控制系统框图

7.4.3 仿真程序及分析

仿真实例

设被控对象为

$$G(s) = \frac{50}{s(0.000046s^2 + 0.006s + 1)}$$

采样时间为 1ms，输入指令信号为一正弦信号 $rin(k) = 1.0\sin(2\pi t)$ ，其中频率 $F = 1.0$ ，幅值 $A = 1.0$ 。在图 7-27 中，取 $K = 1.2$ 。

低通滤波器选取

$$q(s) = \frac{1}{0.2s+1}$$

PID 控制算法采用 PI 形式，其中 $k_p = 1.5, k_i = 10$ 。

取 $M = 1$ ，采用 PID 控制，位置跟踪及跟踪误差如图 7-28 和图 7-29 所示，取 $M = 2$ ，采用重复+PID 控制，位置跟踪、跟踪误差及控制器输出如图 7-30 至图 7-32 所示。

由仿真可见，采用重复控制补偿的 PID 控制可显著提高跟踪精度。

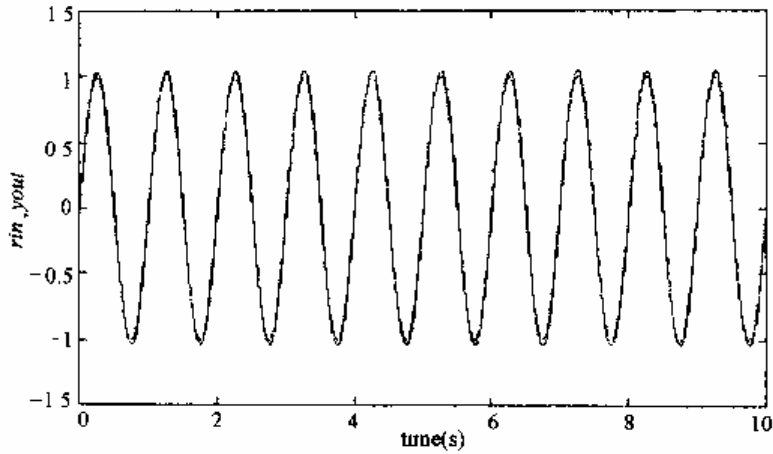


图 7-28 PID 位置跟踪 ($M = 1$)

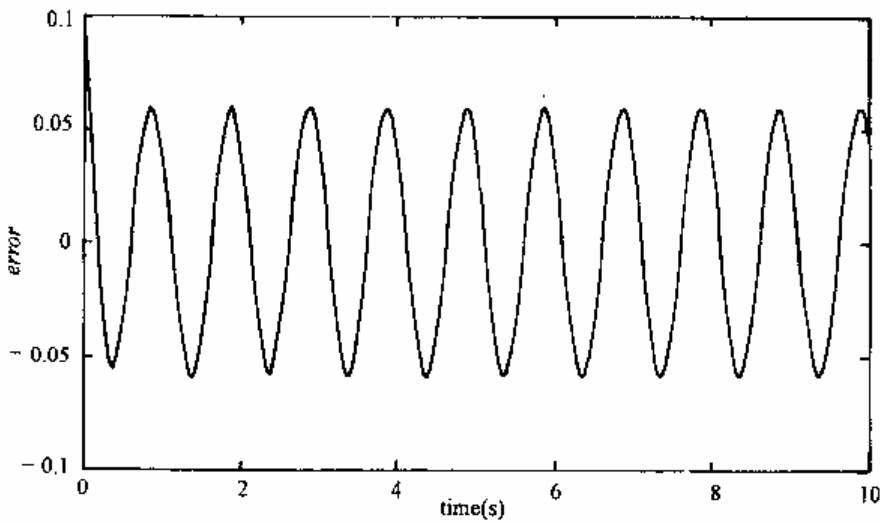


图 7-29 PID 位置跟踪误差

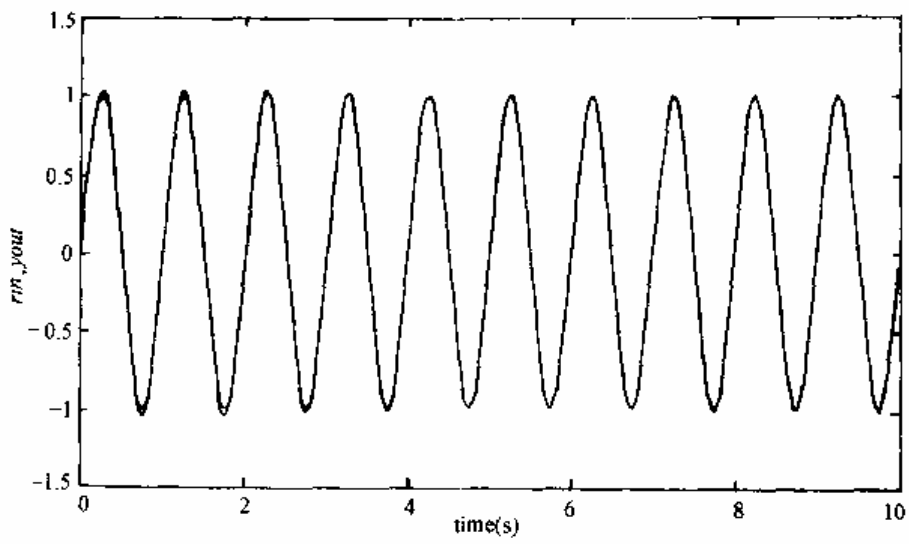


图 7-30 重复控制+PID 位置跟踪 ($M = 2$)

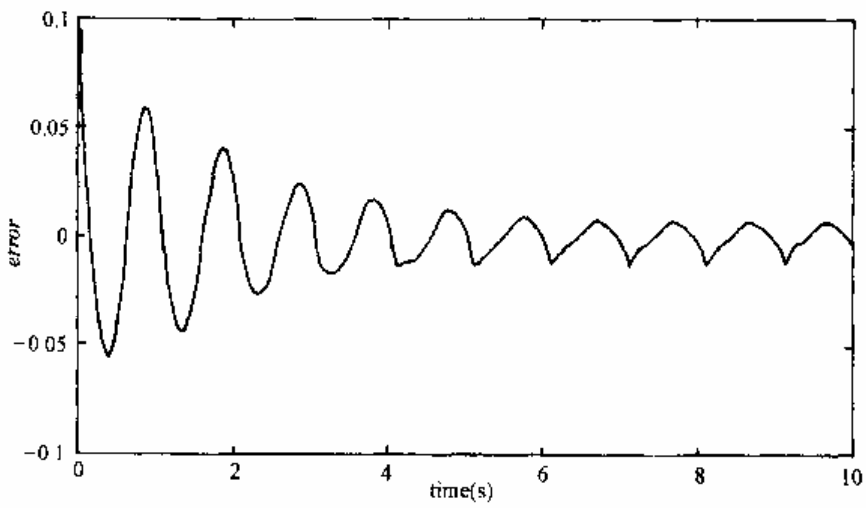


图 7-31 位置跟踪误差

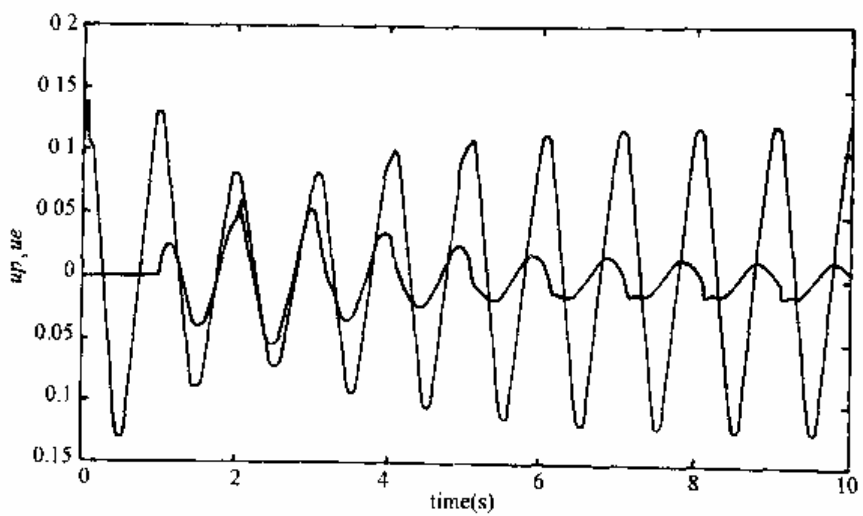


图 7-32 重复控制补偿时的控制器输出 $u_p(k)$ 和 $u_e(k)$

仿真程序: chap7_6.m

```

%PID Control with Repetitive Control compensation
clear all;close all;

ts=0.001;
sys=tf(50,[0.000046,0.006,1,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

q=tf(1,[0.20,1]); %Filter
dq=c2d(q,ts,'z');
[numq,denq]=tfdata(dq,'v');

F=1;
N=1/F*1/ts;

zz=tf([1],[1 zeros(1,N)],ts);
dz=dq*zz;
[numz,denz]=tfdata(dz,'v');

Gr=1/(1-dz); %Repetitive controller

u_1=0;u_2=0;u_3=0;
y_1=0;y_2=0;y_3=0;

ei=0;
ue_1=0;ue_2=0;
ue_N=0;ue_N1=0;ue_N2=0;
e2_N=0;e2_N1=0;e2_N2=0;

e_N1=0;
e1_1=0;
e2_1=0;

for k=1:1:10000
time(k)=k*ts;

rin(k)=1.0*sin(F*2*pi*k*ts);

yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

```

```

e(k)=rin(k)-yout(k);

ei=ei+e(k)*ts;
up(k)=1.5*e(k)+10*ei;

e1(k)=-denq(2)*e1_1+numq(2)*e_N1;
K=1.2;
e2(k)=K*e1(k);

ue(k)=0.8187*ue_1+0.1813*ue_N1+e2(k)-0.8187*e2_1;

M=2;
if M==1
    u(k)=up(k);      %Only using PID
end
if M==2
    u(k)=ue(k)+up(k); %Using REP+PID
end

if k>N
    ue_N=ue(k-N);
    e2_N=e2(k-N);
end
if k>N+1
    ue_N1=ue(k-N-1);
    e2_N1=e2(k-N-1);
    e_N1=e(k-N-1);
end
if k>N+2
    ue_N2=ue(k-N-2);
    e2_N2=e2(k-N-2);
end

e1_1=e1(k);
e2_1=e2(k);
ue_2=ue_1;
ue_1=ue(k);

u_3=u_2;u_2=u_1;u_1=u(k);

```

```

y_3=y_2;y_2=y_1;y_1=yout(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');ylabel('error');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);
plot(time,up,'r',time,ue,'b');
xlabel('time(s)');ylabel('up,ue');

```

7.5 基于零相差前馈补偿的 PID 控制

7.5.1 零相差控制原理

在伺服系统设计中，前馈控制可用于提高系统的跟踪性能，拓宽系统的频带。通常，前馈控制是基于不变性原理，即将前馈控制环节设计成待校正的闭环系统的逆，使校正后系统的传递函数为 1。但当闭环系统为非最小相位系统时，这种方法就不实用了，因为由不变性原理，非最小相位系统的不稳定零点变为前馈控制器的极点，则前馈环节是不稳定的。在实际应用中，随着采样频率的提高，很多最小相位连续系统经零阶保持器离散化后所得的离散系统为非最小相位系统。零相差跟踪控制是一种针对非最小相位系统的数字式前馈控制器。它通过在前馈控制器中引入零点来补偿闭环系统的不稳定零点，在指令超前值为已知时，校正后的系统在全频域范围内相移为零。

考虑如图 7-33 所示的单输入、单输出闭环 PID 控制系统，设 $G_c(z^{-1})$ 为稳定的离散对象，可视为连续的闭环系统经 ZOH 离散化而得， $G_c(z^{-1})$ 具有 s 个非最小相位零点，其静态增益为 1，则 $G_c(z^{-1})$ 可表示为

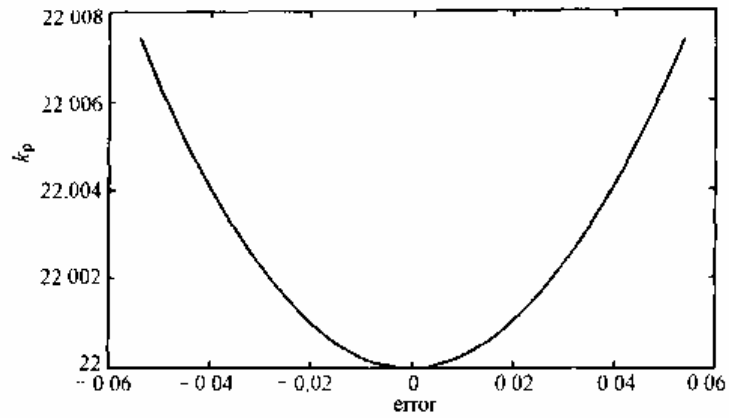
$$G_c(z^{-1}) = \frac{z^{-d} N_u(z^{-1}) N_a(z^{-1})}{D(z^{-1})} \quad (7.19)$$

式中， $N_u(z^{-1}) = n_0 + n_1 z^{-1} + \dots + n_s z^{-s}$, $n_0 \neq 0$ ， $N_u(z^{-1})$ 包含了 $G_c(z^{-1})$ 中所有位于单位圆上及单位圆外的不稳定零点， $N_a(z^{-1})$ 包含了 $G_c(z^{-1})$ 中所有位于单位圆内的稳定零点。

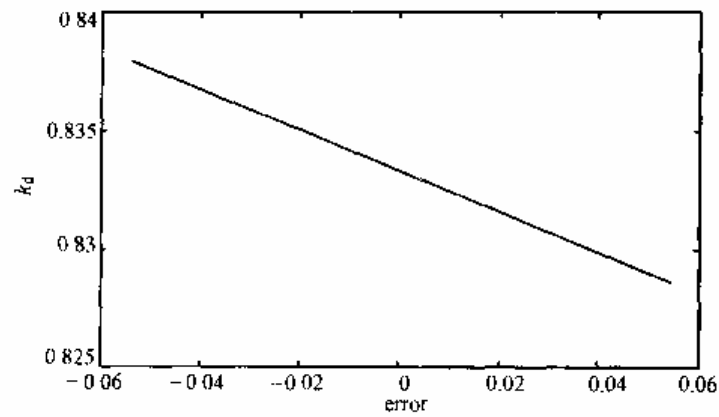
定理 1 设 $H(z^{-1}) = N_u(z) N_u(z^{-1})$ ，则

$$(1) \angle H(e^{-j\omega T}) = 0 \quad \forall \omega \in R$$

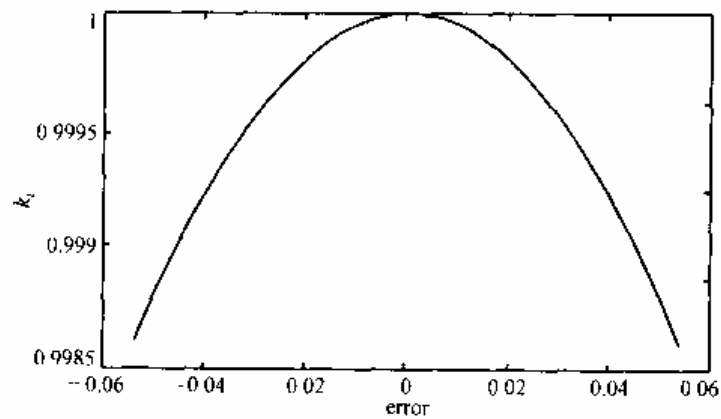
$$(2) \left| \angle H(e^{-j\omega T}) \right|^2 = \text{Re}^2 [N_u(e^{-j\omega T})] + \text{Im}^2 [N_u(e^{-j\omega T})] \quad \forall \omega \in R$$



(a) k_p 变化曲线



(b) k_d 变化曲线



(c) k_i 变化曲线

图 7-33 单输入、输出闭环 PID 控制系统 $G_c(s)$

$$F_u(z^{-1}) = \frac{z^{-s} N_u(z) D(z^{-1})}{N_u(z^{-1}) [N_u(1)]^2} \quad (7.20)$$

则系统的实际输出为

$$y(k) = F_u(z^{-1}) G_c(z^{-1}) r(k+s+d) = \frac{z^{-(s+d)} N_u(z) N_u(z^{-1})}{N_u^2(1)} r(k+s+d)$$

假设

$$G(z^{-1}) = \frac{N_u(z) N_u(z^{-1})}{N_u^2(1)} \quad (7.21)$$

显然 $F_u(z^{-1})$ 与 $G(z^{-1})$ 有相同的频率特性。由定理 1 可得

$$\angle G(e^{-j\omega T}) = 0 \quad \forall \omega \in R \quad (7.22)$$

$$G(1) = 1 \quad (7.23)$$

由此可知：如果理想输出超前值 $r(k+1), \dots, r(k+d+s)$ 已知，则系统在整个频域范围内相移为零，此时系统的输出为序列 $\{r(k-s-d), \wedge, r(k), \wedge, r(k+s+d)\}$ 的移动平均值。

7.5.2 基于零相差前馈补偿的 PID 控制

基于零相差前馈的 PID 控制系统如图 7-34 所示，设计步骤如下：

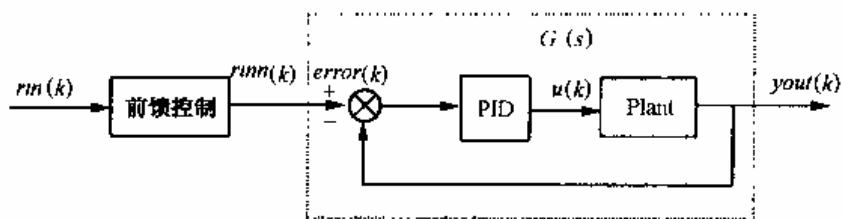


图 7-34 基于零相位前馈的 PID 控制系统结构

(1) 扫频测试，得到闭环系统的幅频、相频特性

取输入信号为正弦信号进行闭环 PID 扫频测试，测量不同频率下的相差和幅差。

设理想输入信号为

$$rin(k) = A \sin(2\pi Ft) \quad (7.24)$$

扫频测试时频率可取为

$$F = F_{mi} + N \times F_{step} \quad (7.25)$$

式中， F_{mi} 为扫频初始频率， F_{step} 为扫频步长。

在闭环 PID 控制器中采用 P 控制方法，可满足闭环正弦位置跟踪相差和幅差为线性，其控制律为

$$\begin{aligned} e(k) &= rin(k) - yout(k) \\ u(k) &= k_p e(k) = k_p (rin(k) - yout(k)) \end{aligned} \quad (7.26)$$

通过扫频测试，并采用最小二乘法，可得到闭环系统在各个频率下的相差和幅差。最小二乘法的原理如下：

设参考信号为 $rin(k) = A_m \sin(\omega t)$ ，当系统为线性时，其输出角位置可表示为

$$\begin{aligned} yout(k) &= A_f \sin(\omega t + \varphi) = A_f \cos \varphi \sin(\omega t) + A_f \sin \varphi \cos(\omega t) \\ &= [\sin(\omega t) \quad \cos(\omega t)] \begin{bmatrix} A_f \cos \varphi \\ A_f \sin \varphi \end{bmatrix} \end{aligned} \quad (7.27)$$

在时间域内取 $t = 0, h, 2h, \dots, nh$ ，其中 $h = 0.001, n = 1000$ 。

设 $Y^T = [y(0) \quad y(h) \quad \dots \quad y(nh)]$ ， $c_1 = A_f \cos \varphi$ ， $c_2 = A_f \sin \varphi$ ，且

$$\Psi^T = \begin{bmatrix} \sin(\omega 0) & \sin(\omega h) & \dots & \sin(\omega nh) \\ \cos(\omega 0) & \cos(\omega h) & \dots & \cos(\omega nh) \end{bmatrix} \quad (7.28)$$

则可以求出 c_1, c_2 的最小二乘解为

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = (\Psi^T \Psi)^{-1} \Psi^T \psi \quad (7.29)$$

从而得到幅频和相频特性

$$20 \lg \left(\frac{\hat{A}_1}{A_m} \right) = 20 \lg \left(\frac{\sqrt{c_1^2 + c_2^2}}{A_m} \right), \quad \varphi = \text{tg}^{-1} \left(\frac{c_2}{c_1} \right) \quad (7.30)$$

(2) 实现闭环系统的拟合, 设计零相差控制器

利用 Bode 图进行拟合, 可得到闭环控制系统的传递函数 $G_c(s)$, 通过零相差原理(7.19)式至(7.23)式, 可得到零相差前馈控制器。

(3) 采用零相差前馈控制

$$\begin{aligned} error(k) &= rinn(k) - yout(k) \\ u(k) &= k_p error(k) \end{aligned} \quad (7.31)$$

式中, $rinn(k)$ 为前馈控制器的输出。

7.5.3 仿真程序及分析

被控对象为三阶传递函数

$$G_p(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

系统采样时间为 1ms。

分为三个阶段进行仿真, 分别由扫频测试(chap7_7_1.m), 闭环系统拟合及前馈控制器的设计(chap7_7_2.m), 位置跟踪(chap7_7_3.m)实现。

(1) 频率测试

设位置跟踪信号为

$$rin(k) = A \sin(2\pi Ft)$$

式中, $t = kT$, $A = 0.50$ 。初始频率取 $F_{ini} = 0.50\text{Hz}$, 扫频步长取 $F_{step} = 0.50\text{Hz}$, 取 $N = 15$, 即从 0.50Hz 到 8.0Hz 进行扫频。

闭环系统采用 P 控制器, 取 $k_p = 7.0$ 。通过扫频测试, 采用最小二乘法求得闭环系统的幅频和相频特性, 如表 7.1 所示。

表 7.1 扫频测试结果

F(Hz)	$20 \lg (A + \Delta A)/A \text{dB}$	$\Delta \varphi (^\circ)$
0.50	-0.0133	-5.1368
1.0	-0.0524	-10.2380
1.5	-0.1152	-15.2706
2.0	-0.1984	-20.2064
2.5	-0.2979	-25.0238
3.0	-0.4087	-29.7087

续表

F(Hz)	$20\lg (A + \Delta A)/A $ dB	$\Delta\varphi(^{\circ})$
3.5	0.5259	-34.2545
4.0	0.6443	-38.6623
4.5	-0.7588	-42.9394
5.0	-0.8648	-47.0995
5.5	-0.9579	-51.1614
6.0	-1.0343	55.1484
6.5	-1.0905	-59.0884
7.0	-1.1234	-63.0135
7.5	-1.1306	-66.9606
8.0	-1.1099	-70.9717

(2) 求闭环系统的拟合传递函数, 设计前馈控制器

根据表 7.1 的扫频数据, 采用 MATLAB 函数 INVREQS, 对闭环系统传递函数进行拟合

$$[B, A] = \text{INVREQS}(H, W, nb, na) \quad (7.32)$$

式中, nb 和 na 为所拟合的传递函数分子与分母的阶数, H 为在频率为 W 时理想的频率响应, B 和 A 为拟合传递函数分子与分母的系数。

取 $na = 3, nb = 1$, 则可得闭环系统的拟合传递函数 $G_c(s)$ 为

$$\text{sysx}(s) = \frac{-178s + 3.664 \times 10^5}{s^3 + 87.49s^2 + 1.029 \times 10^4 s + 3.664 \times 10^5}$$

实际闭环系统传递函数和拟合传递函数 $\text{sysx}(s)$ 的 Bode 图如图 7-35 和 7-36 所示, 将其离散化得

$$G_c(z^{-1}) = \frac{-2.665 \times 10^{-5} z^2 + 0.0002361 z + 0.0001411}{z^3 - 2.906 z^2 + 2.823 z - 0.9162}$$

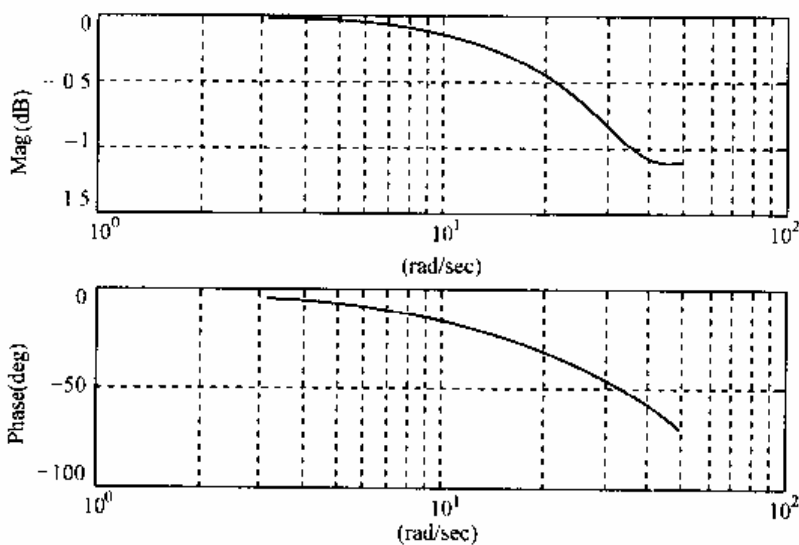


图 7-35 闭环 PID 控制系统 Bode 图拟合

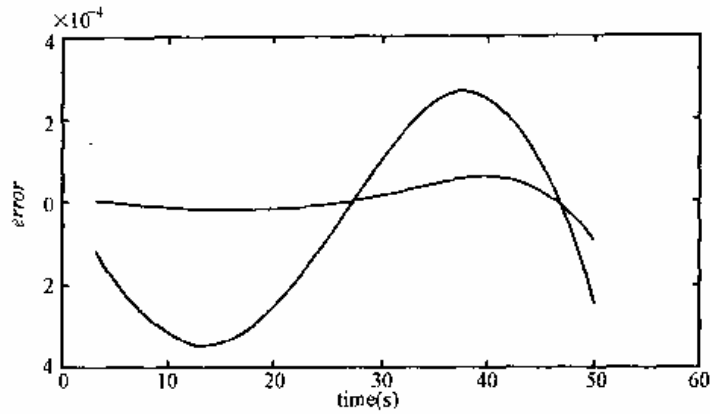


图 7-36 闭环 PID 控制系统 Bode 图拟合误差

闭环系统拟合传递函数 $G_c(z^{-1})$ 的零极点为

$$z = \begin{bmatrix} 9.42221164474527 \\ -0.56177571156134 \end{bmatrix}, \quad p = \begin{bmatrix} 0.97449754250883 + 0.08683973172465i \\ 0.97449754250883 - 0.08683973172465i \\ 0.95720489476102 \end{bmatrix}$$

可见, $G_c(z^{-1})$ 有一个不稳定零点 $z_u = 3.667$, 将该不稳定零点转为闭环系统拟合传递函数的极点, 从而形成前馈控制器的零点, 实现对闭环系统的不稳定零点的补偿, 可得前馈控制器 $F_u(z^{-1})$ 为

$$F_u(z^{-1}) = \frac{4984z^4 - 1.501 \times 10^4 z^3 + 1.561 \times 10^4 z^2 - 6060z + 484.7}{z + 0.5618}$$

由上式可知, 理想的前馈控制需要知道三个采样周期的超前值。考虑到在实际控制时理想输出超前值不知道的情况, 并且采样时间很短 (1ms), 因此可以忽略三个采样周期的超前值, 则实际的前馈控制器为

$$F(z^{-1}) = \frac{4984 - 1.501 \times 10^4 z^{-1} + 1.561 \times 10^4 z^{-2} - 6060z^{-3} + 484.7z^{-4}}{1 + 0.5618z^{-1}}$$

前馈控制器输出为

$$\begin{aligned} rinn(k) = & 4984rin(k) - 1.501 \times 10^4 rin(k-1) + 1.561 \times 10^4 rin(k-2) - 6060rin(k-3) \\ & + 484.7rin(k-4) - 0.5618rinn(k-1) \end{aligned}$$

$G_c(z^{-1})F(z^{-1})$ 的 Bode 图如图 7-37 所示, 该图表明了在一定频带范围内, 通过前馈控制可保证在一定频带范围内实现输出高精度跟踪输入。

