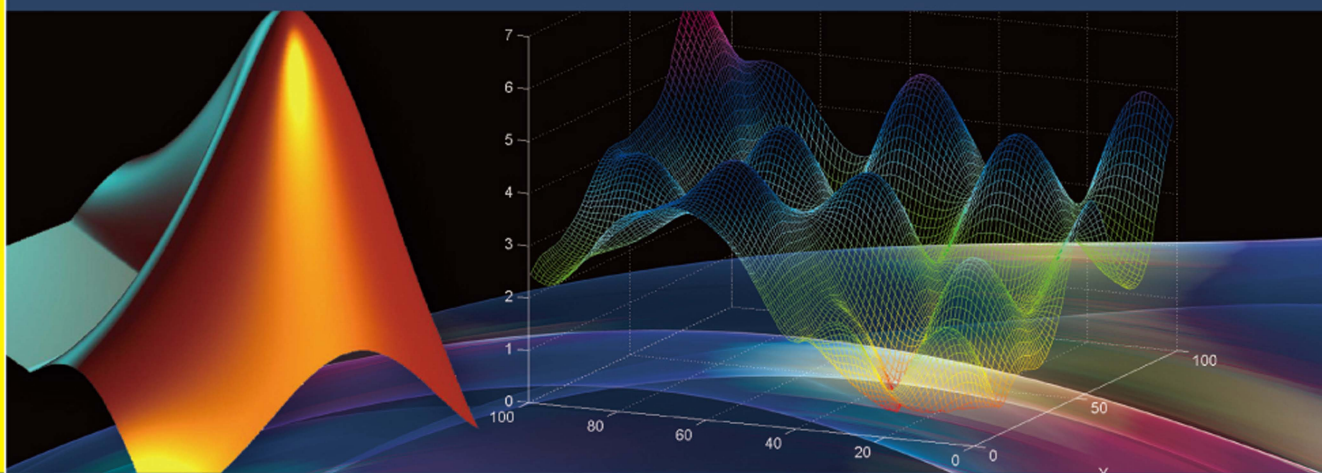


电子工程技术丛书

<http://www.phei.com.cn>

先进 PID 控制 MATLAB 仿真 (第3版)

● 刘金琨 主编



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

电子工程技术丛书

先进 PID 控制 MATLAB 仿真

(第 3 版)

刘金琨 编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书系统地介绍了 PID 控制的几种设计方法,是作者多年来从事控制系统教学和科研工作的结晶,同时融入了国内外同行近年来所取得的最新成果。

全书共分 14 章,包括基本的 PID 控制、PID 控制器的整定、时滞系统的 PID 控制、基于微分器的 PID 控制、基于观测器的 PID 控制、自抗扰控制器及其 PID 控制、PD 鲁棒自适应控制、模糊 PD 控制和专家 PID 控制、神经 PID 控制、基于遗传算法整定的 PID 控制、伺服系统 PID 控制、迭代学习 PID 控制其他控制方法的设计与仿真,以及 PID 实时控制的 C++语言设计及应用。每种方法都给出了算法推导、实例分析和相应的 MATLAB 仿真设计程序。

本书各部分内容既相互联系又相互独立,读者可根据自己的需要选择学习。本书适用于从事生产过程自动化、计算机应用、机械电子和电气自动化领域工作的工程技术人员阅读,也可作为大专院校工业自动化、自动控制、机械电子、自动化仪表、计算机应用等专业的教学参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

先进 PID 控制 MATLAB 仿真/刘金琨编著. —3 版. —北京:电子工业出版社,2011.3
(电子工程技术丛书)
ISBN 978-7-121-13049-6

I. ①先… II. ①刘… III. ①PID 控制②计算机辅助计算—软件包, MATLAB
IV. ①TP273②TP391.75

中国版本图书馆 CIP 数据核字(2011)第 039369 号

策划编辑:赵丽松

责任编辑:李雪梅

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1 092 1/16 印张:30 字数:768 千字

印 次:2011 年 3 月第 1 次印刷

印 数:4 000 册 定价:59.80 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

前 言

PID 控制是最早发展起来的控制策略之一, 由于其算法简单、鲁棒性好和可靠性高, 被广泛应用于过程控制和运动控制中, 尤其适用于可建立精确数学模型的确定性控制系统。然而实际工业生产过程往往具有非线性、时变不确定性, 难以建立精确的数学模型, 应用常规 PID 控制器不能达到理想的控制效果, 而且在实际生产现场中, 由于受到参数整定方法繁杂的困扰, 常规 PID 控制器参数往往整定不良、性能欠佳, 对运行工况的适应性很差。

计算机技术和智能控制理论的发展为复杂动态不确定系统的控制提供了新的途径。采用智能控制技术, 可设计智能 PID 和进行 PID 的智能整定。

有关智能 PID 控制等新型 PID 控制理论及其工程应用, 近年来已有大量的论文发表。作者多年来一直从事智能控制方面的研究和教学工作, 为了促进 PID 控制和自动化技术的进步, 反映 PID 控制设计与应用中的最新研究成果, 并使广大工程技术人员能了解、掌握和应用这一领域的最新技术, 学会用 MATLAB 语言进行 PID 控制器的设计, 作者编写了这本书, 以抛砖引玉, 供广大读者学习参考。

本书是在总结作者多年研究成果的基础上, 进一步理论化、系统化、规范化、实用化而成的, 其特点如下。

(1) PID 控制算法取材新颖, 内容先进, 重点置于学科交叉部分的前沿研究和介绍一些有潜力的新思想、新方法和新技术, 取材着重于基本概念、基本理论和基本方法。

(2) 针对每种 PID 算法给出了完整的 MATLAB 仿真程序, 这些程序都可以在线运行, 并给出了程序的说明和仿真结果, 具有很强的可读性, 很容易转化为其他各种实用语言。

(3) 着重从应用领域角度出发, 突出理论联系实际, 面向广大工程技术人员, 具有很强的工程性和实用性。书中有大量应用实例及其结果分析, 为读者提供了有益的借鉴。

(4) 所给出的各种 PID 算法完整, 程序结构设计力求简单明了, 便于自学和进一步开发。

本书共分 14 章。第 1 章介绍了连续系统 PID 控制和离散系统数字 PID 控制的几种基本方法, 通过仿真和分析进行了说明; 第 2 章介绍了 PID 控制器整定的几种方法; 第 3 章介绍了时滞系统的 PID 控制, 包括串级计算机控制系统的 PID 控制、纯滞后控制系统 Dahlin 算法和基于 Smith 预估的 PID 控制; 第 4 章介绍了基于微分器的 PID 控制, 包括基于全程快速微分器和基于 Levant 微分器的 PID 控制; 第 5 章介绍了基于观测器的 PID 控制, 包括基于干扰观测器、扩张观测器和输出延迟观测器的 PID 控制; 第 6 章介绍了自抗扰控制器及其 PID 控制, 包括非线性跟踪微分器、安排过渡过程及 PID 控制、基于非线性扩张观测器的 PID 控制、非线性 PID 控制和自抗扰控制; 第 7 章介绍了几种 PID 鲁棒自适应控制方法, 包括挠性航天器稳定 PD 鲁棒控制、基于名义模型的机械手 PI 鲁棒控制、基于 Anti-windup 的 PID 抗饱和控制和基于增益自适应调节的模型参考自适应 PD 控制; 第 8 章介绍了专家 PID 和模糊 PID 整定方法, 其中模糊 PID 包括自适应模糊补偿的倒立摆 PD 控制、基于模糊规则表的模糊 PD 控制和模糊自适应整定 PID 控制; 第 9 章介绍了神经网络 PID 控制, 包括基于单神经元网络的 PID 控制、基于 RBF 神经网络整定的 PID 控制和基于自适应神经网络补偿的倒立摆 PD 控制; 第 10 章介绍了基于遗传算法的 PID 控制, 主要包括基于遗传算法整定的 PID 控制和基于遗传算法摩擦模型参数辨识的 PID 控制; 第 11 章介绍了伺服系统的 PID 控制, 包括伺服系统在低速摩擦条件下的 PID 控制、单质量伺服系统 PID 控制和二质量伺服系统 PID 控制;

第 12 章介绍了迭代学习 PID 控制，包括迭代学习 PID 控制基本原理和基本设计方法；第 13 章介绍了其他控制方法，针对每种方法给出了实例说明；第 14 章介绍了 PID 在实时控制中的应用实例，并给出了相应的 Borland C++语言实时控制程序。

本书是基于 MATLAB 7.1 环境下开发的，各个章节的内容具有很强的独立性，读者可以结合自己的方向深入地进行研究。

北京航空航天大学尔联洁教授在伺服系统设计方面提出了许多宝贵意见，东北大学徐心和教授和薛定宇教授给予了大力支持和帮助，北京航空航天大学林岩教授和全权博士也给予了帮助，在此一并表示感谢。

作者在控制系统的分析中，有许多方面得益于与研究生的探讨，这些研究生包括孔建、卢宇、贺庆、郑明慧、张琳军、李晓光等，在此表示感谢。

由于作者水平有限，书中难免存在一些不足和错误之处，欢迎广大读者批评指正。

刘金琨
北京航空航天大学
2010 年 7 月 30 日

再 版 说 明

本书在第 2 版的基础上主要增加了以下内容：PID 控制器的整定、基于微分器的 PID 控制、基于观测器的 PID 控制、自抗扰控制器及其 PID 控制、PD 鲁棒自适应控制、自适应模糊补偿 PD 控制、自适应神经网络补偿 PD 控制、迭代学习 PID 控制及其他控制方法的介绍，给出了基于 GUI 的倒立摆控制动画演示，并针对第 2 版中的某些错误进行了修改。

作者简介

刘金琨：辽宁人，1965年生。分别于1989年7月、1994年3月和1997年3月获东北大学工学学士、工学硕士和工学博士学位。1997年3月至1998年12月在浙江大学工业控制技术研究所做博士后研究工作。1999年1月至1999年7月在香港科技大学从事合作研究。1999年11月至今在北京航空航天大学自动化学院从事教学与科研工作，现任教授，博士生导师。主讲“智能控制”、“工业过程控制”和“系统辨识”等课程。研究方向为控制理论与应用。自从从事研究工作以来，主持国家自然科学基金等科研项目10余项，以第一作者发表学术论文70余篇。曾出版《智能控制》、《先进PID控制及其MATLAB仿真》、《机器人控制系统的设计与MATLAB仿真》、《滑模变结构控制MATLAB仿真》和《微分器设计与应用——信号滤波与求导》等著作。

目 录

第 1 章 基本的 PID 控制	1
1.1 PID 控制原理	1
1.2 连续系统的模拟 PID 仿真	2
1.2.1 基本的 PID 控制	2
1.2.2 线性时变系统的 PID 控制	8
1.3 数字 PID 控制	12
1.3.1 位置式 PID 控制算法	12
1.3.2 连续系统的数字 PID 控制仿真	13
1.3.3 离散系统的数字 PID 控制仿真	19
1.3.4 增量式 PID 控制算法及仿真	25
1.3.5 积分分离 PID 控制算法及仿真	27
1.3.6 抗积分饱和 PID 控制算法及仿真	32
1.3.7 梯形积分 PID 控制算法	35
1.3.8 变速积分 PID 算法及仿真	35
1.3.9 带滤波器的 PID 控制仿真	39
1.3.10 不完全微分 PID 控制算法及仿真	45
1.3.11 微分先行 PID 控制算法及仿真	49
1.3.12 带死区的 PID 控制算法及仿真	52
1.3.13 基于前馈补偿的 PID 控制算法及仿真	56
1.3.14 步进式 PID 控制算法及仿真	59
1.3.15 PID 控制的方波响应	61
1.3.16 基于卡尔曼滤波器的 PID 控制	64
1.4 S 函数介绍	73
1.4.1 S 函数简介	73
1.4.2 S 函数使用步骤	73
1.4.3 S 函数的基本功能及重要参数设定	73
1.4.4 实例说明	74
1.5 PID 研究新进展	74
第 2 章 PID 控制器的整定	76
2.1 概述	76
2.2 基于响应曲线法的 PID 整定	76
2.2.1 基本原理	76
2.2.2 仿真实例	77
2.3 基于 Ziegler-Nichols 的频域响应 PID 整定	81
2.3.1 连续 Ziegler-Nichols 方法的 PID 整定	81

2.3.2	仿真实例	81
2.3.3	离散 Ziegler-Nichols 方法的 PID 整定	84
2.3.4	仿真实例	84
2.4	基于频域分析的 PD 整定	88
2.4.1	基本原理	88
2.4.2	仿真实例	88
2.5	基于相位裕度整定的 PI 控制	91
2.5.1	基本原理	91
2.5.2	仿真实例	94
2.6	基于极点配置的稳定 PD 控制	95
2.6.1	基本原理	95
2.6.2	仿真实例	96
2.7	基于临界比例度法的 PID 整定	98
2.7.1	基本原理	98
2.7.2	仿真实例	99
2.8	一类非线性整定的 PID 控制	101
2.8.1	基本原理	101
2.8.2	仿真实例	103
2.9	基于优化函数的 PID 整定	105
2.9.1	基本原理	105
2.9.2	仿真实例	105
2.10	基于 NCD 优化的 PID 整定	107
2.10.1	基本原理	107
2.10.2	仿真实例	107
2.11	基于 NCD 与优化函数结合的 PID 整定	111
2.11.1	基本原理	111
2.11.2	仿真实例	111
2.12	传递函数的频域测试	113
2.12.1	基本原理	113
2.12.2	仿真实例	114
第 3 章	时滞系统的 PID 控制	117
3.1	单回路 PID 控制系统	117
3.2	串级 PID 控制	117
3.2.1	串级 PID 控制原理	117
3.2.2	仿真实例	118
3.3	纯滞后系统的大林控制算法	122
3.3.1	大林控制算法原理	122
3.3.2	仿真实例	122
3.4	纯滞后系统的 Smith 控制算法	124
3.4.1	连续 Smith 预估控制	125

3.4.2	仿真实例	126
3.4.3	数字 Smith 预估控制	128
3.4.4	仿真实例	129
第 4 章	基于微分器的 PID 控制	134
4.1	基于全程快速微分器的 PID 控制	134
4.1.1	全程快速微分器	134
4.1.2	仿真实例	134
4.2	基于 Levant 微分器的 PID 控制	143
4.2.1	Levant 微分器	143
4.2.2	仿真实例	144
第 5 章	基于观测器的 PID 控制	156
5.1	基于慢干扰观测器补偿的 PID 控制	156
5.1.1	系统描述	156
5.1.2	观测器设计	156
5.1.3	仿真实例	157
5.2	基于干扰观测器的 PID 控制	162
5.2.1	干扰观测器基本原理	162
5.2.2	干扰观测器的性能分析	164
5.2.3	干扰观测器鲁棒稳定性	166
5.2.4	低通滤波器 $Q(s)$ 的设计	167
5.2.5	仿真实例	168
5.3	基于扩张观测器的 PID 控制	172
5.3.1	扩张观测器的设计	172
5.3.2	扩张观测器的分析	173
5.3.3	仿真实例	175
5.4	基于输出延迟观测器的 PID 控制	189
5.4.1	系统描述	189
5.4.2	输出延迟观测器的设计	189
5.4.3	延迟观测器的分析	190
5.4.4	仿真实例	191
第 6 章	自抗扰控制器及其 PID 控制	201
6.1	非线性跟踪微分器	201
6.1.1	微分器描述	201
6.1.2	仿真实例	201
6.2	安排过渡过程及 PID 控制	205
6.2.1	安排过渡过程	205
6.2.2	仿真实例	206
6.3	基于非线性扩张观测器的 PID 控制	212
6.3.1	系统描述	212
6.3.2	非线性扩张观测器	212

6.3.3	仿真实例	213
6.4	非线性 PID 控制	225
6.4.1	非线性 PID 控制算法	225
6.4.2	仿真实例	225
6.5	自抗扰控制	228
6.5.1	自抗扰控制结构	228
6.5.2	仿真实例	228
第 7 章	PD 鲁棒自适应控制	239
7.1	挠性航天器稳定 PD 鲁棒控制	239
7.1.1	挠性航天器建模	239
7.1.2	PD 控制器的设计	240
7.1.3	仿真实例	240
7.2	基于名义模型的机械手 PI 鲁棒控制	245
7.2.1	问题的提出	245
7.2.2	鲁棒控制律的设计	246
7.2.3	稳定性分析	246
7.2.4	仿真实例	247
7.3	基于 Anti-windup 的 PID 控制	255
7.3.1	Anti-windup 基本原理	255
7.3.2	基于 Anti-windup 的 PID 控制	255
7.3.3	仿真实例	256
7.4	基于 PD 增益自适应调节的模型参考自适应控制	259
7.4.1	问题描述	259
7.4.2	控制律的设计与分析	260
7.4.3	仿真实例	261
第 8 章	模糊 PD 控制和专家 PID 控制	270
8.1	倒立摆稳定的 PD 控制	270
8.1.1	系统描述	270
8.1.2	控制律设计	270
8.1.3	仿真实例	271
8.2	基于自适应模糊补偿的倒立摆 PD 控制	274
8.2.1	问题描述	274
8.2.2	自适应模糊控制器设计与分析	275
8.2.3	稳定性分析	276
8.2.4	仿真实例	277
8.3	基于模糊规则表的模糊 PD 控制	284
8.3.1	基本原理	284
8.3.2	仿真实例	285
8.4	模糊自适应整定 PID 控制	288
8.4.1	模糊自适应整定 PID 控制原理	288

8.4.2	仿真实例	291
8.5	专家 PID 控制	296
8.5.1	专家 PID 控制原理	296
8.5.2	仿真实例	297
第 9 章	神经 PID 控制	301
9.1	基于单神经网络的 PID 智能控制	301
9.1.1	几种典型的学习规则	301
9.1.2	单神经元自适应 PID 控制	301
9.1.3	改进的单神经元自适应 PID 控制	302
9.1.4	仿真实例	303
9.1.5	基于二次型性能指标学习算法的单神经元自适应 PID 控制	305
9.1.6	仿真实例	306
9.2	基于 RBF 神经网络整定的 PID 控制	309
9.2.1	RBF 神经网络模型	309
9.2.2	RBF 网络 PID 整定原理	310
9.2.3	仿真实例	311
9.3	基于自适应神经网络补偿的倒立摆 PD 控制	316
9.3.1	问题描述	316
9.3.2	自适应神经网络设计与分析	316
9.3.3	仿真实例	319
第 10 章	基于遗传算法整定的 PID 控制	325
10.1	遗传算法的基本原理	325
10.2	遗传算法的优化设计	326
10.2.1	遗传算法的构成要素	326
10.2.2	遗传算法的应用步骤	326
10.3	遗传算法求函数极大值	327
10.3.1	二进制编码遗传算法求函数极大值	327
10.3.2	实数编码遗传算法求函数极大值	331
10.4	基于遗传算法的 PID 整定	334
10.4.1	基于遗传算法的 PID 整定原理	335
10.4.2	基于实数编码遗传算法的 PID 整定	337
10.4.3	基于二进制编码遗传算法的 PID 整定	341
10.4.4	基于自适应在线遗传算法整定的 PD 控制	347
10.5	基于摩擦模型补偿的 PD 控制	352
10.5.1	摩擦模型辨识	352
10.5.2	仿真实例	353
第 11 章	伺服系统 PID 控制	359
11.1	基于 LuGre 摩擦模型的 PID 控制	359
11.1.1	伺服系统的摩擦现象	359
11.1.2	伺服系统的 LuGre 摩擦模型	359

11.1.3	仿真实例	360
11.2	基于 Stribeck 摩擦模型的 PID 控制	362
11.2.1	Stribeck 摩擦模型描述	362
11.2.2	一个典型伺服系统描述	363
11.2.3	仿真实例	364
11.3	伺服系统三环的 PID 控制	371
11.3.1	伺服系统三环的 PID 控制原理	371
11.3.2	仿真实例	372
11.4	二质量伺服系统的 PID 控制	375
11.4.1	二质量伺服系统的 PID 控制原理	375
11.4.2	仿真实例	377
11.5	伺服系统的模拟 PD+数字前馈控制	379
11.5.1	伺服系统的模拟 PD+数字前馈控制原理	379
11.5.2	仿真实例	380
第 12 章	迭代学习 PID 控制	382
12.1	迭代学习控制方法介绍	382
12.2	迭代学习控制基本原理	382
12.3	基本的迭代学习控制算法	383
12.4	基于 PID 型的迭代学习控制	383
12.4.1	系统描述	383
12.4.2	控制器设计	384
12.4.3	仿真实例	384
第 13 章	其他控制方法的设计与仿真	390
13.1	单级倒立摆建模	390
13.2	倒立摆 PD 控制	391
13.2.1	系统描述	391
13.2.2	仿真实例	391
13.3	单级倒立摆的全状态反馈控制	394
13.3.1	系统描述	394
13.3.2	全状态反馈控制	395
13.3.3	仿真实例	395
13.4	输入/输出反馈线性化	403
13.4.1	系统描述	403
13.4.2	控制律设计	404
13.4.3	仿真实例	404
13.5	倒立摆反演控制	408
13.5.1	系统描述	408
13.5.2	控制律设计	408
13.5.3	仿真实例	409
13.6	倒立摆滑模控制	413

13.6.1	问题描述	413
13.6.2	控制律设计	413
13.6.3	仿真实例	414
13.7	自适应鲁棒控制	419
13.7.1	问题的提出	419
13.7.2	自适应控制律的设计	419
13.7.3	仿真实例	420
13.8	单级倒立摆的 H_∞ 控制	427
13.8.1	系统描述	427
13.8.2	H_∞ 控制器要求	428
13.8.3	基于 Riccati 方程的 H_∞ 控制	429
13.8.4	基于 LMI 的 H_∞ 控制	429
13.8.5	仿真实例	431
13.9	基于 GUI 的倒立摆控制动画演示	438
13.9.1	GUI 介绍	438
13.9.2	演示程序的构成	439
13.9.3	主程序的实现	439
13.9.4	演示界面的 GUI 设计	439
13.9.5	演示步骤	440
第 14 章	PID 实时控制的 C++ 语言 设计及应用	442
14.1	控制系统仿真的 C++ 实现	442
14.2	基于 C++ 的三轴飞行模拟转台伺服系统 PID 实时控制	444
14.2.1	控制系统构成	445
14.2.2	实时控制程序分析	445
14.2.3	仿真实例	449
附录 A	常用符号说明	459
	参考文献	460

第 1 章 基本的 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。

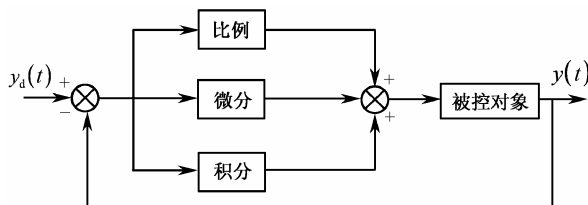


图 1-1 模拟 PID 控制系统原理框图

PID 控制器是一种线性控制器，它根据给定值 $y_d(t)$ 与实际输出值 $y(t)$ 构成控制偏差

$$\text{error}(t) = y_d(t) - y(t) \quad (1.1)$$

PID 的控制规律为

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

或写成传递函数的形式

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

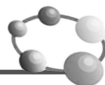
式中， k_p ——比例系数； T_i ——积分时间常数； T_D ——微分时间常数。

简单来说，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 $\text{error}(t)$ ，偏差一旦产生，控制器立即产生控制作用，以减小偏差。

(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

1.2.1 基本的 PID 控制

以二阶线性传递函数 $\frac{133}{s^2 + 25s}$ 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 $y_d(t) = A \sin(2\pi Ft)$ ，其中 $A = 1.0$ ， $F = 0.20\text{Hz}$ 。采用 ODE45 迭代方法，仿真时间为 10s。

仿真之一：连续系统 PID 的 Simulink 仿真

PID 控制器由 Simulink 下的工具箱提供。

仿真程序：chap1_1.mdl（见图 1-2 和图 1-3）

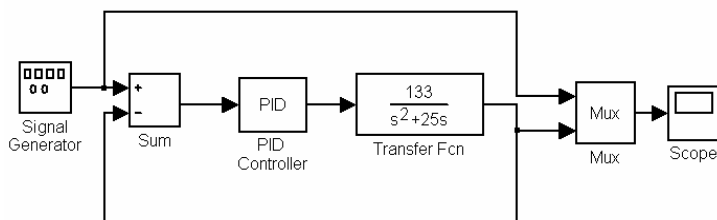


图 1-2 连续系统 PID 控制 Simulink 仿真程序

在 PID 控制器采用 Simulink 封装的形式，其内部结构如图 1-3 所示。

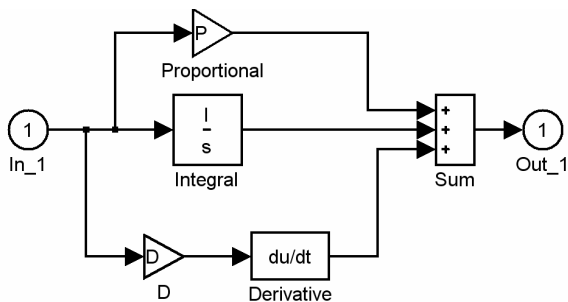


图 1-3 模拟 PID 控制器

连续系统的模拟 PID 控制正弦响应结果如图 1-4 所示。

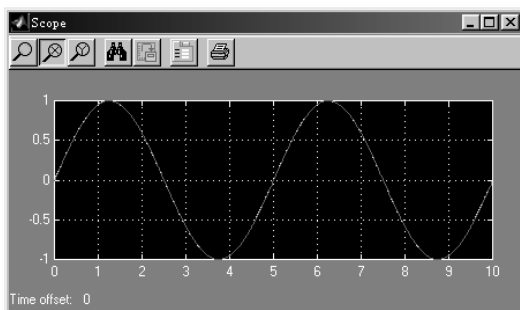


图 1-4 连续系统的模拟 PID 控制正弦响应



仿真之二：连续系统 PID 的 Simulink 仿真

在仿真一的基础上，将仿真结果输出到工作空间中，利用 M 语言作图，仿真结果如图 1-5 所示。

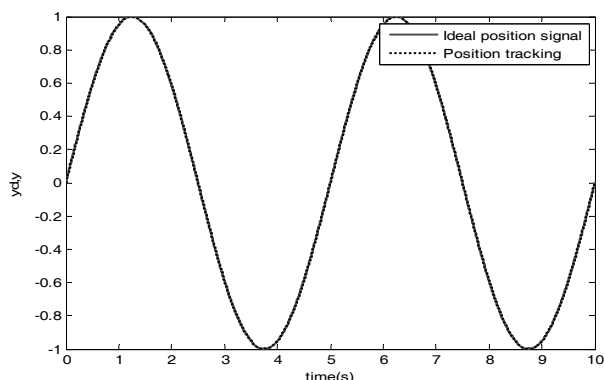


图 1-5 PID 控制正弦响应

仿真程序：chap1_2.mdl（见图 1-6）

程序中同时采用了传递函数 $\frac{133}{s^2 + 25s}$ 的另一种表达方式，即状态方程的形式，其中

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 133 \end{bmatrix}, \quad C = [1 \quad 0], \quad D = 0.$$