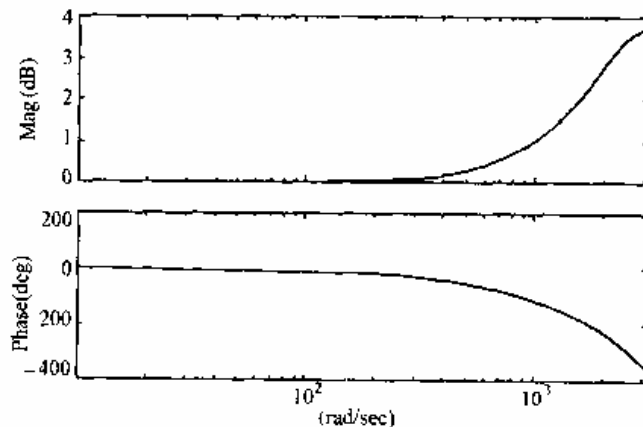


图 7-37 $G_c(z^{-1})F(z^{-1})$ 的 Bode 图

(3) 零相差控制器的验证

取采样时间为 1ms, 输入指令为正弦信号

$$rin(k) = 0.50\sin(6\pi t)$$

假设超前信号已知, 通过将 $G_c(z^{-1})F_0(z^{-1})$ 最小化并进行零极点对消可得系统的输入、输出关系为

$$yout(k) = -0.1328 rin(k+2) + 1.266 rin(k+1) - 0.1328 rin(k)$$

假设超前信号未知, 对输入信号取三个采样时间延迟, 通过将 $G_c(z^{-1})F(z^{-1})$ 最小化并进行零极点对消可得系统的输入、输出关系为

$$yout(k) = -0.1328 rin(k-1) + 1.266 rin(k-2) - 0.1328 rin(k-3)$$

通过上述输入、输出关系可以对零相差控制器进行验证, 其中 $M=1$ 时为超前信号已知, 正弦跟踪结果如图 7-38 至 7-39 所示, $M=2$ 时为超前信号未知, 正弦跟踪结果如图 7-40 至 7-41 所示。仿真结果表明, 当超前信号未知时, 位置跟踪性能下降。

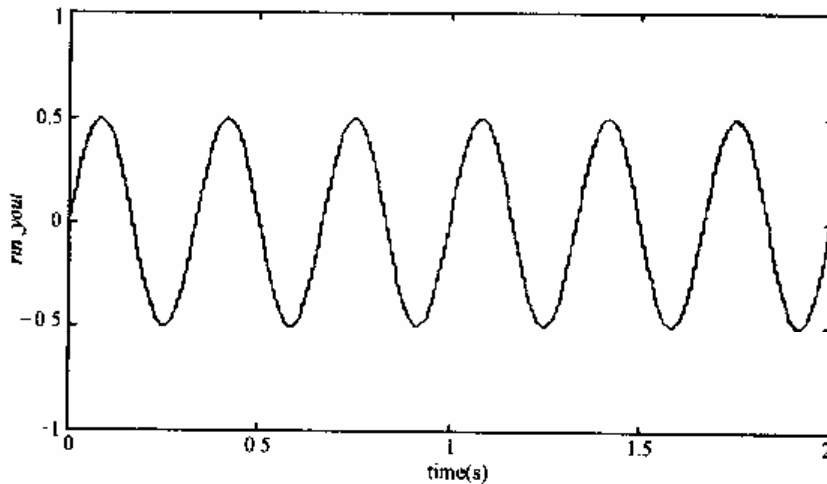


图 7-38 正弦跟踪 ($M=1$)

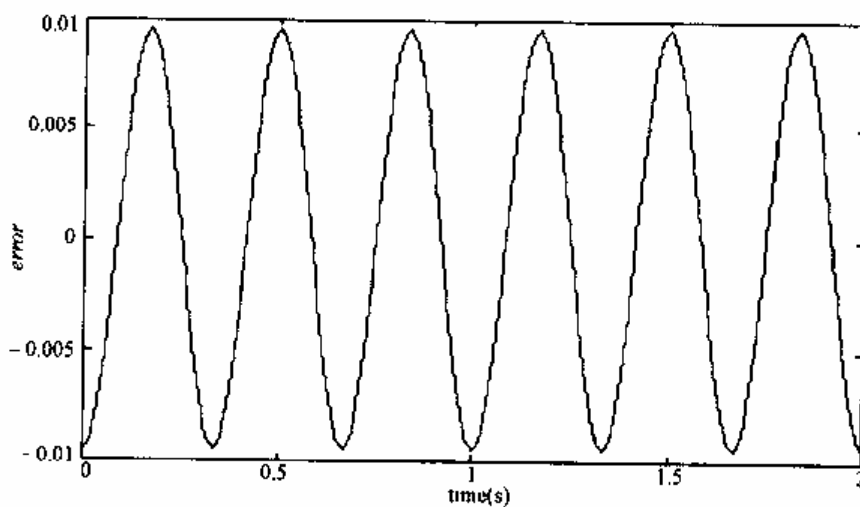


图 7-39 正弦跟踪误差 ($M=1$)

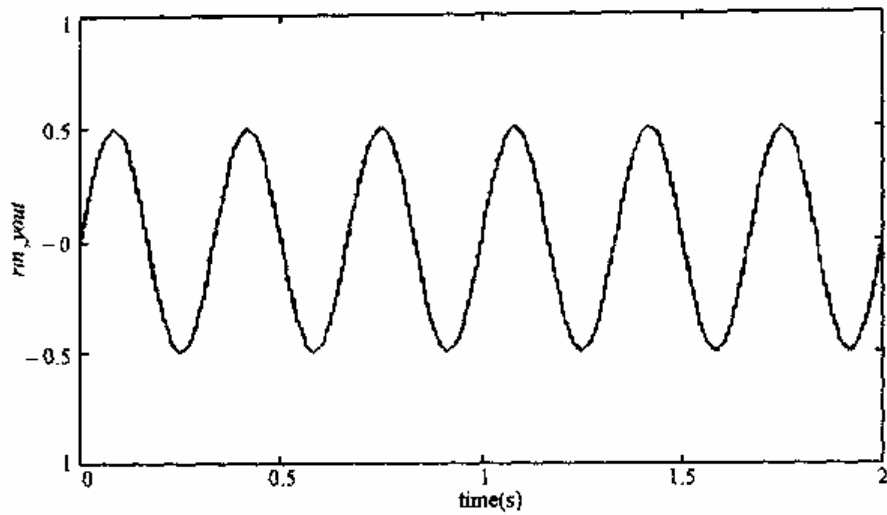


图 7-40 正弦跟踪 ($M=2$)

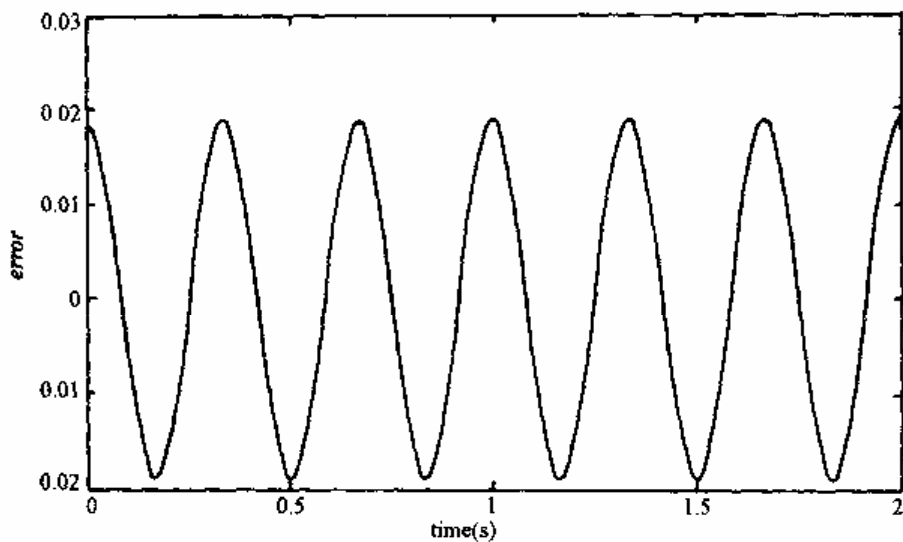


图 7-41 正弦跟踪误差 ($M=2$)

零相差控制器验证程序如下:

```
%Zero Phase Error controller verify
clear all;
close all;
ts=0.001;

rin_5=0;rin_4=0;rin_3=0;rin_2=0;rin_1=0;
F=3;
G=2000;
for k=1:1:G
time(k)=k*ts;
rin(k)=0.50*sin(F*2*pi*k*ts);
```

```

M=1;
if M==1
    rin(k+3)=0.50*sin(F*2*pi*(k+3)*ts);
    rin(k+2)=0.50*sin(F*2*pi*(k+2)*ts);
    rin(k+1)=0.50*sin(F*2*pi*(k+1)*ts);
    yout(k)=-0.1328*rin(k+2)+1.266*rin(k+1)-0.1328*rin(k);
elseif M==2
    yout(k)=-0.1328*rin_1+1.266*rin_2-0.1328*rin_3;
end

rin_5=rin_4;rin_4=rin_3;rin_3=rin_2;rin_2=rin_1;rin_1=rin(k);
end
figure(1);
plot(time(1:1:G),rin(1:1:G),'r',time(1:1:G),yout(1:1:G),'b');
figure(2);
plot(time(1:1:G),rin(1:1:G)-yout(1:1:G),'r');

```

(4) 实现位置跟踪

采用基于零相差前馈补偿的 PID 控制进行正弦位置跟踪，闭环仍采用扫频时的 P 控制。当 $S=1$ 时，输入指令为 $rin(k)=0.50\sin(3\pi Ft)$ ，取 $F=3.0$ ，正弦位置跟踪如图 7-42 所示。当 $S=2$ 时，输入指令为 $rin(k)=0.50\sin(2\pi t)+1.0\sin(6\pi t)+1.0\sin(10\pi t)$ ，正弦叠加信号位置跟踪如图 7-43 所示。

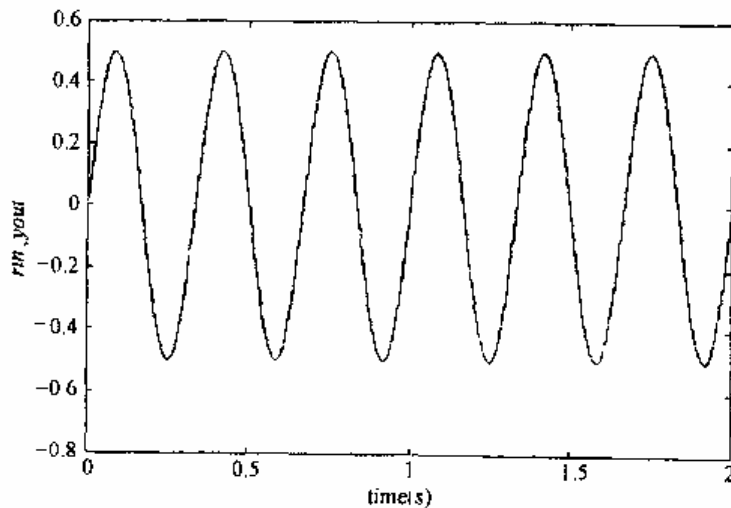


图 7-42 正弦跟踪 ($S=1$)

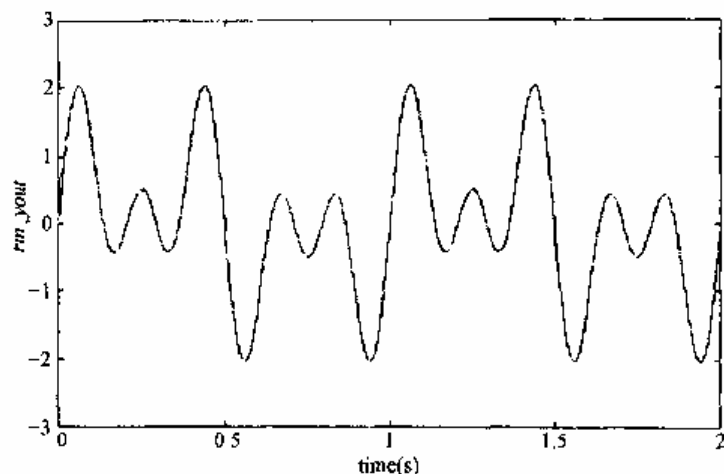


图 7-43 正弦叠加信号跟踪 ($S = 2$)

仿真程序:

程序之一: 扫频测试 chap7_7_1.m

%Zero Phase Error Frequency testing

clear all;

close all;

ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]);

dsys=c2d(sys,ts,'z');

[num,den]=tfdata(dsys,'v');

kp=0.70;

error_1=0;

kk=0; %Frequency steps

for F=0.5:0.5:8

kk=kk+1;

FF(kk)=F;

u_1=0.0;u_2=0.0;u_3=0.0;

y_1=0;y_2=0;y_3=0;

for k=1:1:2000

time(k)=k*ts;

%Tracing Sine high frequency Signal

rin(k)=0.5*sin(1*2*pi*F*k*ts);

%Linear model

```

yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);
u(k)=kp*error(k);    %P Controller

%-----Return of PID parameters-----%
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
end

plot(time,rin,'r',time,yout,'b');
pause(0.00000000000001);

Y=rin(1001:1:2000)';

for i=1:1:1000
    fai(1,i) = sin(2*pi*F*i*ts);
    fai(2,i) = cos(2*pi*F*i*ts);
end

c = inv(fai*fai')*fai*Y;

pinA = sqrt(c(1)*c(1)+c(2)*c(2));
pinF = atan(c(2)/c(1));

Y=yout(1001:1:2000)';

for i=1:1:1000
    fai(1,i) = sin(2*pi*F*i*ts);
    fai(2,i) = cos(2*pi*F*i*ts);
end

c=inv(fai*fai')*fai*Y;

poutA = sqrt(c(1)*c(1)+c(2)*c(2));
poutF = atan(c(2)/c(1));

mag(kk)=20*log10(poutA/pinA);    %Magnitude
ph(kk)=(poutF-pinF)*180/pi;    %Phase error

```



```

end
FF=FF'
mag=mag'
ph=ph'

save freq.mat FF mag ph;
save closed.mat kp;

```

程序之二：闭环系统拟合及前馈控制器设计 chap7_7_2.m

```

%Closed system approaching and zero phase error controller design
clear all;
close all;
load freq.mat;

ts=0.001;
f=FF;
for i=1:length(ph)
    if ph(i)>0 ph(i)=ph(i)-360;
    end
end

%Transfer function approaching
%(1)Freq parameters
w=2*pi*f;          %From Hz to rad/sec
mag1=10.^(mag/20); %From dB to degree
ph1=ph*pi/180;    %From degree to radian

h=mag1.*cos(ph1)+j*mag1.*sin(ph1);

%(2)Continous function
na=3; %Three ranks approaching
nb=1;
%bb and aa are real numerator and denominator of transfer function
[bb,aa]=invfreqs(h,w,nb,na); %w contains the frequency values in radians/s
display('Transfer function approaching is:');
sysx=tf(bb,aa)
[zs,ps,ks]=zpkdata(sysx,'v');

%(3)Discrete function

```

```

Gc=c2d(sysx,ts,'zoh');
zpksys=zpk(Gc);
[z,p,k]=zpkdata(zpksys,'v');    %Getting zero-pole-gain: z,p,k
display('Zeros and Poles of the Transfer function is:');
z
p
%In z-1 format
zGc=tf(Gc);
[nGc,dGc]=tfdata(zGc,'v');
zGc=filt(nGc,dGc,ts);

%(4)Magnitude and Phase to draw Bode
%Frequency response:create the complex frequency response vector: h=a+bi
h=freqs(bb,aa,w);
%Magnitude and phase
sysmag=abs(h);                %Degree
sysmag1=20*log10(sysmag);     %From degree to dB
sysph=angle(h);              %Get radian
sysph1=sysph*180/pi;         %From radian to degree

%(5)Drawing practical plant and its approach function Bode to compare
figure(1);
subplot(2,1,1);
semilogx(w,sysmag1,'r',w,mag,'b');grid on;
xlabel('Frequency(rad/sec)');ylabel('magnitude approach(dB)');

subplot(2,1,2);
semilogx(w,sysph1,'r',w,ph,'b');grid on;
xlabel('Frequency(rad/sec)');ylabel('phase approach(deg)');

figure(2);
magError=sysmag1-mag;
phError=sysph1-ph;
plot(w,phError,'r',w,magError,'b');
xlabel('Frequency(rad/sec)');ylabel('error of phase(deg) and
magnitude(dB)');

%(6)Plant in zero-pole-gain format
zu=z(1);    %Unstable zero point
z1=z(2);

```

```

p1=p;
p1(4)=1/zu;

k1=1;
Gctemp=zpk(z1,p1,k1,ts);
dc=dcgain(Gctemp);      %Getting DC gain
k1=1/dc;                %G(1)=1;

Gcn=zpk(z1,p1,k1,ts);

%(7)Design controller
Fdz=1/Gcn;      %Fdz=zpk(p1,z1,1/k1,ts);
display('ZPE Controller is:');
tfdz=tf(Fdz)    %z^(-3):three rank delay

[nn1,ddl]=tfdata(tfdz,'v');

nF=nn1;
dF(1)=ddl(4);  %z^(-3):three rank delay
dF(2)=ddl(5);
dF(3)=ddl(1);
dF(4)=ddl(2);
dF(5)=ddl(3);

format long;
display('ZPE Controller coefficient is:');
nF
dF
save zpecoeff.mat nF dF;

%Controller
F=filt(nF,dF,ts);  %Equal conversion:from z to z-1
dcgain(F);

%(8)Verify the controller
Gn=series(F,Gc);  %Gn(z)=F(z)*Gc(z)
figure(3);
bode(Gn);
yk=minreal(Gn,ts); %Simpling Gn(z)

```

程序之三：位置跟踪 chap7_7_3.m

```
%Zero Phase Error Position control
clear all;
close all;
load zpecoeff.mat; %ZPE coefficient nF and dF
load closed.mat; %Load kp

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
rin_5=0;rin_4=0;rin_3=0;rin_2=0;rin_1=0;
rinn_1=0;
y_1=0;y_2=0;y_3=0;
error_1=0;

F=3;
S=2;
for k=1:1:2000
time(k)=k*ts;

if S==1 %Sine Signal
rin(k)=0.50*sin(F*2*pi*k*ts);
elseif S==2 %Random Signal
rin(k)=0.50*sin(1*2*pi*k*ts)+1.0*sin(3*2*pi*k*ts)+1.0*sin(5*2*pi*k*ts);
end

rinn(k)=nF(1)*rin(k)+nF(2)*rin_1+nF(3)*rin_2+nF(4)*rin_3+nF(5)*rin_4-
dF(2)*rinn_1;

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-
den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rinn(k)-yout(k);
u(k)=kp*error(k); %P Control

u_3=u_2;u_2=u_1;u_1=u(k);
rin_5=rin_4;rin_4=rin_3;rin_3=rin_2;rin_2=rin_1;rin_1=rin(k);
```

```

rinn_1=rinn(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');

```

7.6 基于卡尔曼滤波器的PID控制

7.6.1 卡尔曼滤波器原理

在现代随机最优控制和随机信号处理技术中，信号和噪声往往是多维非平稳随机过程。因其时变性，功率谱不固定。在 1960 年初提出了卡尔曼滤波理论，该理论采用时域上的递推算法在数字计算机上进行数据滤波处理。

对于离散域线性系统

$$\begin{aligned} x(k) &= Ax(k-1) + B(u(k) + w(k)) \\ y_v(k) &= Cx(k) + v(k) \end{aligned} \quad (7.33)$$

式中， $w(k)$ 为过程噪声信号， $v(k)$ 为测量噪声信号。

离散卡尔曼滤波器递推算法为

$$M_n(k) = \frac{P(k)C^T}{CP(k)C^T + R} \quad (7.34)$$

$$P(k) = AP(k-1)A^T + BQB^T \quad (7.35)$$

$$P(k) = (I_n - M_n(k)C)P(k) \quad (7.36)$$

$$x(k) = Ax(k-1) + M_n(k)(y_v(k) - CAx(k-1)) \quad (7.37)$$

$$y_c(k) = Cx(k) \quad (7.38)$$

误差的协方差为

$$errcov(k) = CP(k)C^T \quad (7.39)$$

卡尔曼滤波器结构如图 7-44 所示。

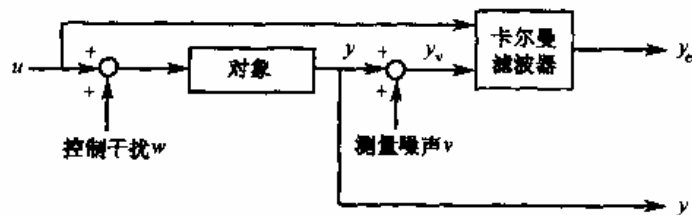


图 7-44 卡尔曼滤波器结构图

7.6.2 仿真程序及分析

仿真实例

验证卡尔曼滤波器的滤波性能。

对象为二阶传递函数

$$G_p(s) = \frac{133}{s^2 + 25s}$$

取采样时间为 1ms，采用 Tustin 变换将对象离散化，并描述为离散状态方程的形式

$$x(k+1) = Ax(k) + B(u(k) + w(k))$$

$$y(k) = Cx(k)$$

带有测量噪声的被控对象输出为

$$y_v(k) = Cx(k) + v(k)$$

式中， $A = \begin{bmatrix} 1.0000000 & 0.0009876 \\ 0.0000000 & 0.9753099 \end{bmatrix}$ ， $B = \begin{bmatrix} 0.0000659 \\ 0.1313512 \end{bmatrix}$ ， $C = [1, 0]$ ， $D = [0]$

控制干扰信号 $w(k)$ 和测量噪声信号 $v(k)$ 幅值均为 0.10 的白噪声信号，输入信号幅值为 1.0、频率为 1.5Hz 的正弦信号。采用卡尔曼滤波器实现信号的滤波，取 $Q=1$ ， $R=1$ 。仿真时间为 3s，原始信号及带有噪声的原始信号、原始信号及滤波后的信号和误差协方差的变化分别如图 7-45 至 7-47 所示。仿真结果表明，该滤波器对控制干扰和测量噪声具有很好的滤波作用。

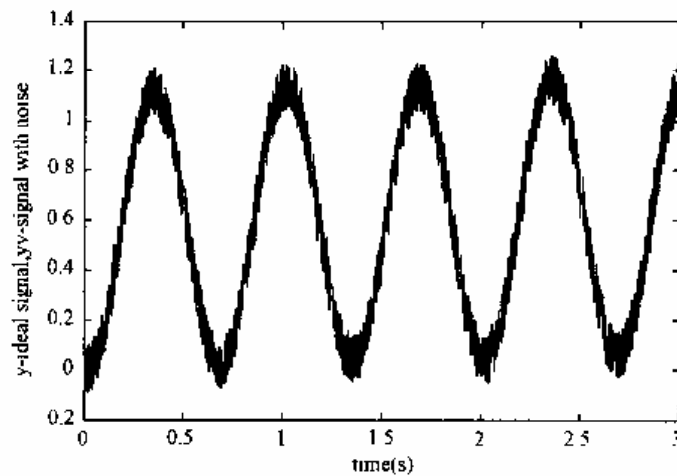


图 7-45 原始信号及带有噪声的原始信号

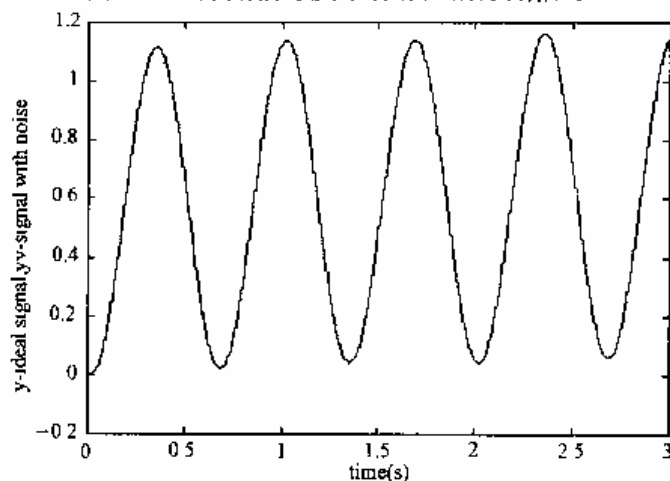


图 7-46 原始信号及滤波后的信号

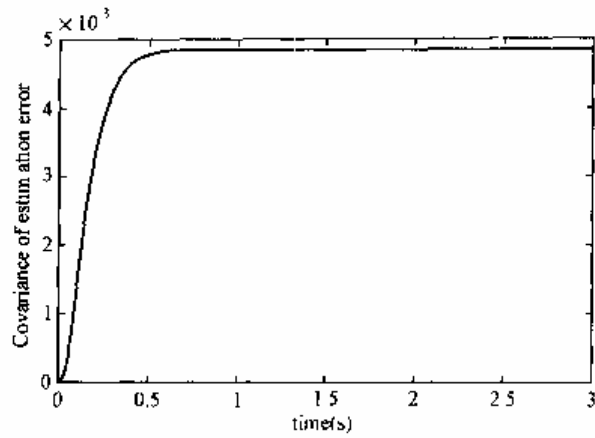


图 7-47 误差协方差的变化

```

仿真程序: chap7_8.m
%Kalman filter
%x=Ax+B(u+w(k));
%y=Cx+D+v(k)
clear all;
close all;

ts=0.001;
M=3000;

%Continuous Plant
a=25;b=133;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

A1=[0 1;0 -a];
B1=[0;b];
C1=[1 0];
D1=[0];
[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

Q=1;          %Covariances of w
R=1;          %Covariances of v

P=B*Q*B';    %Initial error covariance
x=zeros(2,1); %Initial condition on the state

```

```

ye=zeros(M,1);
ycov=zeros(M,1);

u_1=0;u_2=0;
y_1=0;y_2=0;

for k=1:1:M
time(k)=k*ts;

w(k)=0.10*rands(1); %Process noise on u
v(k)=0.10*rands(1); %Measurement noise on y

u(k)=1.0*sin(2*pi*1.5*k*ts);
u(k)=u(k)+w(k);

y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
yv(k)=y(k)+v(k);

%Measurement update
Mn=P*C'/(C*P*C'+R);
P=A*P*A'+B*Q*B';
P=(eye(2)-Mn*C)*P;

x=A*x+Mn*(yv(k)-C*A*x);
ye(k)=C*x+D; %Filtered value

errcov(k)=C*P*C'; %Covariance of estimation error

%Time update
x=A*x+B*u(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=ye(k);
end
figure(1);
plot(time,y,'k',time,yv,'k');
xlabel('time(s)');
ylabel('y-ideal signal; yv-signal with noise')

```



```

figure(2);
plot(time,y,'k',time,ye,'k');
xlabel('time(s)');
ylabel('y-ideal signal; ye-filtered signal')

figure(3);
plot(time,errcov,'k');
xlabel('time(s)');
ylabel('Covariance of estimation error');

```

7.6.3 基于卡尔曼滤波器的PID控制

基于卡尔曼(Kalman)滤波的PID控制系统结构如图7-48所示。

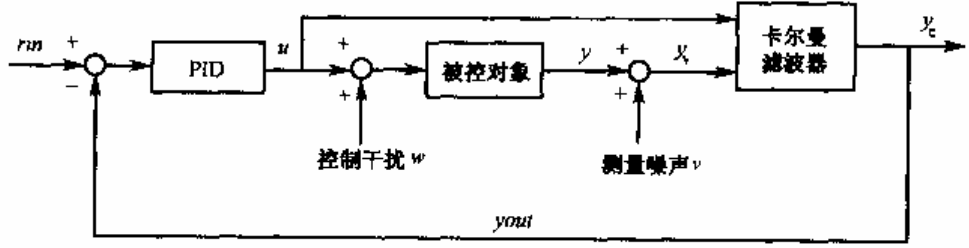


图 7-48 基于卡尔曼滤波的PID控制

7.6.4 仿真程序及分析

仿真实例

采用卡尔曼滤波器的PID控制。
被控对象为二阶传递函数

$$G_p(s) = \frac{133}{s^2 + 25s}$$

离散化结果与7.6.2节的仿真实例相同。采样时间为1ms。

控制干扰信号 $w(k)$ 和测量噪声信号 $v(k)$ 幅值均为 0.002 的白噪声信号，输入信号为一阶跃信号。采用卡尔曼滤波器实现信号的滤波，取 $Q=1$, $R=1$ 。仿真时间为 1s。分两种情况进行仿真： $M=1$ 时为未加滤波， $M=2$ 时为加滤波。在PID控制器中，取 $k_p=8.0, k_i=0.80, k_d=0.20$ 。加入滤波器前后PID阶跃响应如图7-49至图7-50所示，仿真结果表明，通过采用滤波器使控制效果明显改善。

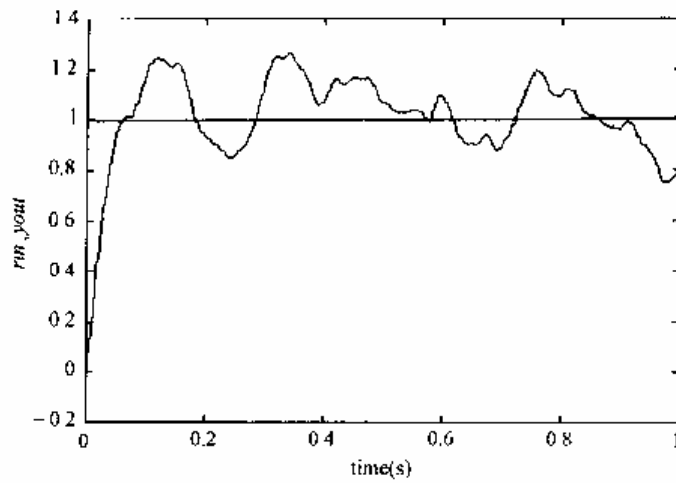


图 7-49 无滤波器时 PID 控制阶跃响应 ($M=1$)

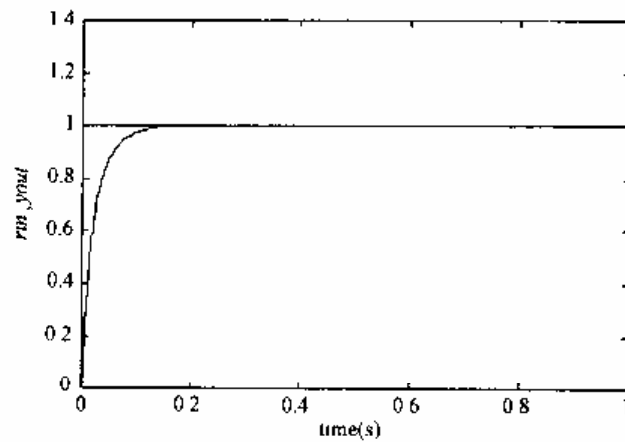


图 7-50 加入滤波器后 PID 控制阶跃响应 ($M=2$)

仿真程序: chap7_9.m

```
%Discrete Kalman filter for PID control
```

```
%Reference kalman_2rank.m
```

```
% $x=Ax+B(u+w(k))$ ;
```

```
% $y=Cx+D+v(k)$ 
```

```
clear all;
```

```
close all;
```

```
ts=0.001;
```

```
%Continuous Plant
```

```
a=25;b=133;
```

```
sys=tf(b,[1,a,0]);
```

```
dsys=c2d(sys,ts,'z');
```

```
[num,den]=tfdata(dsys,'v');
```

```
A1=[0 1;0 -a];
```

```
B1=[0;b];
```

```

C1=[1 0];
D1=[0];
[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

Q=1;           %Covariances of w
R=1;           %Covariances of v

P=B*Q*B';      %Initial error covariance
x=zeros(2,1);  %Initial condition on the state

u_1=0;u_2=0;
y_1=0;y_2=0;
ei=0;
error_1=0;
for k=1:1:1000
time(k)=k*ts;

rin(k)=1;
kp=8.0;ki=0.80;kd=0.20;

w(k)=0.002*rands(1); %Process noise on u
v(k)=0.002*rands(1); %Measurement noise on y

y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
yv(k)=y(k)+v(k);

%Measurement update
Mn=P*C'/(C*P*C'+R);
P=A*P*A'+B*Q*B';
P=(eye(2)-Mn*C)*P;

x=A*x+Mn*(yv(k)-C*A*x);
ye(k)=C*x+D; %Filtered value

M=1;
if M==1 %Not using filter
yout(k)=yv(k);
elseif M==2 %Using filter
yout(k)=ye(k);
end

```

```

error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

u(k)=kp*error(k)+ki*ei+kd*(error(k)-error_1)/ts;    %PID
u(k)=u(k)+w(k);

errcov(k)=C*P*C';    %Covariance of estimation error

%Time update
x=A*x+B*u(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)');
ylabel('rin,yout');

```

7.7 单级倒立摆的 PID 控制

7.7.1 单级倒立摆建模

倒立摆系统的控制问题一直是控制研究中的一个典型问题。控制的目的是通过给小车底座施加一个力 u (控制量)，使小车停留在预定的位置，并使杆不倒下，即不超过一预先定义好的垂直偏离角度范围。图 7-51 为一级倒立摆的示意图，小车质量为 M ，摆的质量为 m ，小车位置为 x ，摆的角度为 θ 。

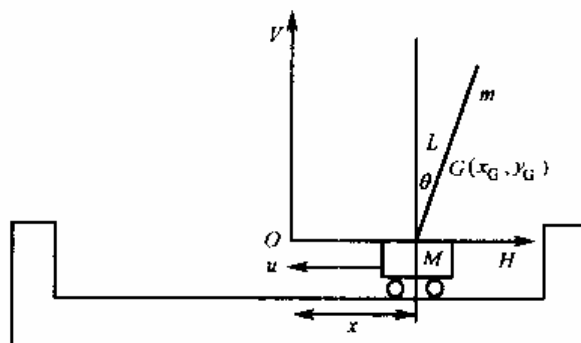


图 7-51 倒立摆系统示意图

设摆杆偏离垂直线的角度为 θ ，同时规定摆杆重心的坐标为 (x_G, y_G) ，则

$$x_G = x + l \sin \theta$$

$$y_G = l \cos \theta$$

则根据牛顿定律，建立水平和垂直运动状态方程。

摆杆围绕其重心的转动运动可用力矩方程来描述

$$I\ddot{\theta} = Vl \sin\theta - Hl \cos\theta$$

式中， I 为摆杆围绕其重心的转动惯量。

摆杆重心的水平运动由下式描述

$$m \frac{d^2}{dt^2} (x + l \sin\theta) = H$$

摆杆重心的垂直运动由下式描述

$$m \frac{d^2}{dt^2} l \cos\theta = V - mg$$

小车的水平运动由下式来描述

$$M \frac{d^2 x}{dt^2} = u - H$$

假设 θ 很小， $\sin\theta \approx \theta$ ， $\cos\theta = 1$ 。则以上各式变为

$$I\ddot{\theta} = Vl\theta - Hl \quad (7.40)$$

$$m(\ddot{x} + l\ddot{\theta}) = H \quad (7.41)$$

$$0 = V - mg \quad (7.42)$$

$$M\ddot{x} = u - H \quad (7.43)$$

由式 (7.41) 和式 (7.43) 得

$$(M + m)\ddot{x} + ml\ddot{\theta} = u \quad (7.44)$$

由式 (7.40) 和式 (7.42) 得

$$(I + ml^2)\ddot{\theta} + ml\ddot{x} = mgl\theta \quad (7.45)$$

由式 (7.44) 和式 (7.45) 可得单级倒立摆方程

$$\ddot{\theta} = \frac{m(m+M)gl}{(M+m)I + Mml^2} \theta - \frac{ml}{(M+m)I + Mml^2} u \quad (7.46)$$

$$\ddot{x} = -\frac{m^2 gl^2}{(M+m)I + Mml^2} \theta + \frac{I + ml^2}{(M+m)I + Mml^2} u \quad (7.47)$$

式中， $I = \frac{1}{12} ml^2$ ， $l = \frac{1}{2}$ 。

控制指标共有 4 个，即单级倒立摆的摆角 θ 、摆速 $\dot{\theta}$ 、小车位置 x 和小车速度 \dot{x} 。将倒立摆运动方程转化为状态方程的形式。令 $x(1) = \theta$ ， $x(2) = \dot{\theta}$ ， $x(3) = x$ ， $x(4) = \dot{x}$ ，则方程 (7.46) 和方程 (7.47) 可表示为状态方程 (7.48)

$$\dot{x} = Ax + Bu \quad (7.48)$$

$$\text{式中， } A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ t_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ t_2 & 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ t_3 \\ 0 \\ t_4 \end{pmatrix}, \quad t_1 = \frac{m(m+M)gl}{(M+m)I + Mml^2}, \quad t_2 = -\frac{m^2 gl^2}{(M+m)I + Mml^2},$$

$$t_3 = -\frac{ml}{(M+m)I + Mml^2}, \quad t_4 = \frac{I + ml^2}{(M+m)I + Mml^2}.$$

7.7.2 单级倒立摆控制

对每个控制指标都选取 PD 控制方式，控制器为

$$u_i(k) = k_{p_i} e_i(k) + k_{d_i} d e_i(k) \quad (7.49)$$

式中， $e_i(k)$ 和 $d e_i(k)$ 为控制指标 i 的误差和误差变化率。

$$u(k) = \sum_{i=1}^4 u_i(k) \quad (7.50)$$

为了进行对比，采用最优控制中的 LQR 方法。该方法针对状态方程 $\dot{x} = Ax + Bu$ ，通过确定最佳控制量 $u(t) = -Kx(t)$ 的矩阵 K ，使得控制性能指标 $J = \int_0^{\infty} (x^T Qx + u^T Ru) dt$ 达到极小，其中 Q 为正定（或半正定）厄米特或实对称矩阵， R 是正定厄米特或实对称矩阵， Q 和 R 分别表示了误差和能量损耗的相对重要性， Q 中对角矩阵的各个元素分别代表各项指标误差的相对重要性。LQR 控制器的增益为

$$K = LQR(A, B, Q, R) \quad (7.51)$$

$$u(k) = -Kx \quad (7.52)$$

7.7.3 仿真程序及分析

仿真中倒立摆的参数为： $g = 9.8 \text{m/s}^2$ （重力加速度）， $M = 1.0 \text{kg}$ （小车质量）， $m = 0.1 \text{kg}$ （杆的质量）， $L = 0.5 \text{m}$ （杆的半长）， $\mu_c = 0.0005$ （小车相对于导轨的摩擦系数）， $\mu_p = 0.000002$ （杆相对于小车的摩擦系数）。 F 为作用于小车上的力，即控制器的输出，在 $[-10, +10]$ 上连续取值。

采样周期 $T = 20 \text{ms}$ ，初始条件取 $\theta(0) = -10^\circ, \dot{\theta}(0) = 0$ ， $x(0) = 0.20, \dot{x}(0) = 0$ ，期望状态为： $\theta(0) = 0, \dot{\theta}(0) = 0$ ， $x(0) = 1.0, \dot{x}(0) = 0$ ，其中摆动角度值应转变为弧度值。

取 $S = 1$ ，采用 PID 控制。由于电机控制方向必须与摆的倒动角度方向相反，故控制器参数选负数，PID 控制参数选取见主程序。使用 PID 时倒立摆响应结果及控制器输出如图 7-52 和 7-53 所示，可见，采用 PID 控制可实现单级倒立摆的控制，但 PID 控制器的参数较难选取。

取 $S = 2$ ，采用 LQR 控制，取 $Q = \begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ ， $R = 0.10$ ，则由式 (7.51) 可得 LQR

控制器增益 $K = (-64.0799, -14.2285, -3.1623, -6.6632)$ 。

仿真程序由两部分组成：(1) 主程序 chap7_10.m；(2) 子程序 chap7_10f.m。

采用 LQR 时倒立摆响应结果及控制器输出如图 7-54 和图 7-55 所示，可见，采用 LQR 可实现单级倒立摆的最优控制。

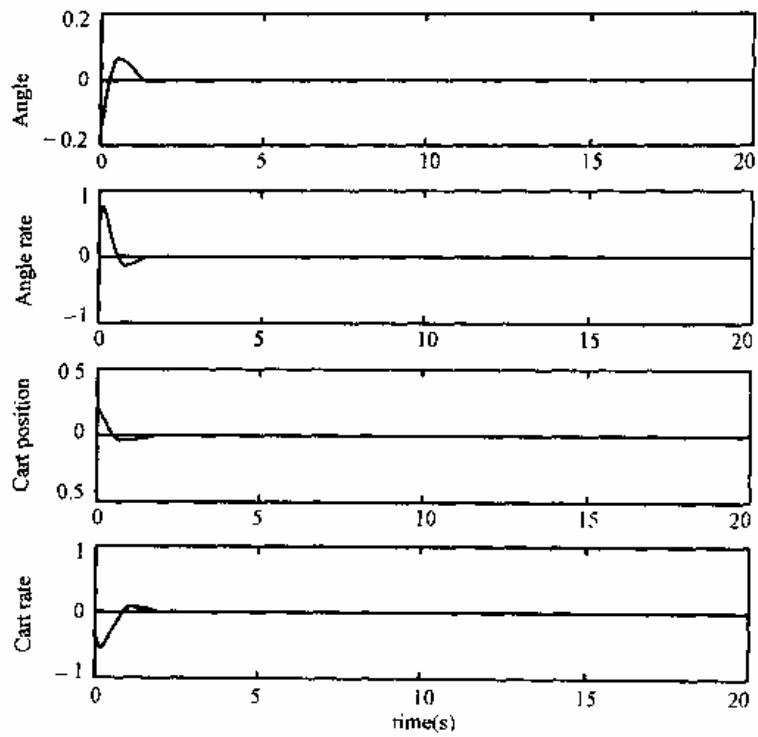


图 7-52 采用 PID 倒立摆响应结果

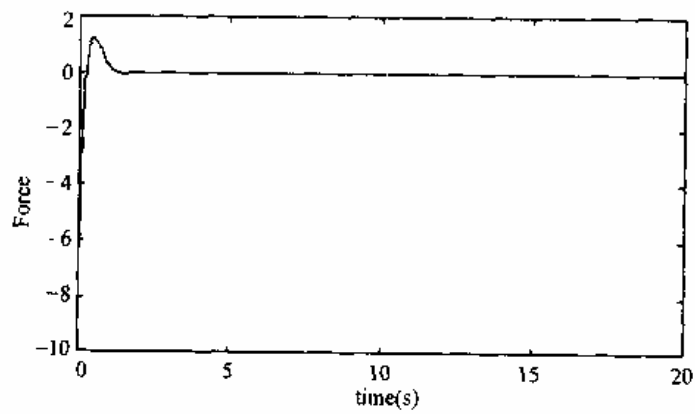


图 7-53 PID 控制器的输出

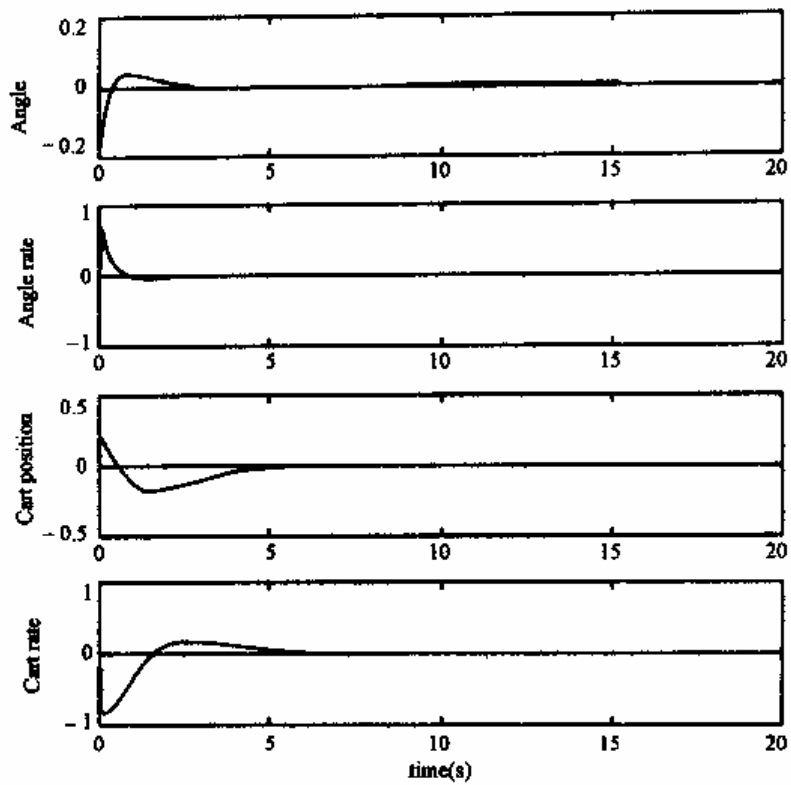


图 7-54 采用 LQR 倒立摆响应结果

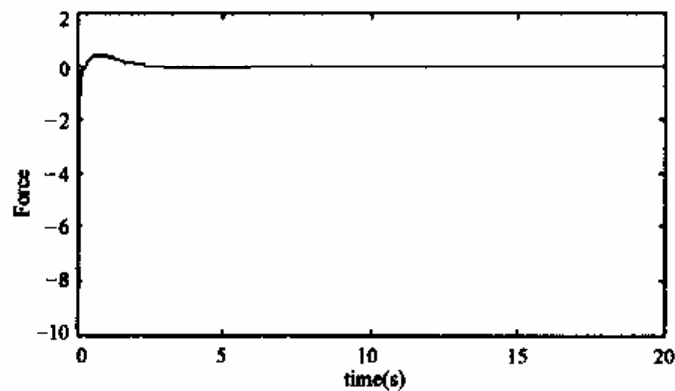


图 7-55 LQR 控制器的输出

仿真程序:

主程序: chap7_10.m

```
%Single Link Inverted Pendulum Control
```

```
clear all;
```

```
close all;
```

```
global A B C D
```

```
%Single Link Inverted Pendulum Parameters
```

```
g=9.8;
```

```
M=1.0;
```

```
m=0.1;
```

```
L=0.5;
```



```

Fc=0.0005;
Fp=0.000002;

I=1/12*m*L^2;
l=1/2*L;
t1=m*(M+m)*g*l/[(M+m)*I+M*m*l^2];
t2=-m^2*g*l^2/[(m+M)*I+M*m*l^2];
t3=-m*l/[(M+m)*I+M*m*l^2];
t4=(I+m*l^2)/[(m+M)*I+M*m*l^2];

A=[0,1,0,0;
   t1,0,0,0;
   0,0,0,1;
   t2,0,0,0];
B=[0;t3;0;t4];
C=[1,0,0,0;
   0,0,1,0];
D=[0;0];

Q=[100,0,0,0; %100,10,1,1 express importance of theta,dtheta,x,dx
   0,10,0,0;
   0,0,1,0;
   0,0,0,1];
R=[0.1];
K=LQR(A,B,Q,R); %LQR Gain

e1_1=0;e2_1=0;e3_1=0;e4_1=0;
u_1=0;
xk=[-10/57.3,0,0.20,0]; %Initial state

ts=0.02;
for k=1:1:1000
time(k)=k*ts;
Tspan=[0 ts];

para=u_1;
[t,x]=ode45('chap7_10f',Tspan,xk,[],para);
xk=x(length(x),:);

r1(k)=0.0; %Pendulum Angle

```

```

r2(k)=0.0;    %Pendulum Angle Rate
r3(k)=0.0;    %Car Position
r4(k)=0.0;    %Car Position Rate

x1(k)=xk(1);
x2(k)=xk(2);
x3(k)=xk(3);
x4(k)=xk(4);

e1(k)=r1(k)-x1(k);
e2(k)=r2(k)-x2(k);
e3(k)=r3(k)-x3(k);
e4(k)=r4(k)-x4(k);

S=1;
if S==1      %LQR
    u(k)=K(1)*e1(k)+K(2)*e2(k)+K(3)*e3(k)+K(4)*e4(k);
elseif S==2 %PD
    de1(k)=e1(k)-e1_1;
    u1(k)=-50*e1(k)-10*de1(k);
    de2(k)=e2(k)-e2_1;
    u2(k)=-10*e2(k)-10*de2(k);
    de3(k)=e3(k)-e3_1;
    u3(k)=-10*e3(k)-10*de3(k);
    de4(k)=e4(k)-e4_1;
    u4(k)=-10*e4(k)-10*de4(k);
    u(k)=u1(k)+u2(k)+u3(k)+u4(k);
end

if u(k)>=10
    u(k)=10;
elseif u(k)<=-10
    u(k)=-10;
end

e1_1=e1(k);
e2_1=e2(k);
e3_1=e3(k);
e4_1=e4(k);
u_1=u(k);
end

```

```

figure(1);
subplot(411);
plot(time,r1,'k',time,x1,'k');    %Pendulum Angle
xlabel('time(s)');ylabel('Angle');
subplot(412);
plot(time,r2,'k',time,x2,'k');    %Pendulum Angle Rate
xlabel('time(s)');ylabel('Angle rate');
subplot(413);
plot(time,r3,'k',time,x3,'k');    %Car Position
xlabel('time(s)');ylabel('Cart position');
subplot(414);
plot(time,r4,'k',time,x4,'k');    %Car Position Rate
xlabel('time(s)');ylabel('Cart rate');
figure(5);
plot(time,u,'k');                %Force F change
xlabel('time(s)');ylabel('Force');

```

子程序: chap7_10f.m

```

function dx=dym(t,x,flag,para)
global A B C D
u=para;
dx=zeros(4,1);

%State equation for one link inverted pendulum
dx=A*x+B*u;

```

7.8 吊车-双摆系统的控制

7.8.1 吊车-双摆系统的建模

实际的吊车要求将货物尽可能快地运送到目的地,并在移动过程中不能有大的晃动,这就要求吊车在移动过程中保持上下摆角的平稳而且小车本身又要达到指定的位置,这些要求可通过电机的控制来实现。吊车-双摆系统如图 7-56 所示。

图中标参数为: M : 吊车质量; m_1 : 上摆质量; m_2 : 下摆质量; x : 小车位置; β : 下摆角; α : 上摆角; l_1 : 上摆杆长度; l_2 : 下摆杆长度; F : 拉小车的力;

通过受力分析,进行线性化处理,令 $x = [x \ \dot{x} \ \alpha \ \dot{\alpha} \ \beta \ \dot{\beta}]$, 可得到关于小车、上摆角、下摆角的状态方程

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{-KeKt}{Ra(Mr^2+J)} & \frac{-(m_1+m_2)gr^2}{Mr^2+J} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{-KeKt}{Ra(Mr^2+J)} & \frac{-[(M+m_1+m_2)r^2+J]g}{(Mr^2+J)l_1} & 0 & \frac{m_2g}{m_1l_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{KeKt}{Ra(Mr^2+J)} & \frac{[(M+m_1+m_2)r^2+J]g}{(Mr^2+J)l_1} & 0 & \frac{-(m_2l_2+l_1m_1+l_1m_2)g}{m_1l_1l_2} & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{rKt}{Ra(Mr^2+J)} \\ 0 \\ \frac{rKt}{Ra(Mr^2+J)l_1} \\ 0 \\ \frac{rKt}{Ra(Mr^2+J)l_1} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x \quad (7.53)$$

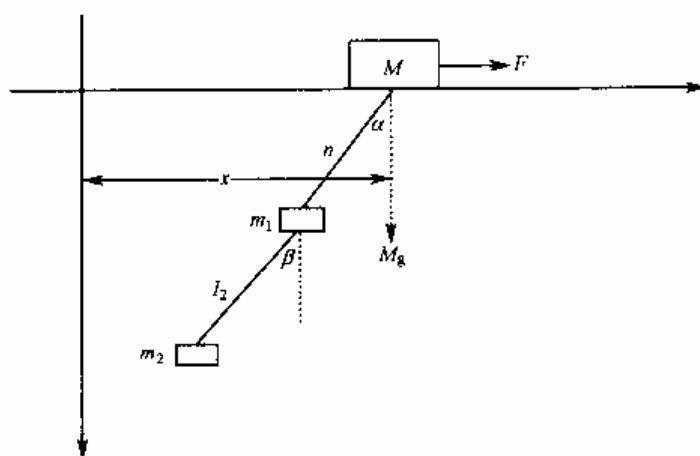


图 7-56 吊车-双摆系统

7.8.2 吊车-双摆系统的仿真

取吊车-双摆参数为：电机负载 $J=1 \times 10^{-4} \text{kg} \cdot \text{m}^2$ ，反电势系数 $Ke=0.4758 \text{V} \cdot \text{s}$ ，电枢电阻 $Ra=13.5 \Omega$ ，力矩系数 $Kt=0.0491 \text{kg} \cdot \text{m/A}$ ，传送轮半径 $r=0.02276 \text{m}$ ， $m_1=0.3 \text{kg}$ ， $m_2=0.5 \text{kg}$ ， $M=0.4 \text{kg}$ ， $l_1=0.205 \text{m}$ ， $l_2=0.156 \text{m}$ 。将上述参数带入式(7.53)，可得实际系统的状态方程

$$\dot{x} = Ax + Bu$$

$$y = Cx + D$$

式中， $A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -58.1558 & -13.3099 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -73.7445 & -112.7311 & 0 & 79.6748 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 73.7445 & 112.7311 & 0 & -247.1962 & 0 \end{bmatrix}$

$$B = [0 \ 9.48888 \ 0 \ 46.275 \ 0 \ -46.275]^T, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad D = [0 \ 0 \ 0 \ 0 \ 0 \ 0]^T.$$

连续系统的控制器采用连续系统的 S 函数来实现。在 S 函数中，只采用初始化和输出函数，即 mdlInitializeSizes 函数和 mdlOutputs 函数。在初始化中采用 sizes 结构，选择 1 个输出，7 个输入，在 7 个输入中，前 6 个输入为系统的运动状态，第 7 个输入为小车的目标位置。在输出中实现控制器，其中 S=1 时为 LQR 控制，S=2 时为 PID 控制。S 函数嵌入在 Simulink 程序中。

系统初始状态为： $x(0)=0, \dot{x}(0)=0, \alpha(0)=1.2, \dot{\alpha}(0)=0, \beta(0)=-1.2, \dot{\beta}(0)=0$ 。期望状态为： $x(0)=0.10, \dot{x}(0)=0, \alpha(0)=0, \dot{\alpha}(0)=0, \beta(0)=0, \dot{\beta}(0)=0$ ，其中角度为弧度单位。

在 Simulink 仿真结果中，角度通过弧度值乘以 $\frac{180}{\pi}$ 以度的形式表示。仿真程序由三部分组成：(1) 主程序 chap7_11.m；(2) Simulink 程序 chap7_11sim.mdl；(3) S 函数程序 chap7_11s.m。

$$\text{取 } S=1, \text{ 采用 LQR 控制, 取 } Q = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 0.10, \text{ 则由式 (7.51)}$$

可得 LQR 控制器增益为 $K = (100, 1.0379, 32.6938, 5.0696, -48.7526, 0.6557)$ 。小车及上下摆的状态响应如图 7-57 至图 7-62 所示，可见，采用 LQR 可实现吊车-双摆的最优控制。

取 $S=2$ ，采用 P 控制，控制参数选取见主程序。以小车位置响应的仿真为例，其响应结果如图 7-63 所示。可见，采用 PID 控制很难实现吊车-双摆的控制。