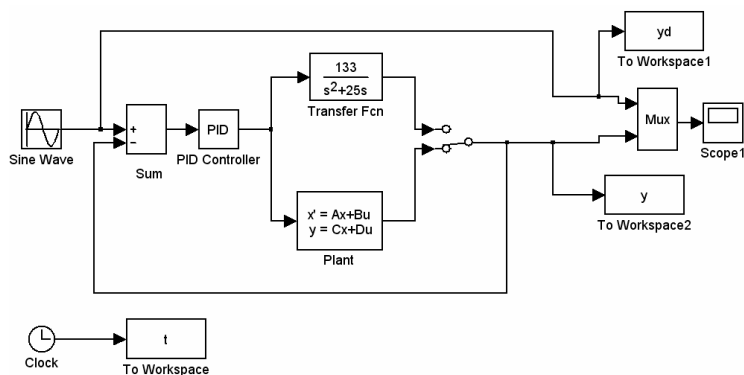


图 1-6 连续系统 PID 控制 Simulink 仿真程序

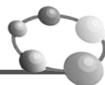
作图程序：chap1_2plot.m

```
close all;

plot(t,yd(:,1),'r','t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('ydy');
legend('Ideal position signal','Position tracking');
```

仿真之三：采用 S 函数实现 Simulink 仿真

仍以二阶线性传递函数为被控对象，进行模拟 PID 控制。被控对象形式为 $\frac{a}{s^2 + bs}$ ，其中 b 在 $[103,163]$ 范围内随机变化， a 在 $[15,35]$ 范围内随机变化，则被控对象的描述方式可转换



为

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -ax_2 + bu\end{aligned}$$

S 函数是 Simulink 的一项重要功能,采用 S 函数可实现在 Simulink 下复杂控制器和复杂被控对象的编程。在仿真一的基础上,利用 S 函数实现上述对象的表达、控制器的设计及仿真结果的输出。

在 S 函数中,采用初始化、微分函数和输出函数,即 `mdlInitializeSizes` 函数、`mdlDerivatives` 函数和 `mdlOutputs` 函数。在初始化中采用 `sizes` 结构,选择 2 个输出,3 个输入,3 个输入实现了 P、I、D 三项的输入。S 函数嵌入在 Simulink 程序中。系统初始状态为: $x(0) = 0, \dot{x}(0) = 0$ 。仿真结果如图 1-7 所示。

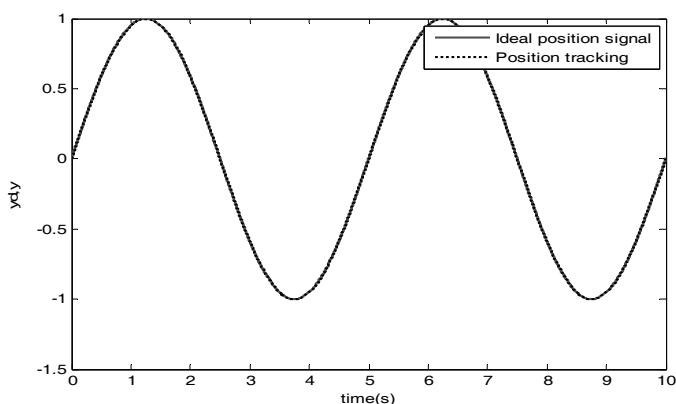


图 1-7 PID 控制正弦响应

仿真程序: `chap1_3.mdl` (见图 1-8)

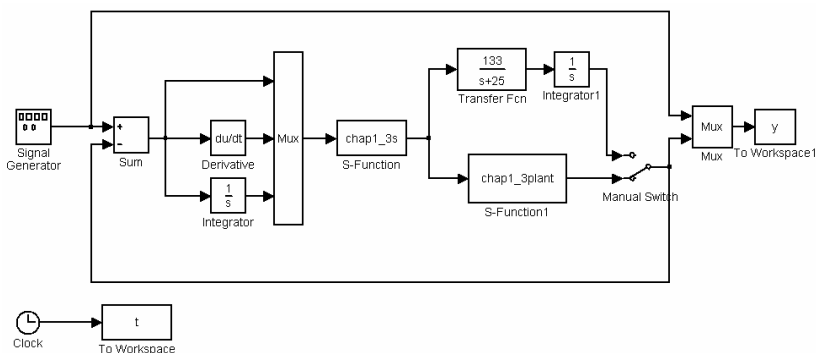


图 1-8 基于 S 函数的 PID 控制 Simulink 仿真程序

S 函数 PID 控制器程序: `chap1_3s.m`

```
%S-function for continuous state equation
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
%Initialization
case 0,
[sys,x0,str,ts]=mdlInitializeSizes;
```



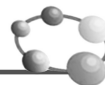
```
%Outputs
case 3,
    sys=mdlOutputs(t,x,u);
%Unhandled flags
case {2, 4, 9 }
    sys = [];
%Unexpected flags
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
error=u(1);
derror=u(2);
errori=u(3);

kp=60;
ki=1;
kd=3;
ut=kp*error+kd*derror+ki*errori;
sys(1)=ut;
```

S 函数被控对象程序: chap1_3plant.m

```
%S-function for continuous state equation
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
%Initialization
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
%Outputs
case 3,
```



```

        sys=mdlOutputs(t,x,u);
    %Unhandled flags
    case {2, 4, 9 }
        sys = [];
    %Unexpected flags
    otherwise
        error(['Unhandled flag = ',num2str(flag)]);
    end

    %mdlInitializeSizes
    function [sys,x0,str,ts]=mdlInitializeSizes
    sizes = simsizes;
    sizes.NumContStates = 2;
    sizes.NumDiscStates = 0;
    sizes.NumOutputs = 1;
    sizes.NumInputs = 1;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 0;

    sys=simsizes(sizes);
    x0=[0,0];
    str=[];
    ts=[];

    function sys=mdlDerivatives(t,x,u)
    sys(1)=x(2);
    %sys(2)=-(25+5*sin(t))*x(2)+(133+10*sin(t))*u;
    sys(2)= -(25+10*rands(1))*x(2)+(133+30*rands(1))*u;

    function sys=mdlOutputs(t,x,u)

    sys(1)=x(1);

```

作图程序: chap1_3plot.m

```

close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');

```

仿真之四：利用简化 S 函数，实现仿真三中 S 函数同样的功能

利用 S 函数简化形式实现被控对象的表达、控制器的设计及仿真结果的输出。在简化 S 函数中，flag=0 时为 S 函数初始化，其中 sys 包括 6 个参数，第 1 个参数表示连续系统的阶数，第 2 个参数表示离散系统的阶数，第 3 个参数表示 S 函数的输出个数，第 4 个参数表示 S 函数的输入个数，第 5 个参数表示直接馈通 (dirFeedthrough)，即输入信号是否在输出端出



现的标识, 取值为 0 或 1, 第 6 个参数表示模块采样周期的个数, S 函数支持多采样周期的系统, $x0=[]$ 为系统初始值设定; $flag=1$ 时为 S 函数被控对象微分方程的描述; $flag=3$ 时为 S 函数输出。仿真结果如图 1-9 所示。

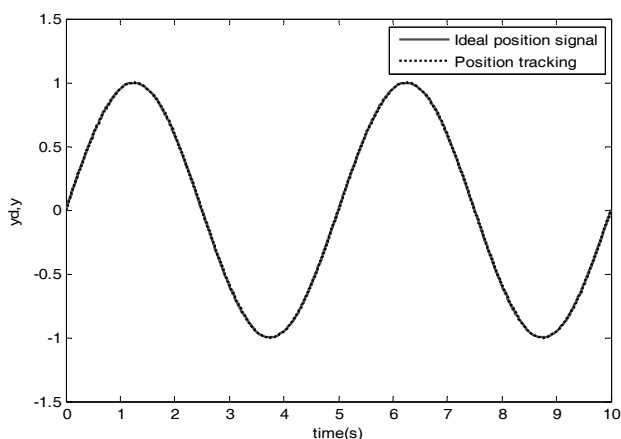


图 1-9 PID 控制正弦响应

仿真程序: chap1_3n.mdl (见图 1-10)

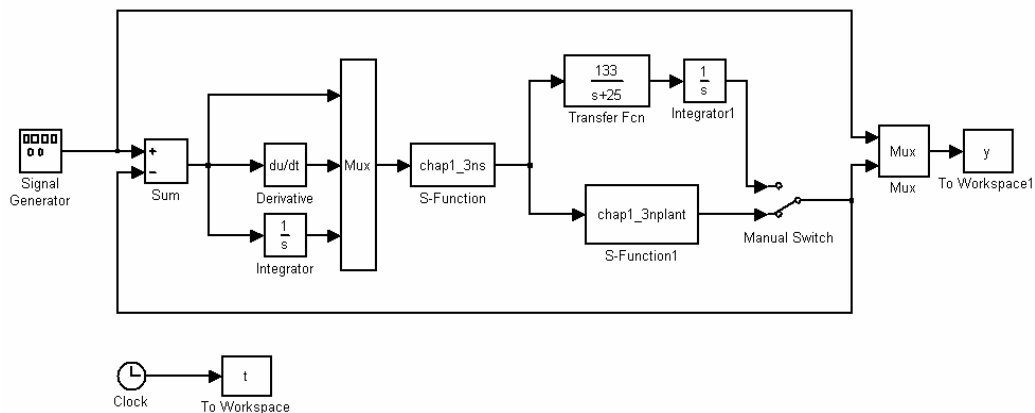
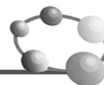


图 1-10 基于简化 S 函数的 Simulink 仿真

简化 S 函数控制器程序: chap1_3ns.m

```
function [sys,x0]=s_function(t,x,u,flag)
kp=60;ki=1;kd=3;
if flag==0
    sys=[0,0,1,3,0,1]; %Outputs=1,Inputs=3,DirFeedthrough=0;
    x0=[];
elseif flag==3
    sys(1)=kp*u(1)+ki*u(2)+kd*u(3);
else
    sys=[];
end
```

简化 S 函数被控对象程序: chap1_3nplant.m



```
function [sys,x0]=s_function(t,x,u,flag)
if flag==0
    sys=[2,0,1,1,0,0];    %ContStates=2,Outputs=1,Inputs=1
    x0=[0,0];
elseif flag==1
    sys(1)=x(2);
    sys(2)=-(25+10*rands(1))*x(2)+(133+30*rands(1))*u;
elseif flag==3
    sys(1)=x(1);
else
    sys=[];
end
```

作图程序: chap1_3nplot.m

```
close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

1.2.2 线性时变系统的 PID 控制

仿真实例

被控对象为

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

输入指令信号为 $0.5\sin(2\pi t)$, $J = 20 + 10\sin(6\pi t)$, $K = 400 + 300\sin(2\pi t)$ 。采用 PD 控制算法进行正弦响应。

仿真之一: 采用 Simulink 模块实现 Simulink 仿真

通过 Simulink 模块实现不确定对象的表示, 取 $k_p = 10$, $k_p = 10$, $k_p = 10$ 。仿真结果如图 1-11 所示。

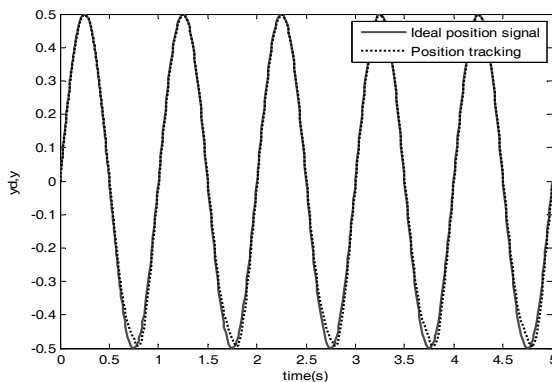


图 1-11 正弦响应



仿真程序: chap1_4.mdl (见图 1-12 和图 1-13)

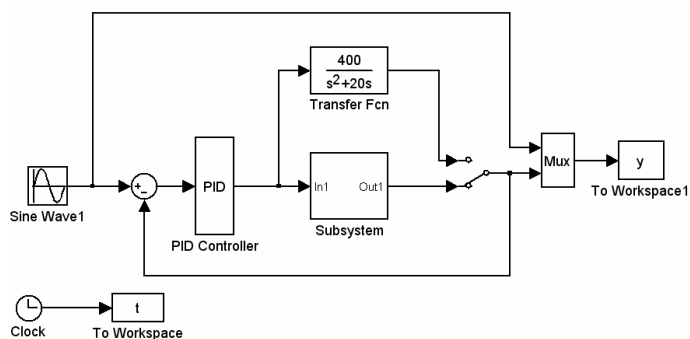


图 1-12 Simulink 主程序

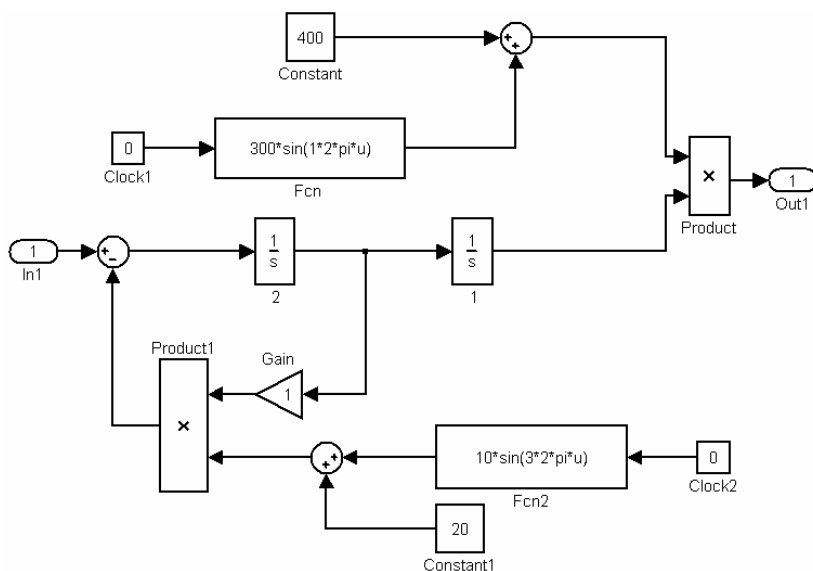


图 1-13 被控对象模块

作图程序: chap1_4plot.m

```
close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

仿真之二: 采用 S 函数进行 Simulink 仿真

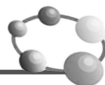
不确定对象的表示、控制器的实现及输出由 S 函数完成。

被控对象为

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

被控对象的描述方式可转换为

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -Jx_2 + Ku \end{aligned}$$



在 S 函数中,采用初始化、微分函数和输出函数,即 `mdlInitializeSizes` 函数、`mdlDerivatives` 函数和 `mdlOutputs` 函数。在初始化中采用 `sizes` 结构,选择 1 个输出,3 个输入,3 个输入实现了 P、I、D 三项的输入。S 函数嵌入在 Simulink 程序中。系统初始状态为: $x(0)=0, \dot{x}(0)=0$ 。取 $k_p=10$, $k_i=2$, $k_d=1$, 仿真结果如图 1-14 所示。

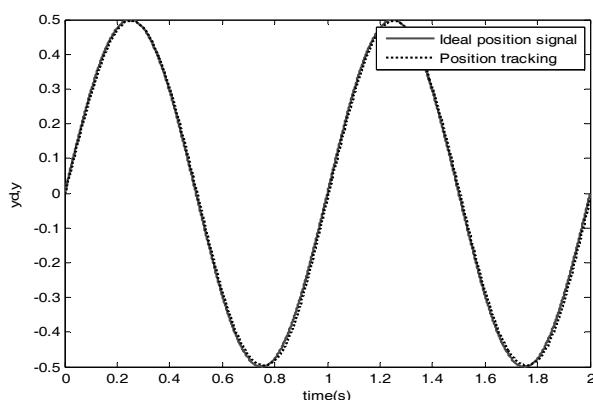


图 1-14 正弦响应

仿真程序

Simulink 主程序: `chap1_5.mdl` (见图 1-15)

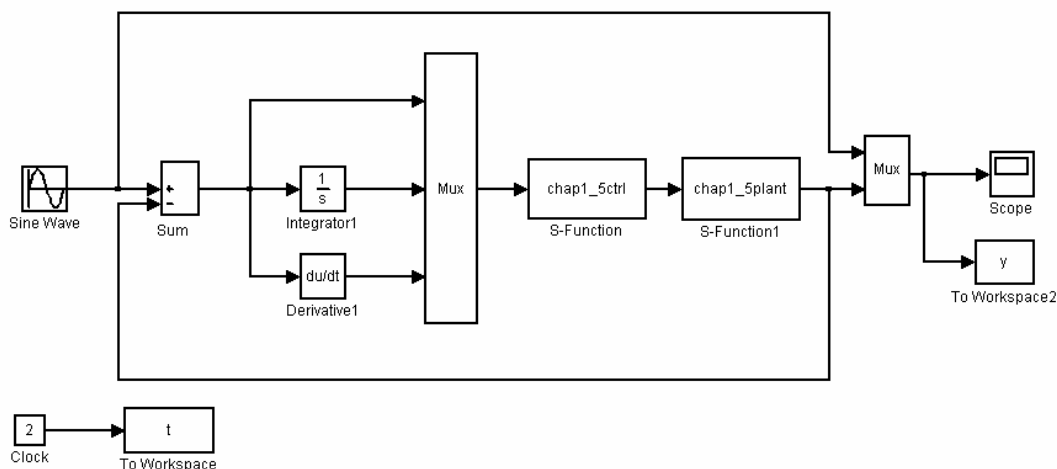


图 1-15 基于 S 函数的 Simulink 主程序

S 函数控制器子程序: `chap1_5ctrl.m`

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

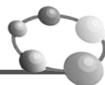
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
```



```
sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
kp=10;
ki=2;
kd=1;
ut=kp*u(1)+ki*u(2)+kd*u(3);
sys(1)=ut;
```

S 函数被控对象子程序: chap1_5plant.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
```



```

sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0 = [0;0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u) %Time-varying model
ut=u(1);
J=20+10*sin(6*pi*t);
K=400+300*sin(2*pi*t);
sys(1)=x(2);
sys(2)=-J*x(2)+K*ut;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);

```

作图程序: chap1_5plot.m

```

close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');s

```

通过本实例的仿真可见, 采用 S 函数, 可以很容易地表示复杂的被控对象及控制算法, 特别适合于复杂控制系统的仿真。



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

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

可得离散 PID 表达式



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

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, $k=1, 2, \dots$, $\text{error}(k-1)$ 和 $\text{error}(k)$ 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 1-16 所示。

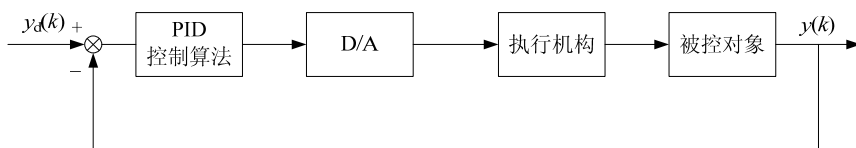


图 1-16 位置式 PID 控制系统

根据位置式 PID 控制算法得到其程序框图如图 1-17 所示。

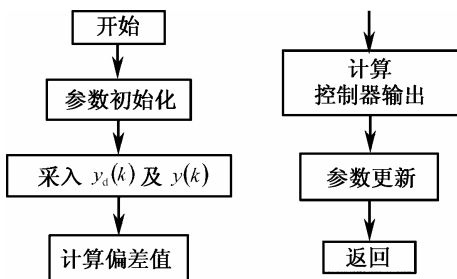


图 1-17 位置式 PID 控制算法程序框图

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

本方法可实现 D/A 及 A/D 的功能, 符合数字实时控制的真实情况, 计算机及 DSP 的实时 PID 控制都属于这种情况。

仿真之一: 采用 M 语言进行仿真

被控对象为一电动机模型传递函数

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

式中, $J = 0.0067, B = 0.10$ 。

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 $y_d(k) = 0.50 \sin(2\pi t)$, 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 1-18 所示。

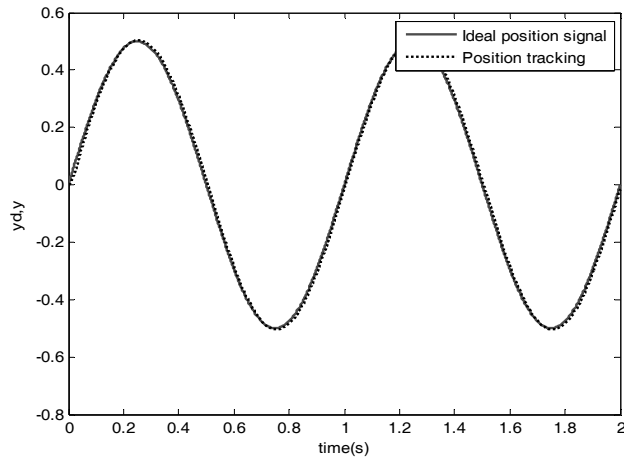
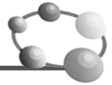


图 1-18 PID 正弦跟踪

控制主程序：chap1_6.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;

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

para=u_1;
tSpan=[0 ts];
[tt,xx]=ode45('chap1_6plant',tSpan,xk,[],para);
xk = xx(length(xx),:);
y(k)=xk(1);

e(k)=yd(k)-y(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,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,yd-y,'r','linewidth',2);
xlabel('time(s)');ylabel('error');
```

连续对象子程序: chap1_6plant.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 $G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$, 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。输入指令信号为正弦信号 $0.05\sin(2\pi t)$ 。采用 PID 方法设计控制器, 其中 $k_p = 2.5$, $k_i = 0.02$, $k_d = 0.50$ 。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。PID 正弦跟踪结果如图 1-19 所示。

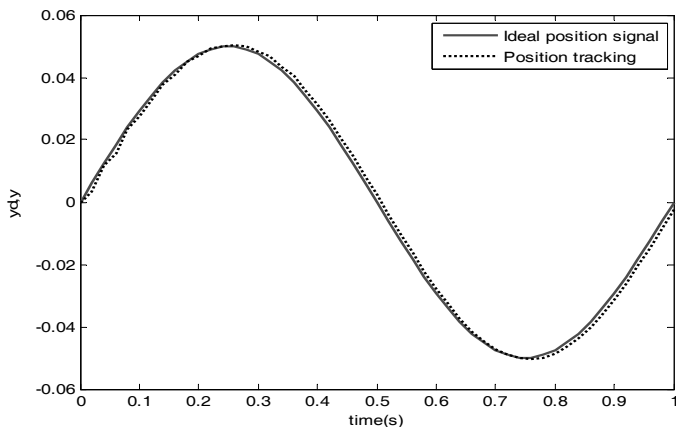


图 1-19 PID 正弦跟踪

控制主程序: chap1_7.mdl (见图 1-20)

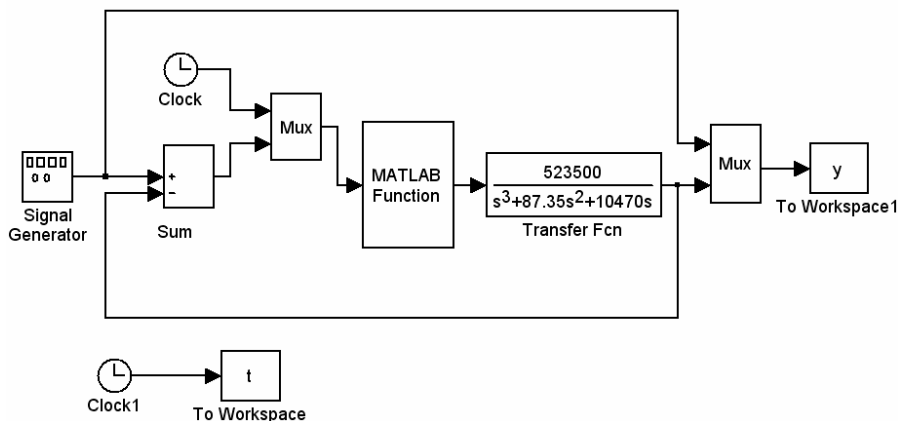
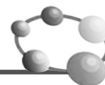


图 1-20 Simulink 主程序

控制器子程序: chap1_7ctrl.m

```
function [u]=pidsimf(u1,u2)
persistent pidmat errori error_1
t=u1;
if t==0
    errori=0;
    error_1=0;
end

kp=2.5;
ki=0.020;
kd=0.50;

error=u2;
error_d=error-error_1;
error_i=errori+error;