通过对吊车-双摆系统控制的仿真，说明了 PID 控制方法的局限性。

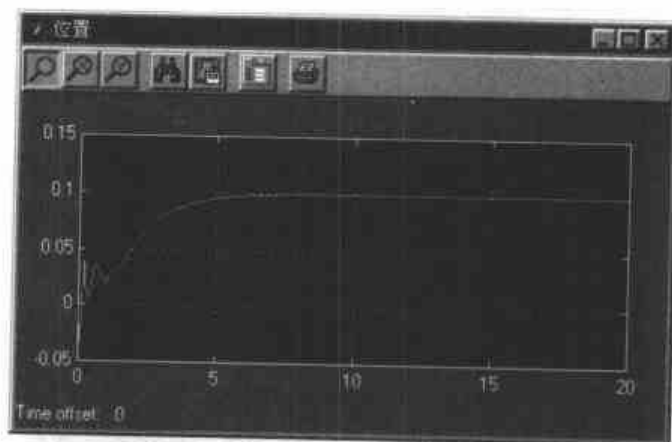


图 7.57 小车位置响应

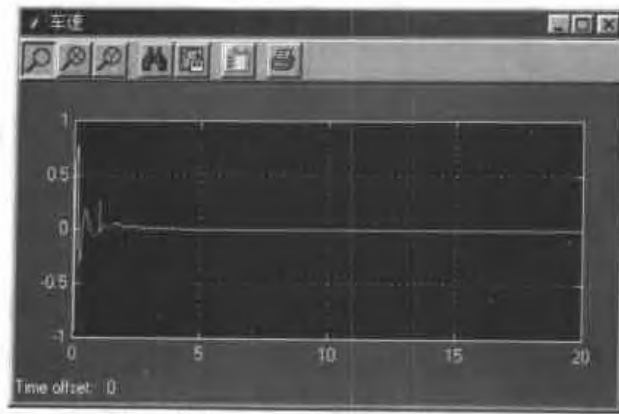


图 7.58 小车速度响应

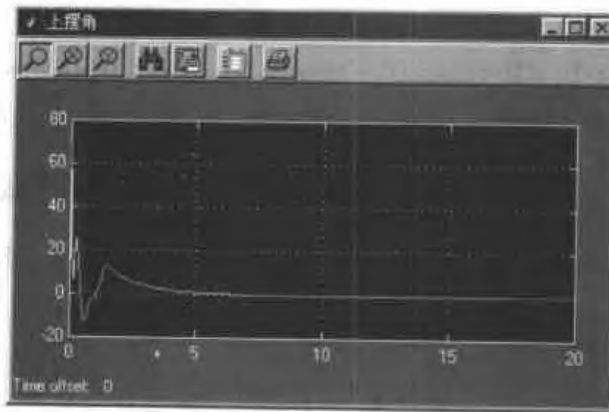


图 7.59 上摆角角度响应

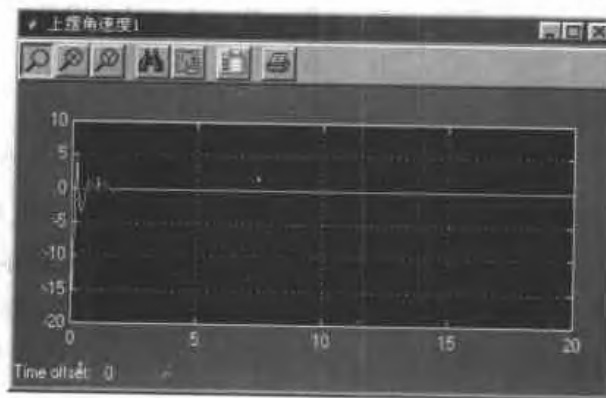


图 7.60 上摆角角速度响应

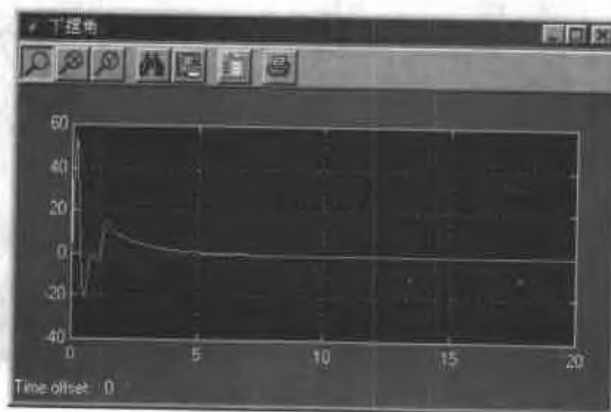


图 7.61 下摆角角度响应

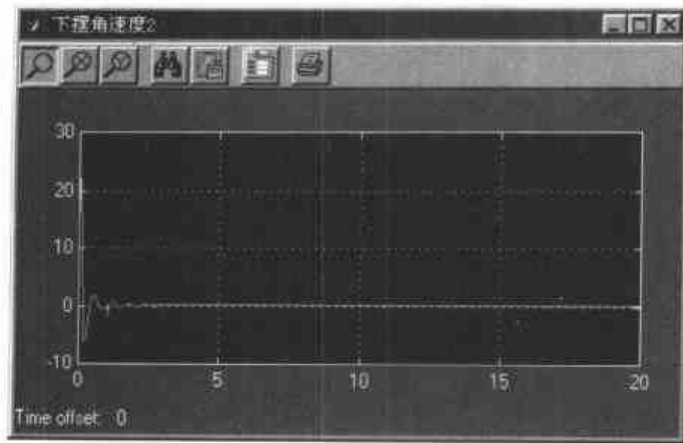


图 7.62 下摆角角速度响应

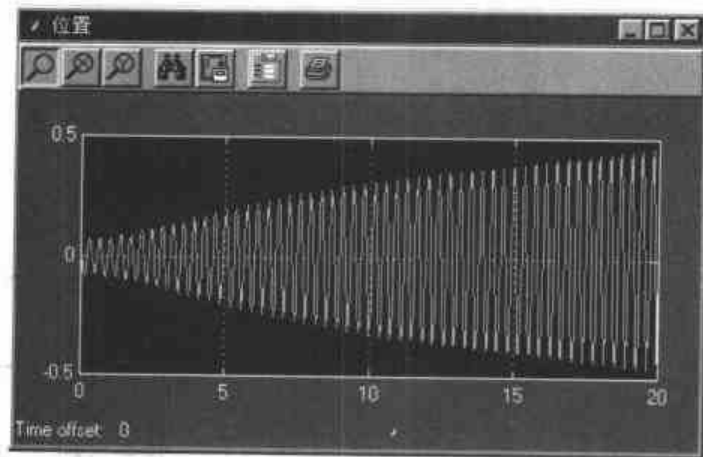


图 7.63 小车位置响应

仿真程序:

主程序: chap7_11.m

%Control for Crane Double Pendulum

clear all;

close all;

global K

```
A=[ 0  1.0000  0  0  0  0
    0 -58.1558 -13.3099  0  0  0
    0  0  0  1.0000  0  0
    0 -73.7445 -112.7311  0  79.6748  0
    0  0  0  0  0  1.0000
    0  73.7445  112.7311  0 -247.1962  0 ]
```

```
B=[0; 9.48888; 0; 46.275; 0; -46.275]
```

```
C=[1 0 0 0 0 0 ;
    0 1 0 0 0 0 ;
    0 0 1 0 0 0 ;
    0 0 0 1 0 0 ;
```



```

    case 3,
        sys=mdlOutputs(t,x,u);
%Unhandled flags
    case {1, 2, 4, 9 }
        sys = [];
%Unexpected flags
    otherwise
        error(['Unhandled flag = ',num2str(flag)]);
end

%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 7;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;

sys=simsizes(sizes);
x0=[];
str=[];
ts=[];

%mdlOutputs
function sys=mdlOutputs(t,x,u)
global K
S=1;
if S==1
    sys=-K*[u(1)-u(7),u(2),u(3),u(4),u(5),u(6)]'; %LQR
elseif S==2
    sys=50*(u(7)-u(1))+10*(0-u(2))+10*(0-u(3))+10*(0-u(4))-10*(0-
u(5))+10*(0-u(6)); %PID
end

```

第 8 章 灰色 PID 控制

8.1 灰色控制原理

部分信息已知、部分信息未知的系统为灰色系统。灰色预测是用灰色模型 GM(M,N) 进行的定量预测,灰色控制是指对本征特性灰色系统的控制,或系统中含灰参数的控制,或用 GM(M,N) 构成的预测控制。

8.1.1 生成数列

(1) 累加生成 (AGO—Accumulated Generating Operation)

如果有一串原始数列,第一个维持不变,第二个数据是原始的第一个加第二个数据,第三个数据是原始的第一个、第二个与第三个相加……。这样得到的新数列,称为累加生成数列,这种处理方式称为累加生成。

基本公式

$$x^{(1)}(k) = \sum_{i=0}^k x^{(0)}(i) \quad (8.1)$$

$$x^{(r)}(k) = x^{(r)}(k-1) + x^{(r-1)}(k) \quad (8.2)$$

累加生成是使任意非负序列、摆动的与非摆动的数据转化为非减的、递增的数据。

没有规律的原始数据,经累加生成后,如果能得到较强的规律,并且接近某一函数,则该函数称为生成函数。生成函数就是一种模型,称为生成模型。通过累加获得的模型称为累加生成模型。

(2) 累减生成

累减生成是指将原始数列前后两个数据相减所得的数据。累减生成是累加生成的逆运算,简记为 IAGO(Inverse Accumulated Generating Operation)。累减运算可将累加生成还原为非生成数列,在建模过程中用来获得增量信息。

基本公式

$$\begin{cases} a^{(1)}(x^{(r)}(k)) = x^{(r-1)}(k) \\ a^{(2)}(x^{(r)}(k)) = x^{(r-2)}(k) \\ a^{(i)}(x^{(r)}(k)) = x^{(r-i)}(k) \\ a^{(r)}(x^{(r)}(k)) = x^{(0)}(k) \end{cases} \quad (8.3)$$

换算公式

$$\begin{cases} x^{(0)}(k) = x^{(1)}(k) - x^{(1)}(k-1) \\ x^{(r-1)}(k) = x^{(r)}(k) - x^{(r)}(k-1) \end{cases} \quad (8.4)$$

8.1.2 GM 灰色模型

GM 模型即灰色模型 (Grey Model), 一般建模是用数据列建立差分方程; 灰色建模则是用原始数据列做生成后建立微分方程。由于系统被噪声污染后, 所以原始数列呈现出离乱的情况。离乱的数列即灰色数列, 或者灰色过程, 对灰色过程建立的模型, 便称为灰色模型。

灰色系统理论 (也可称为灰色理论) 其所以能够建立微分方程的模型, 是基于下述概念、观点、方式和方法:

(1) 灰色理论将随机量当作是在一定范围内变化的灰色量, 将随机过程当作是在一定范围、一定时区内变化的灰色过程。

(2) 灰色理论将无规律的原始数据经生成后, 使其变为较有规律的生成数列 (通过数列间各时刻数据的依个累加以得到新的数据和数列, 累加前的数列称为原始数列, 累加后的数列称为生成数列, 记为 AGO) 再建模。所以灰色建模实际上是生成数列模型。

(3) 灰色理论按开集拓扑定义了数列的时间测度, 进而定义信息浓度, 定义灰导数与灰微分方程。

(4) 灰色理论通过灰数的不同生成方式、数据的不同取舍、不同级别的残差 GM 模型来调整、修正、提高精度。残差是模型计算值与实际值之差。

(5) 灰色理论的模型在考虑残差 GM 模型的补充和修正后, 变成了差分、微分方程。

(6) 灰色理论的模型选择是基于关联度的概念和关联度收敛原理, 关联度收敛是一种有限范围近似的收敛。

(7) 灰色 GM 模型一般采用三种检验, 即残差大小 (或平均值、或最近一个数据的残差值) 的检验、关联度检验、后验差检验。残差大小检验是安点检验, 关联度检验是建立的模型与指定函数之间近似性的检验, 后验差检验是残差分布统计特性的检验。

(8) 对高阶系统建模, 灰色理论是通过 GM(M, N) 模型群解决的。GM 模型群即一阶微分方程组, 也可以通过多级、多次残差 GM 模型的补充修正来解决。

(9) GM 模型所得数据必须经过逆生成 (逆生成是指累减生成, 累减是指前后两个数之差。记为 IAGO) 作还原后才能用。

设 $X_1^{(0)}$ 为系统特征数据序列, $X_i^{(0)} (i=2,3,\dots,N)$ 为相关因素序列, $X_i^{(1)}$ 为诸 $X_i^{(0)} (i=1,2,\dots,n)$ 的 1_AGO 序列, 则称

$$X_1^{(1)} = b_2 X_2^{(0)} + b_3 X_3^{(0)} + \dots + b_N X_N^{(0)} + a \quad (8.5)$$

为 GM(0, N) 模型。

GM(0, N) 模型不含导数, 因此为静态模型, 它形如多元线性回归模型但与一般的多元线性回归模型有着本质的区别, 一般的多元线性回归建模以原始数据序列为基础, GM(0, N) 的建模基础则是原始数据的 1_AGO 序列。

8.2 灰色 PID 控制

8.2.1 灰色 PID 控制的理论基础

灰色系统理论是处理不确定量的一种有效途径。它需要信息少, 通用性好, 计算方便。

采用灰色系统的方法，对于不确定部分建立灰色模型，利用它来使控制系统中的灰量得到一定程度的白化，以提高控制质量及其鲁棒性。

设系统不确定部分符合匹配条件，即为 $bD(x,t)$ ，其中 $D(x,t)$ 包括两部分：一部分与状态 x 成比例，一部分与状态无关，具体可描述为

$$D(x,t) = V_1x_1 + V_2x_2 + \dots + V_nx_n + f(t) = Vx + f(t) \quad (8.6)$$

式中

$$V = (V_1 \ V_2 \ \dots \ V_n) \quad (8.7)$$

$$x^T = (x_1 \ x_2 \ \dots \ x_n) \quad (8.8)$$

设 $V_i (i=1,2,\dots,n)$ 及 $f(t)$ 均为慢时间变量，可视 V_i 及 $f(t)$ 为常数。显然，如果能辨识出 V_i 及 $f(t)$ ，则可得出 $D(x,t)$ 与 x_1, x_2, \dots, x_n 的关系，从而可估计出对应各状态 x 的不确定量 $D(x,t)$ 。灰色系统的研究方法之一，就是将原始数据进行处理，称为数的“生成”，其中累加生成由于它能弱化随机性，增强规律性，因而它在灰色系统建模中，具有特殊的地位。

令 $x^{(0)}$ 为原始的离散时间函数

$$x^{(0)} = (x^{(0)}(1), x^{(0)}(2), \dots, x^{(0)}(n)) \quad (8.9)$$

若

$$x^{(1)}(k) = \sum_{m=1}^k x^{(0)}(m) \quad (8.10)$$

则称 $x^{(1)}(k)$ 为 $x^{(0)}(k)$ 的累加生成，记为

$$AGOx^{(0)} = x^{(1)} \quad (8.11)$$

按灰色系统理论，采用累加生成方法，可建立类似于 GM(0, N) 模型的 $D(x,t)$ 灰色模型。

令离散时间函数为

$$D^{(0)} = (D(1) \ D(2) \ \dots \ D(N)) \quad (8.12)$$

$$f^{(0)} = (f(1) \ f(2) \ \dots \ f(N)) \quad (8.13)$$

$$\left. \begin{aligned} x_1^{(0)} &= (x_1(1) \ x_1(2) \ \dots \ x_1(N)) \\ x_2^{(0)} &= (x_2(1) \ x_2(2) \ \dots \ x_2(N)) \\ &\vdots \\ x_n^{(0)} &= (x_n(1) \ x_n(2) \ \dots \ x_n(N)) \end{aligned} \right\} \quad (8.14)$$

式中， $N \geq n+1$ 。

设 $D^{(1)}$ ， $f^{(1)}$ ， $x_i^{(1)} (i=1,2,\dots,n)$ 为 $D^{(0)}$ ， $f^{(0)}$ ， $x_i^{(0)} (i=1,2,\dots,n)$ 的累加生成数列。

将下述关系

$$D^{(1)}(x,t) = V_1x_1^{(1)} + V_2x_2^{(1)} + \dots + V_nx_n^{(1)} + f^{(1)} \quad (8.15)$$

称为不确定部分 $D(x,t)$ 的灰色模型。

对于慢时变干扰部分，可认为

$$\begin{aligned} f^{(1)}(1) &= f(1) = f \\ f^{(1)}(2) &= 2f(1) = 2f \\ &\vdots \\ f^{(1)}(N) &= (N)f \end{aligned} \quad (8.16)$$

记参数列为

$$\hat{V}^T = (\hat{V}_1 \hat{V}_2 \cdots \hat{V}_n \hat{f})^T \quad (8.17)$$

记数据矩阵为

$$B = \begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \quad (8.18)$$

采用最小二乘法, 若 $(B^T B)$ 可逆, 则有

$$\hat{V}^T = (B^T B)^{-1} B^T D_N^{(1)} \quad (8.19)$$

式中

$$D_N^{(1)} = (D^{(1)}(1)D^{(1)}(2)\cdots D^{(1)}(N))^T \quad (8.20)$$

将累加值还原, 可得 (8.6) 式的估计模型

$$\hat{D}(k) = \hat{V}_1 x_1(k) + \hat{V}_2 x_2(k) + \cdots + \hat{V}_n x_n(k) + \hat{f} \quad (8.21)$$

8.2.2 连续系统灰色 PID 控制

考虑由下列 N 个非线性不确定子系统组成的复合非线性不确定系统

$$\dot{x} = Ax(t) + bu(t) + bD(x, t) \quad (8.22)$$

式中, $x \in R^n, u \in R$, A 为 $n \times n$ 维矩阵, b 为 n 维矩阵, $D(x, t) \in R$ 。

$bD(x, t)$ 代表系统满足匹配条件的不确定部分, 它包括参数不确定与外干扰等。

$$D(x, t) = V_1 x_1 + V_2 x_2 + \cdots + V_n x_n + f(t) \quad (8.23)$$

采用 PID 控制

$$u(t) = k_p e(k) + k_i \sum_{k=1}^n e(k)T + k_d de(k) \quad (8.24)$$

为了减弱不确定部分的影响, 改善控制性能并提高鲁棒性, 在控制器启动过程中, 首先采用灰色估计器将不确定部分模型参数 V 粗略地估计出来, 然后对 $D(x, t)$ 加以一定程度的补偿。由于这种灰色估计不要求连续实时地进行, 故不存在通常实时辨识中存在的发散问题。

不确定部分的 $D(x)$ 无法直接测量, 可由测量数据间接计算估出。

离散化为

$$D(x, k) = \frac{1}{b} (\dot{x}(t) - Ax(t) - bu_p(t)) \quad (8.25)$$

式中, $t = kT$, T 为采样周期。

灰色估计器的具体算法如下:

第一步: 建立 $x_i^{(0)}(k)$ 原始离散数列

$$\begin{aligned} i &= 1, 2, \cdots, n \\ k &= 1, 2, \cdots, N \\ N &\geq n \end{aligned} \quad (8.26)$$

第二步：计算 $x_i^{(1)}(k)$ 累加离散数列

$$\begin{aligned} i &= 1, 2, \dots, n \\ k &= 1, 2, \dots, N \end{aligned} \quad (8.27)$$

第三步：计算

$$B = \begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \quad (8.28)$$

式中， $B^T B$ 必须可逆，若不可逆，则应适当增加 N ，直到 $B^T B$ 可逆。

第四步：根据状态 $x^{(0)}(k)$ 及 (8-25) 式，计算 $D^{(0)}(k)$ 离散数列，其中

$$k = 1, 2, \dots, N \quad (8.29)$$

第五步：计算 $D^{(1)}(k)$ 累加离散数列

$$D_N^{(1)} = (D^{(1)}(1) \ D^{(1)}(2) \ \cdots \ D^{(1)}(N)) \quad (8.30)$$

第六步：计算不确定参数估计值

$$\hat{V} = (B_b^T B_b)^{-1} B_b^T D_N^{(1)} \quad (8.31)$$

$$\hat{V} = (\hat{V}_1 \ \hat{V}_2 \ \cdots \ \hat{V}_n \ \hat{f}_D)^T \quad (8.32)$$

具有灰色估计器的 PID 控制方法分为两个阶段：

第一阶段：

采用 PID 进行控制，对不确定部分的模型参数 V 进行估计。

第二阶段：

在 N 步后，在上述控制律基础上，按估计参数 \hat{V} 加上补偿控制 u_c ，此时

$$u = u_p + u_c \quad (8.33)$$

$$u_c = - \left[\sum_{i=1}^n \hat{V}_i x_i + \hat{f} \right] \quad (8.34)$$

设

$$\tilde{D}(x, t) = \sum_{i=1}^n (V_i - \hat{V}_i) x_i + (f(t) - \hat{f}) \quad (8.35)$$

采用控制律(8.33)式，系统性能将大为改善，鲁棒性大为提高。

8.2.3 仿真程序及分析

仿真实例

被控对象为

$$G(s) = \frac{133}{s(s+25)}$$

将该传递函数转化为状态方程的形式

$$\dot{x} = Ax + bu + bD(x, t)$$

式中, $A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}$, $b = \begin{bmatrix} 0 \\ 133 \end{bmatrix}$ 。

外加干扰为

$$D(x,t) = V_1 x_1 + V_2 x_2 + f$$

取干扰参数 $V = [5.0, 5.0, 5.0]$, 采用连续系统灰色 PID 预测, 经过三个采样时间, 得到干扰参数估计结果

$$\tilde{V} = [4.6421, 5.0129, 5.0597]$$

取 $M = 1$, 不采用灰色预估补偿, PID 控制跟踪误差及变化率如图 8-1 和 8-2 所示, 取 $M = 2$, 采用灰色预估补偿, PID 跟踪误差及变化率如图 8-3 和 8-4 所示。

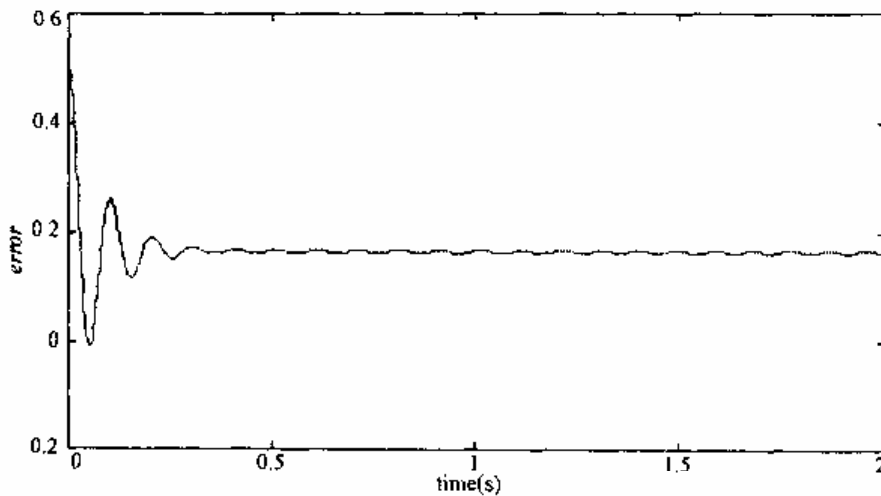


图 8-1 PID 跟踪误差 ($M = 1$)

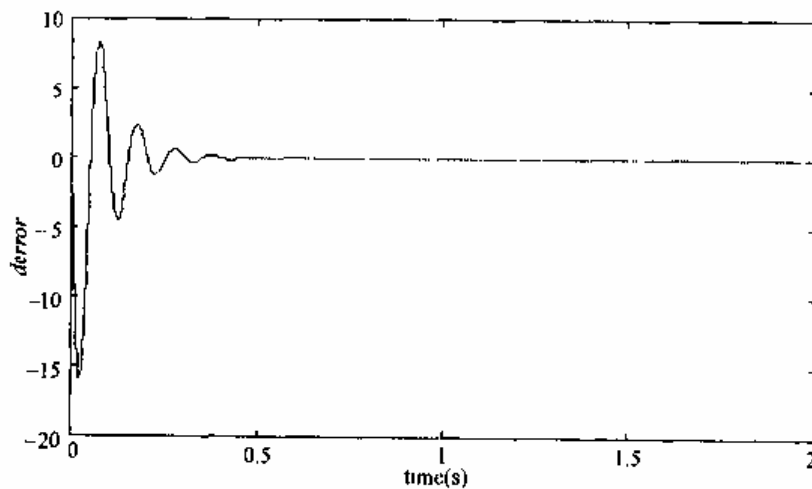


图 8-2 PID 跟踪误差变化率

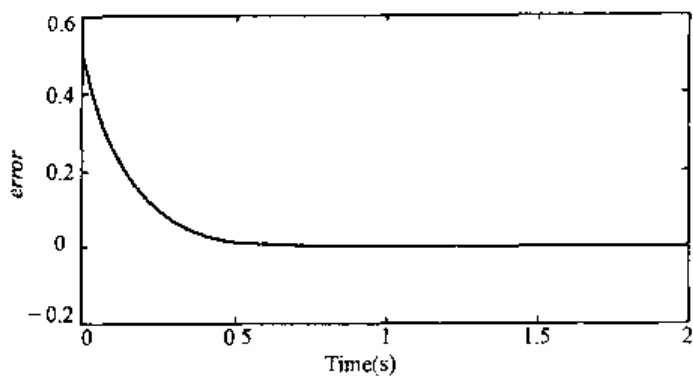


图 8-3 灰色 PID 跟踪误差 ($M = 2$)

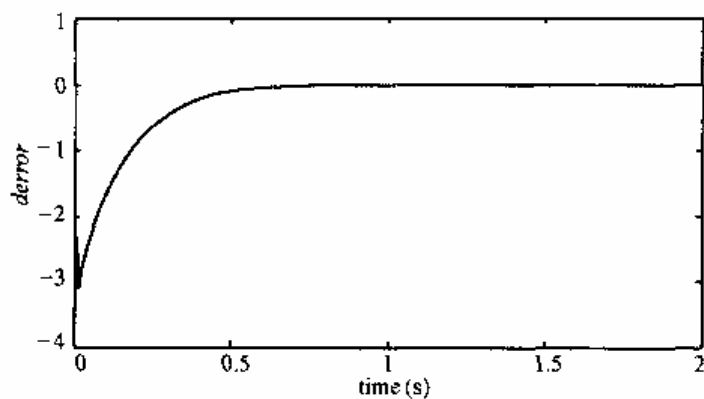


图 8-4 灰色 PID 跟踪误差变化率

仿真程序:

主程序: chap8_1.m

```
%PID Control based on Grey model compensation
```

```
clear all;close all;
```

```
global a21 a22 b A B kp kd
```

```
para=[];
```

```
BB=zeros(1,3);
```

```
ts=0.001;
```

```
N=3; %Needing N>=2(2 is x demension number)
```

```
TimeSet=[0:ts:ts*N];
```

```
%Using grey model to predict disturbance
```

```
V=zeros(1,3);
```

```
[t,x]=ode45('chap8_1f',TimeSet,[0.5 0.5],[],para,V);
```

```
x11=x(:,1);
```

```
x22=x(:,2);
```



```

x1=x(2,:); %It is the first value(not including initial value)
BB=[x1(1,:),1];
for k=2:1:N
    x1=[x1;x1(k-1,:)+x(k+1,:)];
    BB=[BB;[x1(k,:) k]];
end

D=zeros(N+1,1);
for k=2:1:N+1
    ddx(k)=(x22(k)-x22(k-1))/ts;
    up(k)=b*kp*x11(k)+b*kd*x22(k);
    D(k)=1/b*(ddx(k)-a21*x11(k)-a22*x22(k)-up(k));
end
D1=zeros(N,1);
D1(1)=D(1)+D(2);
for k=2:1:N
    D1(k)=D1(k-1)+D(k+1);
end

V1=inv(BB'*BB)*BB'*D1;
V=V1'

%Grey PID control using grey prediction results
N1=2000;
TimeSet1=[0:ts:ts*N1];
[t,x]=ode45('chap8_1f',TimeSet1,[0.50 0.50],[],para,V);
x1=x(:,1);
x2=x(:,2);
for k=1:1:N1+1
    kpd=[kp kd];
    up(k)=kpd*[x1(k);x2(k)];
    uc(k)=-V*[x1(k);x2(k);1]; %Grey Compensation
    u(k)=up(k)+uc(k);
end
figure(1);
plot(t,x1);
xlabel('time(s)');ylabel('error');
figure(2);
plot(t,x2);ylabel('derror');
xlabel('time(s)');ylabel('derror');

```

```

figure(3);
plot(t,u);
xlabel('time(s)');ylabel('u');

M 函数程序: chap8_if.m
%Dynamic model
function dx=DynamicModel(t,x,flag,para,V)
global a21 a22 b A B kp kd
dx=zeros(2,1);

a21=0;a22=-25;
b=133;

B=[0;b];
A=[0 1;a21 a22];

V1=[5 5];
f=5;

DD=V1*x+f; %System true disturbance
%Control law
kp=-35;
kd=-5;
up=kp*x(1)+kd*x(2);

M=1;
if M==1
    uc=0; %No Grey Compensation
elseif M==2
    uc=-V*[x;1]; %Grey Compensation
end
u=up+uc;
dx=A*x+B*(u+DD);

```

8.2.4 离散系统灰色 PID 控制

直接针对离散系统模型的 PID 控制具有很多对实际应用有价值的优点,但由于不确定部分对控制的影响不能忽略,使控制器难于适应各种不确定性。为此,提出对不确定部分建立灰色估计模型,在有限步数后,根据参数对不确定部分进行一定的补偿,以减小其影响。考虑单输入离散系统

$$x(k+1) = Ax(k) + bu(k) + bD(x,k) \quad (8.36)$$

式中

$$x \in R^n; u \in R; D(x, k) \in R$$

式中, A 为 $n \times n$ 系数矩阵, b 为 n 维矩阵, $bD(x, k)$ 代表系统满足匹配条件的不确定部分。它包括参数不确定与外干扰等。

$$D(x, k) = V_1 x_1 + V_2 x_2 + \cdots + V_n x_n + d(k) \quad (8.37)$$

控制律分为两个阶段:

(1) 采用 PID 控制进行灰色预测

$$u(k) = u_p(k) \quad (8.38)$$

$$D(x, k) = \frac{1}{b} (\ddot{x} - a_{21} \dot{x} - a_{22} x - bu_p(t)) \quad (8.39)$$

可计算离散数列向量

$$D^{(0)} \stackrel{\Delta}{=} (D(0) \quad D(1) \quad \cdots \quad D(N-1))^T \quad (8.40)$$

$$D^{(1)}(k) = \sum_{l=0}^k D(l) \quad (8.41)$$

$$D^{(1)} \stackrel{\Delta}{=} (D^{(1)}(0) \quad D^{(1)}(1) \quad \cdots \quad D^{(1)}(N-1))^T \quad (8.42)$$