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

作图程序: chap1_7plot.m

```
close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

仿真之三: 利用 S 函数实现 PID 离散控制器的 Simulink 仿真

被控对象为三阶传递函数 $G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$ ，在 S 函数中，采用初始化、更新

函数和输出函数，即 `mdlInitializeSizes` 函数、`mdlUpdates` 函数和 `mdlOutputs` 函数。在初始化中采用 `sizes` 结构，选择 1 个输出，2 个输入。其中一个输入为误差信号，另一个输入为误差



信号上一时刻的值。S 函数嵌入在 Simulink 程序中，采样时间为 1 ms。仿真结果如图 1-21 所示。

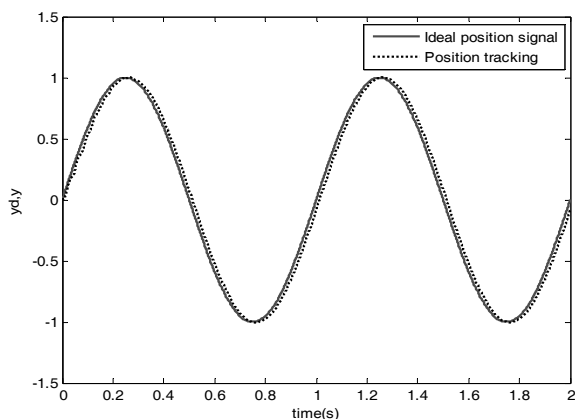


图 1-21 PID 正弦跟踪

仿真程序：

Simulink 主程序：chap1_8.mdl（见图 1-22）

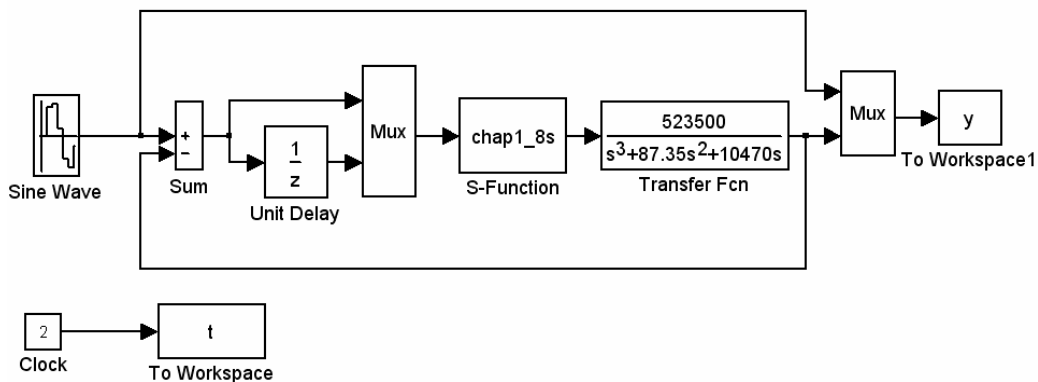
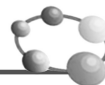


图 1-22 基于 S 函数离散控制器的 Simulink 仿真程序

PID 控制器程序：chap1_8s.m

```
function [sys,x0,str,ts]=exp_pidf(t,x,u,flag)
switch flag,
case 0          % initializations
    [sys,x0,str,ts] = mdlInitializeSizes;
case 2          % discrete states updates
    sys = mdlUpdates(x,u);
case 3          % computation of control signal
    % sys = mdlOutputs(t,x,u,kp,ki,kd,MTab);
    sys=mdlOutputs(t,x,u);
case {1, 4, 9}  % unused flag values
    sys = [];
otherwise       % error handling
    error(['Unhandled flag = ',num2str(flag)]);
```



```

end;

%=====
% when flag=0, perform system initialization
%=====
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;           % read default control variables
sizes.NumContStates = 0;    % no continuous states
sizes.NumDiscStates = 3;    % 3 states and assume they are the P/I/D components
sizes.NumOutputs = 1;       % 2 output variables: control u(t) and state x(3)
sizes.NumInputs = 2;        % 4 input signals
sizes.DirFeedthrough = 1;   % input reflected directly in output
sizes.NumSampleTimes = 1;   % single sampling period
sys = simsizes(sizes);      %
x0 = [0; 0; 0];            % zero initial states
str = [];
ts = [-1 0];               % sampling period
%=====
% when flag=2, updates the discrete states
%=====
function sys = mdlUpdates(x,u)
T=0.001;
sys=[ u(1);
      x(2)+u(1)*T;
      (u(1)-u(2))/T];

%=====
% when flag=3, computes the output signals
%=====
function sys = mdlOutputs(t,x,u,kp,ki,kd,MTab)

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

%sys=[kp,ki,kd]*x;
sys=kp*x(1)+ki*x(2)+kd*x(3);

```

作图程序: chap1_8plot.m

```

close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');

```



1.3.3 离散系统的数字PID控制仿真

控制对象为

$$G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$$

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

$$y(k) = -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) - \text{den}(4)y(k-3) \\ + \text{num}(2)u(k-1) + \text{num}(3)u(k-2) + \text{num}(4)u(k-3)$$

其中 num 和 den 为离散化系数。

仿真之一：指令为阶跃信号、正弦信号和方波信号

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 1-23 至图 1-25 所示。

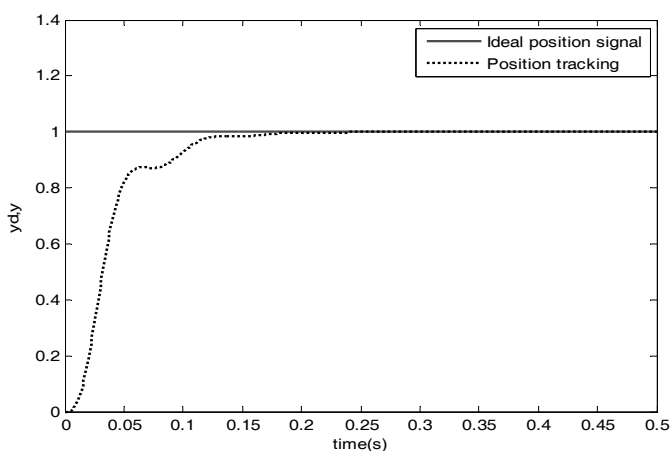


图 1-23 PID 阶跃跟踪 ($S=1$)

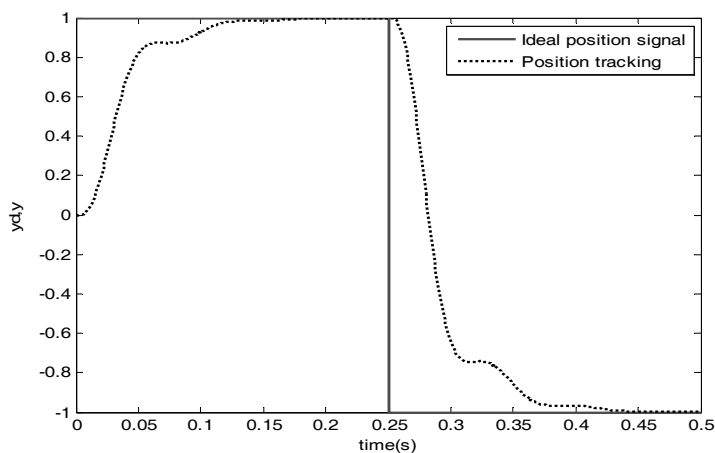
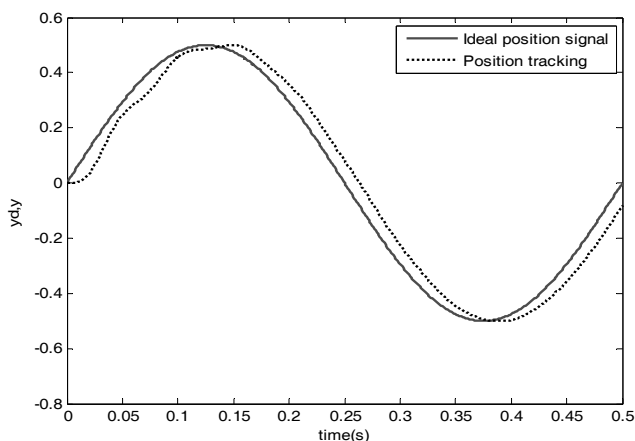
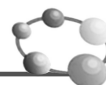


图 1-24 PID 方波跟踪 ($S=2$)

图 1-25 PID 正弦跟踪 ($S=3$)

仿真程序: chap1_9.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;
        yd(k)=1; %Step Signal
    elseif S==2
        kp=0.50;ki=0.001;kd=0.001;
        yd(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
        yd(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
y(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)=yd(k)-y(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=y(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,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

仿真之二：指令为三角波、锯齿波和随机信号

设计离散 PID 控制器，各信号的跟踪结果如图 1-26 至图 1-28 所示，其中 S 代表输入指令信号的类型。通过取余指令 `mod` 实现三角波和锯齿波。当 $S=1$ 时为三角波， $S=2$ 时为锯齿波， $S=3$ 时为随机信号。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。

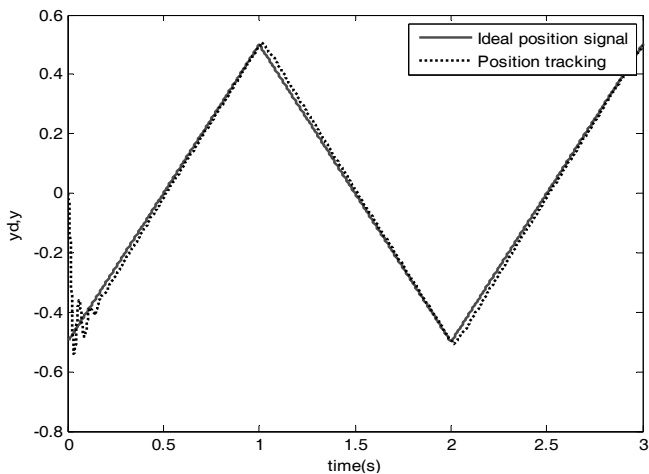
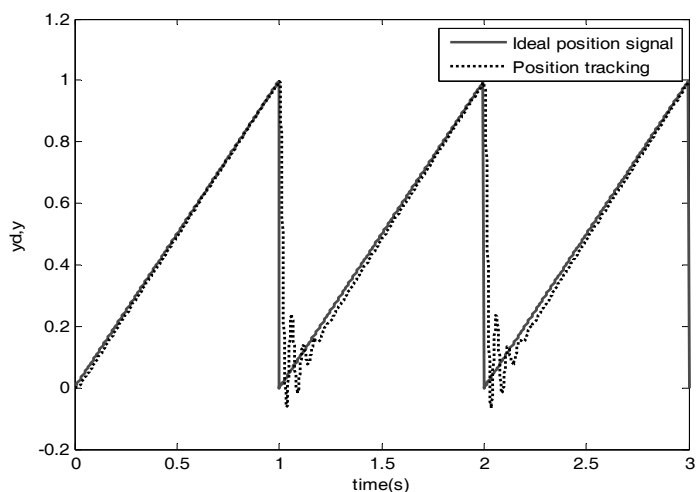
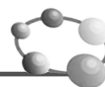
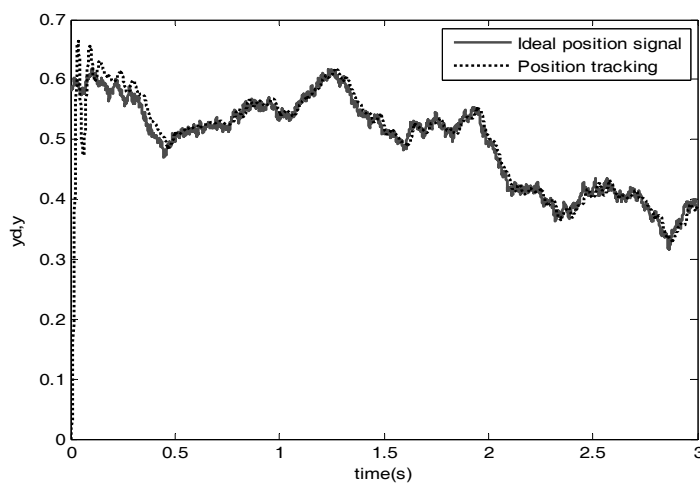


图 1-26 PID 三角波跟踪 ($S=1$)

图 1-27 PID 锯齿波跟踪 ($S=2$)图 1-28 PID 随机信号跟踪 ($S=3$)

仿真程序: chap1_10.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;
yd_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=3;
if S==1    %Triangle Signal
    if mod(time(k),2)<1
        yd(k)=mod(time(k),1);
    else
        yd(k)=1-mod(time(k),1);
    end
    yd(k)=yd(k)-0.5;
end
if S==2    %Sawtooth Signal
    yd(k)=mod(time(k),1.0);
end
if S==3    %Random Signal
    yd(k)=rand;
    dyd(k)=(yd(k)-yd_1)/ts;    %Max speed is 5.0
    while abs(dyd(k))>=5.0
        yd(k)=rand;
        dyd(k)=abs((yd(k)-yd_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
y(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)=yd(k)-y(k);

yd_1=yd(k);

u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(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,yd,'b',time,y,'r');
    pause(0.0000000000000000);
end
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');

```

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 $\text{error}(k)$ 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, 可采用增量式 PID 控制算法。

仿真之三: 采用 Simulink 实现离散 PID 控制器

离散 PID 控制的封装界面如图 1-29 所示, 在该界面中可设定 PID 的三个系数、采样时间及控制输入的上下界。仿真结果如图 1-30 所示。

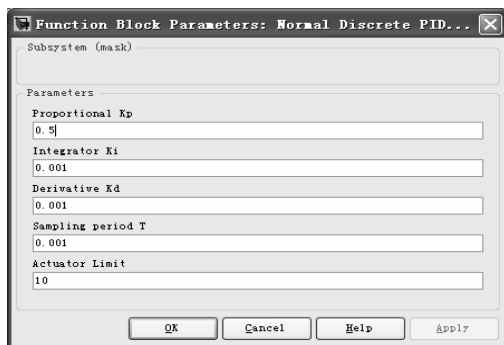


图 1-29 离散 PID 控制的封装界面

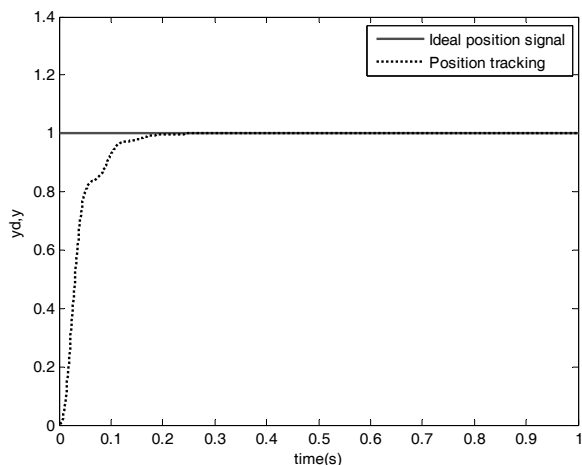


图 1-30 阶跃响应仿真结果

仿真程序: chap1_11.mdl, 其中离散 PID 控制的比例、微分和积分三项分别由 Simulink 模块实现 (见图 1-31 和图 1-32)

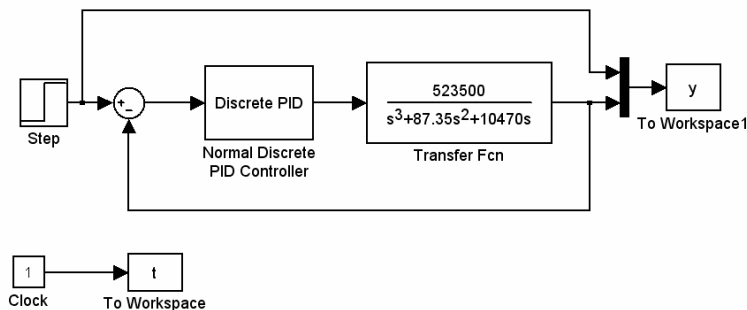


图 1-31 离散 PID 控制的 Simulink 主程序

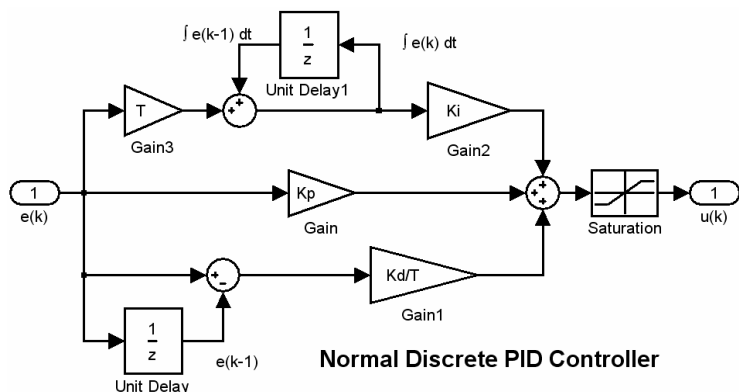


图 1-32 离散 PID 控制器子程序

作图程序: chap1_11plot.m

```
close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

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（例如驱动步进电动机）时，应采用增量式 PID 控制。根据递推原理可得

$$u(k-1) = k_p (\text{error}(k-1) + k_i \sum_{j=0}^{k-1} \text{error}(j) + k_d (\text{error}(k-1) - \text{error}(k-2))) \quad (1.6)$$

增量式 PID 控制算法

$$\Delta u(k) = u(k) - u(k-1)$$

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

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： $k_p = 8$, $k_i = 0.10$, $k_d = 10$ 。增量式 PID 阶跃跟踪结果如图 1-33 所示。

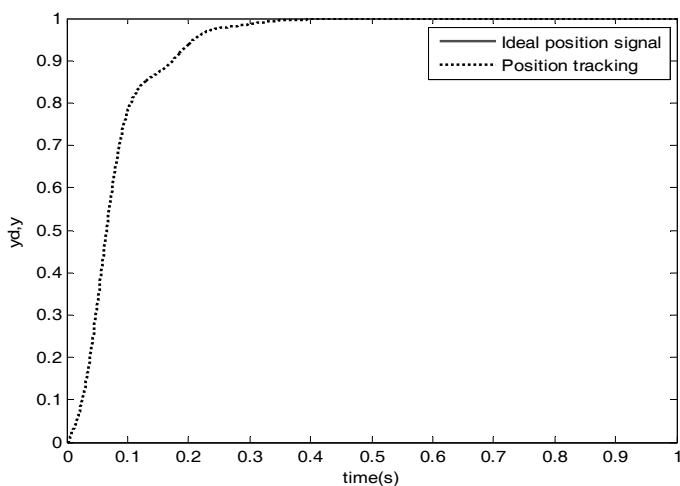
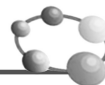


图 1-33 增量式 PID 阶跃跟踪

仿真程序: chap1_12.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;

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

error=yd(k)-y(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(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
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position value','tracking position value');
```

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

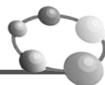
在普通 PID 控制中引入积分环节的目的，主要是为了消除静差，提高控制精度。但在过程的启动、结束或大幅度增减设定时，短时间内系统输出有很大的偏差，会造成 PID 运算的积分积累，致使控制量超过执行机构可能允许的最大动作范围对应的极限控制量，引起系统较大的超调，甚至引起系统较大的振荡，这在生产中是绝对不允许的。

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：^[68]

- (1) 根据实际情况，人为设定阈值 $\varepsilon > 0$ ；
- (2) 当 $|\text{error}(k)| > \varepsilon$ 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 $|\text{error}(k)| \leq \varepsilon$ 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

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



式中, T 为采样时间, β 为积分项的开关系数。

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

根据积分分离式 PID 控制算法得到其程序框图如图 1-34 所示。

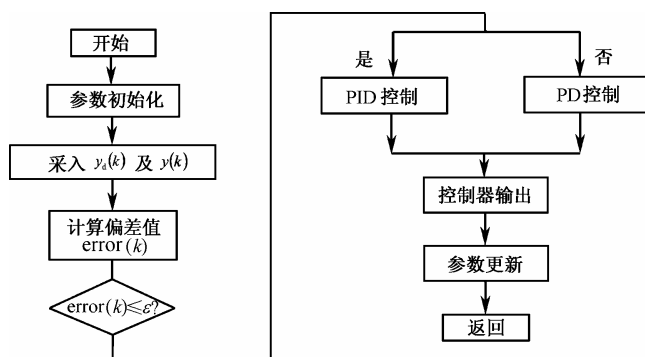


图 1-34 积分分离式 PID 控制算法程序框图

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, 被控对象离散化为

$$y(k) = -\text{den}(2)y(k-1) + \text{num}(2)u(k-5)$$

仿真之一: 采用 M 语言进行仿真

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 $y_d(k) = 40$, 控制器输出限制在 $[-110, 110]$, 其阶跃式跟踪结果如图 1-35 所示。取 $M=2$, 采用普通 PID 控制, 其阶跃式跟踪结果如图 1-36 所示。

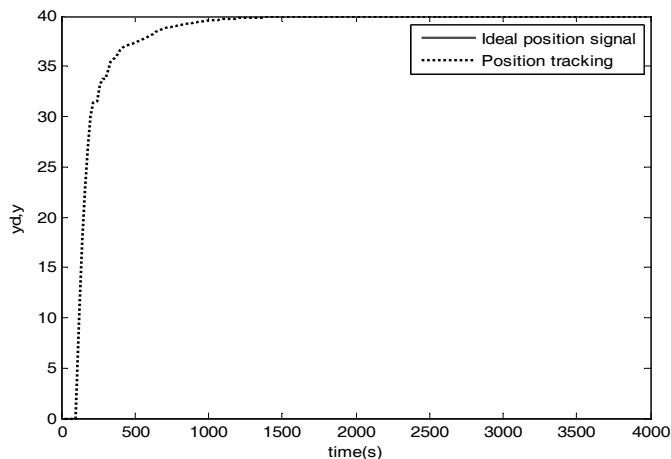
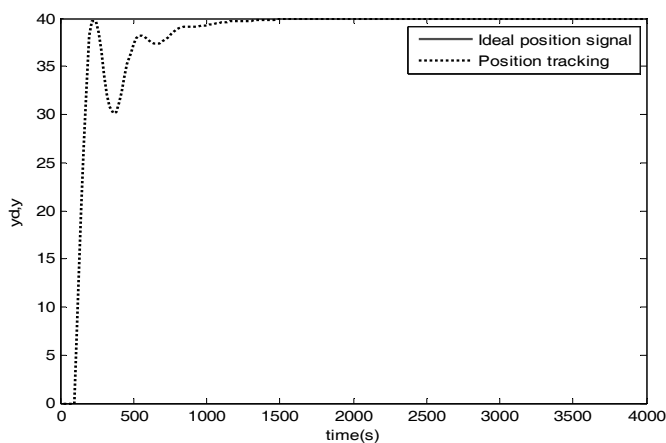


图 1-35 积分分离式 PID 阶跃跟踪 ($M=1$)

图 1-36 采用普通 PID 阶跃跟踪 ($M=2$)

仿真程序: chap1_13.m

```
%Integration Separation PID Controller
clear all;
close all;

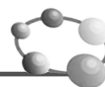
ts=20;
%Delay plant
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

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

%I separation
yd(k)=40;
error(k)=yd(k)-y(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=y(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```

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 可做相应变化。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。

仿真之二：采用 Simulink 仿真

通过 Simulink 模块实现积分分离 PID 控制算法，仿真结果如图 1-37 所示。

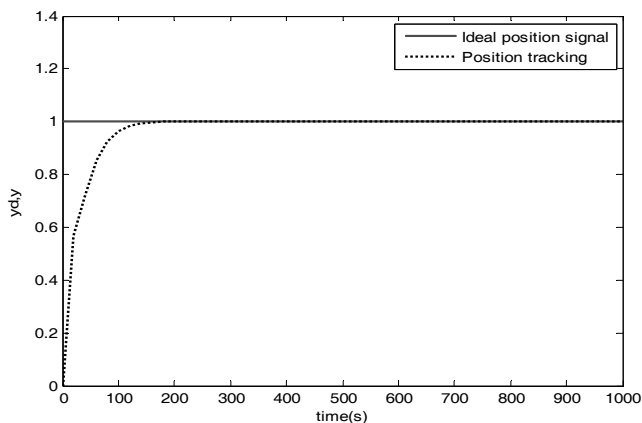


图 1-37 阶跃响应仿真结果

仿真程序:

初始化程序: chap1_14int.m

```
clear all;
close all;

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

kp=1.80;
ki=0.05;
kd=0.20;
```

Simulink 主程序: chap1_14.mdl (见图 1-38)

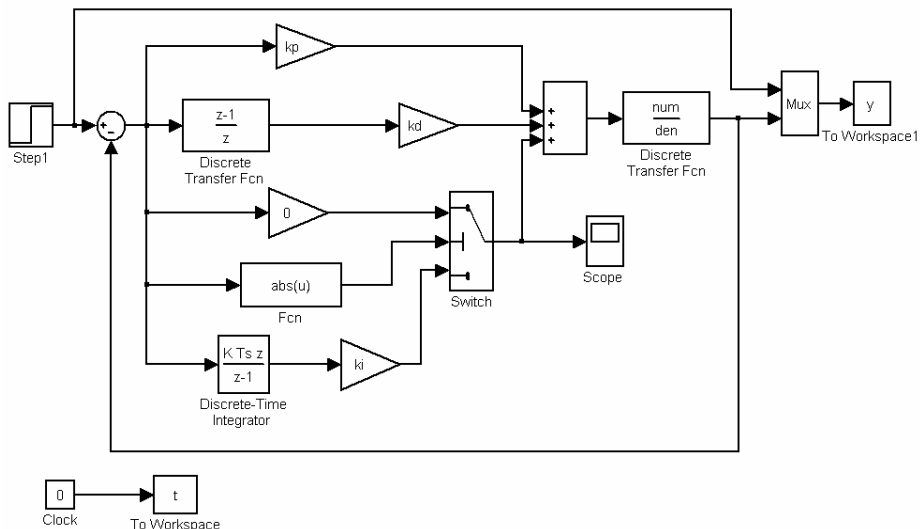


图 1-38 积分分离 PID 控制主程序



作图程序: chap1_14plot.m

```
close all;

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```

1.3.6 抗积分饱和和 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 1-39 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。^[68]一旦系统出现反向偏差, $u(k)$ 逐渐从饱和区退出。进入饱和区越深则退出饱和区所需时间越长。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。

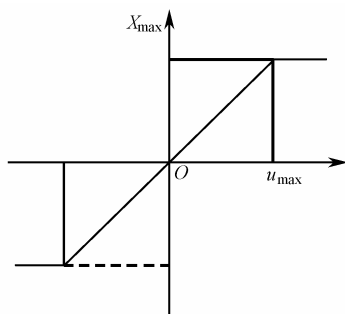


图 1-39 执行机构饱和特性

2. 抗积分饱和和算法

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。^[68]

若 $u(k-1) > u_{\max}$, 则只累加负偏差; 若 $u(k-1) < u_{\max}$, 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

$$G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$$

采样时间为 1ms, 取指令信号 $y_d(k) = 30$, $M=1$, 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 1-40 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 1-41 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。