由此可见, 在 N 步后, 即可估计出灰色模型的参数向量 \hat{V}^T

$$D^{(1)}(x, k) = V_1 x_1^{(1)}(k) + V_2 x_2^{(1)}(k) + \cdots + V_n x_n^{(1)}(k) + d^{(1)}(k) \quad (8.43)$$

$$\hat{V} = (\hat{V}_1 \quad \hat{V}_2 \quad \cdots \quad \hat{V}_n \quad \hat{f}_D)^T \quad (8.44)$$

式中

$$d^{(1)}(k) = \sum_{l=0}^k d(l) \quad (8.45)$$

$d(k)$ 为慢时变扰动, 在控制过程中可看作不变的常量, 将 $d(k)$ 记作 d 。

按最小二乘法公式, 可求得

$$\hat{V}^T = (B^T B)^{-1} B^T D^{(1)} \quad (8.46)$$

式中

$$B = \begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \quad (8.47)$$

且

$$|\det(B^T B)| > \varepsilon > 0 \quad (8.48)$$

(2) 采用灰色 PID 控制

在 $(n+1)$ 步后, 增加补偿控制 u_c , 此时,

$$u = u_p + u_c \quad (8.49)$$

$$u_c = -[\sum_{i=1}^n \hat{V}_i x_i + \hat{d}] \quad (8.50)$$

在第二阶段，估计器停止工作。

8.2.5 仿真程序及分析

仿真实例

被控对象为

$$G(s) = \frac{133}{s(s+25)}$$

以采样时间为 1ms，将传递函数转化为离散状态方程的形式

$$x(k+1) = Ax(k) + bu(k) + bD(x, k)$$

式中， $A = \begin{bmatrix} 1.0 & 0.0010 \\ 0.0 & 0.9753 \end{bmatrix}$ ， $b = \begin{bmatrix} -0.0001 \\ -0.1314 \end{bmatrix}$

外加干扰参数 $V = [0.50, 0.50, 0.50]$ ，采用离散系统灰色 PID 预测，经过 5 个采样时间的 PID 控制，得到干扰估计结果为： $\hat{V} = [0.50000026835824, 0.50000000007557, 0.49999987408683]$ 。

取 $M = 1$ ，不采用灰色预估补偿，而只采用 PID 控制，跟踪误差及变化率如图 8-5 和 8-6 所示。取 $M = 2$ ，采用灰色预估补偿，跟踪误差、误差变化率及控制器输出如图 8-7 至 8-9 所示。

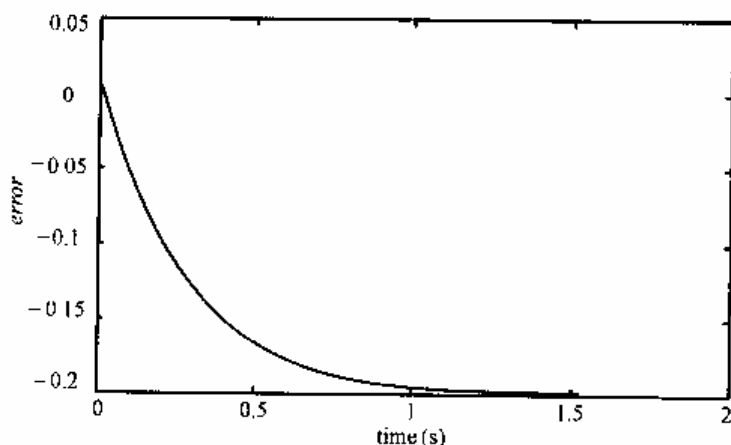


图 8-5 PID 跟踪误差 ($M = 1$)

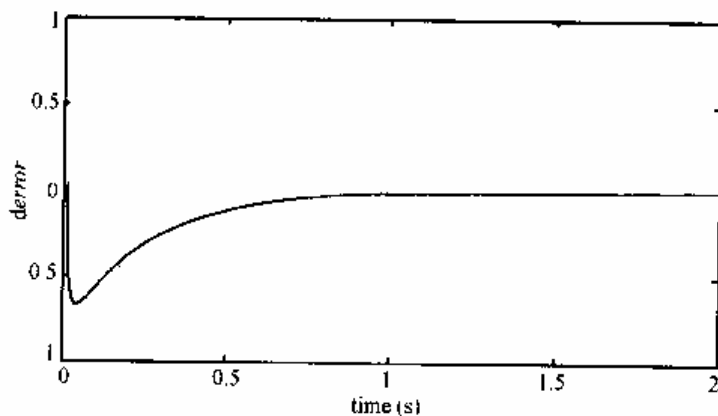


图 8-6 PID 跟踪误差变化率

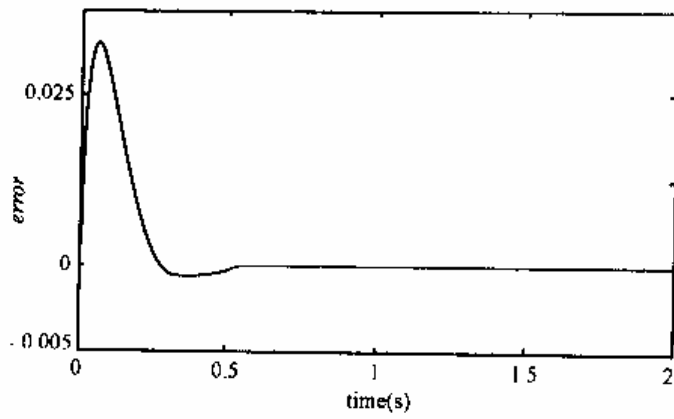


图 8-7 灰色 PID 跟踪误差 ($M = 2$)

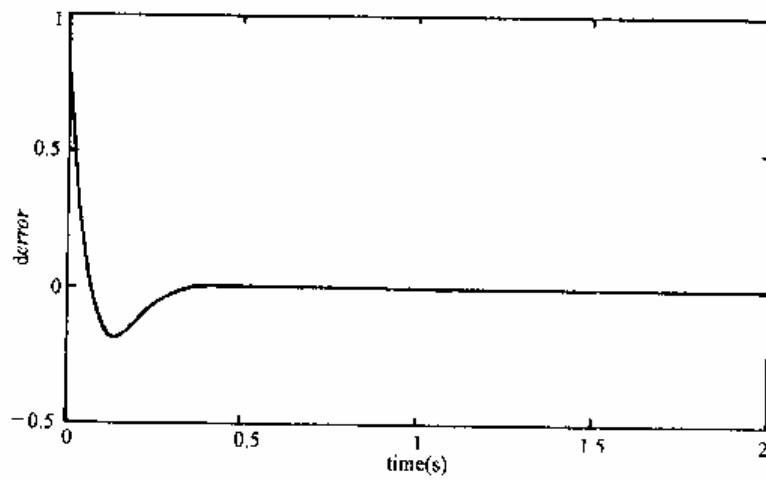


图 8-8 灰色 PID 跟踪误差变化率

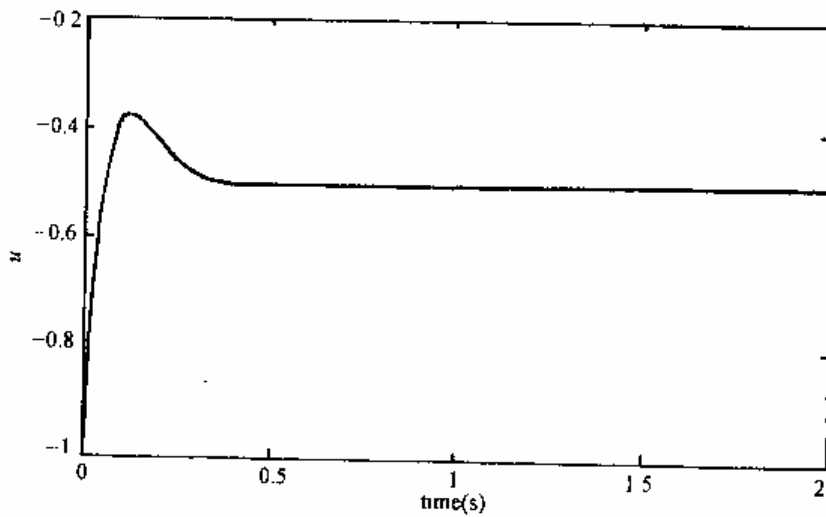


图 8-9 灰色 PID 控制器输出

仿真程序: chap8_2.m

```
%Discreted PID with grey model prediction
clear all;
close all;
```

```

ts=0.001;

n=2;
N=n+3;

a=25;b=133;
J=1/133;
q=25/133;
sys=tf(1,[J,q,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

A1=[0,1;0,-a];
b1=[0;b];
C1=[1,0];
D1=0;
[A,b,C,D]=c2dm(A1,b1,C1,D1,ts,'z');
A=A;
b=-b;

x_0=[0;1];
x_1=x_0;

%Uncertain Parameters
V=[0.5 0.5];d=0.5;
%Initial Value
x_1=[1;1];

%Grey prediction
for k=1:1:N
    time(k)=k*ts;
    x1(k)=x_1(1);
    x2(k)=x_1(2);

    D(k)=V*x_1+d;

    kp=2.0;
    up(k)=kp*x1(k);
    u(k)=up(k);

```

```

    D=V*x_1+d;
    x=A*x_1+b*u(k)+b*D;
    x_1=x;
end
xx1(1)=x1(2);
xx2(1)=x2(2);
BB=[xx1(1) xx2(1) 1];

for i=2:1:N-2
    xx1(i)=xx1(i-1)+x1(i+1);
    xx2(i)=xx2(i-1)+x2(i+1);
    BB=[BB;xx1(i) xx2(i) i];
end

for i=1:1:N-1
    D(i)=1/b*([x1(i+1);x2(i+1)]-A*[x1(i);x2(i)]-b*up(i));
end
D1(1)=D(2);
for i=2:1:N-2
    D1(i)=D1(i-1)+D(i+1);
end

%abs(det(BB'*BB));
V1=inv(BB'*BB)*BB'*D1';
Vp=V1'

%Grey PID control
x_1=x_0;
N1=2000;
for k=1:1:N1
    time(k)=k*ts;
    x1(k)=x_1(1);
    x2(k)=x_1(2);

    D(k)=V*x_1+d;

%Control law
M=1;
if M==1    %No Grey Compensation
    uc(k)=0;

```

```

elseif M==2 %Grey Compensation
    uc(k)=-(Vp(1)*x_1(1)+Vp(2)*x_1(2)+Vp(3));
end
up(k)=kp*x1(k);
u(k)=up(k)+uc(k);

D=V*x_1+d;
x=A*x_1+b*u(k)+b*D;
x_1=x;

end
figure(1);
plot(time,x1);
xlabel('time(s)');ylabel('error');
figure(2);
plot(time,x2);
xlabel('time(s)');ylabel('derror');
figure(3);
plot(time,u);
xlabel('time(s)');ylabel('u');

```

8.3 灰色 PID 的位置跟踪

8.3.1 连续系统灰色 PID 位置跟踪

考虑单输入连续系统

$$\dot{x} = Ax(t) + bu(t) + bD(x, t) \quad (8.51)$$

式中

$$x \in R^n; u \in R; D(x, t) \in R$$

式中, A 为 $n \times n$ 维矩阵, b 为 n 维矩阵, $bD(x, t)$ 代表系统满足匹配条件的不确定部分, 它包括参数不确定与外干扰等。

$$D(x, t) = V_1 x_1 + V_2 x_2 + \dots + V_n x_n + f(t) \quad (8.52)$$

取输入信号为

$$r(t) = 0.5 \sin(2\pi Ft) \quad (8.53)$$

离散化

$$r(k) = 0.5 \sin(2\pi FkT) \quad (8.54)$$

式中, F 为输入信号频率, T 为采样时间。

令

$$R^T(t) = [r(t) \quad \dot{r}(t)] \quad (8.55)$$

控制律分为两个阶段

(1) 采用 PID 控制进行灰色预测

$$u(t) = u_p(t) \quad (8.56)$$

$$D(x, k) = \frac{1}{h}(\ddot{y} + a\dot{y} - bu) \quad (8.57)$$

计算离散数列向量

$$D^{(0)} \stackrel{\Delta}{=} (D(1) \ D(2) \ \dots \ D(N))^T \quad (8.58)$$

$$D^{(1)}(k) \stackrel{\Delta}{=} \sum_{l=0}^k D(l) \quad (8.59)$$

$$D^{(1)} \stackrel{\Delta}{=} (D^{(1)}(2) \ D^{(1)}(3) \ \dots \ D^{(1)}(N))^T \quad (8.60)$$

在 N 步后, 即可估计出灰色模型的参数向量 \hat{V}^T

$$D^{(1)}(x, t) = V_1 x_1^{(1)}(t) + V_2 x_2^{(1)}(t) + \dots + V_n x_n^{(1)}(t) + f^{(1)}(t) \quad (8.61)$$

$$\hat{V} = (\hat{V}_1 \ \hat{V}_2 \ \dots \ \hat{V}_n \ \hat{f})^T \quad (8.62)$$

式中

$$f^{(1)}(k) = \sum_{l=0}^k f(l) \quad (8.63)$$

$f(t)$ 为慢时变扰动, 在控制过程中可看作不变的常量, 将 $f(t)$ 记作 f 。
按最小二乘法公式, 求得

$$\hat{V}^T = (B^T B)^{-1} B^T D^{(1)} \quad (8.64)$$

式中

$$B = \begin{bmatrix} x_1^{(1)}(2) & \dots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \dots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \dots & x_n^{(1)}(N) & N-1 \end{bmatrix} \quad (8.65)$$

且

$$|\det(B^T B)| > \varepsilon > 0 \quad (8.66)$$

(2) 采用灰色 PID 控制
加入补偿控制 u_c , 此时

$$u = u_p + u_c \quad (8.67)$$

$$u_c = -\left[\sum_{i=1}^n \hat{V}_i x_i + \hat{f}\right] \quad (8.68)$$

在第二阶段, 估计器停止工作。
控制系统状态方程为

$$\dot{x} = Ax + bu + bD(x, t) \quad (8.69)$$

$$D(x, t) = V_1 x_1 + V_2 x_2 + f \quad (8.70)$$

采用带有灰色估计器的补偿 PID 控制

$$u = u_p + u_c \quad (8.71)$$

不加灰色估计器，只用 PID 控制

$$u = u_p \quad (8.72)$$

8.3.2 仿真程序及分析

仿真实例

考虑单输入单输出连续系统

$$\dot{x} = Ax(t) + bu(t) + bD(x, t)$$

式中， $A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}$ ， $b^T = (0 \quad 133)$

外加干扰参数为 $V = [5, -5]$ ， $d = 5$ ，经过 4 个采样时间，干扰参数估计结果为 $V = [4.7117, 5.0109, 5.2018]$ 。

指令信号为一个幅值为 0.50，频率为 1.0 的正弦信号。取 $M = 1$ ，不采用灰色预估补偿，即 $u_c(k) = 0$ ，位置跟踪如图 8-10 所示。取 $M = 2$ ，采用灰色预估补偿，位置跟踪如图 8-11 所示。

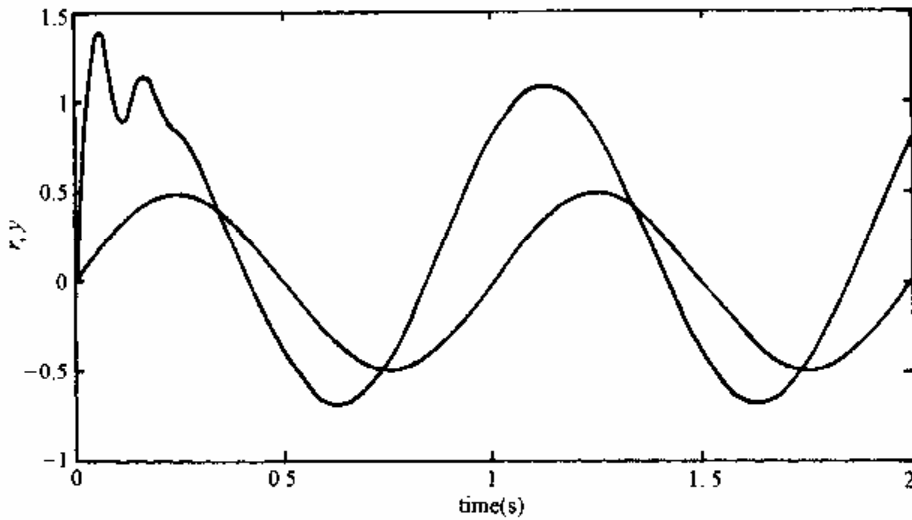


图 8-10 PID 位置跟踪 ($M = 1$)

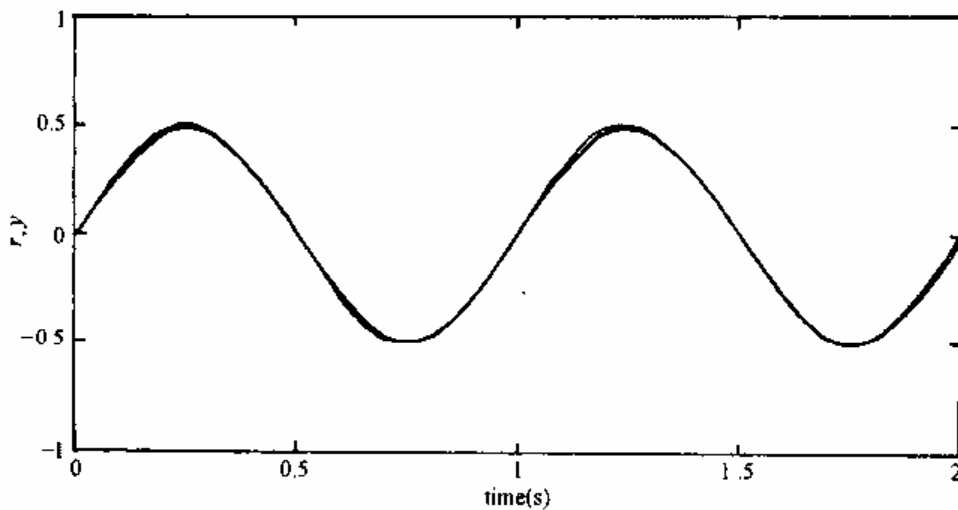


图 8-11 灰色 PID 位置跟踪 ($M = 2$)

```

仿真清单:
主程序: chap8_3.m

%Grey model PID Control
clear all;close all;
global kp kd a b A B F AA

F=1.0;
para=[];
AA=0.50;

%Disturbance Prediction
BB=zeros(1,3);
ts=0.001;
N=4;
ab=abs(det(BB'*BB));
    TimeSet=[0:ts:ts*N];
    V=zeros(1,3);
    [t,y]=ode45('chap8_3f',TimeSet,[0 0],[],para,V); %Grey Prediction

w=2*pi*F;
r=AA*(sin(w*t));
dr=AA*w*cos(w*t);

y0=y;
y1=y0(2,:);
BB=[y1(1,:),1];

for k=2:1:N
    y1=[y1;y1(k-1,:)+y0(k+1,:)];
    BB=[BB;y1(k,:),k];
end

y11=y(:,1);
y22=y(:,2);

D=zeros(N+1,1);
for k=2:1:N+1
    D(k)=1/b*((y22(k)-y22(k-1))/ts-a*y22(k)-b*kp*(r(k)-y11(k))-b*kd*(dr(k)-y22(k)));
end

D1=zeros(N,1);

```

```

D1(1)=D(2);

for k=2:1:N
    D1(k)=D1(k-1)+D(k+1);
end
ab=abs(det(BB'*BB))

V1=inv(BB'*BB)*BB'*D1;
V=V1'

%PID Control
N1=2000;
TimeSet1=[0:ts:ts*N1];
[t,y]=ode45('chap8_3f',TimeSet1,[0 0],[],para,V);

y1=y(:,1);
y2=y(:,2);
r=AA*(sin(w*t));
dr=AA*w*cos(w*t);
ddr=-AA*w^2*sin(w*t);

e=r-y1;
de=dr-y2;
for k=1:1:N1+1
    up(k)=kp*e(k)+kd*de(k);
    uc(k)=-V*[y1(k);y2(k);1]; %Grey Compensation
    u(k)=up(k)+uc(k);
end
figure(1);grid on;
plot(t,r,'r',t,y(:,1),'b');
xlabel('time(s)');ylabel('r,y');

figure(2);grid on;
plot(t,r-y(:,1));
xlabel('time(s)');ylabel('error');

figure(3);grid on;
plot(t,u);
xlabel('time(s)');ylabel('u');

```

```

M 函数程序: chap8_3f.m
%Dynamic model
function dy=DynamicModel(t,y,flag,para,V)
global kp kd a b A B F AA
dy=zeros(2,1);

%Input signal
w=2*pi*F;
rr=AA*sin(w*t);
dr=AA*w*cos(w*t);
ddr=-AA*(w^2)*sin(w*t);

r=[rr;dr];

b=133;a=-25;
B=[0;b];
A=[0 1;0 a];

V1=[5 5];f=5;

DD=V1*y+f; %True disturbance
%Control law
kp=30;kd=5.0;kpd=[kp,kd];
up=kpd*(r-y);

M=2;
if M==1
    uc=0; %No Grey Compensation
elseif M==2
    uc=-V*[y;1]; %Grey Compensation
end
u=up+uc;
dy=A*y+B*(u+DD);

```

8.3.3 离散系统灰色 PID 位置跟踪

考虑单输入离散系统

$$x(k+1) = Ax(k) + bu(k) + bD(x,k) \quad (8.73)$$

式中, $x \in R^n; u \in R; D(x,k) \in R$, A 为 $n \times n$ 维矩阵, b 为 n 维矩阵。

$bD(x,k)$ 代表系统满足匹配条件的不确定部分, 它包括参数不确定与外干扰等。

$$D(x, k) = V_1 x_1 + V_2 x_2 + \cdots + V_n x_n + d(k) \quad (8.74)$$

取输入信号为

$$r(t) = 0.5 \sin(2\pi Ft) \quad (8.75)$$

离散化为

$$r(k) = 0.5 \sin(2\pi FkT) \quad (8.76)$$

式中, F 为输入信号频率, T 为采样时间。

令

$$R^T(k) = [r(k) \quad \dot{r}(k)] \quad (8.77)$$

控制律分为两个阶段:

(1) 采用 PID 控制进行灰色预测

$$u(k) = u_p(k) \quad (8.78)$$

$$D(x, k) = b^{-1}(x(k+1) - Ax(k) - Bu(k)) \quad (8.79)$$

可计算离散数列向量

$$D^{(0)} \triangleq (D(0) \quad D(1) \quad \cdots \quad D(N-1))^T \quad (8.80)$$

$$D^{(1)}(k) \triangleq \sum_{l=0}^k D(l)$$

$$D^{(1)} \triangleq (D^{(1)}(0) \quad D^{(1)}(1) \quad \cdots \quad D^{(1)}(N-1))^T \quad (8.81)$$

由此可见, 经过 N 步后, 即可估计出灰色模型

$$D^{(1)}(x, k) = V_1 x_1^{(1)}(k) + V_2 x_2^{(1)}(k) + \cdots + V_n x_n^{(1)}(k) + d^{(1)}(k) \quad (8.82)$$

的参数向量 \hat{V}^T

$$\hat{V} = (\hat{V}_1 \quad \hat{V}_2 \quad \cdots \quad \hat{V}_n \quad \hat{f}_D)^T \quad (8.83)$$

式中

$$d^{(1)}(k) = \sum_{l=0}^k d(l) \quad (8.84)$$

$d(k)$ 为慢时变扰动, 可看作不变的常量, 将 $d(k)$ 记作 d 。

按最小二乘法公式求得

$$\hat{V}^T = (B^T B)^{-1} B^T D^{(1)} \quad (8.85)$$

式中

$$B = \begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \quad (8.86)$$

且

$$|\det(B^T B)| > \varepsilon > 0 \quad (8.87)$$

(2) 采用灰色 PID 控制

增加补偿控制 u_c , 此时控制律为

$$u = u_p + u_c \quad (8.88)$$

$$u_c = -\left[\sum_{i=1}^n \hat{V}_i x_i + \hat{d}\right] \quad (8.89)$$

在第二阶段，估计器停止工作。

8.3.4 仿真程序及分析

仿真实例

考虑单输入连续系统

$$\dot{x} = Ax + bu + bD(x, t)$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}, b^T = (0 \quad 133)$$

以采样时间 1ms 离散化为

$$x(k+1) = Ax(k) + bu(k) + bD(x, k)$$

$$A = \begin{bmatrix} 1.0000 & 0.0010 \\ 0 & 0.9753 \end{bmatrix}, b^T = (0.0001 \quad 0.1314)$$

外加干扰参数为 $V = [5, -5]$, $d = 5$, 经过 5 个采样时间, 可得到干扰参数估计结果

$$V_p = [5.0000 \quad -5.0000 \quad 5.0000]$$

指令信号为一个幅值为 0.50, 频率为 3.0 的正弦信号。当 $G=1$ 时为 PID 灰色预测, 当 $G=2$ 时为 PID 灰色控制。在灰色 PID 控制时, 取 $M=1$ 为不采用灰色预估补偿, 即 $u_c(k)=0$, 位置跟踪如图 8-12 所示; 取 $M=2$ 为采用灰色预估补偿, 位置跟踪如图 8-13 所示。

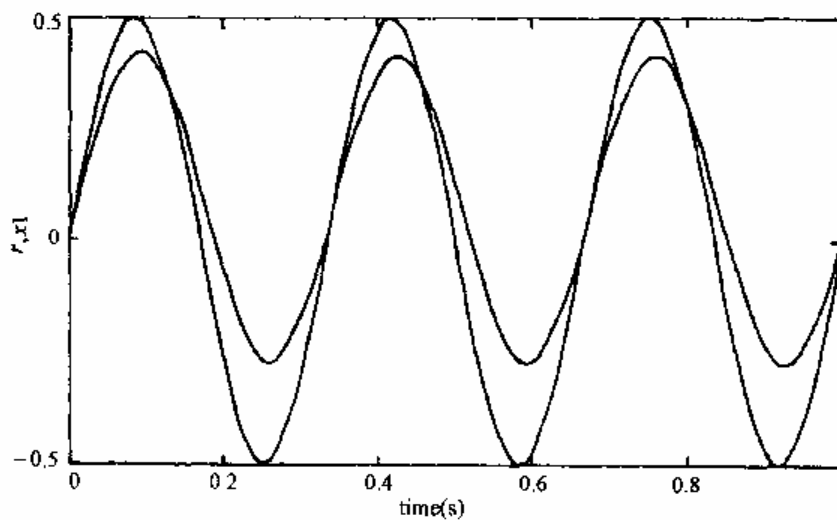


图 8-12 未采用灰色预估补偿 PID 位置跟踪 ($M=1$)

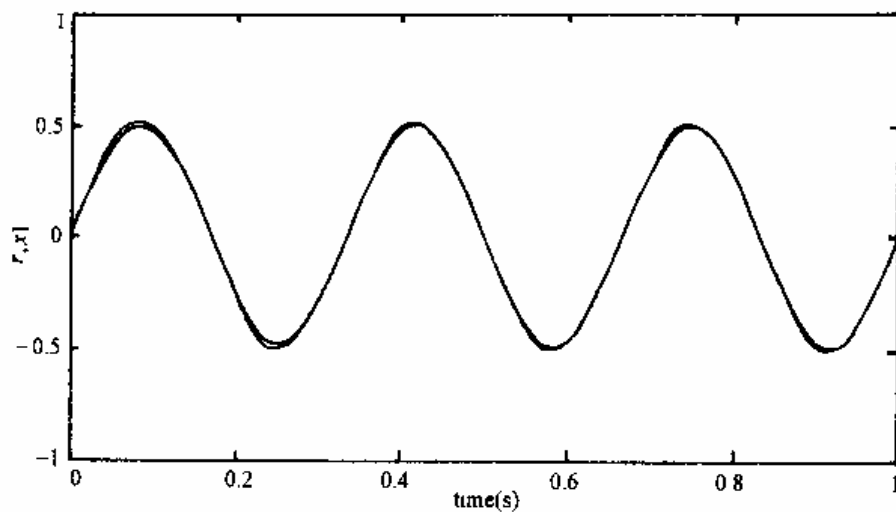


图 8-13 灰色 PID 位置跟踪 ($M=2$)

仿真程序: chap8_4.m

%Discrete PID Control with grey model prediction

clear all;

close all;

ts=0.001;

n=2;

AA=0.5;

F=3.0;

N1=1000;

N=n+3;

w=2*pi*F;

%System model

A1=[0 1;0 -25];

B1=[0;133];

C1=[1 0];

D1=[0];

[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

V=[5 -5];d=5;

x_1=[0;0];

for G=1:2

 for k=1:1:N

 time(k)=k*ts;


```

r(k)=AA*(sin(w*k*ts));
dr(k)=AA*w*cos(w*k*ts);

x1(k)=x_1(1);
x2(k)=x_1(2);

%Control law
if G==1      %For Grey Prediction
    uc(k)=0;
elseif G==2  %For Grey PID Control
    M=2;
    if M==1   %No Grey Compensation
        uc(k)=0;
    elseif M==2 %Grey Compensation
        uc(k)=-(Vp(1)*x_1(1)+Vp(2)*x_1(2)+Vp(3));
    end
end

end

kp=80;kd=10;
e(k)=r(k)-x1(k);
de(k)=dr(k)-x2(k);

up(k)=kp*e(k)+kd*de(k);
u(k)=up(k)+uc(k);

%Plant
DD=V*x_1+d;
x=A*x_1+B*u(k)+B*DD;
x_1=x;
end

if G==1      %Grey prediction
    xx1(1)=x1(2);xx2(1)=x2(2);
    BB=[xx1(1) xx2(1) 1];

for i=2:1:N-2
    xx1(i)=xx1(i-1)+x1(i+1);
    xx2(i)=xx2(i-1)+x2(i+1);
    BB=[BB;xx1(i) xx2(i) i];
end

```

```

for i=1:1:N-1
    u(i)=kp*e(i)+kd*de(i);
    DDD(i)=1/B*([x1(i+1);x2(i+1)]-A*[x1(i);x2(i)])-u(i);
end
D1(1)=DDD(2);

for i=2:1:N-2
    D1(i)=D1(i-1)+DDD(i+1);
end
end

xp=[x1' x2'];

if G==1
    ab=abs(det(BB'*BB))
    V1=inv(BB'*BB)*BB'*D1';
    Vp=V1'
end

N-N1; %If G=2
end
figure(1);grid on;
plot(time,r,'r',time,x1,'b');
xlabel('time(s)');ylabel('r,x1');
figure(2);grid on;
plot(time,r-x1,'b');
xlabel('time(s)');ylabel('error');
figure(3);grid on;
plot(time,u);
xlabel('time(s)');ylabel('u');

```

第 9 章 伺服系统 PID 控制

9.1 伺服系统低速摩擦条件下 PID 控制

9.1.1 Stribeck 摩擦模型描述

伺服系统的摩擦动态特性是非常复杂的，目前已经提出了许多摩擦模型，其中 Stribeck 曲线是比较著名的摩擦模型，如图 9-1 所示。该图表明了在不同的摩擦阶段，摩擦力矩与速度之间的关系，该关系即为 stribeck 曲线。

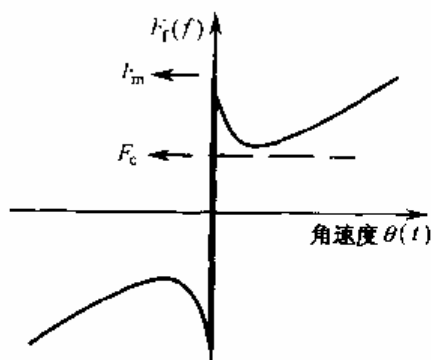


图 9-1 摩擦-速度曲线 (Stribeck 曲线)

Stribeck 摩擦模型可表示为:

当 $|\dot{\theta}(t)| < \alpha$ 时，静摩擦为

$$F_f(t) = \begin{cases} -F_m & F(t) > F_m \\ -F(t) & -F_m < F < F_m \\ F_m & F(t) < -F_m \end{cases} \quad (9.1)$$

当 $|\dot{\theta}(t)| > \alpha$ 时，动摩擦为

$$F_f(t) = \left(F_c + (F_m - F_c) e^{-\alpha_1 |\dot{\theta}(t)|} \right) \text{sgn}(\dot{\theta}(t)) \quad (9.2)$$

$$F(t) = -J\ddot{\theta}(t) \quad (9.3)$$

式中， $F(t)$ 为驱动力， F_m 为最大静摩擦力， F_c 为库仑摩擦力， $\dot{\theta}(t)$ 为转动角速度， α 和 α_1 为非常小的、正的常数。

9.1.2 一个典型伺服系统描述

以飞行模拟转台伺服系统为例，它是三轴伺服系统，正常情况下可简化为线性二阶环节的系统，在低速情况下具有较强的摩擦现象，此时控制对象就变为非线性，很难用传统控制方法达到高精度控制。则任意框的伺服结构可表达为如图 9-2 所示的形式，该系统采用直流电机，忽略电枢电感，并假设电流环和速度环为开环。

其中 K_u 为 PWM 功率放大器放大系数， R 为电枢电阻， K_m 为电机力矩系数， C_e 为电压反馈系数， J 为该框的转动惯量， $\dot{\theta}(t)$ 为转速， $r(t)$ 为指令信号， $u(t)$ 为控制输入。

根据伺服系统的结构，飞行模拟转台位置状态方程可描述如下

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{K_m C_e}{JR} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ K_u \frac{K_m}{JR} \end{bmatrix} u(t) - \begin{bmatrix} 0 \\ 1 \end{bmatrix} F_f(t) \quad (9.4)$$

式中， $x_1(t) = \theta(t)$ 为转角， $x_2(t) = \dot{\theta}(t)$ 为转速。

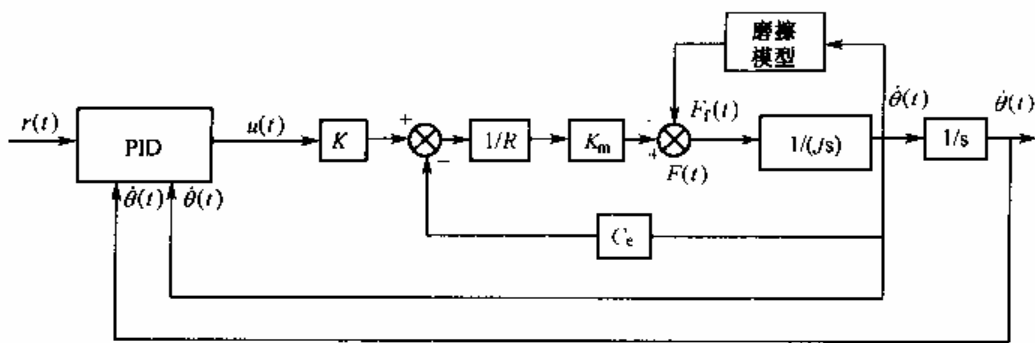


图 9-2 转台伺服系统结构

9.1.3 仿真程序及分析

仿真实例

设某转台某框伺服系统参数如下

$$R = 7.77\Omega, K_m = 6\text{N}\cdot\text{m}/\text{A}, C_e = 1.2\text{V}/(\text{rad}/\text{s}), J = 0.6\text{kg}\cdot\text{m}^2,$$

$$K_u = 11\text{V}/\text{V}, F_c = 1.5\text{N}\cdot\text{m}, F_m = 50\text{N}\cdot\text{m}, a_1 = 1.0, \alpha = 0.05$$

S 为信号选择变量, 其中 $S = 1$ 为正弦信号, $S = 2$ 为阶跃信号, $S = 3$ 为方波信号。本仿真选取 $S = 1$, 低速正弦跟踪信号指令为 $r(t) = 0.050\sin(2\pi t)$

采用 PD 控制

$$u(t) = 180e(t) + 50\dot{e}(t)$$

假设伺服系统无摩擦, 即取 $M = 0$, 此时 $F_f(t) = 0$, 采用 PD 控制, 速度和位置跟踪结果如图 9-3 和 9-4 所示。该结果表明, 无摩擦时, 采用 PID 控制方法具有较好的控制效果。

取 $M = 1$, 带有摩擦环节的 PID 控制跟踪结果如图 9-5 至图 9-8 所示。仿真结果表明在带有摩擦条件下, 位置跟踪存在“平顶”现象, 速度跟踪存在“死区”现象。采用 PID 控制鲁棒性差, 不能达到高精度跟踪。

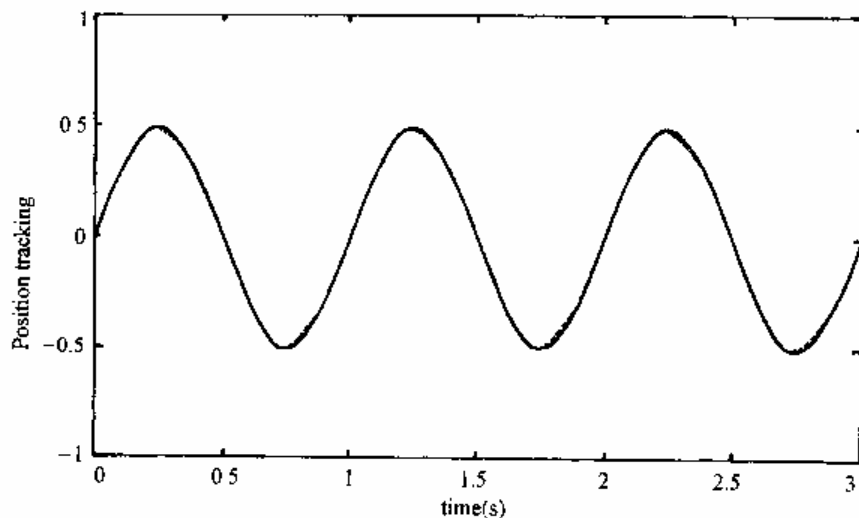


图 9-3 无摩擦时位置跟踪 ($M = 0$)

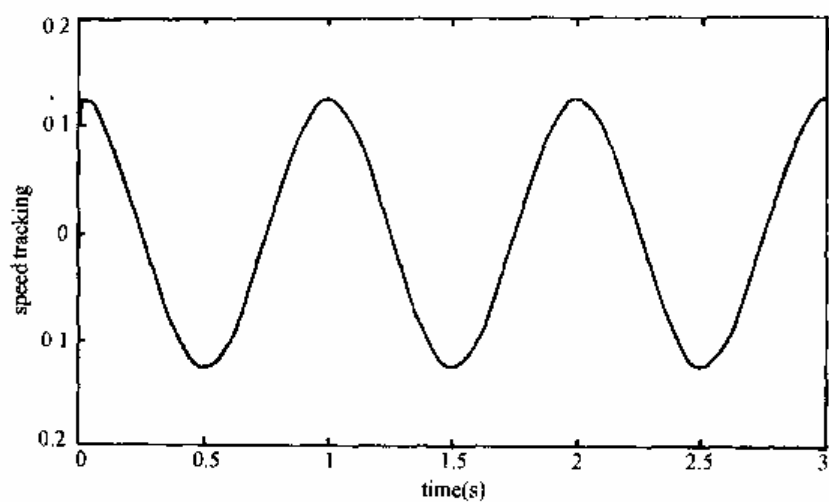


图 9-4 无摩擦时速度跟踪

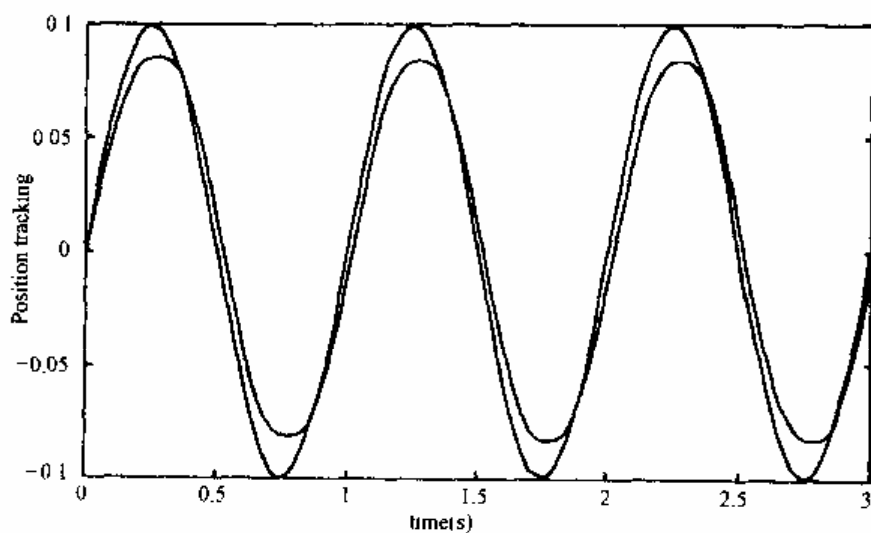


图 9-5 带摩擦时位置跟踪 ($M=1$)

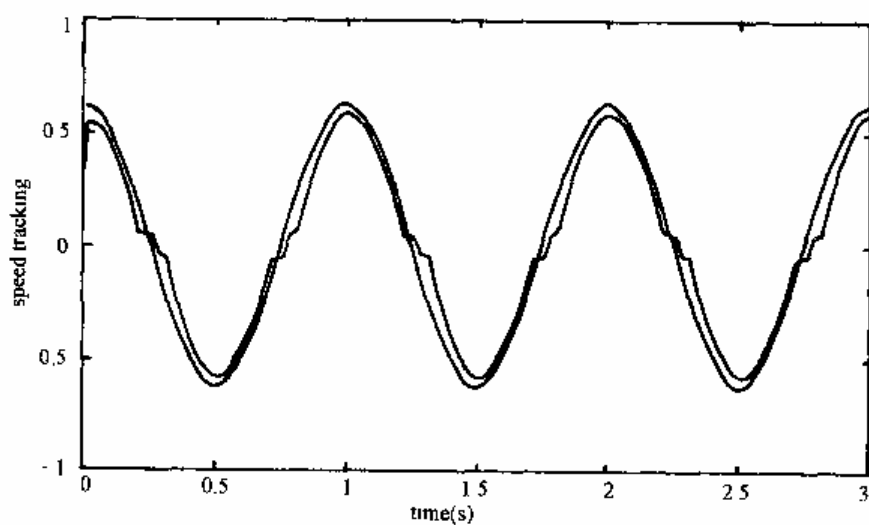


图 9-6 带摩擦时速度跟踪

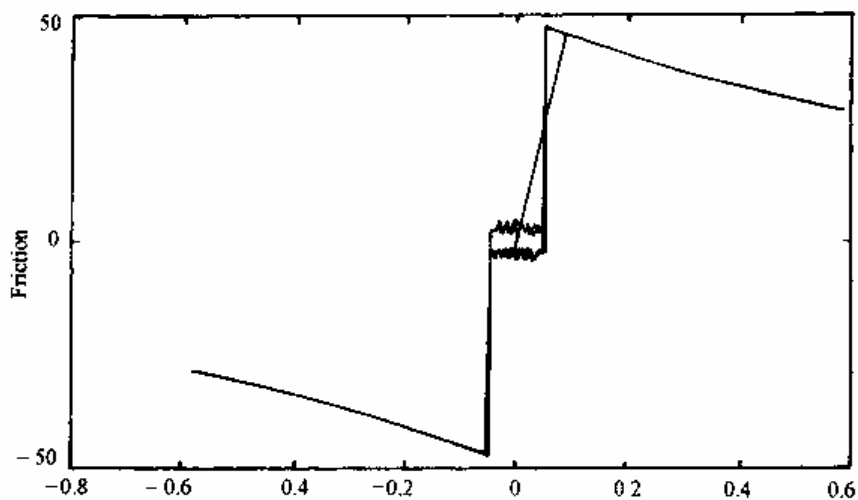


图 9-7 摩擦 $F_f(t)$ 的变化

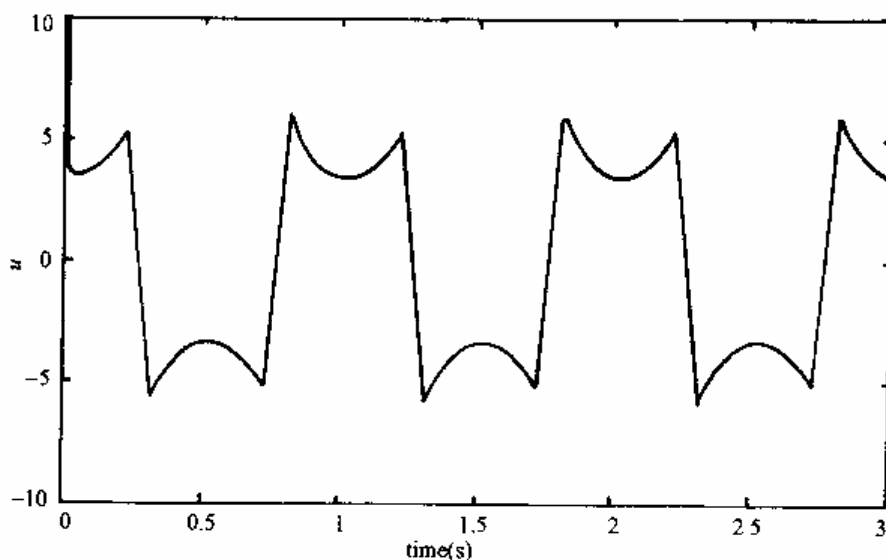


图 9-8 带摩擦时控制器输出

仿真程序:

主程序: chap9_1.m

%PID Control with Stribeck Friction Model

clear all;

close all;

global w A alfa J Ce R Km Ku S al Fm Fc M

%Servo system Parameters

J=0.6;Ce=1.2;Km=6;

Ku=11;R=7.77;

w=1*2*pi;A=0.10;

```

alfa=0.05;

T=3;
ts=0.001; %Sampling time
TimeSet=[0:ts:T];

M=1;      %If M=0, No Friction works
S=1;
[t,x]=ode45('chap9_1f',TimeSet,[0,0,0]);
x1=x(:,1);
x2=x(:,2);
x3=x(:,3);

if S==1
    rin=A*sin(w*t);
    drin=A*w*cos(w*t);
    ddrin=-A*w*w*sin(w*t);
    error=rin-x(:,1);
    derror=drin-x(:,2);
end
if S==2
    for kk=1:1:T/ts+1
        rin(kk)=1;
        drin(kk)=0;
        ddrin(kk)=0;
        error(kk)=rin(kk)-x1(kk);
        derror(kk)=drin(kk)-x2(kk);
    end
end
if S==3
    for kk=1:1:T/ts+1
        rin=A*sign(sin(0.4*2*pi*t));
        drin(kk)=0;
        ddrin(kk)=0;
        error(kk)=rin(kk)-x1(kk);
        derror(kk)=drin(kk)-x2(kk);
    end
end

F=-J*x(:,3);

```

```

x2=x(:,2);
for kk=1:1:T/ts+1
    time(kk)=(kk-1)*ts;

    if abs(x2(kk))<=alfa
        if F(kk)>Fm
            Ff(kk)=-Fm;
        elseif F(kk)<=-Fm
            Ff(kk)=Fm;
        else
            Ff(kk)=-F(kk);
        end
    end

    if x2(kk)>alfa
        Ff(kk)=Fc+(Fm-Fc)*exp(-a1*x2(kk));
    elseif x2(kk)<=-alfa
        Ff(kk)=-Fc-(Fm-Fc)*exp(a1*x2(kk));
    end

    if M==0
        Ff(kk)=0; %No Friction
    end

    u(kk)=200*error(kk)+40*derror(kk); %PID Control

    if u(kk)>=10
        u(kk)=10;
    end
    if u(kk)<=-10
        u(kk)=-10;
    end
end
figure(1);
plot(t,rin,'r',t,x(:,1),'b');
xlabel('time(s)');ylabel('position tracking');
figure(2);
plot(t,drin,'r',t,x(:,2),'b');
xlabel('time(s)');ylabel('speed tracking');

```



```

figure(3);
plot(t,error);
xlabel('time(s)');ylabel('error');
figure(4);
plot(x(:,2),Ff);
xlabel('speed');ylabel('Friction');
figure(5);
plot(time,u);
xlabel('time(s)');ylabel('u');

```

子程序: chap9_1f.m

```

function dx=Model(t,x)
global w A alfa J Ce R Km Ku S ai Fm Fc M
persistent aa
dx=zeros(3,1);

alfa=1.0; %Effect on the shape of friction curve
Fm=50;
Fc=1.5;

F=-J*x(3);
if t==0
    aa=0;
end
dF=-J*aa;

if abs(x(2))<=alfa
    if F>Fm
        Ff=-Fm;
        dFf=0;
    elseif F<-Fm
        Ff=Fm;
        dFf=0;
    else
        Ff=-F;
        dFf=dF;
    end
end
if x(2)>alfa
    Ff=Fc+(Fm-Fc)*exp(-alfa*x(2));

```

```

    dFf=(Fm-Fc)*exp(-a1*x(2))*(-a1)*x(3);
elseif x(2)<=-alfa
    Ff=-Fc-(Fm-Fc)*exp(a1*x(2));
    dFf=-(Fm-Fc)*exp(a1*x(2))*a1*x(3);
end

if S==1
    rin=A*sin(w*t);
    drin=A*w*cos(w*t);
    ddrin=-A*w*w*sin(w*t);
    dddrin=-A*w*w*w*cos(w*t);
end
if S==2
    rin=1;
    drin=0;
    ddrin=0;
    dddrin=0;
end
if S==3
    rin=A*sign(sin(0.4*2*pi*t));
    drin=0;
    ddrin=0;
    dddrin=0;
end
error=rin-x(1);
derror=drin-x(2);
dderror=ddrin-x(3);

u=200*error+40*derror; %PID
du=200*derror+40*dderror;

if u>=10
    u=10;
end
if u<=-10
    u=-10;
end

if M==0
    Ff=0;dFf=0; %No Friction

```

```

end
dx(1)=x(2);
dx(2)=-Km*Ce/(J*R)*x(2)+Ku*Km*u/(J*R)-Ff/J;
dx(3)=-Km*Ce/(J*R)*x(3)+Ku*Km*du/(J*R)-dFf/J;
aa=dx(3);

```

9.2 伺服系统三环的 PID 控制

9.2.1 伺服系统三环的 PID 控制原理

以转台伺服系统为例，其控制结构如图 9-9 所示，其中 r 为框架参考角位置输入信号， θ 为输出角位置信号。

伺服系统执行机构为典型的直流电动驱动机构，电机输出轴直接与负载-转动轴相连，为使系统具有较好的速度和加速度性能，引入测速机信号作为系统的速度反馈，直接构成模拟式速度回路。由高精度圆感应同步器与数字变换装置构成数字式角位置伺服回路。

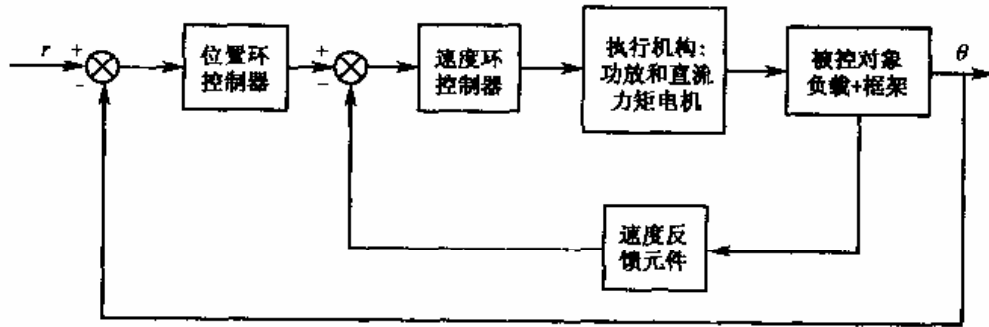


图 9-9 转台伺服系统框图

转台伺服系统单框的位置环、速度环和电流环框图如图 9-10、9-11 和 9-12 所示。

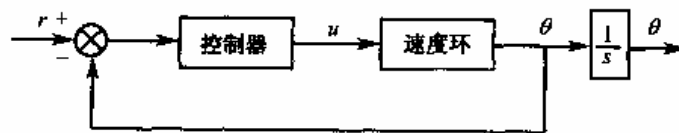


图 9-10 伺服系统位置环框图

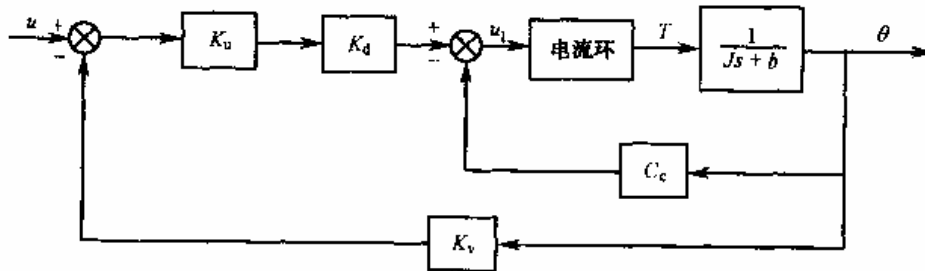


图 9-11 伺服系统速度环框图

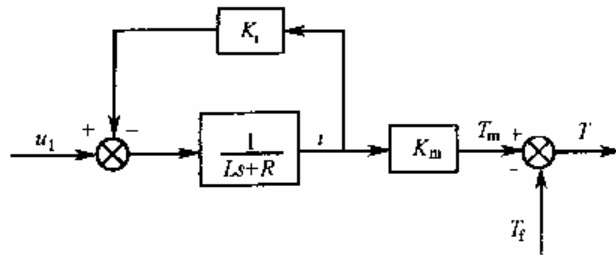


图 9-12 伺服系统电流环框图

图中符号含义如下： r 为位置指令； θ 为转台转角； K_u 为 PWM 功率放大倍数； K_d 为速度环放大倍数； K_v 为速度环反馈系数； K_i 为电流反馈系数； L 为电枢电感； R 为电枢电阻； K_m 为电机力矩系数； C_e 为电机反电动势系数； J 为等效到转轴上的转动惯量； b 为粘性阻尼系数，其中 $J = J_m + J_L$ ， $b = b_m + b_L$ ， J_m 和 J_L 分别为电机和负载的转动惯量， b_m 和 b_L 分别为电机和负载的粘性阻尼系数； T_f 为扰动力矩，包括摩擦力矩和耦合力矩。

假设在速度环中的外加干扰为粘性摩擦模型

$$F_f(t) = F_c \cdot \text{sgn}(\dot{\theta}) + b_c \cdot \dot{\theta} \quad (9.5)$$

控制器采用 PID 控制+前馈控制的形式，加入前馈摩擦补偿控制表示为

$$u_f(t) = F_{c1} \cdot \text{sgn}(\dot{\theta}) + b_{c1} \cdot \dot{\theta} \quad (9.6)$$

式中， F_{c1} 和 b_{c1} 为粘性摩擦模型等效到位置环的估计系数，该系数可以根据经验确定，或根据计算得出。

9.2.2 仿真程序及分析

仿真实例

被控对象为一个具有三环结构的伺服系统。伺服系统参数和控制参数在程序中给予了描述，系统采样时间为 1ms。取 $M=2$ ，此时输入指令为正弦迭加信号： $r(t) = A \sin(2\pi Ft) + 0.5A \sin(1.0\pi Ft) + 0.25A \sin(0.5\pi Ft)$ ，其中 $A = 0.50$ ， $F = 0.50$ 。

考虑到 K_i ， L 和 C_e 的值很小，前馈补偿系数 F_{c1} 和 b_{c1} 等效到摩擦力矩端的系数可近似写为

$$\text{Gain} \approx K_u \times K_d \times 1/R \times K_m \times K_g$$

式中， K_g 为经验系数。则摩擦模型估计系数 F_{c1} 和 b_{c1} 为

$$F_{c1} \approx F_c / \text{Gain}$$

$$b_{c1} \approx b_c / \text{Gain}$$

系统总的控制输出为

$$u(t) = u_p(t) + u_f(t)$$

式中， $u_p(t)$ 为 PID 控制的输出，其三项系数为 $k_{pp} = 15$ ， $k_{ii} = 0.10$ ， $k_{dd} = 1.5$ 。

根据是否加入摩擦干扰和前馈补偿分别进行仿真。

仿真程序：

初始化程序：chap9_2f.m

```
%Three Loop of Flight Simulator Servo System with Direct Current Motor
```

```

clear all;
close all;
%(1)Current loop
L=0.001; %L<<1 Inductance of motor armature
R=1; %Resistance of motor armature
ki=0.001; %Current feedback coefficient

%(2)Velocity loop
kd=6; %Velocity loop amplifier coefficient
kv=2; %Velocity loop feedback coefficient

J=2; %Equivalent moment of inertia of frame and motor
b=1; %Viscosity damp coefficient of frame and motor

km=1.0; %Motor moment coefficient
Ce=0.001; %Voltage feedback coefficient

%Friction model: Coulomb&Viscous Friction
Fc=100.0;bc=30.0; %Practical friction

%(3)Position loop: PID controller
ku=11; %Voltage amplifier coefficient of PWM
kpp=150;
kii=0.1;
kdd=1.5;

%Friction Model compensation
%Equavalent gain from feedforward to practical friction
Gain=ku*kd*1/R*km*1.0;
Fcl=Fc/Gain; bcl=bc/Gain; %Feedforward compensation

%Input signal initialize
F=0.50;
A=0.50;
ts=0.001; %Sampling time

M=2;
if M==1 %Sine Signal
    k=5000;