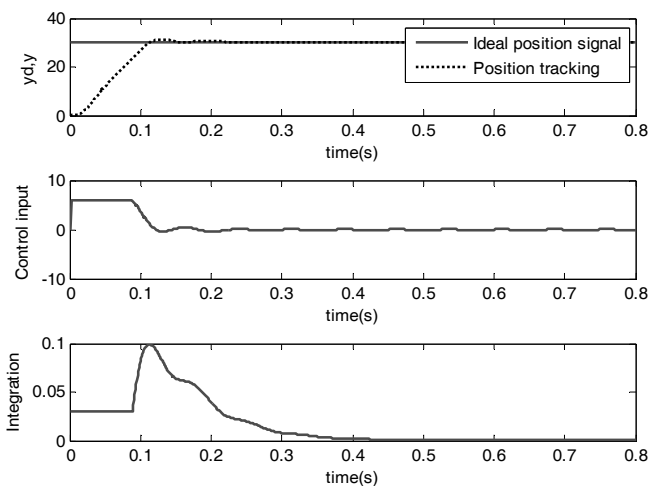


图 1-40 抗积分饱和仿真 ($M=1$)

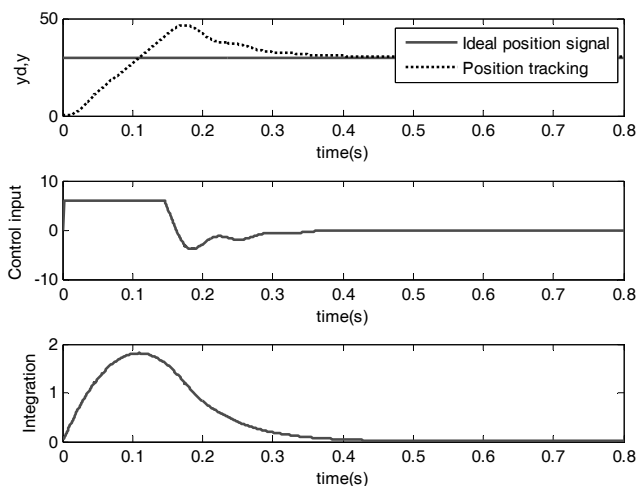


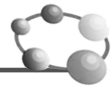
图 1-41 普通 PID 算法进行离散系统阶跃响应 ($M=2$)

仿真程序: chap1_15.m

```
%PID Controller with integration saturation
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;
for k=1:1:800
time(k)=k*ts;

yd(k)=30;                %Step Signal
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
y(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)=yd(k)-y(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
end

elseif M==2                %Not using intergration sturation
    alpha=1;
end
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=y(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,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
subplot(312);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
subplot(313);
plot(time,xi,'r','linewidth',2);
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

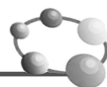
$$\int_0^t e(t)dt = \sum_{i=0}^k \frac{e(i) + e(i-1)}{2} T \quad (1.10)$$

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^[68]



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

其中系数 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，当偏差 $|e(k)|$ 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 $u_i(k) = k_i \sum_{i=0}^k e(i)T$ ，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，其积分速度在 $k_i \sum_{i=0}^{k-1} e(i)T$ 和 $k_i \sum_{i=0}^k e(i)T$ 之间。变速积分 PID 算法为

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

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 1-42 和图 1-43 所示。取 $M=2$ ，采用普通 PID 控制，其结果如图 1-44 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。

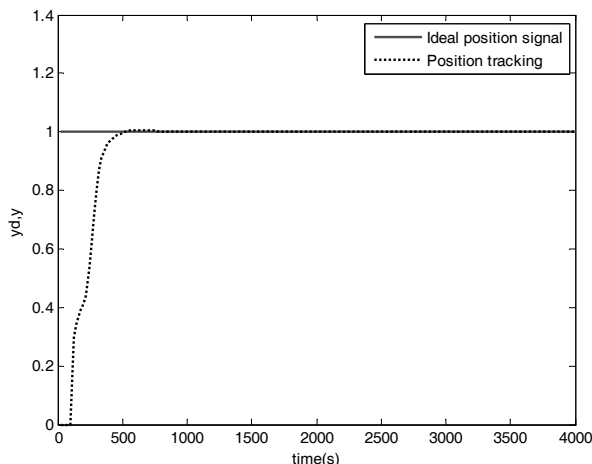


图 1-42 变速积分阶跃响应 ($M=1$)

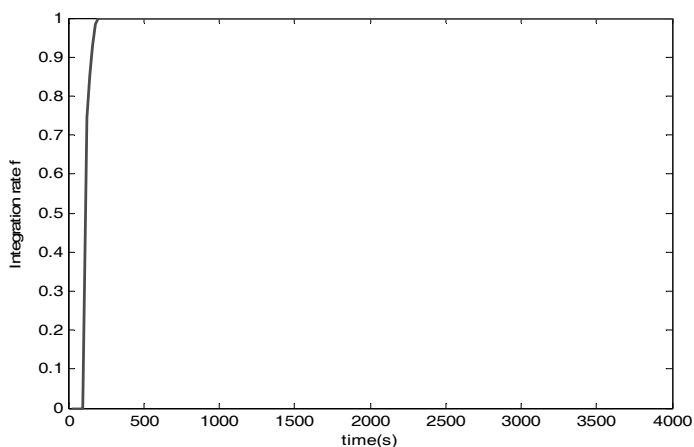
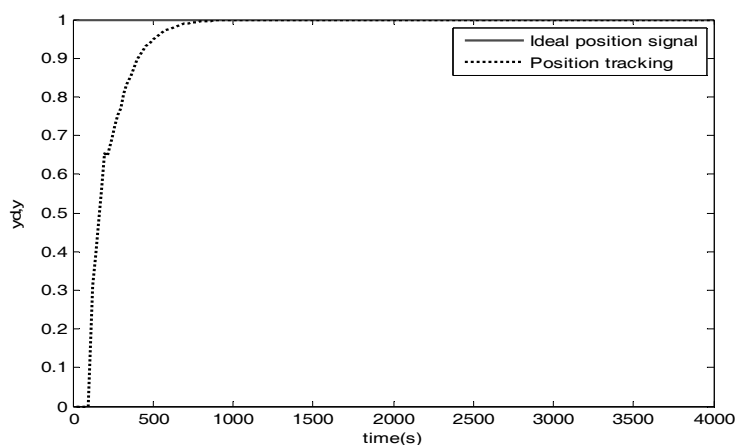


图 1-43 变速积分参数的变化

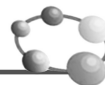
图 1-44 普通 PID 控制阶跃响应 ($M=2$)

仿真程序: chap1_16.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;
```



```

yd(k)=1.0; %Step Signal

%Linear model
y(k)=-den(2)*y_1+num(2)*u_5;
error(k)=yd(k)-y(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=1;
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=y(k);
error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,f,'r','linewidth',2);
xlabel('time(s)');ylabel('Integration rate f');

```




1.3.9 带滤波器的PID控制仿真

采用低通滤波器可有效地滤掉噪声信号，在控制系统的设计中是一种常用的方法。通过以下两个实例验证低通滤波器的滤波性能。

仿真实例 1：基于低通滤波器的信号处理

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 1-45 和图 1-46 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。

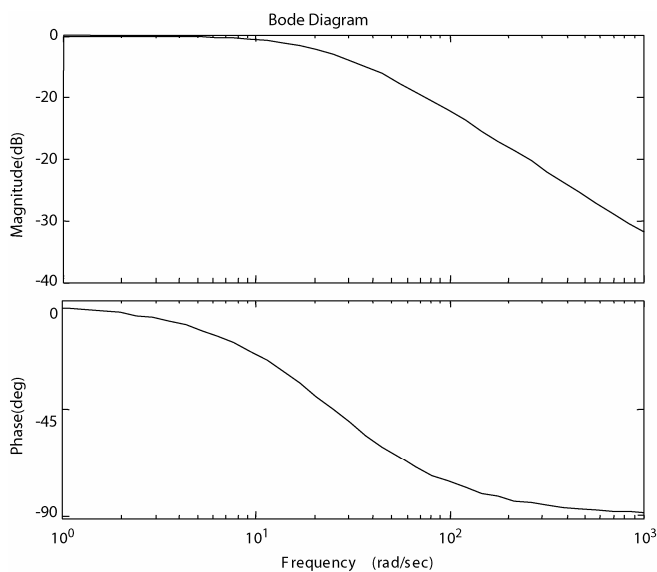


图 1-45 低通滤波器 Bode 图

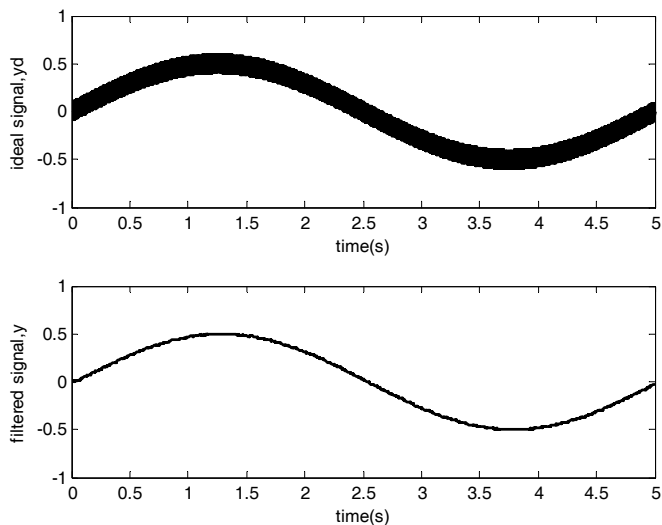
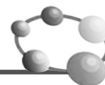


图 1-46 原始信号及滤波后信号



仿真程序：chap1_17.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,'tucsin');
[num,den]=tfdata(Qz,'v');

y_1=0;y_2=0;
yd_1=0;yd_2=0;
for k=1:1:5000
time(k)=k*ts;

%Input Signal with noise
n(k)=0.10*sin(100*2*pi*k*ts); %Noise signal
yd(k)=n(k)+0.50*sin(0.2*2*pi*k*ts); %Input Signal

y(k)=-den(2)*y_1+num(1)*yd(k)+num(2)*yd_1;

y_2=y_1;y_1=y(k);
yd_2=yd_1;yd_1=yd(k);
end
figure(1);bode(Q);
figure(2);
subplot(211);
plot(time,yd,'k','linewidth',2);
xlabel('time(s)');ylabel('ideal signal,yd');
subplot(212);
plot(time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('filtered signal,y');
```

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

$$G_p(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$$

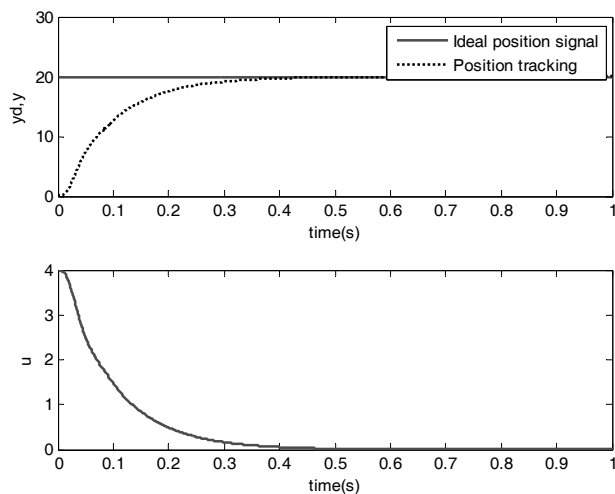
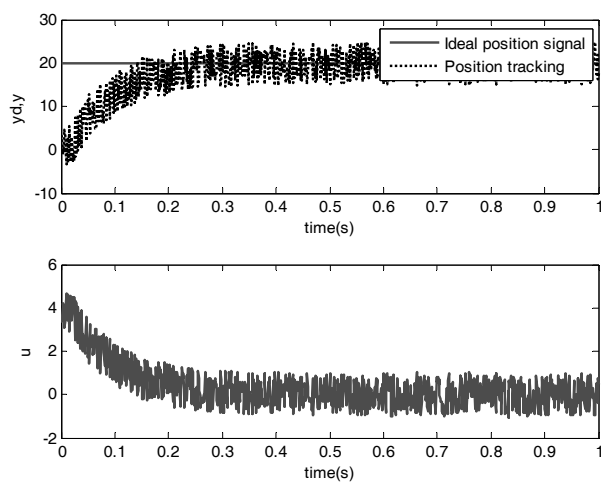
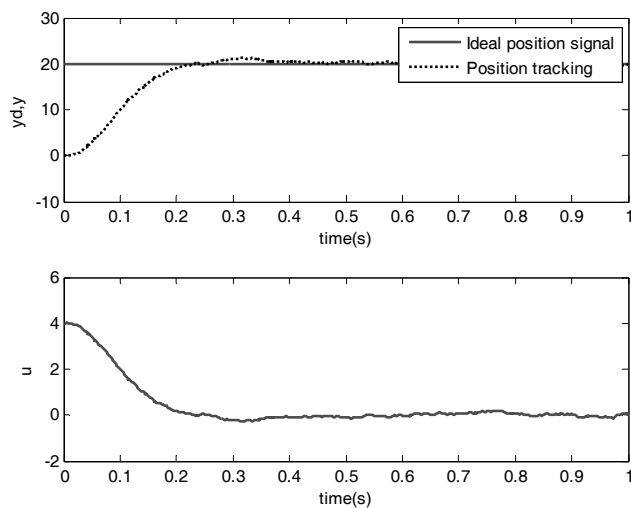
低通滤波器为

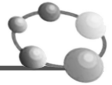
$$Q(s) = \frac{1}{0.04s + 1}$$

采样时间为 1ms，噪声信号加在对象的输出端。

仿真之一：采用 M 语言进行仿真

分三种情况进行： $M=1$ 时，为未加噪声信号； $M=2$ 时，为加噪声信号未加滤波； $M=3$ 时，为加噪声信号加滤波。阶跃响应结果如图 1-47 至图 1-49 所示。

图 1-47 普通 PID 控制阶跃响应 ($M=1$)图 1-48 无滤波器时 PID 控制阶跃响应 ($M=2$)图 1-49 加入滤波器后 PID 控制阶跃响应 ($M=3$)



仿真程序: chap1_18.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;
yn_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,'tucsin');
[num1,den1]=tfdata(dsys1,'v');
f_1=0;

M=3;
for k=1:1:1000
time(k)=k*ts;

yd(k)=20;  %Step Signal
%Linear model
y(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 noisy signal
    error(k)=yd(k)-y(k);
    filty(k)=y(k);
end

n(k)=5.0*rands(1);           %Noisy signal
yn(k)=y(k)+n(k);

if M==2                      %No filter
    filty(k)=yn(k);
    error(k)=yd(k)-filty(k);
end
if M==3                      %Using low frequency filter
    filty(k)=-den1(2)*f_1+num1(1)*(yn(k)+yn_1);
    error(k)=yd(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;
%-----Return of PID parameters-----
yd_1=yd(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=y(k);

f_1=filty(k);
yn_1=yn(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,filty,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
subplot(212);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('u');
figure(2);
plot(time,n,'r','linewidth',2);
xlabel('time(s)');ylabel('Noisy signal');
```

仿真之二：采用 Simulink 进行仿真

控制器采用积分分离 PI 控制，即当误差的绝对值小于等于 0.80 时，加入积分控制，仿真结果如图 1-50 和图 1-51 所示。

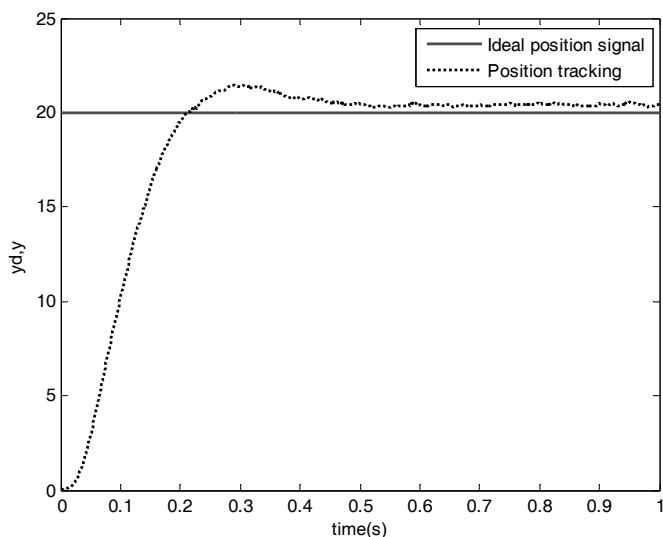


图 1-50 加入滤波器时 PID 控制阶跃响应

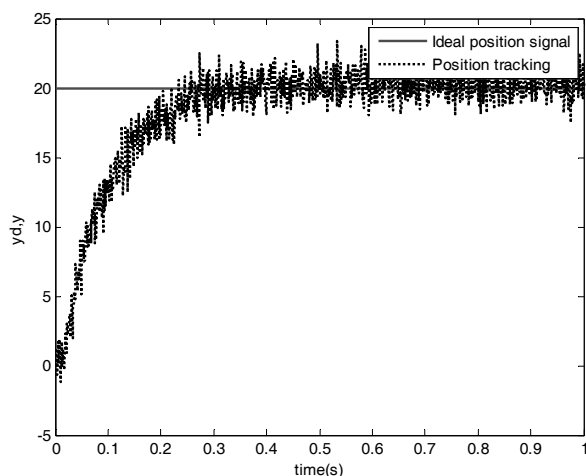
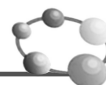


图 1-51 无滤波器时 PID 控制阶跃响应

仿真程序:

初始化程序: chap1_19int.m

```
clear all;
close all;

ts=0.001;
%Low Filter
Q=tf([1],[0.04,1]);
Qz=c2d(Q,ts,'tustin');
[numQ,denQ]=tfdata(Qz,'v');

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

kp=0.20;
ki=0.05;
```

Simulink 主程序: chap1_19.mdl (见图 1-52 和图 1-53)

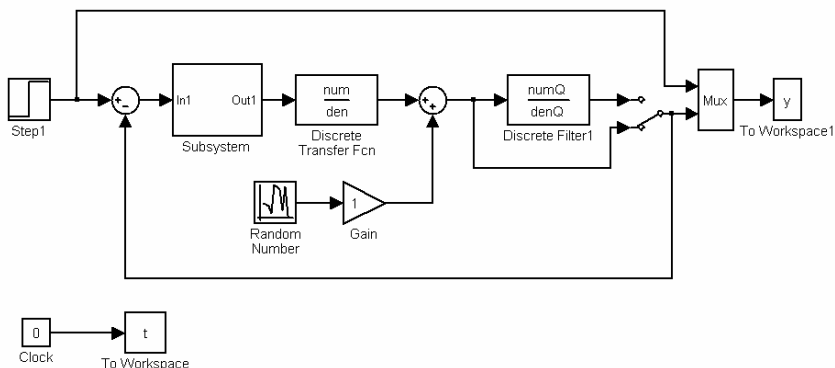


图 1-52 Simulink 仿真主程序

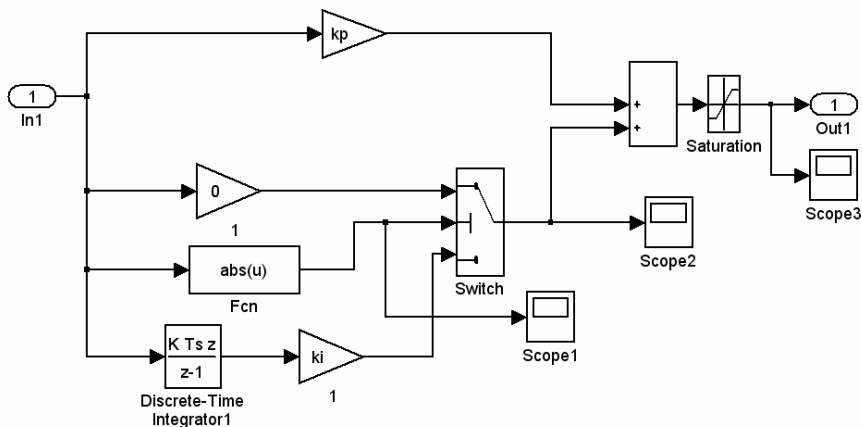


图 1-53 PI 控制器子程序

作图程序: chap1_19plot.m

```
close all;  
  
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('yd,y');  
legend('Ideal position signal','Position tracking');
```

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 1-54 (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。

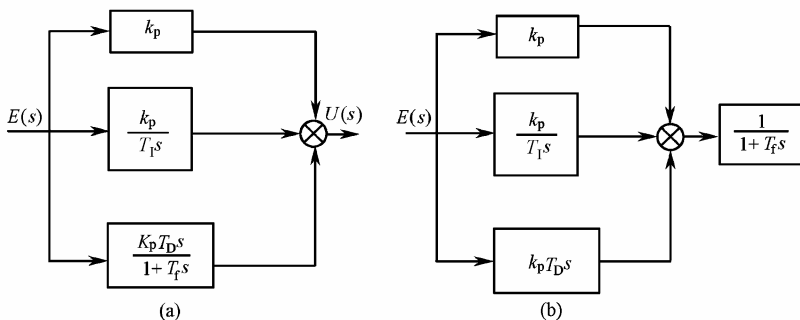
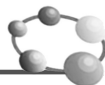


图 1-54 不完全微分算法结构图



对图 (a) 所示的不完全微分结构, 其传递函数为

$$U(s) = (k_p + \frac{k_p / T_I}{s} + \frac{k_p T_D s}{T_f s + 1}) E(s) = u_p(s) + u_I(s) + u_D(s) \quad (1.14)$$

将式 (1.14) 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

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

取采样时间为 T_s , 将上式离散化为

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

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

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

式中, $K_D = k_p \cdot T_D / T_s$ 。

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

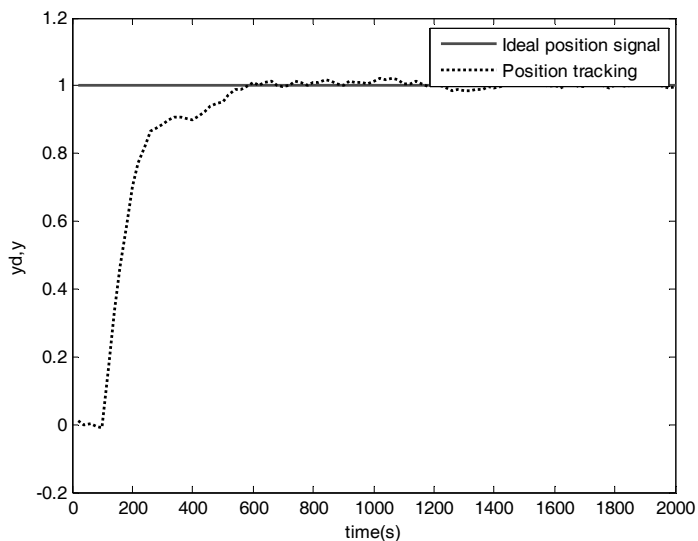
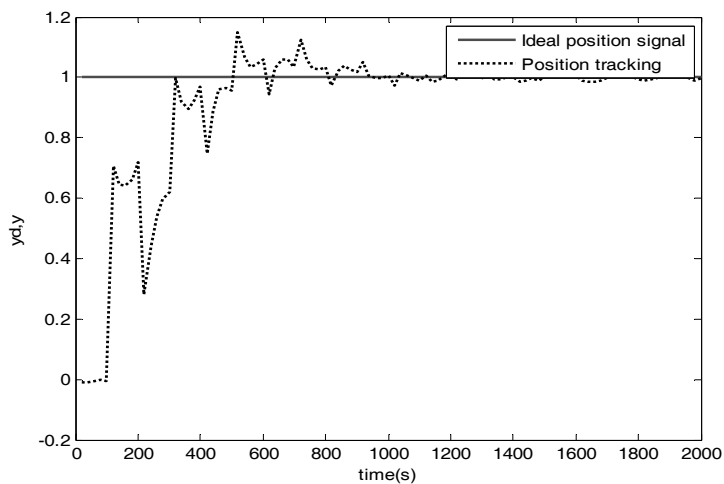
$$G(s) = \frac{e^{-80s}}{60s + 1}$$

在对象的输出端加幅值为 0.01 的随机信号 $n(k)$ 。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 1-55 所示。取 $M=2$, 采用普通 PID 方法, 阶跃响应结果如图 1-56 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。

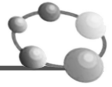
图 1-55 不完全微分控制阶跃响应 ($M=1$)图 1-56 普通 PID 控制阶跃响应 ($M=2$)

仿真程序: chap1_20.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;

yd(k)=1.0;

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

n(k)=0.01*rands(1);
y(k)=y(k)+n(k);

error(k)=yd(k)-y(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=y(k);

```



```

error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('u');
figure(3);
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 1-57 所示^[68]，其特点是只对输出量 $y(k)$ 进行微分，而对给定值 $y_d(k)$ 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 $y_d(k)$ 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^[68]

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

式中， $\frac{1}{\gamma T_D s + 1}$ 相当于低通滤波器。

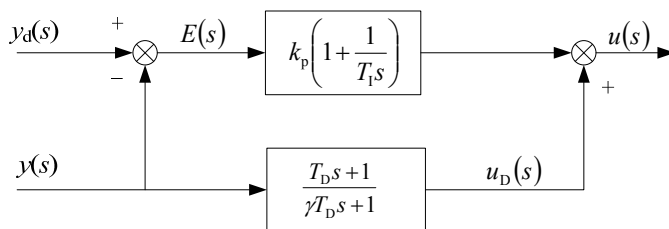
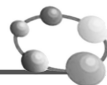


图 1-57 微分先行 ID 控制结构图

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

由差分得

$$\begin{aligned} \frac{du_D}{dt} &\approx \frac{u_D(k) - u_D(k-1)}{T} \\ \frac{dy}{dt} &\approx \frac{y(k) - y(k-1)}{T} \\ \gamma T_D \frac{u_D(k) - u_D(k-1)}{T} + u_D(k) &= T_D \frac{y(k) - y(k-1)}{T} + y(k) \end{aligned}$$



$$u_D(k) = \left(\frac{\gamma T_D}{\gamma T_D + T} \right) u_D(k-1) + \left(\frac{T_D + T}{\gamma T_D + T} \right) y(k) - \left(\frac{T_D}{\gamma T_D + T} \right) y(k-1)$$

$$u_D(k) = c_1 u_D(k-1) + c_2 y(k) - c_3 y(k-1) \quad (1.22)$$

式中,

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

PI 控制部分传递函数为

$$\frac{u_{PI}(s)}{E(s)} = k_p \left(1 + \frac{1}{T_I s} \right) \quad (1.24)$$

式中, T_I 为积分时间常数。

离散控制律为

$$u(k) = u_{PI}(k) + u_D(k) \quad (1.25)$$

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干扰的方波信号: $y_d(t) = 1.0 \operatorname{sgn}(\sin(0.0005\pi t) + 0.05 \sin(0.03\pi t))$ 。