```

```

time=[0:ts:k*ts]'; %Simulation time
rin=A*sin(2*pi*F*time);
drin=2*pi*F*A*cos(2*pi*F*time);
elseif M==2 %Random Signal
T=4999;
time=zeros(T,1);
rin=zeros(T,1);
drin=zeros(T,1);
rin(1)=0;
drin(1)=0;
for k=1:1:T
time(k+1)=k*ts;
%Random signal
rin(k+1)=A*sin(2*pi*F*k*ts)+0.5*A*sin(2*pi*0.5*F*k*ts)+...
0.25*A*sin(2*pi*0.25*F*k*ts);
drin(k+1)=(rin(k+1)-rin(k))/ts;
end
end

```

控制系统的 Simulink 程序: chap9_2.mdl, 如图 9-13 和 9-14 所示。

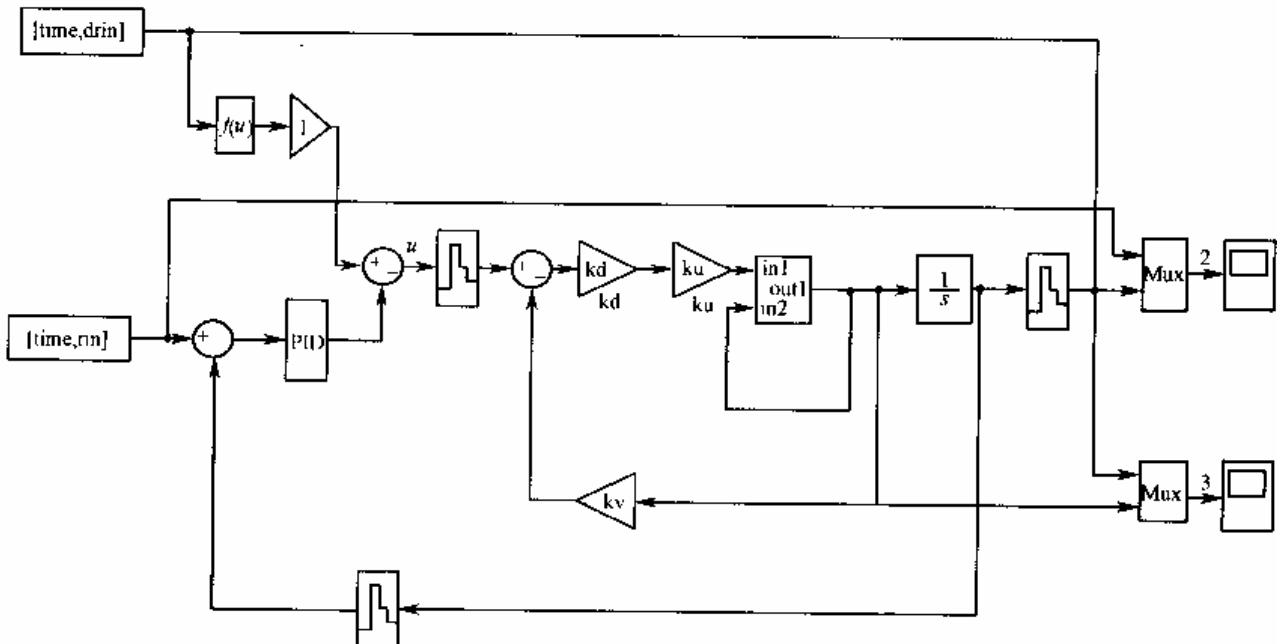


图 9-13 三环控制的 Simulink 仿真程序

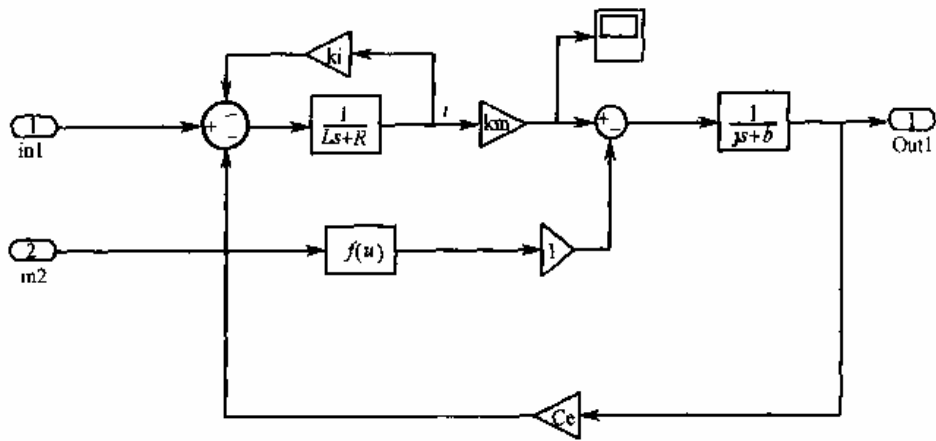


图 9-14 电机模型的 Simulink 仿真程序

电机模型的 Simulink 子模块:

(1) 无摩擦无前馈时的仿真, 正弦叠加信号跟踪结果如图 9-15 和图 9-16 所示。

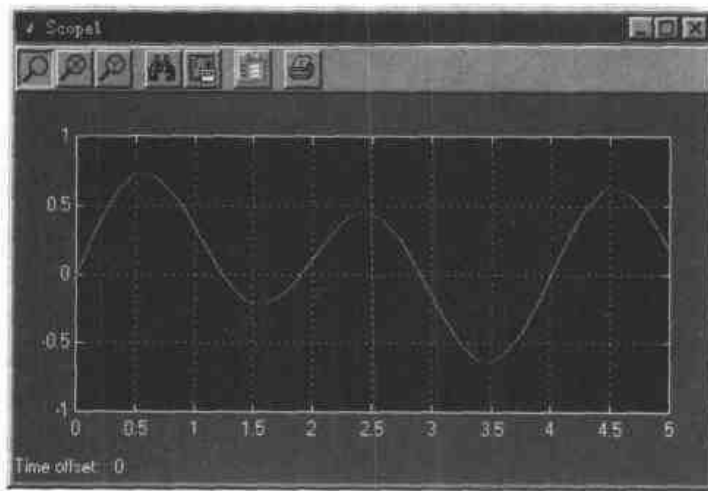


图 9-15 正弦叠加信号位置跟踪

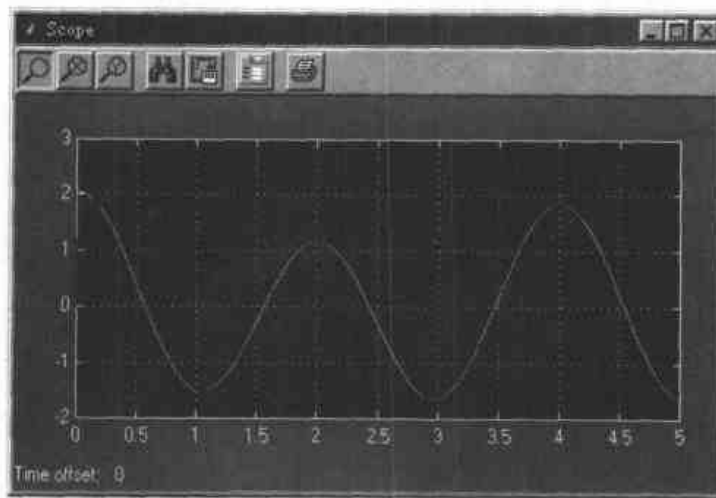


图 9-16 正弦叠加信号速度跟踪

(2) 带摩擦无前馈补偿时的仿真

正弦叠加信号跟踪如图 9-17 和图 9-18 所示, 由于静摩擦的作用, 在低速跟踪时, 位置跟踪存在“平顶”现象, 速度跟踪存在“死区”现象。

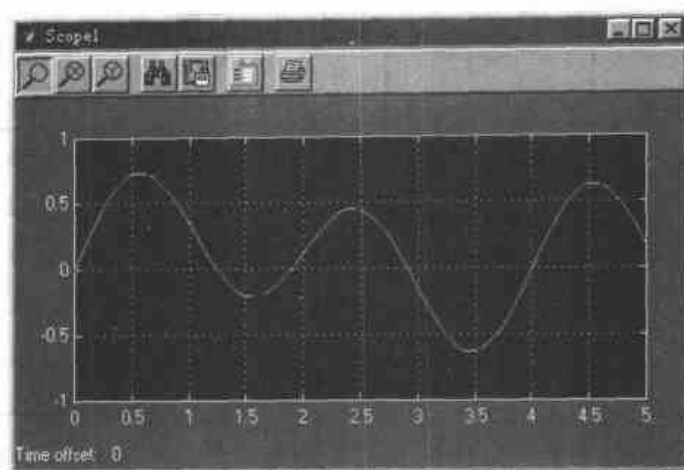


图 9-17 正弦叠加信号位置跟踪

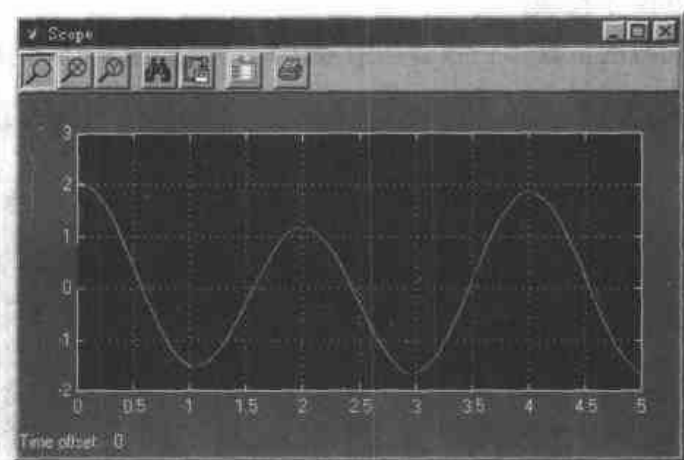


图 9-18 正弦叠加信号速度跟踪

(3) 带摩擦有前馈补偿时的仿真

正弦叠加信号跟踪如图 9-19 和图 9-20 所示, 采用 PID 控制+前馈控制可很大程度地克服摩擦的影响, 基本消除了位置跟踪的“平顶”和速度跟踪的“死区”, 实现了较高的位置跟踪和速度跟踪精度。

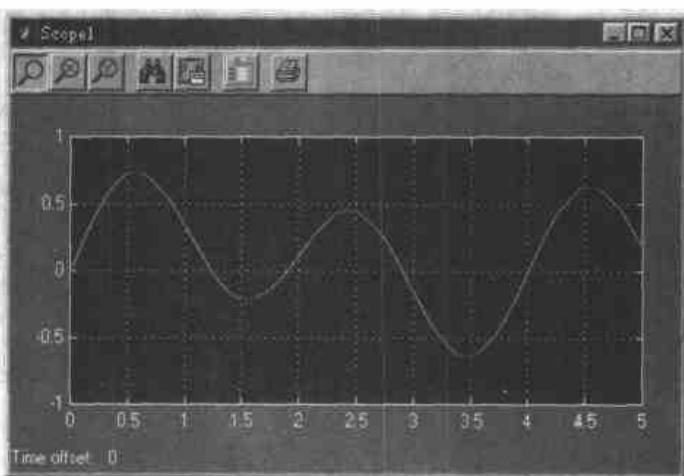


图 9-19 正弦叠加信号位置跟踪

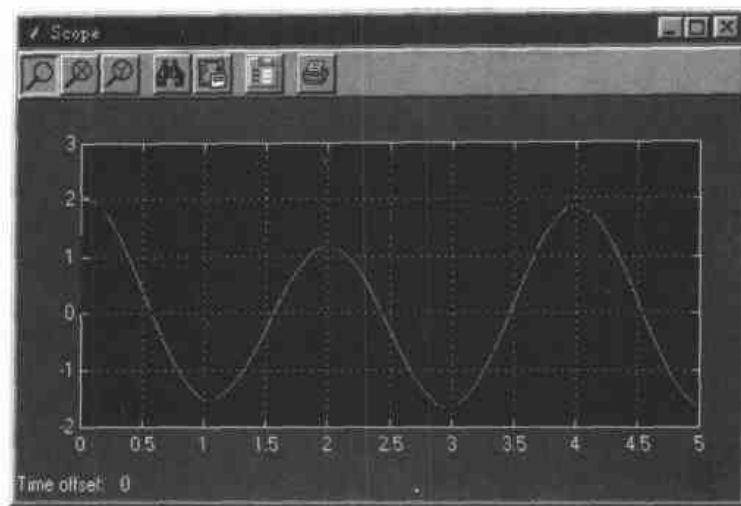


图 9-20 正弦叠加信号速度跟踪

9.3 二质量伺服系统的 PID 控制

9.3.1 二质量伺服系统的 PID 控制原理

如图 9-9 所示, 伺服系统是把电机与负载作为一个刚体来考虑的, 称为单质量伺服系统, 该系统与实际特性有很大差别。对于实际系统, 尽管电机与负载是直接耦合的, 但传动本质上是弹性的, 而且轴承和框架也都不完全是刚性的。在电机驱动力矩的作用下, 机械轴会受到某种程度的弯曲和变形。对于加速度要求大、快速性和精度要求高的系统或是转动惯量大、性能要求高的系统, 弹性变形对系统性能的影响不能忽略。由于传动轴的弯曲和变形, 在传递运动时含有储能元件。如果速度阻尼小, 则在它的传递特性中将出现较高的机械谐振, 此谐振对系统的动态性能影响较大。因此应将被控对象视为图 9-21 所示由电机、纯惯性负载以及联结二者的等效传递轴所组成的三质量系统。

根据图 9-21 可得传动轴动力学方程, 根据伺服系统电机框图 9-22 可得电机电动力学方程。根据伺服系统负载框图 9-23 (不考虑干扰时) 可得负载动力学方程。

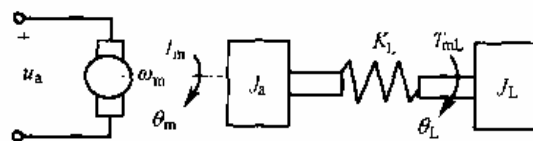


图 9-21 电机-传动轴-负载模型

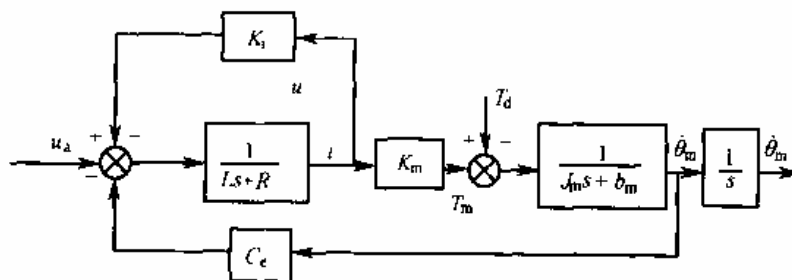


图 9-22 伺服系统电机框图

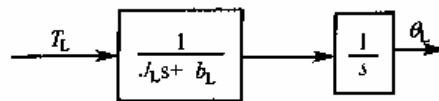


图 9-23 负载框图

三质量伺服系统的电学方程和动力学方程

电机
$$iR + Li = u_a - C_e \dot{\theta}_m - K_f i \quad (9.7)$$

$$T_m = iK_m \quad (9.8)$$

$$J_m \ddot{\theta}_m = T_m - b_m \dot{\theta}_m - K_L (\theta_m - \theta_L) \quad (9.9)$$

传动轴
$$J_a (\ddot{\theta}_m - \ddot{\theta}_L) = K_L (\theta_m - \theta_L) - T_{mL} \quad (9.10)$$

负载
$$J_L \ddot{\theta}_L = T_{mL} - b_L \dot{\theta}_L \quad (9.11)$$

式中， J_a 为传动轴的转动惯量， θ_m 和 θ_L 分别为电机和负载的转角， J_m 和 J_L 分别为电机和负载的转动惯量， b_m 和 b_L 分别为电机和负载的粘性阻尼系数； K_L 为电机和框架之间的耦合刚度系数， T_{mL} 为负载端输出力矩。

一般 J_a 相对于 J_L 很小，而且其质量分布在轴的长度上，因此可以忽略或计入到 J_a 中，于是上述三质量系统可以简化为二质量系统。二质量系统的电学和动力学方程为

电机
$$iR + Li = u_a - C_e \dot{\theta}_m - K_f i \quad (9.12)$$

$$T_m = iK_m \quad (9.13)$$

$$J_m \ddot{\theta}_m = T_m - b_m \dot{\theta}_m - K_L (\theta_m - \theta_L) \quad (9.14)$$

负载
$$J_L \ddot{\theta}_L = T_{mL} - b_L \dot{\theta}_L \quad (9.15)$$

$$K_L (\theta_m - \theta_L) - T_{mL} = 0 \quad (9.16)$$

根据上述描述，参考伺服系统结构图 9-10 至 9-12，得到二质量伺服系统部分结构图（其余部分与单质量伺服系统结构相同），如图 9-24 所示。

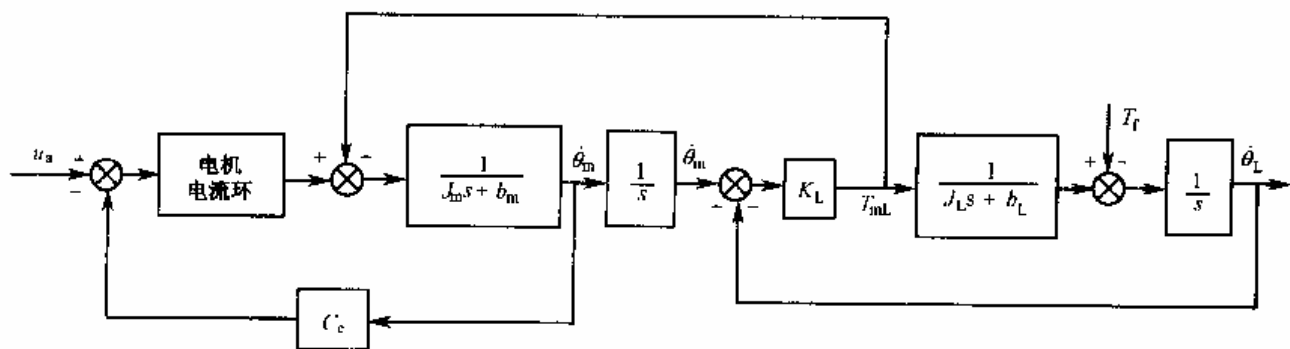


图 9-24 二质量伺服系统部分结构框图

9.3.2 仿真程序及分析

仿真实例

被控对象为一个具有三环结构的二质量伺服系统。伺服系统参数和控制参数在程序中给予了描述，系统采样时间为 1ms。

输入指令为正弦信号： $r(t) = A \sin(2\pi Ft)$ ，其中 $A = 0.50$ ， $F = 0.50$ 。

假设在速度环中的外加干扰 T_f 为粘性摩擦模型: $F_f(t) = F_c \cdot \text{sgn}(\dot{\theta}) + b_c \cdot \dot{\theta}$
 控制器采用 PID 控制, 参数选为 $k_{pp} = 60$, $k_{ii} = 3.0$, $k_{dd} = 20$ 。
 根据是否加入摩擦干扰分别进行仿真。

仿真程序:

初始化程序: chap9_3f.m

%Three Loop of Flight Simulator Servo System with two-mass of Direct Current
 Motor

```
clear all;
close all;

%(1)Current loop
L=0.001;    %L<<1,Inductance of motor armature
R=1.0;      %Resistance of motor armature
ki=0.001;   %Current feedback coefficient

%(2)Velocity loop
kd=6;       %Velocity loop amplifier coefficient
kv=2;       %Velocity loop feedback coefficient

Jm=0.005;   %Equivalent moment of inertia of motor
bm=0.010;   %Viscosity damp coefficient of motor

km=10;      %Motor moment coefficient
Ce=0.001;   %Voltage feedback coefficient

Jl=0.15;    %Equivalent moment of inertia of frame
bl=8.0;     %Viscosity damp coefficient of frame

kl=5.0;     %Motor moment coefficient between frame and motor

%Friction model: Coulomb&Viscous Friction
Fc=10;bc=3; %Practical friction

%(3)Position loop: PID controller
ku=11;      %Voltage amplifier coefficient of PWM

kpp=60;
kii=3.0;
kdd=20;
```

```

%Input Signal Initialize
F=0.50;
A=0.50;
ts=0.001;      %Sampling time

k=5000;
time=[0:ts:k*ts]'; %Simulation time

```

```

rin=A*sin(2*pi*F*time);
drin=2*pi*F*A*cos(2*pi*F*time);

```

控制系统的 Simulink 程序: chap9_3.mdl, 如图 9-25 至 9-27 所示。

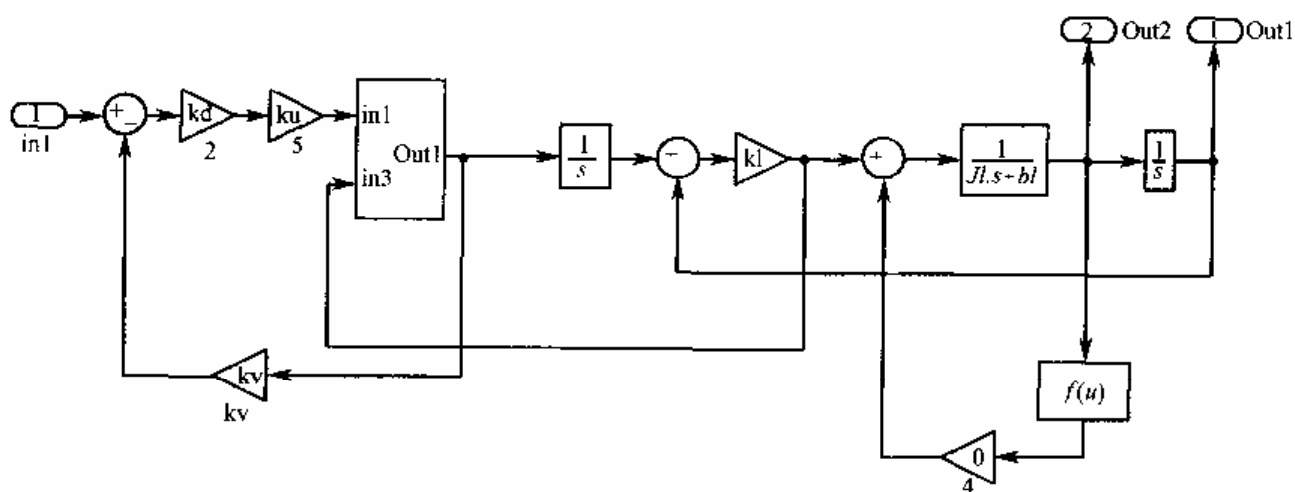


图 9-25 二质量伺服系统一环控制的 Simulink 仿真程序

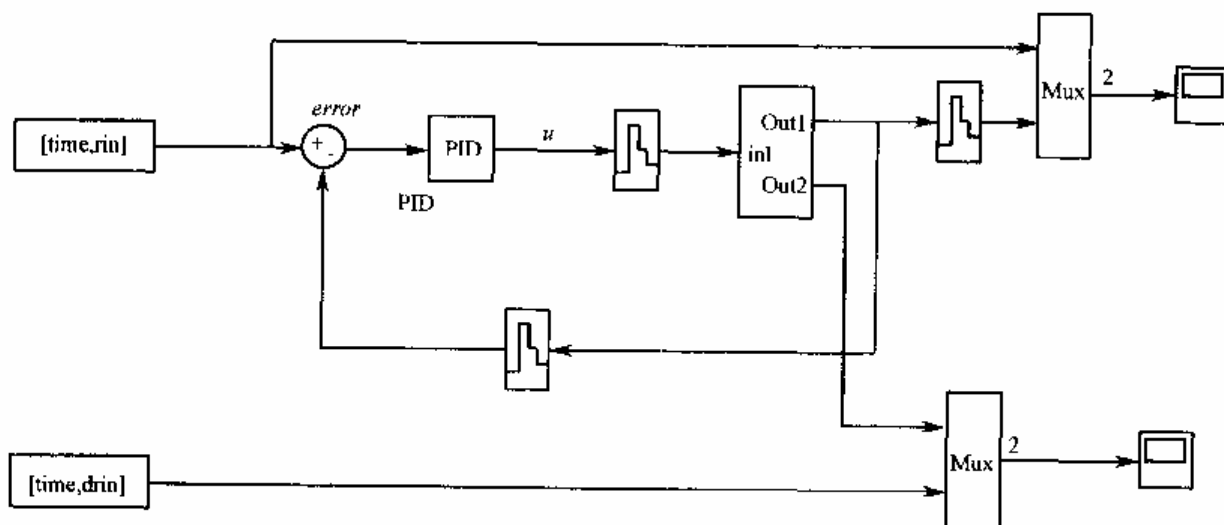


图 9-26 广义被控对象的 PID 控制 Simulink 仿真程序

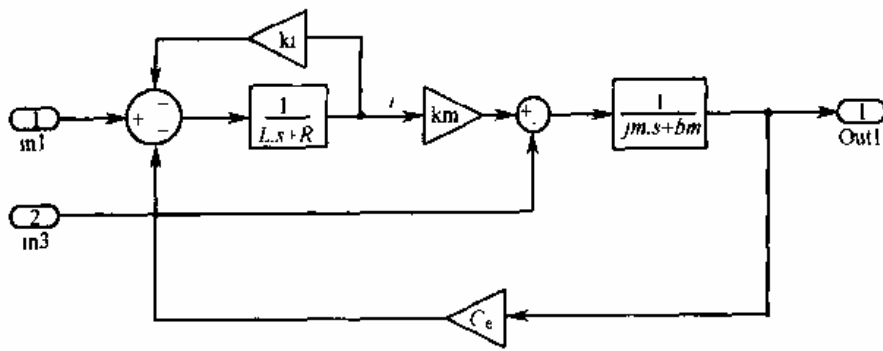


图 9-27 电机模型的 Simulink 仿真程序

(1) 无摩擦时，正弦位置跟踪和速度跟踪的结果如图 9-28 和图 9-29 所示。

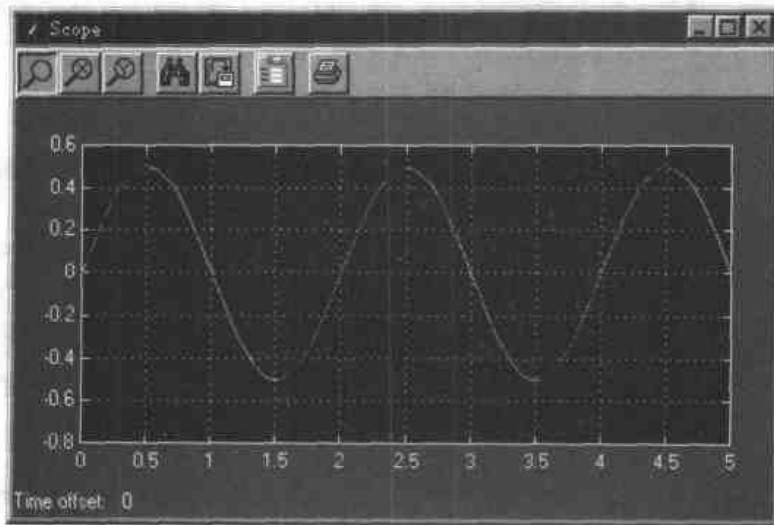


图 9-28 正弦位置跟踪

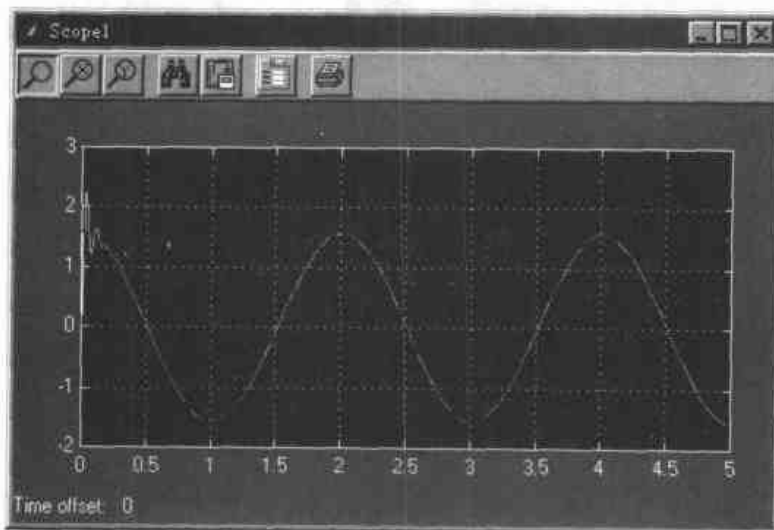


图 9-29 正弦速度跟踪

(2) 带摩擦时，正弦位置跟踪和速度跟踪的结果如图 9-30 和图 9-31 所示，由于静摩擦的作用，在低速跟踪时，位置跟踪存在“平顶”现象，速度跟踪存在“死区”现象。