取 $M=1$, 采用微分先行 PID 控制方法, 其方波响应结果如图 1-58 所示。取 $M=2$, 采用普通 PID 方法, 其方波响应控制结果如图 1-59 所示。由仿真结果可以看出, 对于给定值 $y_d(k)$ 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。

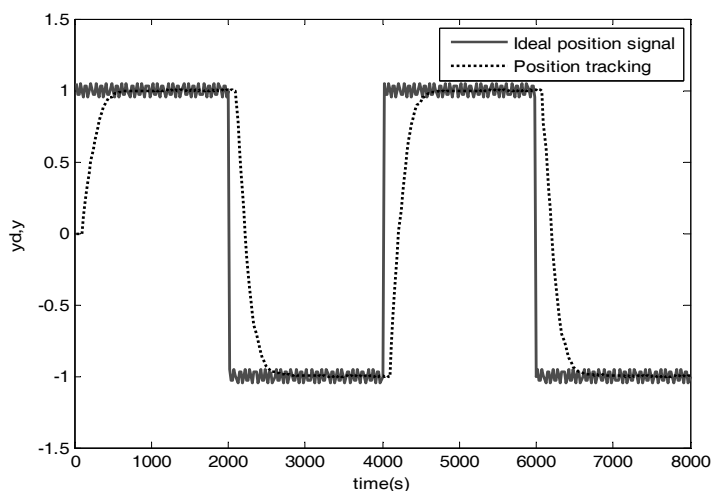
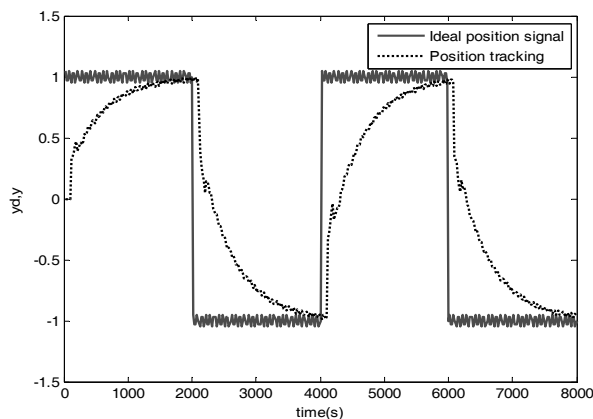


图 1-58 微分先行 PID 控制方波响应 ($M=1$)

图 1-59 普通 PID 控制方波响应 ($M=2$)

仿真程序: chap1_21.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;

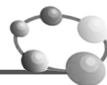
%Linear model
y(k)=-den(2)*y_1+num(2)*u_5;

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

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

error(k)=yd(k)-y(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*y(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=y(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
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

$$e(k) = \begin{cases} 0 & |e(k)| \leq |e_0| \\ e(k) & |e(k)| > |e_0| \end{cases} \quad (1.26)$$

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 1-60 所示。



仿真实例

被控对象为

$$G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$$

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

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

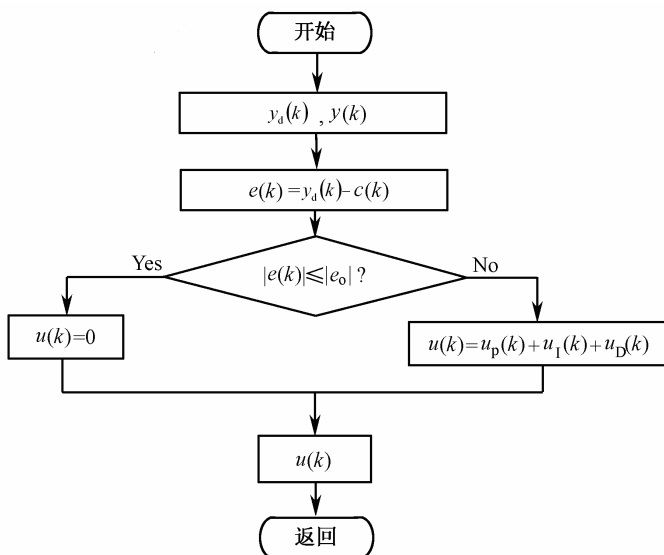


图 1-60 带死区的 PID 控制算法流程图

取 $M = 1$ ，采用一般积分分离式 PID 控制方法，其控制结果如图 1-61 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 1-62 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。

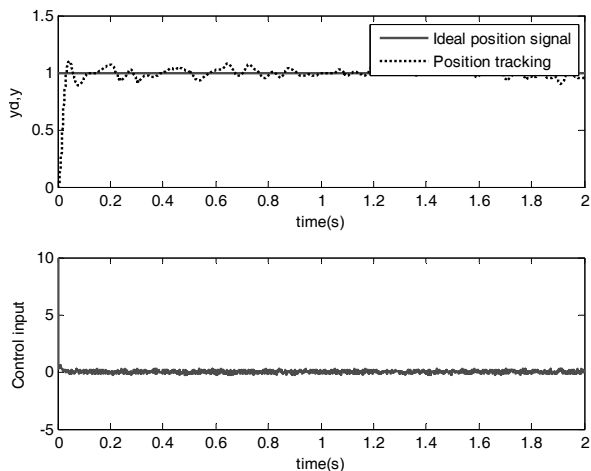
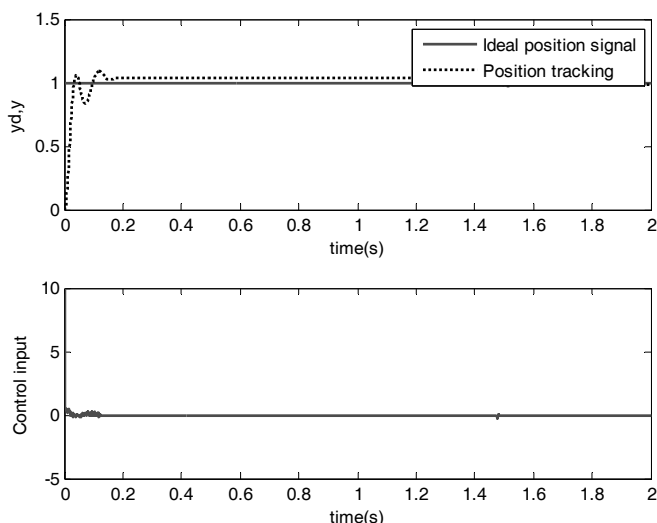
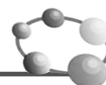


图 1-61 不带死区 PID 控制 ($M=1$)

图 1-62 带死区 PID 控制 ($M=2$)

仿真程序: chap1_22.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;
yn_1=0;
error_1=0;error_2=0;ei=0;

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

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

yd(k)=1;                        %Step Signal

%Linear model
y(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;

n(k)=0.50*rands(1);            %Noisy signal
```




```
yn(k)=y(k)+n(k);

%Low frequency filter
filty(k)=-den1(2)*f_1+num1(1)*(yn(k)+yn_1);
error(k)=yd(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 control
    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-----
yd_1=yd(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(k);

f_1=filty(k);
yn_1=yn(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
```



```
subplot(212);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
figure(2);
plot(time,n,'r','linewidth',2);
xlabel('time(s)');ylabel('Noisy 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的, 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 1-63 所示。

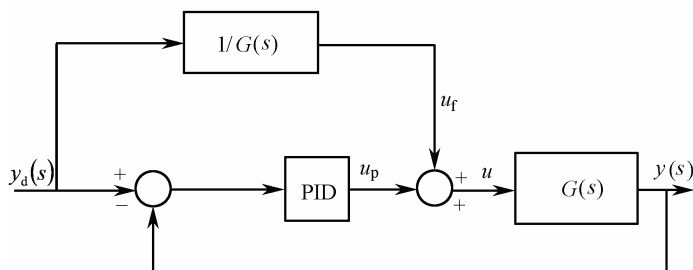


图 1-63 PID 前馈补偿控制结构

设计前馈补偿控制器为

$$u_f(s) = y_d(s) \frac{1}{G(s)} \quad (1.27)$$

总控制输出为 PID 控制输出加上前馈控制输出

$$u(t) = u_p(t) + u_f(t) \quad (1.28)$$

写成离散形式为

$$u(k) = u_p(k) + u_f(k) \quad (1.29)$$

仿真实例

被控对象为

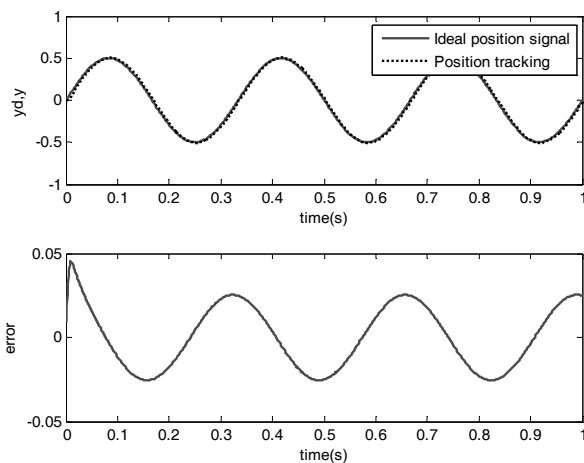
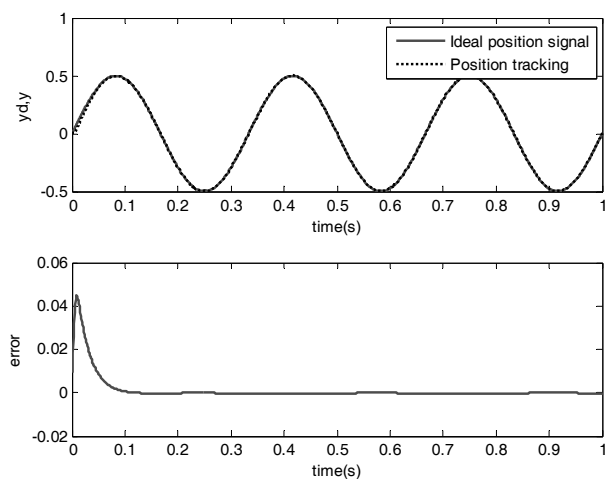
$$G(s) = \frac{133}{s^2 + 25s}$$

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

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

只采用 PID 正弦跟踪控制方法的结果如图 1-64 所示, 采用前馈 PID 控制方法的跟踪结果如图 1-65 所示。可见通过前馈补偿可大大提高系统的跟踪性能。

本方法的不足之处是需要被控对象的精确模型。

图 1-64 PID 正弦跟踪及跟踪误差 ($M=1$)图 1-65 PID+前馈补偿正弦跟踪及跟踪误差 ($M=2$)

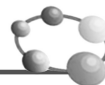
仿真程序: chap1_23.m

```
%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;
yd(k)=A*sin(F*2*pi*k*ts);
dyd(k)=A*F*2*pi*cos(F*2*pi*k*ts);
ddy(k)=-A*F*2*pi*F*2*pi*sin(F*2*pi*k*ts);

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

error(k)=yd(k)-y(k);

ei=ei+error(k)*ts;

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

uf(k)=25/133*dyd(k)+1/133*ddy(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=y(k);
error_1=error(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
subplot(212);
plot(time,error,'r','linewidth',2);
xlabel('time(s)');ylabel('error');
figure(2);
plot(time,up,'k',time,uf,'b',time,u,'r','linewidth',2);
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。

在步进式PID控制的仿真中，取位置指令为 $R=20$ ，实际输入指令 $y_d(k)$ 采用 0.25 的步长变化，逐渐逼近输入指令信号 R 。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，需要的控制输入信号大；而采用步进式PID控制，虽然响应速度慢，但阶跃响应平稳，需要的控制输入信号小，具有很好的工程实用价值。

仿真实例

被控对象为

$$G(s) = \frac{523\,500}{s^3 + 87.35s^2 + 10\,470s}$$

采样时间为 1ms，输入指令信号为 $R=20$ 。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 1-66， $M=2$ 时为步进式积分分离PID控制，响应结果及输入信号的变化如图 1-67 和图 1-68 所示。

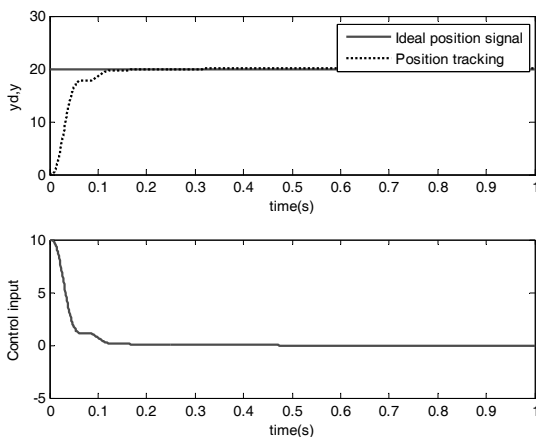


图 1-66 积分分离阶跃响应 ($M=1$)

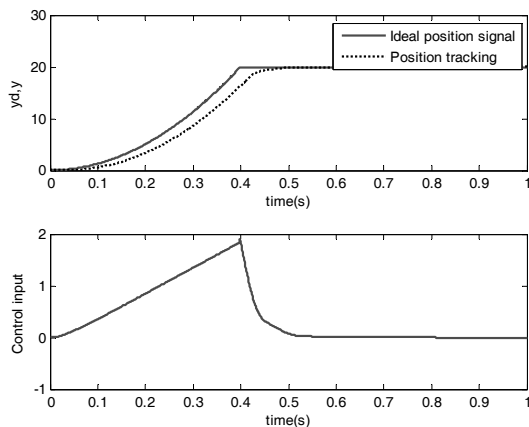
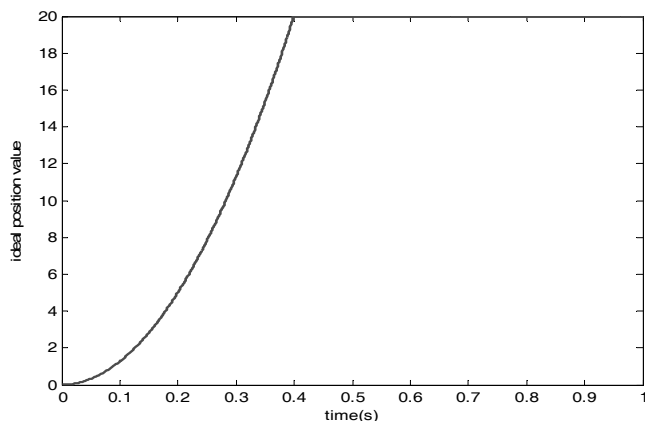
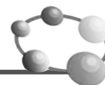


图 1-67 步进式积分分离阶跃响应和控制输入 ($M=2$)

图 1-68 步进式阶跃信号 $y_d(k)$ 的变化

仿真程序: chap1_24.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;
ydi=0.0;

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

%Linear model
y(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
    yd(k)=R;
    error(k)=yd(k)-y(k);
end
if M==2    %Using Gradual approaching input value
    if ydi<R-0.25
```



```
        ydi=ydi+k*ts*rate;
    elseif ydi>R+0.25
        ydi=ydi-k*ts*rate;
    else
        ydi=R;
    end
    yd(k)=ydi;
    error(k)=yd(k)-y(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;

%-----Return of PID parameters-----
yd_1=yd(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
subplot(212);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
figure(2);
plot(time,yd,'r','linewidth',2);
xlabel('time(s)');ylabel('ideal position value');
```

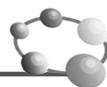
1.3.15 PID 控制的方波响应

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, 被控对象离散化为

$$y(k) = -\text{den}(2)y(k-1) + \text{num}(2)u(k-5)$$



由于方波信号的速度、加速度不连续，当位置跟踪指令为方波信号时，如采用滤波器对指令信号进行滤波，将滤波输出作为给定信号，可使方波响应及执行器的动作更加平稳，在工程上具有一定意义。

为了保证滤波后幅值不变，取三阶离散滤波器为

$$F(z-1) = a_1 + a_2 z^{-1} + a_1 z^{-2}$$

$$2a_1 + a_2 = 1$$

取方波信号为 $y_d(t) = \text{sgn}(\sin(0.0001\pi t))$ ，滤波器参数取 $a_1 = 0.10, a_2 = 0.80$ 。

分两种情况进行仿真： $M=1$ 时，为普通方波指令信号，方波响应结果如图 1-69 所示； $M=2$ 时，为加滤波的方波指令信号，方波响应结果如图 1-70 所示。

可见，将方波指令信号加滤波后，方波响应更加平稳，控制输入信号的抖动消除。

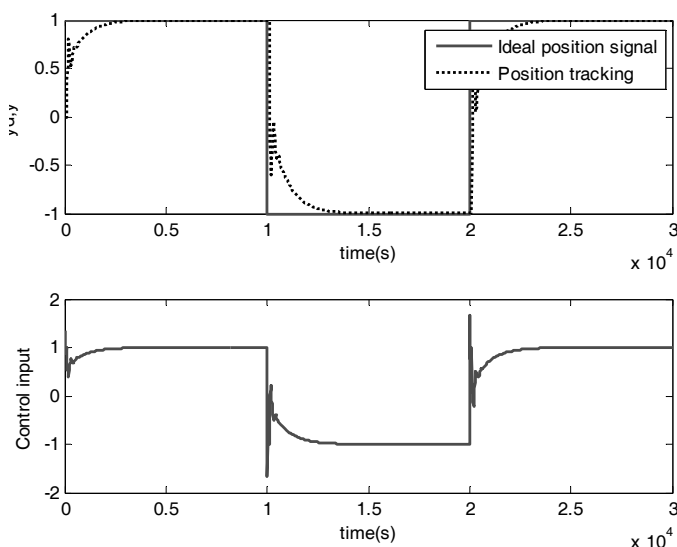


图 1-69 普通方波指令信号的 PID 响应和控制输入 ($M=1$)

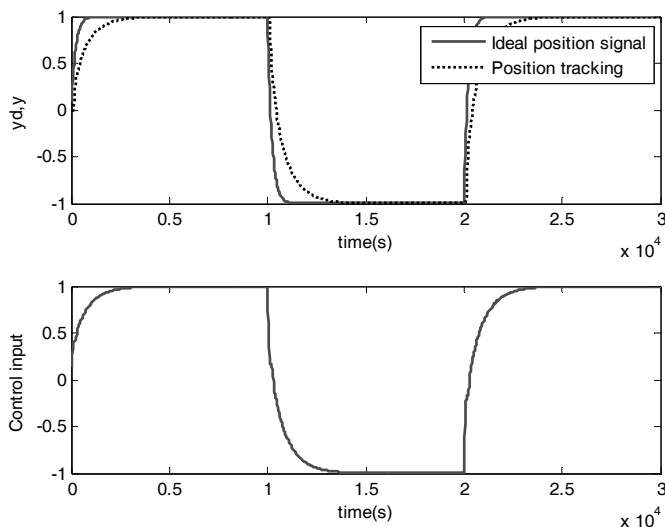


图 1-70 带滤波器的方波指令信号 PID 响应和控制输入 ($M=2$)



仿真程序: chap1_25.m

```
%PID Controller for Square Tracking with Filtered Signal
```

```
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;
```

```
y_1=0;
```

```
error_1=0;
```

```
ei=0;
```

```
yd_1=0;yd_2=0;
```

```
for k=1:1:1500
```

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

```
yd(k)=1.0*sign(sin(0.00005*2*pi*k*ts));
```

```
M=1;
```

```
switch M;
```

```
case 1
```

```
    yd(k)=yd(k);
```

```
case 2
```

```
    yd(k)=0.10*yd(k)+0.80*yd_1+0.10*yd_2;
```

```
end
```

```
%Linear model
```

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

```
kp=0.80;
```

```
kd=10;
```

```
ki=0.002;
```

```
error(k)=yd(k)-y(k);
```

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

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

```
%Update parameters
```

```
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
```

```
y_1=y(k);
```



```

error_2=error_1;
error_1=error(k);
yd_2=yd_1;
yd_1=yd(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
subplot(212);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```

1.3.16 基于卡尔曼滤波器的PID控制

1.3.16.1 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

$$\begin{aligned} x(k) &= Ax(k-1) + B(u(k) + w(k)) \\ y_v(k) &= Cx(k) + v(k) \end{aligned} \quad (1.30)$$

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

$$M_n(k) = \frac{P(k)C^T}{CP(k)C^T + R} \quad (1.31)$$

$$P(k) = AP(k-1)A^T + BQB^T \quad (1.32)$$

$$P(k) = (I_n - M_n(k)C)P(k) \quad (1.33)$$

$$x(k) = Ax(k-1) + M_n(k)(y_v(k) - CAx(k-1)) \quad (1.34)$$

$$y_e(k) = Cx(k) \quad (1.35)$$

误差的协方差为

$$\text{errcov}(k) = CP(k)C^T \quad (1.36)$$

卡尔曼滤波器结构如图 1-71 所示。

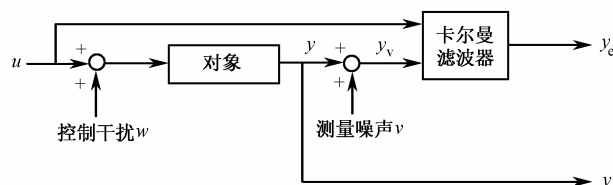


图 1-71 卡尔曼滤波器结构图



1.3.16.2 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 Z 变换将对象离散化，并描述为离散状态方程的形式

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

$$y(k) = Cx(k)$$

带有测量噪声的被控对象输出为

$$y_v(k) = Cx(k) + v(k)$$

式中， $A = \begin{bmatrix} 1.0000000 & 0.0009876 \\ 0.0000000 & 0.9753099 \end{bmatrix}$ ， $B = \begin{bmatrix} 0.0000659 \\ 0.1313512 \end{bmatrix}$ ， $C = [1, 0]$ ， $D = [0]$

仿真方法之一：采用 M 语言进行仿真

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 1-72 至图 1-74 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。

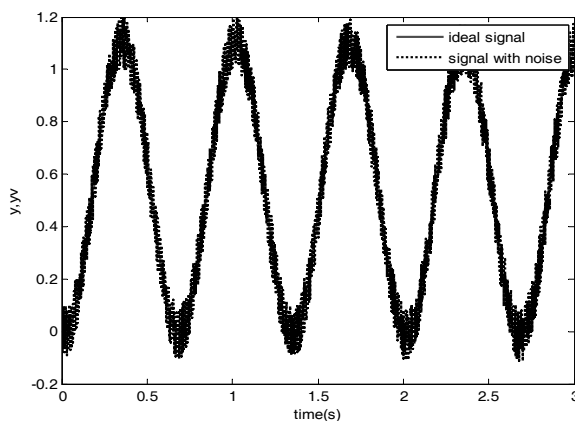


图 1-72 原始信号及带有噪声的原始信号

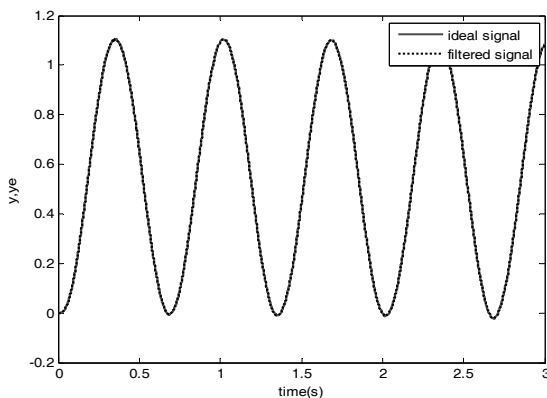


图 1-73 原始信号及滤波后的信号

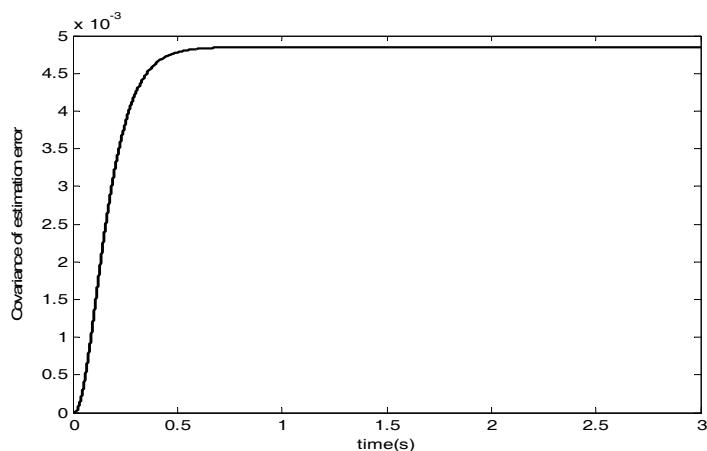
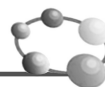


图 1-74 误差协方差的变化

仿真程序: chap1_26.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

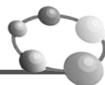
errcov(k)=C*P*C';        %Covariance of estimation error

%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,'r',time,yv,'k:','linewidth',2);
xlabel('time(s)');ylabel('y,yv')
legend('ideal signal','signal with noise');
figure(2);
plot(time,y,'r',time,ye,'k:','linewidth',2);
xlabel('time(s)');ylabel('y,ye')
legend('ideal signal','filtered signal');
figure(3);
plot(time,errcov,'k','linewidth',2);
xlabel('time(s)');ylabel('Covariance of estimation error');
```

仿真之二：采用 Simulink 进行仿真

Kalman 算法由 M 函数实现。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 0.5Hz 的正弦信号。采用卡尔曼滤波器实现信号的滤