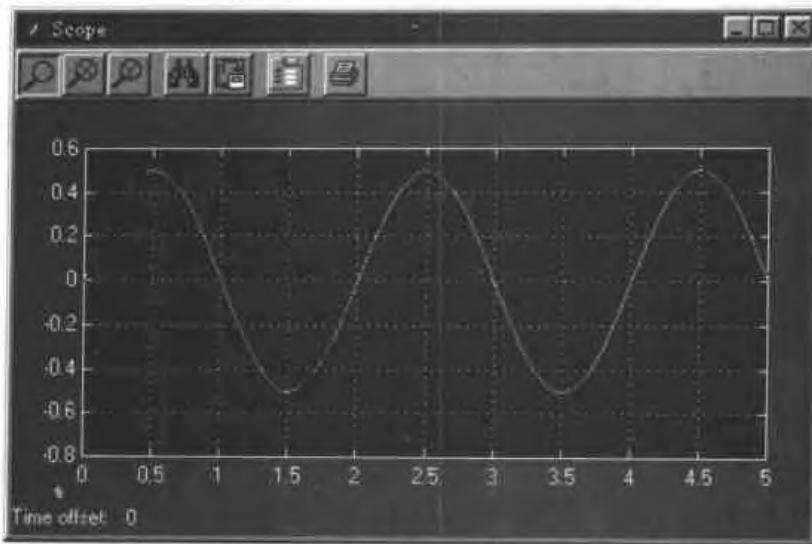


图 9-30 正弦位置跟踪

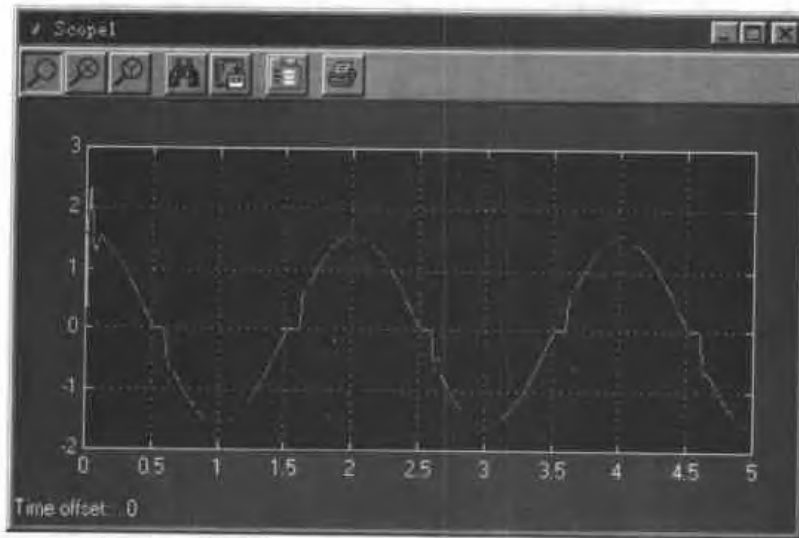


图 9-31 正弦速度跟踪

第 10 章 PID 实时控制的 C + + 语言设计及应用

10.1 M 语言的 C++ 转化

仿真实例

对象采用二阶传递函数为

$$G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms, 被控对象可离散化为

$$y(k) = -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) - \text{den}(4)y(k-3) \\ + \text{num}(2)u(k-1) + \text{num}(3)u(k-2) + \text{num}(4)u(k-3)$$

取输入指令为阶跃信号, 进行位置跟踪的仿真。PID 控制参数取 $k_p = 25, k_d = 0, k_i = 0.28$ 。M 语言的仿真程序为 chap10_1.m, PID 阶跃响应如图 10-1 所示。转化的 C++ 语言 (Borland C++) 程序为 chap10_2.cpp。

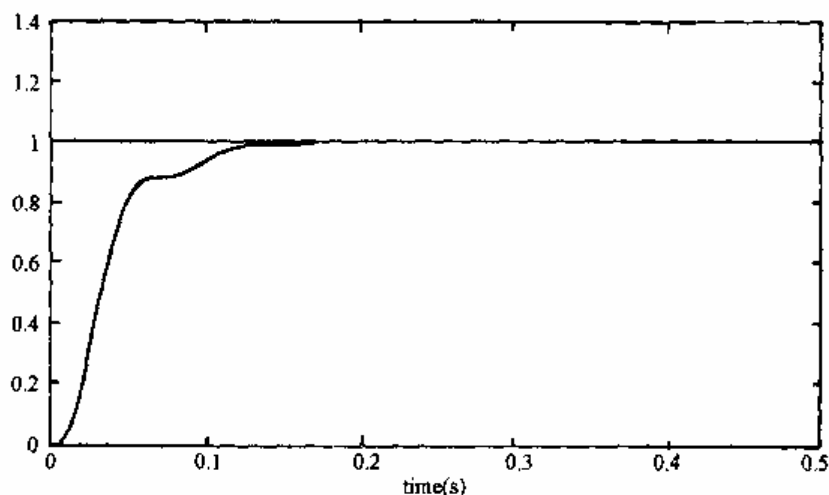


图 10-1 PID 的阶跃响应

M 语言仿真程序: chap10_1.m

```
%PID Controller
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
```

```

dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0.0;y_2=0.0;y_3=0.0;
x=[0,0,0]';
error_1=0;

for k=1:1:500
time(k)=k*ts;

rin(k)=1;                %Tracing Step Signal
kp=0.50;ki=0.001;kd=0.001;

u(k)=kp*x(1)+kd*x(2)+ki*x(3);    %PID Controller

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);

%Return of parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                %Calculating P
x(2)=(error(k)-error_1)/ts;    %Calculating D
x(3)=x(3)+error(k)*ts;        %Calculating I
xi(k)=x(3);

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');

```

C++语言仿真程序: chap10_2.cpp

//PID Realtime Control

#include <math.h>

int k;


```

double yout,u,rin,u_1=0,u_2=0,u_3=0,yout_1=0,yout_2=0,yout_3=0;
double Error_1=0.0,Error_2=0.0;
double ts=0.001;
double timek,pi=3.1415926;
double kp=0.50,ki=0.001,kd=0.001;
double Error,Perror,Ierror,Derror;

void main()
{
for (k=1;k<=5000;k++)
{
    timek=k*ts;
    rin=1;

    //Practical Position Degree Signal at time t
    yout=2.9063*yout_1-
2.8227*yout_2+0.9164*yout_3+0.0853*0.001*u_1+0.3338*0.001*u_2+0.0817*0.001*u_3;

    Error=yout-rin;
//PID Controller
    Perror=Error;           //Getting P
    Ierror= Ierror+Error*ts; //Getting i
    Derror=(Error-Error_1)/ts; //Getting D

    u=kp*Perror+ki*Ierror+kd*Derror; //PID Controller

//Update Parameters
    Error_2=Error_1;
    Error_1=Error;

    yout_3=yout_2;
    yout_2=yout_1;
    yout_1=yout; //Positional Signal at k-1
    u_3=u_2;
    u_2=u_1;
    u_1=u;
}
}

```

10.2 基于 C++ 的三轴飞行模拟转台伺服系统 PID 实时控制

10.2.1 控制系统构成

三轴电动飞行模拟转台伺服系统由机械台体、电控柜两部分组成。参见图 10-2。机械台体是三轴仿真系统的主体，由基座和三个运动框架组成，用以安装试验负载，模拟飞行器的三维姿态运动；电控柜由转台系统的测控、显示、电子伺服装置和控制计算机组成，用来构成转台的伺服控制系统，实施对转台的检测和控制。

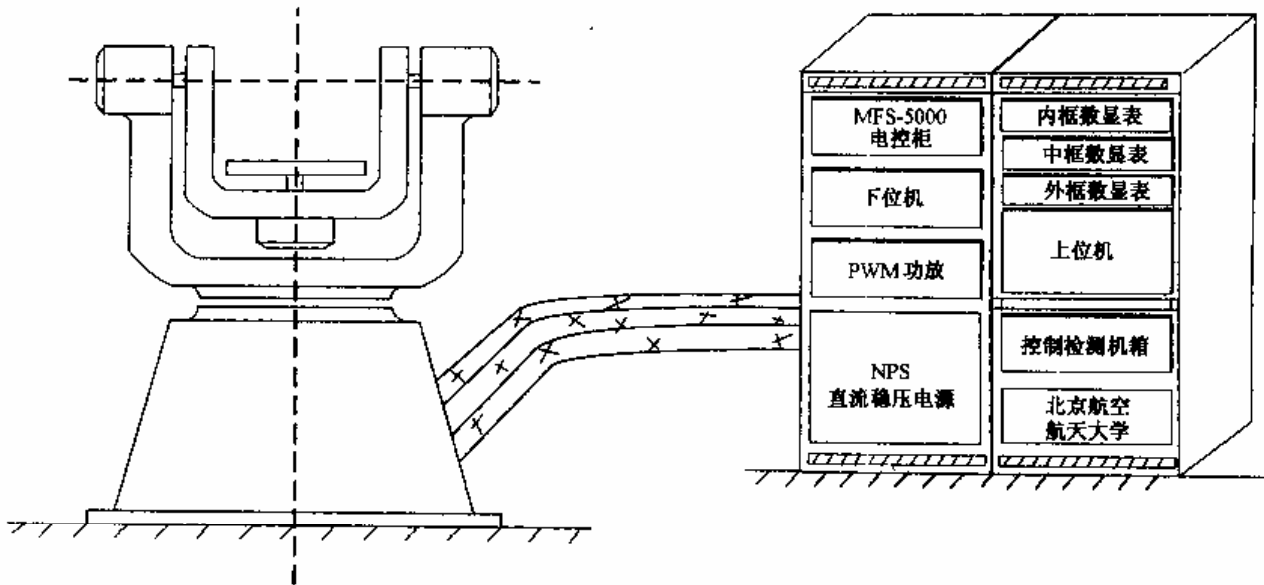


图 10-2 转台系统总体结构图

内、中、外框三个通道的数字伺服控制系统，其原理结构完全相同，如图 10-3 所示。系统采用了微机控制下的脉冲调宽功放装置 (PWM)，直流力矩马达直接驱动转台框架的数字伺服调速体制。高灵敏度的直流测速机直接构成连续式速度控制回路，有利于系统刚度的提高和频带的拓宽。由高精度测角元件圆感应同步器和数字变换装置构成数字式角位置反馈回路，可以满足系统的精度和性能要求。采用工业控制机作为该伺服系统的主控计算机，能够保证系统快速性能的实现，同时也能很好地完成系统控制律的形成、回路内部的数据采集处理、故障诊断和监控管理，使系统的动态性能与安全可靠性都得到充分的保证。

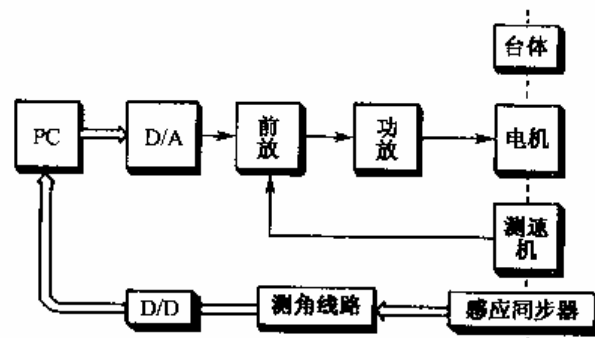


图 10-3 内、中、外三通道数字伺服控制系统原理结构

10.2.2 系统各部分功能的软件设计

该程序实现了实时控制的几个基本功能:

(1) 中断: 采用函数 `IniTimer()`、`interrupt()`和 `IrqHook()`实现。其中 `IniTimer()`实现中断初始化, `interrupt()`实现中断处理, `IrqHook()`实现中断地址勾挂。中断时间为 1ms, 控制算法及绘图在中断时间内完成, 采用定时器 8253 模式来实现中断, 中断控制器采用 8259A 模式。中断参数在头文件 `pidctrl.h` 中设定;

(2) 位置信号的采集: 采用 D/D 板实现, 由 `ReadD_D()`函数完成;

(3) 控制器输出: 采用 D/A 板实现, 由 `write_DA()`函数完成;

(4) 控制: 由控制函数 `control()`实现实时控制, 由函数 `ReleaseHardware()`实现控制电机的清零;

(5) 图形显示: 由作图命令及函数 `DynamicDisplay()`实现;

(6) 键盘管理: 由函数 `KbGetKey()`和 `SetupKeyReaction()`实现, 键盘参数在头文件 `pidctrl.h` 中设定, 其中数据保存由 F5 键实现, 退出由 ALT+X 键实现;

(7) 数据存储: 由函数 `DataSaveRoutine()`实现, 按 F5 键则保存数据, 保存文件名为 `xyz.dat`;

(8) 数显表的初始化: 由函数 `ResetShuXianBiao()`实现。

10.2.3 仿真程序及分析

仿真实例

转台中框为二阶传递函数

$$G(s) = \frac{1770}{s^2 + 60s + 1770}$$

采样时间为 1ms, 离散化为

$$y(k) = 1.94y(k-1) - 0.94y(k-2) + 0.0008674u(k-1) + 0.0008503u(k-2)$$

`channel` 和 `Signal` 框选和输入信号类型变量, 其中 `channel=1` 为转台中框, `Signal=1` 为正弦输入信号, 其幅值为 0.010, 频率为 0.50Hz。采用 PD 进行位置跟踪的实时控制, 其中 $k_p=10, k_s=1.0$, 当 $M=1$ 时位置信号由 D/D 板采集而得, 为 PID 实时控制; $M=2$ 时位置信号由传递函数给出, 为 PID 仿真控制。位置跟踪结果如图 10-4 所示。

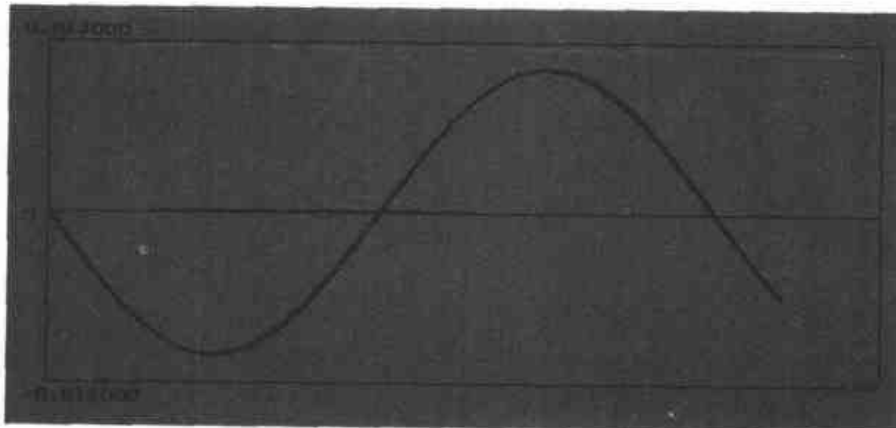


图 10-4 位置跟踪实时控制结果

```

仿真程序: chap10_3.cpp
//Real-time PID Control
//Options->Linker->Libraries->Graphics Library
//Copy c:\Borlandc\bgi\*.bgi files to the current directory
//Copy the H file "chap10_3.h" to \include directory
//Set:(1)c:\borlandc\include;(2)c:\borlandc\lib;
#include <math.h> //Using sin()
#include <conio.h> //Using clrscr() and getch() and kbhit()
#include <graphics.h>
#include <stdio.h> //Using sprintf()
#include <dos.h> //Using _disable() and _enable()
#include <string.h>
#include <bios.h> //Specially for std_fun bioskey()
#include <stdlib.h> //Using exit()
#include <chap10_3.h> //Interupt parameters

#define OUTPORT outportb
#define INPORT inportb
#define T 500
#define DD_PORT 0x380 //D/D board base address

int channel=1; //three frame: 0:inner; 1:middle; 2:outter
int Signal=1; //Sine Signal
//int Signal=2; //Step Signal

double PositionCommand[TIMER_RATIO];
double Line1[TIMER_RATIO];
double CurrentPosition[TIMER_RATIO];
double Line2[TIMER_RATIO];

double A,F;
double u=0.0,ts=0.001;
double pi=3.14159265358979;
double timezt=0;
int flag=0;
unsigned short UpdateFlag=0;
double mStep,mStep_1,Step,SaveStep=0; //mStep_1(t)=mStep(t-10)
//mStep must defined double
unsigned short ItemNum;
unsigned short TimerCount;

```

```

unsigned long    Counter;

unsigned short   flagF5;
unsigned short   FinishSimulate;
FILE *xyz;

#define DD_PORT  0x380  //Inner frame
void ResetShuXianBiao()
{
    outport(DD_PORT,0x0);
    outport(DD_PORT,0xf);
    outport(DD_PORT+4,0x0);
    outport(DD_PORT+4,0xf);
    outport(DD_PORT+8,0x0);
    outport(DD_PORT+8,0xf);
}

double angle_1;
double ReadD_D(unsigned short channel)
{
    unsigned short Data_H1,Data_L1,Data_H2,Data_L2;
    long int data,angle_degree,angle_minute,angle_second;
    double angle,angle1,error;
    do {
        Data_L1=inport(DD_PORT+4*channel);
        Data_H1=inport(DD_PORT+2+4*channel);
        Data_L2=inport(DD_PORT+4*channel);
        Data_H2=inport(DD_PORT+2+4*channel);
    }while((Data_L1!=Data_L2)||((Data_H1!=Data_H2)));

    data=((Data_H1 & 0x3f)*65536+Data_L1);
    angle1=data*0.9;    //Second per data
    if((Data_H1 & 0x80) ==0x80) //negative if Data_H.7 ==1
        angle1=-angle1;
    angle=angle1/3600.0; //Change from Second to degree
    return(angle);
}

#define Da_Board  0x1A0
void Write_DA(double dValue, int channel)

```

```

{
    unsigned int hi,low,hilow;
    int state;
    double voltage;
    voltage=dValue;
    if(voltage>2.0)voltage=2.0;
    if(voltage<-2.0)voltage=-2.0;
    hilow=(unsigned int)((voltage)*65535/20);

    hi=hilow&0xff00;
    hi=hi>>8;
    low=hilow&0xff;

    outportb(Da_Board+0,channel); //Select channel
do
    {
        state=(inportb(Da_Board+0))&0x80;
        } while(state==1); //read bit D7; if D7=0 write data

    outportb(Da_Board+1,low); //write low byte
    outportb(Da_Board+2,hi); //write high byte
    outportb(Da_Board+3,0); //start da
}

int KbGetKey(int *ScanCode)
{
    int Key;
    int KeyCode;
    Key=bioskey(0);
    if((Key&0x00ff)==0)
    {
        KeyCode=0;
        *ScanCode=(Key>>8)&127;
    }
    else
    {
        KeyCode=Key&0xff;
        *ScanCode=0;
    }
    return(KeyCode);
}

```

```

}

int SavedFlag, SaveCounter;
void SetupKeyReaction(void)
{
    int Key, Scan;
    if(kbhit())
    {
        Key=KbGetKey(&Scan);
        if (Key==KB_C_N_F5 && Scan==KB_S_N_F5)
        {
            SavedFlag=1;
            SaveCounter=0;
        }
        if(Key==KB_C_A_X && Scan==KB_S_A_X)
        {
            FinishSimulate = 1;
        }
    }
}

double huge DataSaved[DATA_DIMENTION][DATA_LENGTH];
void DataSaveRoutine( long int SaveSpan)
{
    if(SavedFlag==1)
    {
        if((SaveCounter<=SaveSpan*DATA_LENGTH)) //DATA_LENGTH is Defined by
chap10_3.h
        {
            int SaveIndex = SaveCounter/SaveSpan;
            //Save data(rin,yout) to xyz.dat
            DataSaved[0][SaveIndex]=PositionCommand[ItemNum];
            DataSaved[1][SaveIndex]=CurrentPosition[ItemNum];
        }
        SaveCounter++;
        if(SaveCounter>=SaveSpan*DATA_LENGTH) { SavedFlag=0; SaveCounter=0; }
    }
    if(SavedFlag==0) SaveCounter=0;
}

```

```

int M;
double yout,error,derror;
double u_1=0,u_2=0,y_1=0,y_2=0,error_1=0,error_2=0,ei=0;
double Control(double rin,unsigned short channel)
{
M=2;
if( M==1) //Realtime control
{
yout=ReadD_D(channel); //Read realtime data
CurrentPosition[ItemNum]=yout;
}
if( M==2 ) //Simulation control
{
yout=1.94*y_1-0.94*y_2+0.0008674*u_1+0.0008503*u_2;
}

CurrentPosition[ItemNum]=yout;
yout=CurrentPosition[ItemNum];

F=0.50;
A=0.010; if(A==0.0) { A=0.0001; }
rin=A*sin(F*2*pi*timezt);
error=rin-yout;

u=10.0*error+1.0*derror+0*ei;
//Update Parameters
y_2=y_1;y_1=yout;
u_2=u_1;u_1=u;
error_2=error_1;
error_1=error;

return u;
}
int Cycles=5; //Display cycle times
void DynamicDisplay()
{
//Make a window
char strA[50];
int i,Spoint;
int bottom,middle,top,right,left;

```



```

    bottom=300;middle=200;top=100;left=50;right=550;
    setcolor(RED);
    outtextxy(270,50,"PID Controller");
    line(left,top,left,bottom);        //lineleft
    sprintf(strA,"%f",A*6.0/5.0);
    outtextxy(36,top-10,strA);
    line(left,top,right,top);          //linetop
    outtextxy(36,middle,"0");
    line(left,middle,right,middle);    //linemiddle
    sprintf(strA,"%f",-A*6.0/5.0);
    outtextxy(36,bottom+5,strA);
    line(left,bottom,right,bottom);    //linedown
    line(right,top,right,bottom);      //lineright

//Make Curve Range
for(i=0;i<TIMER_RATIO;i++) //Plot 10 points once a time
{
    Spoint=mStep_1/Step-TIMER_RATIO+i; //Start point

    Spoint=Spoint/Cycles; //25000/(10*T)=One Cycle
    if(Spoint%T==0)
    {
        clrscr();
    }
    putpixel(Spoint-(Spoint/T)*T+50,-100*5.0/6.0*
        (Line1[i]/A)+200,BLUE); //Practical output
    putpixel(Spoint-(Spoint/T)*T+50,-100*5.0/6.0*
        (Line2[i]/A)+200,RED); //Ideal output
    putpixel(Spoint-(Spoint/T)*T+50,-100*5.0/6.0*
        u+200,BLACK); //Control output
}
}

void IniTimer( unsigned char timer_mode ) //Initial 8253
{
    unsigned char value;
    _disable();

//Seting control word
    OUTPORT(TIMER_BASE+3,timer_mode);
}

```

```

    value = TIMER_VALUE ;      //Low 8 bits
    OUTPUT( TIMER_BASE , value );
    value = TIMER_VALUE / 256 ; //High 8 bits
    OUTPUT( TIMER_BASE , value );
    _enable();
}

void interrupt ( *OldIrqVect )( __CPPARGS );
void interrupt TimerIrqVect( __CPPARGS ) //Interrupt processing
{
    unsigned short i;
    _disable();
    ItemNum = Counter%TIMER_RATIO;

    if(Counter%20000==0)
    {
        Counter=0;
        SaveStep=SaveStep+mStep;
        mStep=0;
    }
    mStep += Step;          //k*ts: TIMER_CYCLE / TIMER_RATIO;

    if (Signal==1) //Dynamic Signal
    { PositionCommand[ItemNum]=A*sin(F*2*pi*mStep); }
    if (Signal==2) //Static Signal
    { PositionCommand[ItemNum]=A; }

    u=Control(PositionCommand[ItemNum],channel);

    Write_DA(u,channel);
    DataSaveRoutine(1);

    Counter++;
    if((--TimerCount)==0) //Updating 1 times while interrupted by 10 times
    {
        UpdateFlag=1;
        mStep_1=mStep;
        TimerCount=TIMER_RATIO;
    }
    OUTPUT(EOI,0x20); //OCW2 value:0010 0000

```

```

    _enable();
}

void IrqHook( unsigned short irqnumber, void interrupt
( *newVect )( __CPPARGS ) )
{
    _disable();
    OldIrqVect = getvect ( irqnumber );
    setvect ( irqnumber , newVect );
    _enable();
}

void ReleaseHardware()
{
    _disable();
    setvect (Irqnumber,OldIrqVect);
    Write_DA(0.0,0);
    Write_DA(0.0,1);
    Write_DA(0.0,2);
    _enable();
}

main(void)
{
    int i,j,k;
    unsigned char timer0_mode,strtime[10];
    int driver,mode;
    driver=DETECT;
    initgraph(&driver,&mode,"");
    timer0_mode=T8253_MODE_3|T8253_CHANNEL_0|
        T8253_BIN_MODE|T8253_LOW_FIRST; //8253 timer setting

    Step=TIMER_CYCLE/TIMER_RATIO; //Step=0.01/10=0.001s=ts
    mStep=0;
    mStep_1=0;
    Counter=0;
    TimerCount=TIMER_RATIO;

    clrscr();
    ResetShuXianBiao();
}

```

```

//Using interrupt function
    IniTimer(timer0_mode); //8253 timer setting
    IrqHook(Irqnumber,TimerIrqVect);

while(1)          //1 is true
{
    timezt=SaveStep+mStep;
    if(UpdateFlag==1)
    {
        UpdateFlag=0;
        for(i=0;i<TIMER_RATIO;i++)
        {
            Line1[i]=PositionCommand[i];
            Line2[i]=CurrentPosition[i];
        }
        SetupKeyReaction();
        DynamicDisplay();
    }

    sprintf(strtime,"%f",timezt);
    bar(260,380,360,360);
    outtextxy(285,385,"time(s)");
    outtextxy(280,370,strtime);

//    if(kbhit()) { break; } //Break by any key
    if(FinishSimulate==1) //Break by "ALT+X"
        break;
} //End of for() loop

//Open xyz.dat for save data
if ((xyz = fopen("xyz.dat", "w"))== NULL)
{
    printf("Cannot open output file xyz.dat.\n");
    exit( 1 );
}
//xyz=fopen("xyz.dat","w");

for( i=0;i<DATA_LENGTH;i++)
{
    for( j=0; j<DATA_DIMENTION; j++)

```

```

        {
            fprintf(xyz, " %10.6f ",DataSaved[j][i]);
        }
        fprintf(xyz, "\n");
    }
fclose(xyz);

ReleaseHardware(); //Clear all output u
getch(); //Any key to exit
closegraph();
//Restores the original video mode detected by initgraph
restorecrtmode();
return 0;
} //End of main{}

```

参数初始化程序: chap10_3.h

```

#define TIMER_BASE      0x40    //Timer base address
#define TIMER_RATIO     10     //Display timer rate
#define TIMER_CYCLE     0.001*TIMER_RATIO //Display time cycle

#define TIMER_VALUE 1193 //Define interupt time:sampling time(ts=0.001s)

#define Irqnumber 0x08 //Realtime interupt kind:Clock Interupt

//Save data parameters
#define DATA_DIMENTION      2
#define DATA_LENGTH        100

//8253 timer(P225)
#define T8253_MODE_0        0x00
#define T8253_MODE_1        0x02
#define T8253_MODE_2        0x04
#define T8253_MODE_3        0x06
#define T8253_MODE_4        0x08
#define T8253_MODE_5        0x0a

#define T8253_CHANNEL_0     0x00
#define T8253_CHANNEL_1     0x40
#define T8253_CHANNEL_2     0x80

```

```
#define T8253_BIN_MODE      0x00
#define T8253_BCD_MODE     0x01

#define T8253_COUNT_LOCK   0x00
#define T8253_COUNT_LOW   0x10
#define T8253_COUNT_HI    0x20
#define T8253_LOW_FIRST   0x30
```

```
//8259 control(P197)
```

```
#define EOI          0x20
```

```
//Define key value//
```

```
#define KB_C_N_F4  0
```

```
#define KB_S_N_F4  62
```

```
#define KB_C_N_F5  0
```

```
#define KB_S_N_F5  63
```

```
#define KB_S_A_X   45
```

```
#define KB_C_A_X   0
```

```
#ifdef __cplusplus
```

```
    #define __CPPARGS ...
```

```
#endif
```