波，取 $Q=1$ ， $R=1$ 。仿真结果如图 1-75 和图 1-76 所示。

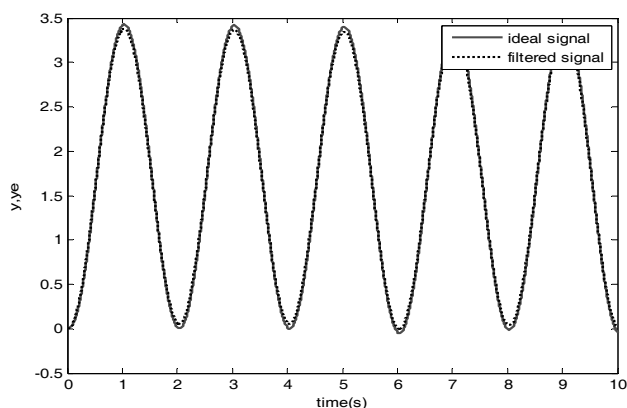


图 1-75 原始信号 y 及滤波后的信号 y_e

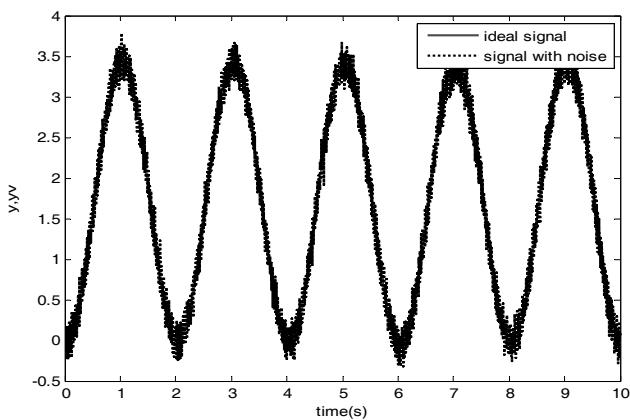


图 1-76 原始信号 y 及带有噪声的原始信号 y_v

仿真程序：

Simulink 主程序：chap1_27.mdl（见图 1-77）

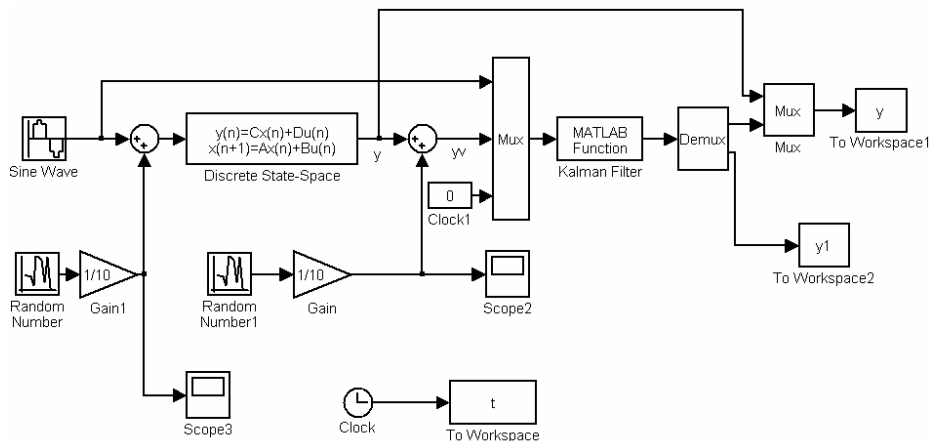


图 1-77 基于 Kalman 滤波器的 Simulink 仿真



Kalman 滤波子程序: chap1_27f.m

```
%Discrete Kalman filter
%x=A*x+B*(u+w(k));
%y=C*x+D+v(k)
function [u]=kalman(u1,u2,u3)
persistent A B C D Q R P x

yv=u2;
if u3==0
    x=zeros(2,1);
    ts=0.001;
    a=25;b=133;
    sys=tf(b,[1,a,0]);
    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
end

%Measurement update
Mn=P*C'/(C*P*C'+R);

x=A*x+Mn*(yv-C*A*x);

P=(eye(2)-Mn*C)*P;

ye=C*x+D;              %Filtered value

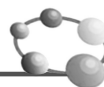
u(1)=ye;               %Filtered signal
u(2)=yv;               %Signal with noise

errcov=C*P*C';         %Covariance of estimation error

%Time update
x=A*x+B*u1;
P=A*P*A'+B*Q*B';
```

作图程序: chap1_27plot.m

```
close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('y,ye');
```



```

legend('ideal signal','filtered signal');

figure(2);
plot(t,y(:,1),'r',t,y1(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y,yv');
legend('ideal signal','signal with noise');

```

1.3.16.3 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 1-78 所示。

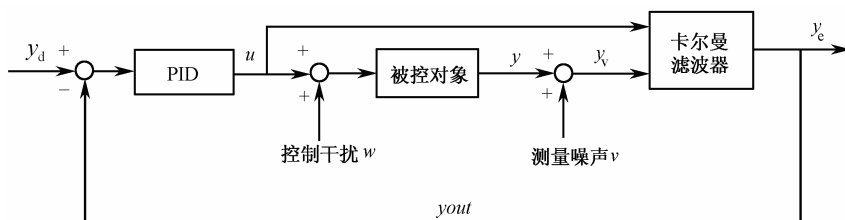


图 1-78 基于卡尔曼滤波的 PID 控制系统结构

1.3.16.4 仿真实例

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 1.3.16.2 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 1-79 和图 1-80 所示，仿真结果表明，通过采用滤波器使控制效果明显改善。

本方法的不足之处是设计卡尔曼滤波器时需要被控对象的精确模型。

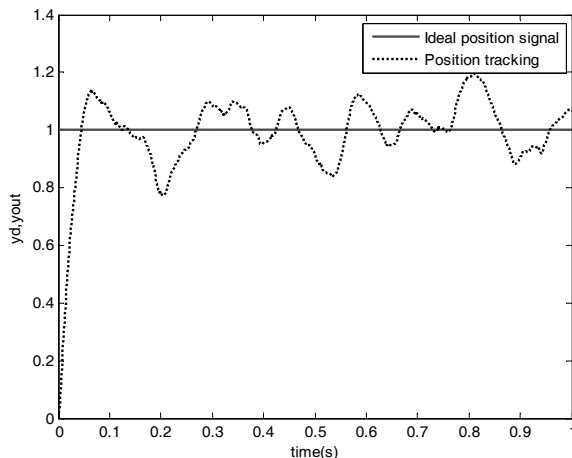
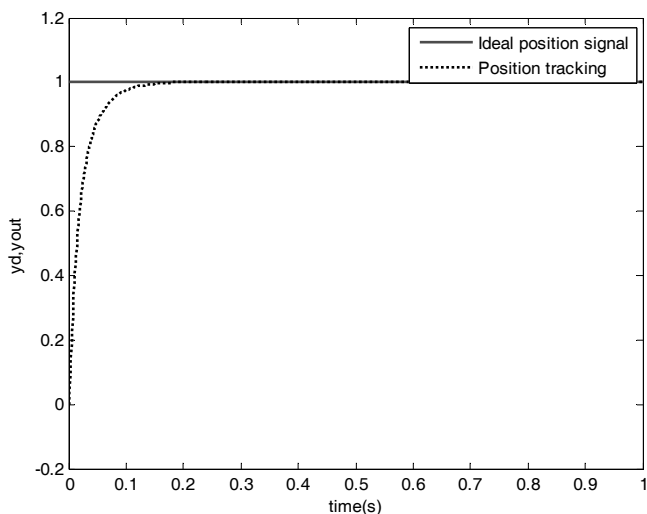


图 1-79 无滤波器时 PID 控制阶跃响应 ($M=1$)

图 1-80 加入滤波器后 PID 控制阶跃响应 ($M=2$)

仿真程序: chap1_28.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;

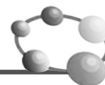
ts=0.001;
%Continuous Plant
a=25;b=133;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

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;

yd(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          %No kalman filter
    yout(k)=yv(k);
elseif M==2      %Using kalman filter
    yout(k)=ye(k);
end
error(k)=yd(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,yd,'r',time,yout,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,yout');

```



```
legend('Ideal position signal','Position tracking');
```



1.4 S 函数介绍

S 函数中使用文本方式输入公式和方程, 适合复杂动态系统的数学描述, 并且在仿真过程中可以对仿真参数进行更精确的描述。在本书的控制系统的 Simulink 仿真中, 主要使用 S 函数来实现控制律、自适应律和被控对象的描述。

1.4.1 S 函数简介

S 函数模块是整个 Simulink 动态系统的核心, 也可以说 S 函数是 Simulink 最具魅力的地方。

S 函数是系统函数 (system function) 的简称, 是指采用非图形化的方式 (即计算机语言, 区别于 Simulink 的系统模块) 描述的一个功能块。用户可以采用 MATLAB 代码、C、C++ 等语言编写 S 函数。S 函数由一种特定的语法构成, 用来描述并实现连续系统、离散系统及复合系统等动态系统, S 函数能够接受来自 Simulink 求解器的相关信息, 并对求解器发出的命令做出适当的响应, 这种交互作用非常类似于 Simulink 系统模块与求解器的交互作用。一个结构体系完整的 S 函数包含了描述动态系统所需的全部能力, 所有其他的使用情况都是这个结构体系的特例。

1.4.2 S 函数使用步骤

一般而言, S 函数的使用步骤如下:

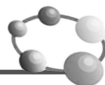
- (1) 创建 S 函数源文件。创建 S 函数源文件有多种方法, Simulink 提供了很多 S 函数模板和例子, 用户可以根据自己的需要修改相应的模板或例子。
- (2) 在动态系统的 Simulink 模型框图中添加 S-function 模块, 并进行正确的设置。
- (3) 在 Simulink 模型框图中按照定义好的功能连接输入/输出端口。

为了方便 S 函数的使用和编写, Simulink 的 Functions&Tables 模块库还提供了 S-function demos 模块组, 该模块组为用户提供了编写 S 函数的各种例子, 以及 S 函数模块。

1.4.3 S 函数的基本功能及重要参数设定

(1) S 函数功能模块: 各种功能模块完成不同的任务, 这些功能模块 (函数) 称为仿真例程或回调函数 (call-back functions), 包括初始化 (initialization)、导数 (mdlDerivative)、输出 (mdlOutput) 等。

- (2) NumContStates 表示 S 函数描述的模块中连续状态的个数。
- (3) NumDiscStates 表示离散状态的个数。
- (4) NumOutputs 和 NumInputs 分别表示模块输入和输出的个数。
- (5) 直接馈通 (dirFeedthrough) 为输入信号是否在输出端出现的标识, 取值为 0 或 1。



(6) NumSampleTimes 为模块采样周期的个数，S 函数支持多采样周期的系统。

除了 sys 外，还应设置系统的初始状态变量 x_0 、说明变量 str 和采样周期变量 t_s 。 t_s 变量为双列矩阵，其中每一行对应一个采样周期。对连续系统和单个采样周期的系统来说，该变量为 $[t_1, t_2]$ ， t_1 为采样周期， $t_1 = -1$ 表示继承输入信号的采样周期， t_2 为偏移量，一般取为 0。对连续系统来说， t_s 取为 $[-1, 0]$ 。

1.4.4 实例说明

在控制系统仿真中，通常采用 S 函数描述控制律、自适应律和被控对象。本章采用 S 函数进行控制系统仿真的例子见 1.2.1 节的仿真之三和仿真之四，1.2.2 节的仿真之二和 1.3.2 节的仿真之三。

以 1.2.1 节的仿真之三为例，采用 S 函数实现了控制系统的 Simulink 仿真，仿真主程序为 chap1_3.mdl。分别采用 S 函数描述控制律和被控对象，介绍如下。

(1) 采用 S 函数描述被控对象，见程序 chap1_3plant.m。利用 S 函数的微分子模块描述被控对象

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -ax_2 + bu\end{aligned}$$

程序如下：

```
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-(25+10*rands(1))*x(2)+(133+30*rands(1))*u;
```

(2) 采用 S 函数描述 PID 控制器，见程序 chap1_3s.m。利用 S 函数的输出子模块描述 PID 控制算法，程序如下：

```
function sys=mdlOutputs(t,x,u)
error=u(1);
derror=u(2);
errori=u(3);

kp=60;
ki=1;
kd=3;
ut=kp*error+kd*derror+ki*errori;
sys(1)=ut;
```



1.5 PID 研究新进展

(1) 在理论研究方面

A.N.Gundes 等研究了线性 MIMO 系统的 PID 控制稳定性问题，V.A.Oliveira 等针对时滞系统的 PID 控制问题进行了深入探讨，J.A.Ramirez 等研究了控制输入受限下的 PID 控制算法。



(2) 在应用研究方面

韩京清提出了基于自抗扰控制理论的非线性 PID 控制^[4], 该理论结合微分器、扩张观测器和过渡过程, 有效地提高了控制性能。Y.X.Su 等^[5]将韩京清所提出的方法成功地应用于机械手的控制中。

(3) 先进 PID 整定研究

J.Chen 等^[6]采用神经网络实现了 PID 的在线整定, T.K.Teng 等^[7]采用遗传算法实现了 PID 的在线整定。T.H.Kim 等^[8]以 H^∞ 为优化性能指标, 采用粒子群算法实现了 PID 的整定。K.S.Tang 等^[9]采用模糊逻辑, 并利用遗传算法优化, 实现了 PID 的整定。F.Zheng 等^[10]采用线性矩阵不等式 LMI 方法实现了 PID 参数的整定, 使之满足最优性能。

(4) PID 控制器产品

文^[11,12]深入分析了 P、I、D 三项的独立整定效果和对闭环系统稳定性的影响, 并介绍了国际上有代表性的 PID 控制算法专利、PID 控制软件包和 PID 硬件系统。

(5) 有关 PID 控制相关著作

著作^[13]探讨了针对延迟系统的 PID 控制器设计方法, 著作^[14]探讨了针对延迟系统的 PID 控制器设计方法, 著作^[15]针对实际工程探讨了 PID 控制器设计方法, 著作^[16]给出了各种情况下 PID 控制器的调节规则。

第2章 PID控制器的整定

2.1 概述

几十年来, PID 控制的参数整定方法和技术处于不断发展中, 特别是近年来, 国际自动控制领域对 PID 控制的参数整定方法的研究仍在继续, 许多重要国际杂志不断发表新的研究成果。Astrom 和 Hagglund 于 1988 年出版了专著《PID 控制器自整定》^[17], 并于 1995 年再次出版了《PID 控制器:理论、设计及整定》一书, 介绍了 PID 整定的常用方法^[18]。

自 Ziegler 和 Nichols^[19]提出 PID 参数整定方法起, 有许多技术已经被用于 PID 控制器的手动和自动整定。根据发展阶段的划分, 可分为常规 PID 参数整定方法及智能 PID 参数整定方法; 按照被控对象个数来划分, 可分为单变量 PID 参数整定方法^[20]及多变量 PID 参数整定方法^[21], 前者包括现有大多数整定方法, 后者是研究的热点及难点; 按控制量的组合形式来划分, 可分为线性 PID 参数整定方法及非线性 PID 参数整定方法, 前者用于经典 PID 调节器, 后者用于由非线性跟踪—微分器和非线性组合方式生成的非线性 PID 控制器^[22]。

2.2 基于响应曲线法的 PID 整定

工业生产中常用的 PID 整定方法有经验法、衰减曲线法和响应曲线法。其中经验法也叫试凑法, 具体包括先比例、后积分、再微分三个步骤。衰减曲线法包括 4:1 衰减曲线法和 10:1 衰减曲线法。下面以响应曲线法和临界比例度法的 PID 整定为例来介绍。

2.2.1 基本原理

可根据带有时滞环节的一阶近似模型的阶跃响应来整定 PID。该模型表示为

$$G(s) = \frac{K}{Ts+1} e^{-\tau s} \quad (2.1)$$

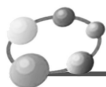
式中, 一阶响应的特征参数 K 、 T 和 τ 可以由图 2-1 所构成的示意图提取出来。

响应曲线法是根据给定对象的瞬态响应特性参数 K 、 T 和 τ 来确定 PID 控制器的参数, 整定公式如表 2-1 所示^[18]。如果单位阶跃响应曲线为 S 形曲线, 则可用此法, 否则不能用。

PID 控制算法为

$$u(t) = \frac{1}{\delta} \left(e + \frac{1}{T_I} \int_0^t e dt + T_D \frac{de}{dt} \right) \quad (2.2)$$

式中, δ 为比例度, T_I 为积分时间, T_D 为微分时间。



如果取 $k_p = \frac{1}{\delta}$, $k_i = \frac{k_p}{T_i}$, $k_d = k_p T_D$, 则 PID 控制律表示为

$$u(t) = k_p e + k_i \int_0^t e dt + k_d \frac{de}{dt} \quad (2.3)$$

该方法首先要通过实验测定开环系统对阶跃输入信号的响应曲线, 具体步骤为: ^[18]

- (1) 首先进行开环控制, 断开反馈通道, 给被控对象一个阶跃输入信号 Δu ;
- (2) 记录被控对象的输出特性曲线;
- (3) 从曲线上求得参数 $u_{\min}$ 、 $u_{\max}$ 、 $y_{\min}$ 、 $y_{\max}$ 、 T 和 τ ;
- (4) 计算 K 和飞升速度 ε ;
- (5) 根据所求的 τ 和 ε , 按表 2-1 的经验公式求出不同类型的控制器参数。

K 和 ε 按下式计算:

$$K = \frac{\frac{\Delta y}{y_{\max} - y_{\min}}}{\frac{\Delta u}{u_{\max} - u_{\min}}}, \quad \varepsilon = \frac{K}{T} \quad (2.4)$$

式中, Δu 为输入信号的阶跃值, $u_{\min}$ 和 $u_{\max}$ 为输入信号的最大最小值, $y_{\min}$ 和 $y_{\max}$ 为对象输出的最大最小值。

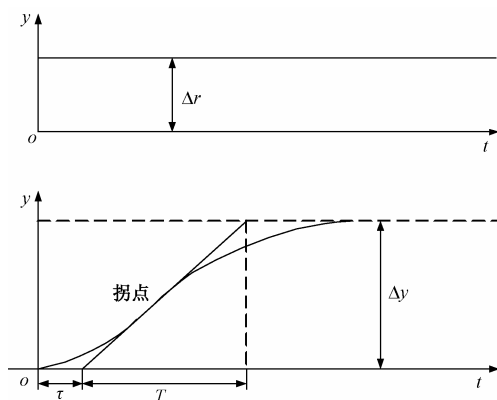


图 2-1 开环系统对阶跃输入信号的响应曲线示意图

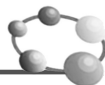
表 2-1 响应曲线法整定 PID 参数

控制器类型	比例度 $\delta(\%)$	积分时间 T_i	微分时间 T_D
P	$\varepsilon\tau$		
PI	$1.1\varepsilon\tau$	3.3τ	
PID	$0.85\varepsilon\tau$	2τ	0.5τ

2.2.2 仿真实例

设被控对象为

$$G_p(s) = \frac{K}{Ts+1} e^{-\tau s}$$



响应曲线法整定分以下三步：

(1) 首先断开反馈通道，给被控对象一个阶跃输入信号，仿真程序为 chap2_1sim.mdl，这是一个反复测试的过程，如图 2-2 所示；

(2) 由图 2-2 可以近似得到 $\tau = 80$ ， $T = 60$ ，从而得到

$$K = \frac{\frac{\Delta y}{y_{\max} - y_{\min}}}{\frac{\Delta u}{u_{\max} - u_{\min}}} = 1, \quad \varepsilon = \frac{K}{T} = \frac{1}{60}$$

则对象模型可表示为

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

(3) 采用 PID 控制算法，根据表 2-1 可计算得： $\delta = 0.85\varepsilon\tau = 0.85 \times \frac{1}{60} \times 80 = 1.1333$ ，即

$$k_p = \frac{1}{\delta} = 0.8824, \quad T_i = 2\tau = 160, \quad T_D = 0.5\tau = 40。$$

连续系统控制仿真结果如图 2-3 所示。取采样周期为 $T_s = 20$ ，将被控对象和 PID 控制器离散化，离散控制仿真结果如图 2-4 所示。

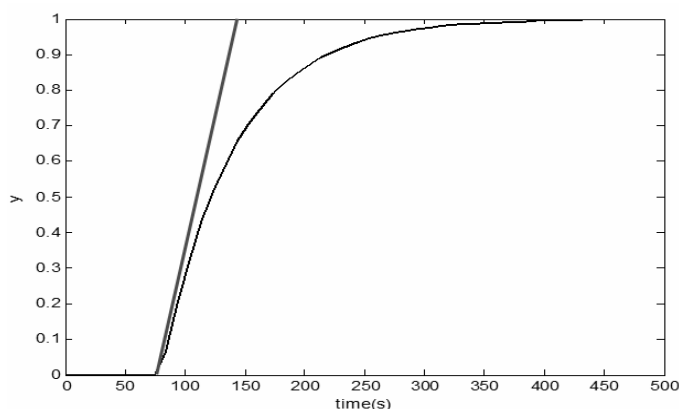


图 2-2 开环系统对阶跃输入信号的响应曲线

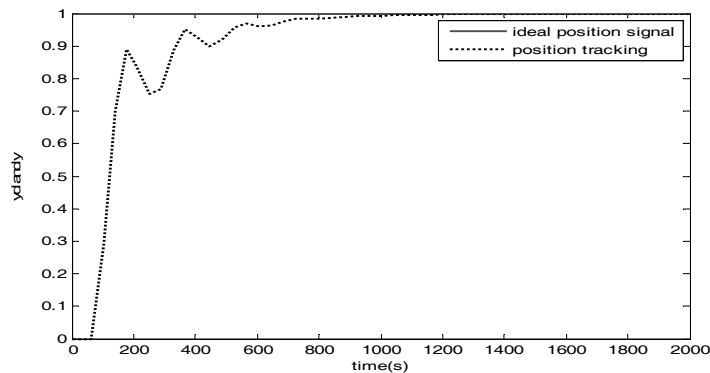


图 2-3 连续 PID 控制单位阶跃响应曲线

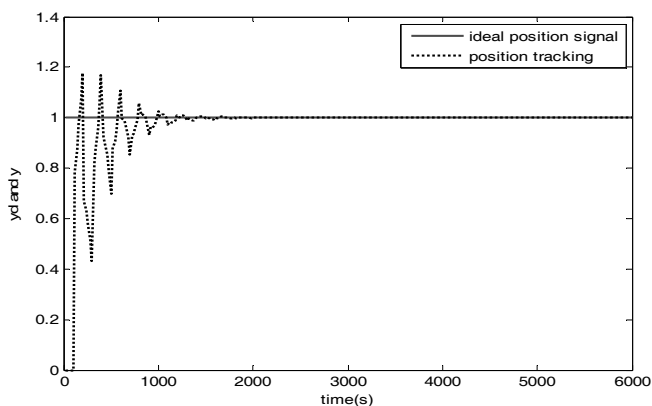


图 2-4 离散 PID 控制单位阶跃响应曲线

仿真程序

(1) 对象开环测试

Simulink 主程序: chap2_1sim.mdl (见图 2-5)

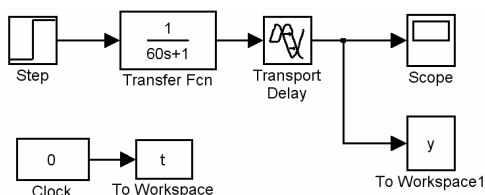


图 2-5 对象开环测试 Simulink 主程序

作图程序: chap2_1plot.m

```
close all;

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');
```

(2) 连续 PID 控制仿真

Simulink 主程序: chap2_2sim.mdl (见图 2-6)

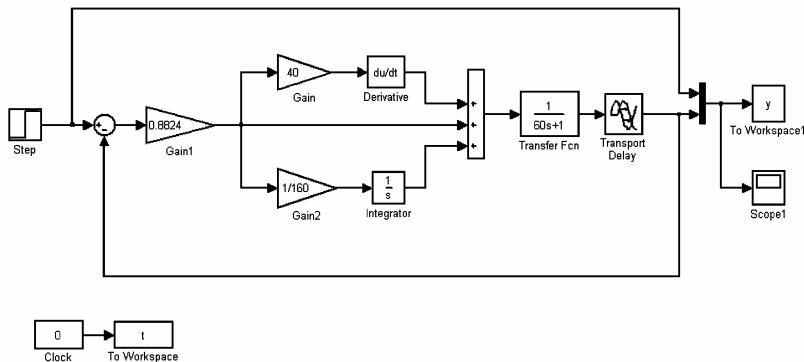


图 2-6 连续 PID 控制的 Simulink 主程序



作图程序: chap2_2plot.m

```
close all;

figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd and y');
legend('ideal position signal','position tracking');
```

(3) 离散 PID 控制仿真: chap2_3.m

```
clear all;
close all;
Ts=20;

%Delay plant
K=1;
Tp=60;
tol=80;
sys=tf([K],[Tp,1],'inputdelay',tol);
dsys=c2d(sys,Ts,'zoh');
[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;
e_1=0;
ei=0;
y_1=0.0;
for k=1:1:300
time(k)=k*Ts;

yd(k)=1.0;      %Tracing Step Signal

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

e(k)=yd(k)-y(k);
de(k)=(e(k)-e_1)/Ts;
ei=ei+Ts*e(k);

delta=0.885;
TI=160;
TD=40;

u(k)=delta*(e(k)+1/TI*ei+TD*de(k));

e_1=e(k);
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=y(k);
```



```
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd and y');
legend('ideal position signal','position tracking');
```



2.3 基于 Ziegler-Nichols 的频域响应 PID 整定

2.3.1 连续 Ziegler-Nichols 方法的 PID 整定

Ziegler-Nichols 频域整定方法是基于稳定性分析的频域响应 PID 整定方法。该方法整定的思想是：对于给定的被控对象传递函数，可以得到其根轨迹，对应穿越 $j\omega$ 轴的点，增益即为 K_m ，而此点的 ω 值即为 ω_m 。

整定公式如下：^[19]

$$K_p = 0.6K_m, \quad K_d = \frac{K_p \pi}{4\omega_m}, \quad K_i = \frac{K_p \omega_m}{\pi} \quad (2.5)$$

式中， K_m 为系统开始振荡时的增益 K 值， ω_m 为振荡频率。

2.3.2 仿真实例

设被控对象为

$$G(s) = \frac{400}{s(s^2 + 30s + 200)}$$

运行整定程序 chap2_4tuning.m，可得图 2-7 至图 2-9。如图 2-7 所示为未整定时开环系统的根轨迹图，在该图上可选定穿越 $j\omega$ 轴时的点（共有两个点，任选其一），从而获得增益 K_m 和该点的 ω 值即 ω_m 。

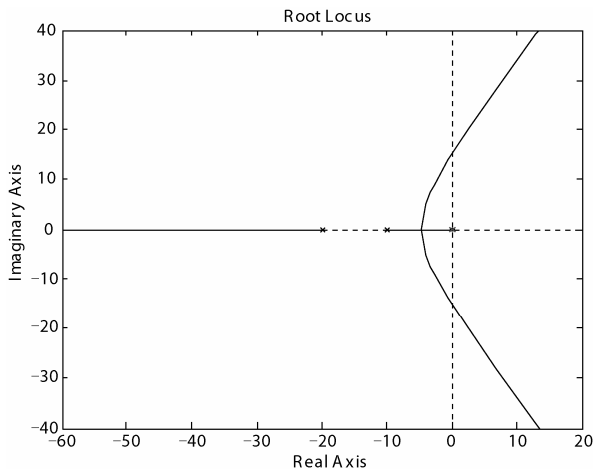


图 2-7 未整定时开环系统的根轨迹图

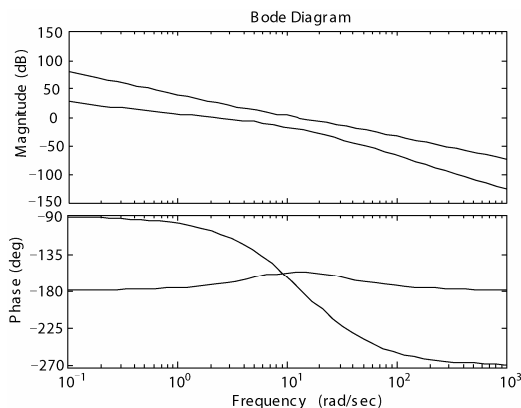
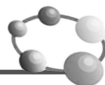


图 2-8 整定前后系统的波特图（实线为整定前，虚线为整定后）

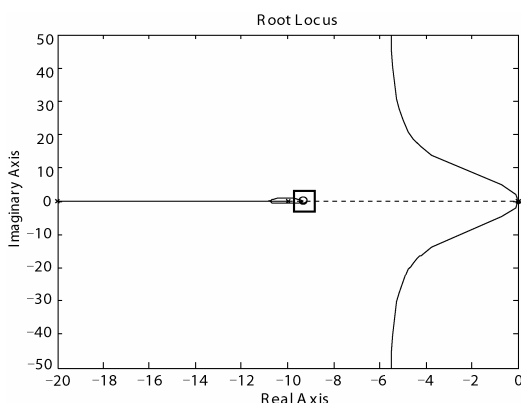


图 2-9 整定后闭环系统的根轨迹

使用 `rlocus` 及 `rlocfind` 命令可求得穿越增益 $K_m=14$ 和穿越频率 $\omega_m=14\text{rad/s}$ 。采用 Ziegler-Nichols 整定方法 (2.5) 可求得 PID 参数:

$$K_p = 8.8371, \quad K_d = 0.4945, \quad K_i = 39.4847。$$

整定程序中, `sys_pid` 和 `sysc` 分别为控制器和闭环系统的传递函数。如图 2-8 所示为整定前后系统的波特图, 可见, 该系统整定后, 频带拓宽, 相移超前。如图 2-9 所示为整定后闭环系统的根轨迹, 所有极点位于负半面, 达到完全稳定状态。

运行 Simulink 控制程序 `chap2_4.mdl`, 通过开关切换进行两种方法的仿真, 可得图 2-10 和图 2-11, 如图 2-10 所示为整定前的正弦跟踪, 如图 2-11 所示为整定后的正弦跟踪。

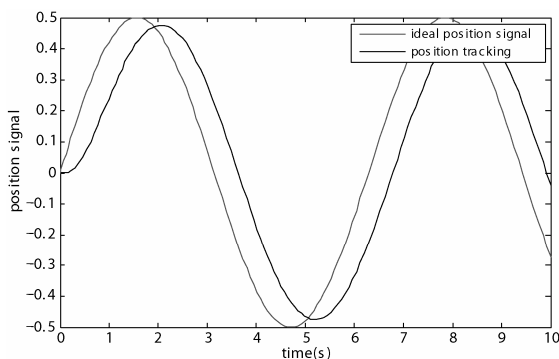


图 2-10 整定前的正弦跟踪

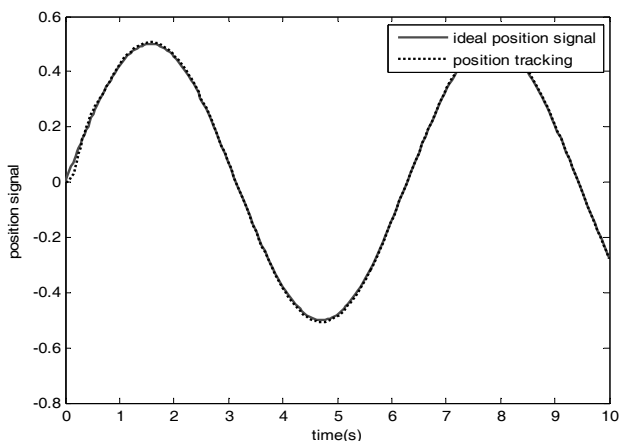


图 2-11 整定后的正弦跟踪

仿真程序：分为 PID 整定程序、Simulink 主程序和作图程序三个部分。

PID 整定程序：chap2_4tuning.m

```
%PID Controler Based on Ziegler-Nichols
```

```
clear all;
```

```
close all;
```

```
sys=tf(400,[1,30,200,0]);
```

```
figure(1);
```

```
rlocus(sys);
```

```
[km,pole]=rlocfind(sys)
```

```
wm=imag(pole(2));
```

```
kp=0.6*km
```

```
kd=kp*pi/(4*wm)
```

```
ki=kp*wm/pi
```

```
figure(2);
```

```
grid on;
```

```
bode(sys,'r');
```

```
sys_pid=tf([kd,kp,ki],[1,0])
```

```
sysc=series(sys,sys_pid)
```

```
hold on;
```

```
bode(sysc,'b')
```

```
figure(3);
```

```
rlocus(sysc);
```

Simulink 主程序：chap2_4sim.mdl（见图 2-12）

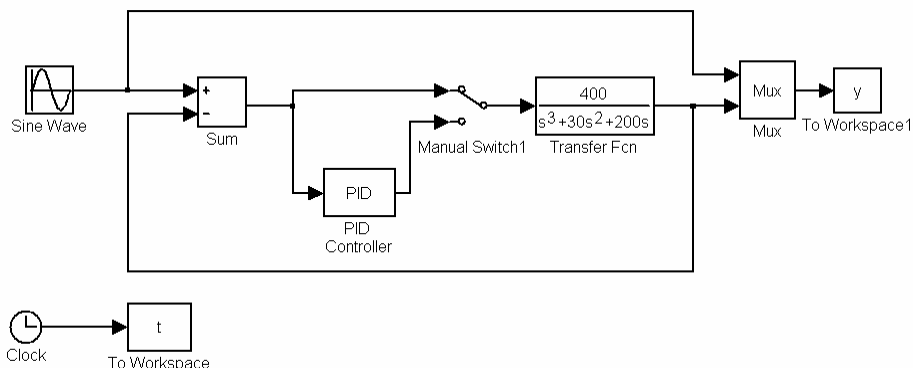
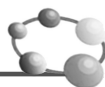


图 2-12 Simulink 主程序

作图程序: chap2_4plot.m

```
close all;
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('position signal');
legend('ideal position signal','position tracking');
```

2.3.3 离散 Ziegler-Nichols 方法的 PID 整定

Ziegler-Nichols 方法同样也适用于离散系统的 PID 整定。该方法整定比例系数的思想是：^[19]对于给定的被控对象传递函数，选择其离散系统的根轨迹图与 z 平面单位圆交点（共有两个点，任选其一），从而获得增益 K_m 和该点的 ω_m 值即 ω_m 。整定公式如下：

$$K_p = 0.6K_m, \quad K_d = \frac{K_p \pi}{4\omega_m}, \quad K_i = \frac{K_p \omega_m}{\pi} \quad (2.6)$$

式中， K_m 为系统开始振荡时的 K_p 值， ω_m 为振荡频率。振荡频率 ω_m 可以由极点位于单位圆上的角度 θ 得到， $\omega_m = \frac{\theta}{T}$ （ T 为采样周期）。

2.3.4 仿真实例

设被控对象为

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

采样周期为 $T=0.25s$ 。

采用零阶保持器将对象离散化，使用 `rlocus` 及 `rlocfind` 命令做出 $G(z)$ 的根轨迹图，可求得振荡增益 $K_m=11.2604$ 和振荡频率 $\omega_m=1.0546\text{rad/s}$ 。采用 Ziegler-Nichols 公式 (2.6) 可求得离散 PID 参数

$$K_p = 6.7562, \quad K_d = 5.0318, \quad K_i = 2.2679。$$

运行整定程序 `chap2_5tuning.m`，可得图 2-13 和图 2-14。如图 2-13 所示为系统未补偿的根轨迹图与 z 平面单位圆的比较，在根轨迹图上选定与 z 平面单位圆上的交点（共有两个点，



任选其一), 则求得所对应的增益 K_m 和该点对应的 ω_m 。整定程序中, dsysc 为校正后的离散闭环系统。如图 2-14 所示为 PID 整定后闭环系统的根轨迹与单位圆比较。

运行控制程序 chap2_5.m, 通过开关切换进行两种方法的仿真, 可得图 2-15 和图 2-16, 如图 2-15 所示为系统采用 PID 校正后的正弦跟踪, 如图 2-16 所示为系统未校正的正弦跟踪。

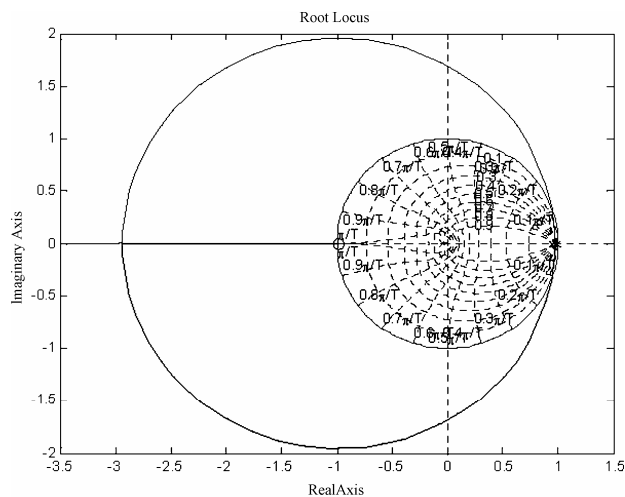


图 2-13 未整定时开环系统的根轨迹图与单位圆比较

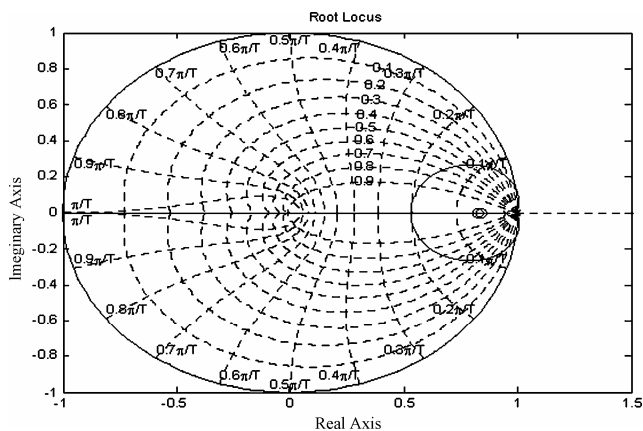


图 2-14 整定后闭环系统的根轨迹图与单位圆比较

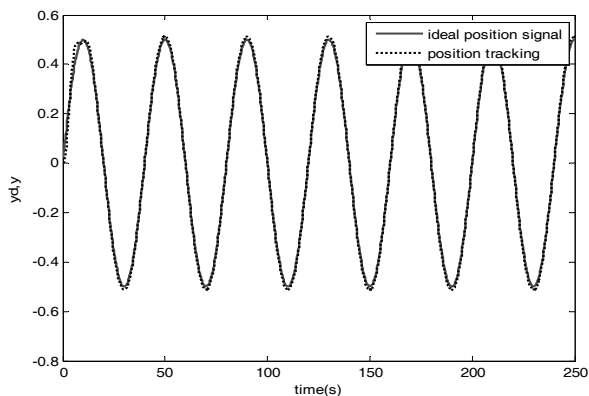
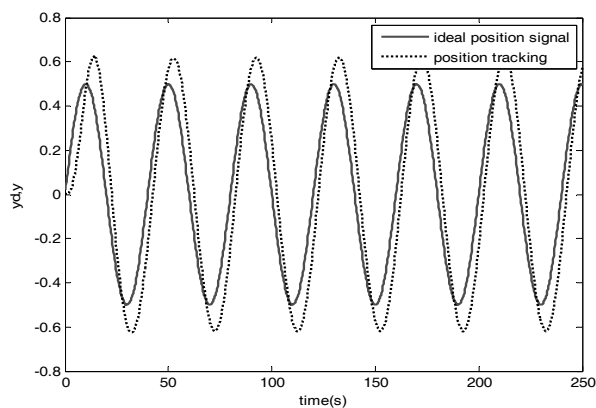
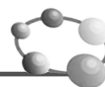


图 2-15 校正后的正弦跟踪 ($M=1$)

图 2-16 未校正的正弦跟踪 ($M=2$)

仿真程序：分为 PID 整定程序和 PID 控制程序两部分。

PID 整定程序：chap2_5tuning.m

```
%PID Controler Based on Ziegler-Nichols
```

```
clear all;
```

```
close all;
```

```
ts=0.25;
```

```
sys=tf(1,[10,2,0]);
```

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

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

```
axis('normal'),zgrid('new');
```

```
figure(1);
```

```
rlocus(dsys);
```

```
[km,pole]=rlocfind(dsys)
```

```
wm=angle(pole(1))/ts;
```

```
kp=0.6*km
```

```
kd=kp*pi/(4*wm)
```

```
ki=kp*wm/pi
```

```
sysc=tf([kd,kp,ki],[10,2,0,0]);
```

```
dsysc=c2d(sysc,ts,'zoh');
```

```
figure(2);
```

```
rlocus(dsysc);
```

```
axis('normal'),zgrid;
```

PID 控制程序：chap2_5.m

```
close all;
```

```
ts=0.25;
```




```
sys=tf(1,[10,2,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;
y_1=0;y_2=0;

x=[0,0,0]';
error_1=0;
for k=1:1:1000
time(k)=k*ts;

%yd(k)=1.0;
yd(k)=0.5*sin(0.025*2*pi*k*ts);

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

M=1;
switch M
    case 1          %Using PID
        u(k)=kp*x(1)+kd*x(2)+ki*x(3);
    case 2          %No PID
        u(k)=error(k);
end

u_2=u_1;
u_1=u(k);

y_2=y_1;
y_1=y(k);

error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position signal','position tracking');
figure(2);
plot(time,yd-y,'r');
xlabel('time(s)');ylabel('error');
```



2.4 基于频域分析的 PD 整定

2.4.1 基本原理

针对二阶系统传递函数，采用频域分析方法^[23]，可实现 PD 的整定。二阶系统传递函数的标准形式为

$$\phi(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \quad (2.7)$$

二阶系统的动态特性可用系统阻尼比 ξ 和固有频率 ω_n 来描述，它的动态特性为

$$s^2 + 2\xi\omega_n s + \omega_n^2 = 0 \quad (2.8)$$

下面以二阶系统为例来说明 PD 控制机理。被控对象为

$$G(s) = \frac{K}{s(\tau s + 1)} \quad (2.9)$$

闭环控制器采用 PD 控制

$$D(s) = K_p + K_d s \quad (2.10)$$

则闭环系统的传递函数为 $\frac{D(s)G(s)}{1 + D(s)G(s)}$ ，其特征方程为 $1 + D(s)G(s) = 0$ ，即

$$\tau s^2 + (1 + KK_d)s + KK_p = 0 \quad (2.11)$$

根据式 (2.11)，可得系统的固有频率为

$$\omega_n = \sqrt{KK_p / \tau} \quad (2.12)$$

由式 (2.11) 可见，PD 控制律中的比例项 K_p 决定了系统的固有频率，即响应速度。

根据式 (2.11)，可得系统的阻尼比 ξ 为

$$\xi = \frac{1 + KK_d}{2\tau} \sqrt{\frac{\tau}{KK_p}} \quad (2.13)$$

由式 (2.13) 可见，系统的阻尼特性取决于微分项 K_d 。

由上述分析可见：在 PD 控制中，当增加 K_p 提高系统的响应速度时，系统的阻尼将下降。微分项 K_d 起到增加阻尼的作用，提高了系统的相对稳定性。

2.4.2 仿真实例

为了分析 K_p 和 K_d 对 PD 控制效果的影响，仿真中分别取 $M = 1, 2, 3$ ，采用 3 组不同的 K_p 和 K_d 进行分析。仿真结果如图 2-17 至图 2-19 所示。

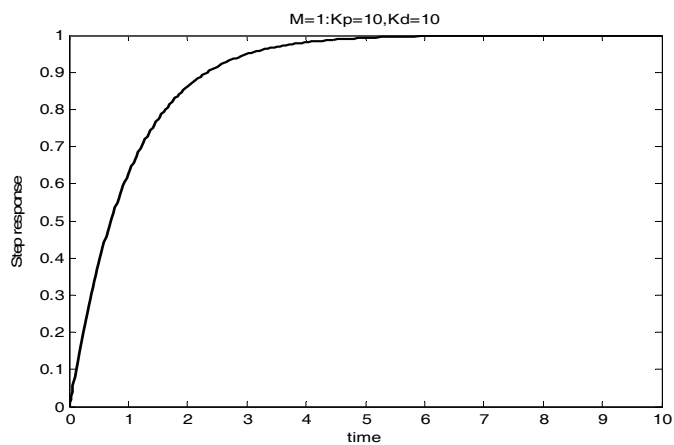
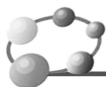


图 2-17 $K_p = 10$, $K_d = 20$ 时仿真结果 ($M=1$)

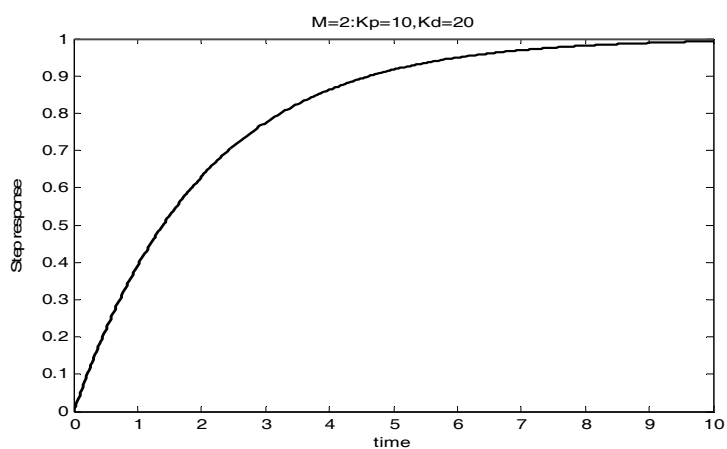


图 2-18 $K_p = 10$, $K_d = 20$ 时仿真结果 ($M=2$)

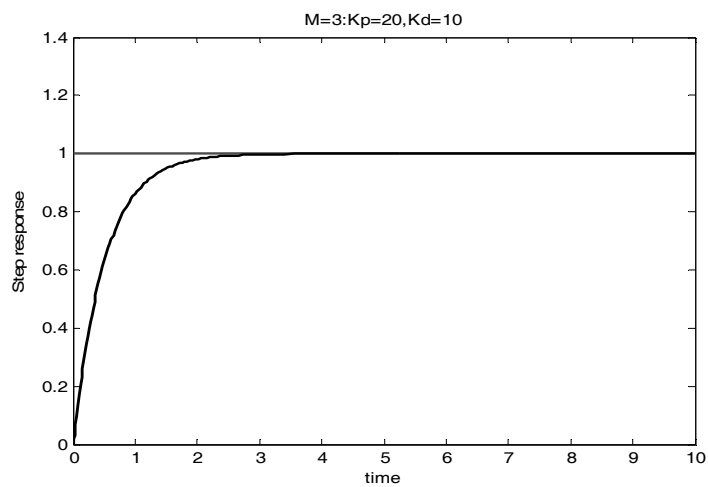
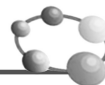


图 2-19 $K_p = 20$, $K_d = 10$ 时仿真结果 ($M=3$)



仿真程序：

初始化程序：chap2_6int.m

```
clear all;
close all;
K=10;
tol=1.0;

M=3;
if M==1
    Kp=10;Kd=10;
elseif M==2
    Kp=10;Kd=20;
elseif M==3
    Kp=20;Kd=10;
end
```

Simulink 主程序：chap2_6sim.mdl（见图 2-20）

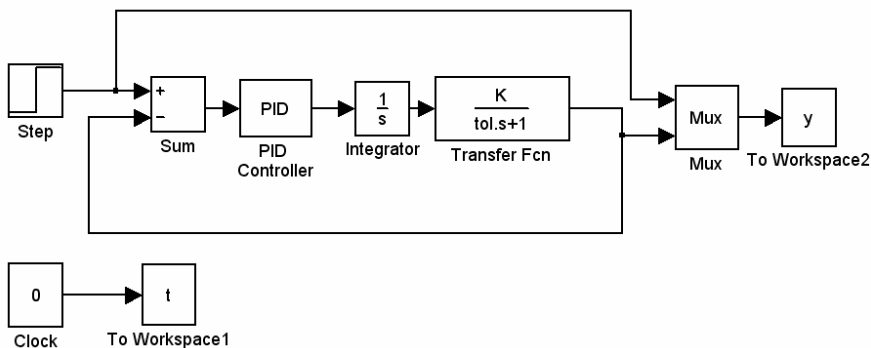
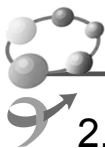


图 2-20 PD 控制的 Simulink 主程序

作图程序：chap2_6plot.m

```
close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time');ylabel('Step response');

if M==1
    title('M=1:Kp=10,Kd=10');
elseif M==2
    title('M=2:Kp=10,Kd=20');
elseif M==3
    title('M=3:Kp=20,Kd=10');
end
```



2.5 基于相位裕度整定的PI控制

2.5.1 基本原理

相位裕度是衡量系统稳定度的一个重要指标。它是指频率的回路增益等 0dB (单位增益) 时, 反馈信号总的相位偏移与 -180° 的差。

相位裕度可以看作是系统进入不稳定状态之前可以增加的相位变化, 相位裕度越大, 系统越稳定, 但同时时间响应速度减慢了, 因此必须要有一个比较合适的相位裕度。经研究发现, 相位裕度至少要 45° , 最好是 60° 。^[23]

针对具有一阶加积分形式的被控对象:

$$G(s) = \frac{K_g}{s(T_g s + 1)} = \frac{K_g \omega_g}{s(s + \omega_g)} \quad (2.14)$$

式中, $\omega_g = \frac{1}{T_g}$ 。

PI 控制器表示为:

$$K(s) = K_p \left(1 + \frac{1}{T_i s} \right) = K_p \left(1 + \frac{\omega_i}{s} \right) \quad (2.15)$$

一阶加积分形式的被控对象具有如下频率特性: 当转折频率是对称分布时, 闭环系统 $K(s)G(s)$ 的相位裕度有最大值, 如图 2-17 所示。^[23]

根据图 2-21, 有如下关系

$$2 \lg \omega_c = \lg \omega_i + \lg \omega_g$$

即

$$\frac{\omega_g}{\omega_c} = \frac{\omega_c}{\omega_i} \quad (2.16)$$

令 $\alpha = \frac{\omega_g}{\omega_i}$, 则有

$$\omega_c^2 = \omega_g \omega_i = \omega_g \frac{\omega_g}{\alpha}, \text{ 即 } \omega_c = \omega_g / \sqrt{\alpha} \quad (2.17)$$

$$\omega_i = \omega_g / \alpha \quad (2.18)$$

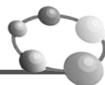
$$\frac{\omega_c}{\omega_i} = \frac{\omega_g / \sqrt{\alpha}}{\omega_g / \alpha} = \sqrt{\alpha} \quad (2.19)$$

通过参考文献^[23], 下面给出 K_p 和 x 的设计方法:

(1) K_p 的设计:

闭环系统可写为

$$G(s)K(s) = \frac{K_g}{s(T_g s + 1)} K_p \left(1 + \frac{1}{T_i s} \right) = K_g K_p \frac{1}{s} \frac{1}{T_i} \frac{1}{T_g s + 1} (T_i s + 1)$$



根据典型环节的频率特性可知：积分环节 $\frac{1}{s}$ 的幅频为 $\frac{1}{\omega}$ ；一阶惯性环节 $\frac{1}{T_g s + 1}$ 的幅频为 $\frac{1}{\sqrt{(T_g \omega)^2 + 1}}$ ；一阶微分环节 $T_i s + 1$ 的幅频为 $\sqrt{(T_i \omega)^2 + 1}$ ；则闭环系统 $G(s)K(s)$ 的幅频 $A(\omega)$ 为 $K_g K_p \frac{1}{\omega} \frac{1}{\omega} \frac{1}{T_i} \frac{1}{\sqrt{(T_g \omega)^2 + 1}} \sqrt{(T_i \omega)^2 + 1}$ 。

由图 2-21 可知，当闭环系统相位裕度为最大时， $\lg A(\omega_c) = 0$ ，即 $A(\omega_c) = 1$ ，此时

$$K_g K_p \frac{1}{\omega_c} \frac{1}{\omega_c} \frac{1}{T_i} \frac{\sqrt{(T_i \omega_c)^2 + 1}}{\sqrt{(T_g \omega_c)^2 + 1}} = 1 \quad (2.20)$$

由于

$$\frac{1}{T_i} \frac{\sqrt{(T_i \omega_c)^2 + 1}}{\sqrt{(T_g \omega_c)^2 + 1}} = \omega_1 \frac{\sqrt{\left(\frac{\omega_c}{\omega_1}\right)^2 + 1}}{\sqrt{\left(\frac{\omega_c}{\omega_g}\right)^2 + 1}} = \omega_1 \frac{\sqrt{\left(\frac{\omega_c}{\omega_1}\right)^2 + 1}}{\sqrt{\left(\frac{\omega_1}{\omega_c}\right)^2 + 1}} = \omega_1 \frac{\frac{1}{\omega_1} \sqrt{\omega_c^2 + \omega_1^2}}{\frac{1}{\omega_c} \sqrt{\omega_1^2 + \omega_c^2}} = \omega_c$$

则式 (2.20) 变为

$$K_g K_p \frac{1}{\omega_c} \frac{1}{\omega_c} \frac{1}{T_i} \frac{\sqrt{(T_i \omega_c)^2 + 1}}{\sqrt{(T_g \omega_c)^2 + 1}} = K_g K_p \frac{1}{\omega_c} \frac{1}{\omega_c} \omega_c = 1$$

即

$$\begin{aligned} K_g K_p \frac{1}{\omega_c} &= 1 \\ K_p &= \omega_c / K_g \end{aligned} \quad (2.21)$$

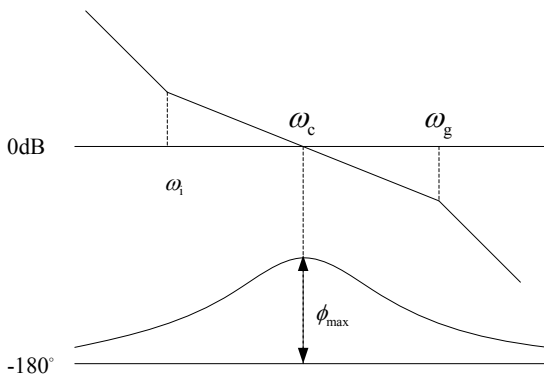


图 2-21 闭环系统 $K(s)G(s)$ 的相位裕度 (对称整定法)

(2) 参数 α 与最大相位裕度的关系。

根据典型环节的频率特性可知：积分环节 $\frac{1}{s}$ 的相频为 $-\frac{\pi}{2}$ ；一阶惯性环节 $\frac{1}{s+1}$ 的相频为 $-\arctan T\omega$ ；一阶微分环节 $\tau s + 1$ 的相频为 $\arctan \tau\omega$ 。



则 $G(s) = \frac{K_g}{s(T_g s + 1)}$ 的相位为 $\varphi_G(\omega) = -\frac{\pi}{2} - \arctan T_g \omega$; PI 控制器

$$K(s) = K_p \left(1 + \frac{1}{T_i s} \right) = K_p \frac{1}{T_i} \frac{1}{s} (T_i s + 1) \text{ 的相位为 } \varphi_K(\omega) = -\frac{\pi}{2} + \arctan T_i \omega。$$

根据相位裕度的定义, 可得闭环系统 $G(s)K(s)$ 的相位裕度为:

$$\begin{aligned} \varphi_{GK}(\omega) &= \varphi_G(\omega) + \varphi_K(\omega) - (-\pi) \\ &= -\frac{\pi}{2} - \arctan T_g \omega - \frac{\pi}{2} + \arctan T_i \omega - (-\pi) \\ &= -\arctan T_g \omega + \arctan T_i \omega \end{aligned}$$

闭环系统 $G(s)K(s)$ 的最大相位裕度为:

$$\begin{aligned} \phi_m &= \varphi_{GK}(\omega_c) = -\arctan T_g \omega_c + \arctan T_i \omega_c \\ &= -\arctan \frac{\omega_c}{\omega_g} + \arctan \frac{\omega_c}{\omega_i} = -\arctan \frac{1}{\sqrt{\alpha}} + \arctan \sqrt{\alpha} \end{aligned}$$

根据 $\tan(\alpha - \beta) = \frac{\tan \alpha - \tan \beta}{1 + \tan \alpha \tan \beta}$, 则

$$\tan \phi_m = \tan \left(\arctan \sqrt{\alpha} - \arctan \frac{1}{\sqrt{\alpha}} \right) = \frac{\sqrt{\alpha} - \frac{1}{\sqrt{\alpha}}}{1 + \sqrt{\alpha} \frac{1}{\sqrt{\alpha}}} = \frac{\sqrt{\alpha} - \frac{1}{\sqrt{\alpha}}}{2} = \frac{\alpha - 1}{2\sqrt{\alpha}}$$

由 $\tan \phi_m = \frac{\sin \phi_m}{\cos \phi_m}$ 代入, 整理得

$$\alpha \cos \phi_m - 2\sqrt{\alpha} \sin \phi_m - \cos \phi_m = 0$$

令 $x = \sqrt{\alpha}$, 则上式写为

$$x^2 \cos \phi_m - 2x \sin \phi_m - \cos \phi_m = 0 \quad (2.22)$$

解方程, 得

$$x_{1,2} = \frac{2 \sin \phi_m \pm \sqrt{4 \sin^2 \phi_m - 4 \cos \phi_m (-\cos \phi_m)}}{2 \cos \phi_m} = \frac{\sin \phi_m \pm 1}{\cos \phi_m}$$

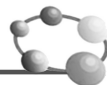
即方程 (2.22) 的解为 $x_1 = \frac{\sin \phi_m + 1}{\cos \phi_m}$, $x_2 = \frac{\sin \phi_m - 1}{\cos \phi_m}$ 。由于 $x = \sqrt{\alpha} > 0$, 当取最大相位

裕度为 $\phi_m = 60^\circ$ 时, $x_2 = \frac{\sin \phi_m - 1}{\cos \phi_m}$ 不成立, 故方程 (2.22) 的唯一解为 $x_1 = \frac{\sin \phi_m + 1}{\cos \phi_m}$, 即

$\sqrt{\alpha} = \frac{\sin \phi_m + 1}{\cos \phi_m}$, 从而可得相位裕度 ϕ_m 与参数 α 的关系为

$$\alpha = \left(\frac{1 + \sin \phi_m}{\cos \phi_m} \right)^2 \quad (2.23)$$

因此, 在进行 PI 整定时, 可以根据控制系统所要求的 ϕ_m 来计算 α , 并通过式 (2.21) 和式 (2.18) 可求得 PI 控制的参数 K_p 和 ω_i 。



2.5.2 仿真实例

被控对象为

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

对比式 (2.14), 可得 $K_g = 133/25$, $T_g = 1/25$, $\omega_g = \frac{1}{T_g} = 25$ 。

取最大相位裕度为 $\phi_m = 60$, 根据式 (2.23) 得 $\alpha = \left(\frac{1 + \sin \phi_m}{\cos \phi_m} \right)^2 = 13.9282$ 。根据式 (2.17)

至式 (2.21) 得: $\omega_c = \omega_g / \sqrt{\alpha} = 6.6987$, $\omega_i = \omega_g / \alpha = 1.7949$, $K_p = \omega_c / K_g = 1.2592$ 。 $K(s)G(s)$ 的 Bode 图如图 2-22 所示, 可见, 所设计的方法符合对称整定法, 闭环系统阶跃响应如图 2-23 所示。

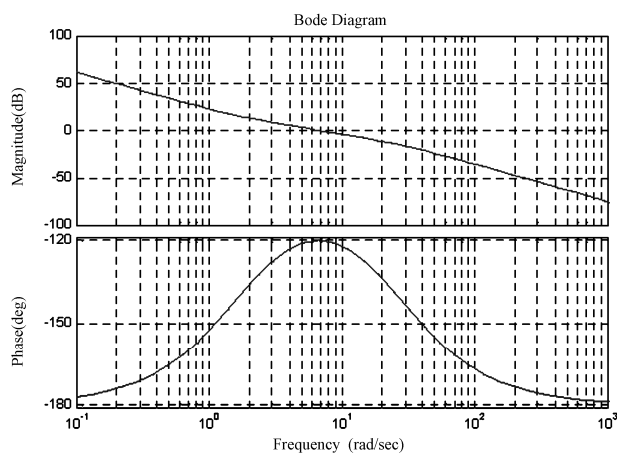


图 2-22 $K(s)G(s)$ 的 Bode 图 (对称整定法)

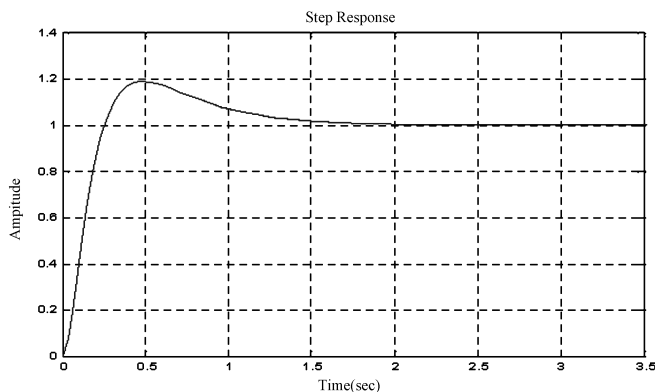


图 2-23 闭环系统阶跃响应

仿真程序: chap2_7.m

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




```

G=tf(133,[1 25 0]);
Kg=133/25;
Tg=1/25;
wg=1/Tg;

ph_max=60*pi/180;    %Designed maximum phase margin
alfa=((1+sin(ph_max))/cos(ph_max))^2;

wc=wg/sqrt(alfa);
wi=wg/alfa;
Kp=wc/Kg;

K=Kp*tf([1 wi],[1 0]);

figure(1);
bode(G*K);

Gc=K*G/(1+K*G);
figure(2);
step(Gc);

```



2.6 基于极点配置的稳定 PD 控制

2.6.1 基本原理

本节介绍一种基于精确模型极点配制的 PD 控制器设计方法。

假设被控对象为一电动机模型传递函数

$$G(s) = \frac{133}{s^2 + 25s} \quad (2.24)$$

取 $a = 25$ ， $b = 133$ ，则被控对象可表示为

$$\ddot{\theta} = -a\dot{\theta} + bu$$

式中， θ 为位置信号， u 为控制输入。

假设理想位置指令为 r ，且跟踪误差为 $e = \theta - r$ ，则有 $\dot{\theta} = \dot{e} + \dot{r}$ ， $\ddot{\theta} = \ddot{e} + \ddot{r}$ ，则被控对象可写为

$$\ddot{e} + \ddot{r} = -a(\dot{e} + \dot{r}) + bu$$

将控制律设计为 PD 控制+前馈的形式，即

$$u = \frac{1}{b}(-h_1 e - h_2 \dot{e} + a\dot{r} + \ddot{r}) \quad (2.25)$$

将控制律代入式 (2.24)，可得闭环系统

$$\ddot{e} + \ddot{r} = -a(\dot{e} + \dot{r}) + (-h_1 e - h_2 \dot{e} + a\dot{r} + \ddot{r})$$



整理得

$$\ddot{e} + (h_2 + a)\dot{e} + h_1 e = 0$$

为了使闭环系统稳定，需要满足 $s^2 + (h_2 + a)s + h_1$ 为 Hurwitz，即需要使 $s^2 + (h_2 + a)s + h_1 = 0$ 的特征根为负实部。

对于 $k > 0$ ，取特征根为 $-k$ ，由 $(s + k)^2 = 0$ 可得 $s^2 + 2ks + k^2 = 0$ ，从而可设计 $h_2 + a = 2k$ ， $h_1 = k^2$ ，即 $h_2 = 2k - a$ ， $h_1 = k^2$ 。因此，可以通过 k 的设计得到 h_1 和 h_2 ，从而实现控制律式 (2.25) 的设计。

2.6.2 仿真实例

被控对象为式 (2.24)，控制律采用式 (2.25)，取对于 $k = 3$ ，则 $h_2 = 2k - a = -19$ ， $h_1 = 9$ 。仿真结果如图 2-24 所示。

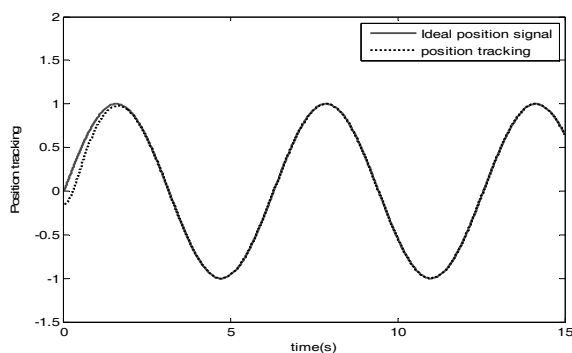


图 2-24 正弦跟踪

仿真程序

Simulink 主程序: chap2_8sim.mdl (见图 2-25)

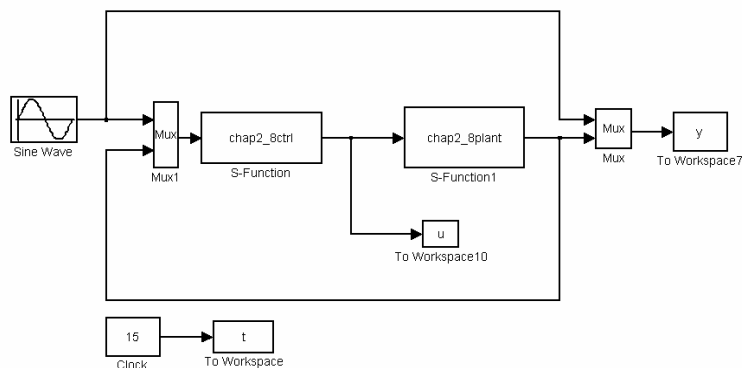


图 2-25 极点配置 PD 控制主程序

控制器子程序: chap2_8ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
```



```
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
yd=u(1);
dyd=cos(t);
ddy=-sin(t);

th=u(2);
dth=u(3);

e=th-yd;
de=dth-dyd;

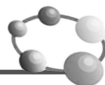
a=25;
b=133;

k=3;
h1=k^2;
h2=2*k-a;

ut=1/b*(-h1*e-h2*de+a*dyd+ddy);
sys(1)=ut;
```

被控对象子程序: chap2_8plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
```



```

case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[-0.15 -0.15];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-25*x(2)+133*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);

```

作图子程序: chap2_8plot.m

```

close all;

figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
axis([0 15 -1.5 2]);
legend('Ideal position signal','position tracking','location','NorthEast');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```



2.7 基于临界比例度法的 PID 整定

2.7.1 基本原理

考虑如下被控对象



$$G_p(s) = \frac{1}{T_p s + 1} e^{-\tau s} \quad (2.26)$$

PID 控制算法为

$$u(t) = \frac{1}{\delta} \left(e + \frac{1}{T_I} \int_0^t e dt + T_D \frac{de}{dt} \right) \quad (2.27)$$

步骤如下:

(1) 先采用比例控制, 从较大的比例度 δ 开始, 逐步减小比例度, 使系统对阶跃输入的响应达到临界振荡状态。将此时的比例度记作 δ_r , 临界振荡周期记作 T_r 。临界振荡曲线如图 2-26 所示。

(2) 根据 Ziegler-Nichols 提供的临界比例度法经验公式确定 PID 控制器参数^[19], 见表 2-2。本方法适用于有自平衡能力的被控对象。

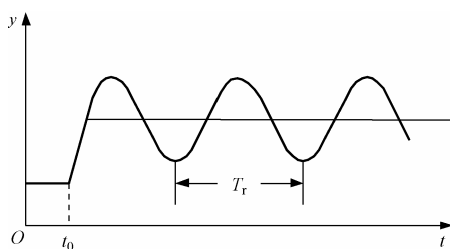


图 2-26 临界振荡曲线

表 2-2 临界比例度法整定 PID 参数

控制器类型	比例度 $\delta\%$	积分时间 T_i	微分时间 T_d
P	$2\delta_r$		
PI	$2.2\delta_r$	$0.85T_r$	
PID	$1.7\delta_r$	$0.5T_r$	$0.13T_r$

2.7.2 仿真实例

设被控对象为

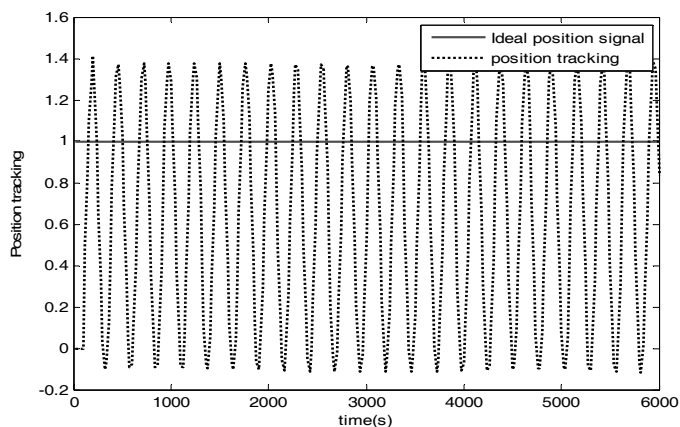
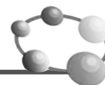
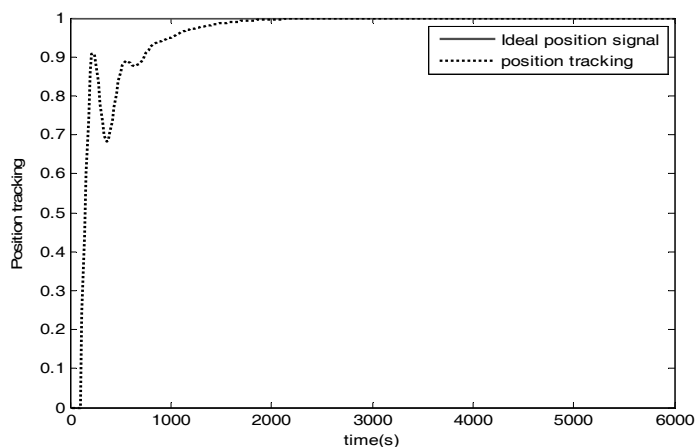
$$G_p(s) = \frac{1}{60s + 1} e^{-80s}$$

临界比例度法整定分成两步:

(1) 采样周期取为 $T_s = 20$, 将被控对象和 PID 控制器离散化, 首先采用纯比例控制 (程序中取 $M=1$), 取 $\delta_r = 0.575$, 使系统对阶跃输入的响应达到临界振荡状态, 如图 2-22 所示, 这是一个反复测试的过程。

(2) 然后按照临界比例度法, 根据图 2-27 可得到 $T_r = 500 - 200 = 300$ 。

(3) 根据表 2-2, 采用 PI 控制算法, 可计算得到 K_p 、 T_i 的值, 程序中取 $M=3$, 仿真结果如图 2-28 所示。

图 2-27 等幅振荡曲线 ($M=1$)图 2-28 闭环 PID 控制单位阶跃响应曲线 ($M=3$)

仿真程序: chap2_9.m

```
clear all;
close all;
Ts=20;

%Delay plant
deltar=0.575;
Tr=1000/4; %From 4000 to 5000

kp=1;
Tp=60;
tol=80;
sys=tf([kp],[Tp,1],'inputdelay',tol);
dsys=c2d(sys,Ts,'zoh');
[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;
e_1=0;
ei=0;
```



```
y_1=0.0;
for k=1:1:300
    time(k)=k*Ts;

    yd(k)=1.0;    %Tracing Step Signal

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

    e(k)=yd(k)-y(k);
    de(k)=(e(k)-e_1)/Ts;
    ei=ei+Ts*e(k);

    u(k)=1/deltar*e(k);

    M=1;
    if M==1    %Critical testing
        delta=deltar;
        u(k)=1/delta*e(k);
    elseif M==2    %P
        delta1=2*deltar;
        u(k)=1/delta1*e(k);
    elseif M==3    %PI
        delta2=2.2*deltar;
        TI2=0.85*Tr;
        u(k)=1/delta2*(e(k)+1/TI2*ei);
    elseif M==4    %PID
        delta3=1.7*deltar;
        TI3=0.5*Tr;
        TD3=0.13*Tr;
        u(k)=1/delta3*(e(k)+1/TI3*ei+TD3*de(k));
    end
    e_1=e(k);
    u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
    y_1=y(k);
end
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position signal','position tracking','location','NorthEast');
```



2.8 一类非线性整定的 PID 控制

2.8.1 基本原理

设图 2-29 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,实现 PID 的三个调节参数在一定范围内的整定。

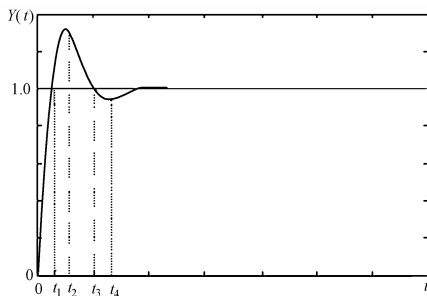
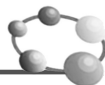


图 2-29 一般系统阶跃响应曲线

肖永利等^[24]利用图 2-29, 对非线性 PID 整定进行了如下分析:

(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 2-30 (a) 所示, 根据该图可以构造如下非线性函数

$$k_p(e_p(t)) = a_p + b_p(1 - \text{sech}(c_p e_p(t))) \quad (2.28)$$

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 2-30 (b) 所示, 所构造的非线性函数为

$$k_d(e_p(t)) = a_d + b_d / (1 + c_d \exp(d_d \cdot e_v(t))) \quad (2.29)$$

式中, $e_v = \dot{e}_p$ 为误差变化速率, a_d, b_d, c_d, d_d 为正实常数, 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, 以防止响应产生振荡, 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 2-30 (c) 所示, 其非线性函数可表示为

$$k_i(e_p(t)) = a_i \text{sech}(c_i e_i(t)) \quad (2.30)$$

式中, $k_i(e_p(t)) = a_i \text{sech}(c_i e_i(t))$ 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

$$u(t) = k_p(e_p(t))e_p(t) + k_i(e_p(t)) \int_0^t e_p(t) dt + k_d(e_p(t), e_v(t)) \frac{de_p(t)}{dt} \quad (2.31)$$

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2-30所示。

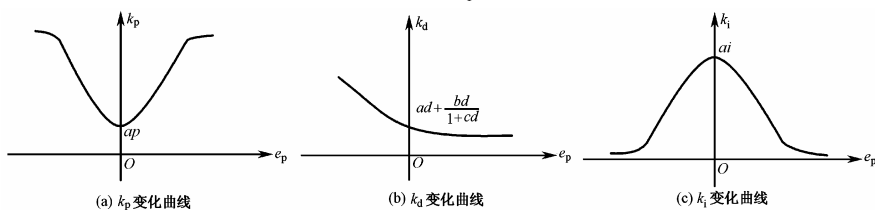


图2-30 非线性增益调节参数变化曲线

2.8.2 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。

针对阶跃进行仿真,阶跃响应如图2-31所示,其中 k_p, k_i, k_d 随偏差的变化曲线如图2-32所示, k_p, k_i, k_d 随时间的变化曲线如图2-33所示。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。

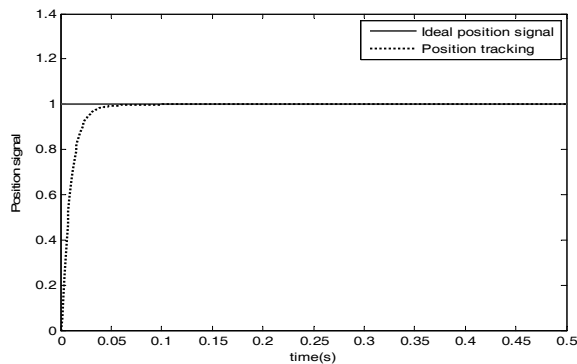
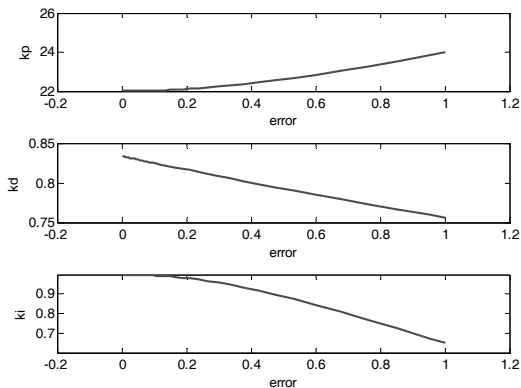
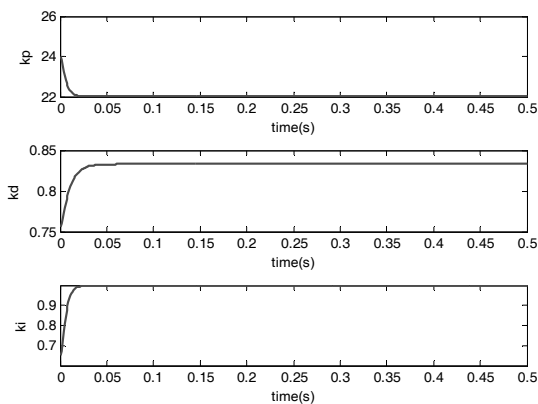
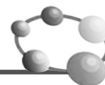


图2-31 阶跃响应

图2-32 k_p, k_i, k_d 随偏差的变化曲线

图 2-33 k_p, k_i, k_d 随时间的变化曲线

仿真程序: chap2_10.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');

u_1=0;u_2=0;
y_1=0;y_2=0;
error_1=0;
ei=0;
for k=1:1:500
time(k)=k*ts;

yd(k)=1.0;
y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=yd(k)-y(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
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=y(k);
error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('Position signal');
legend('Ideal position signal','Position tracking','location','NorthEast');
figure(2);
subplot(311);
plot(error,kp,'r','linewidth',2);xlabel('error');ylabel('kp');
subplot(312);
plot(error,kd,'r','linewidth',2);xlabel('error');ylabel('kd');
subplot(313);
plot(error,ki,'r','linewidth',2);xlabel('error');ylabel('ki');
figure(3);
subplot(311);
plot(time,kp,'r','linewidth',2);xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,kd,'r','linewidth',2);xlabel('time(s)');ylabel('kd');
subplot(313);
plot(time,ki,'r','linewidth',2);xlabel('time(s)');ylabel('ki');
```

2.9 基于优化函数的 PID 整定

2.9.1 基本原理

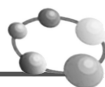
采用 MATLAB 优化工具箱提供的各类函数可实现 PID 的整定。一种整定方法为^[70]：利用 MATLAB 非线性最小平方函数 `lsqnonlin()`，按照最小平方指标 $J = \int e^2 dt$ 进行 PID 参数寻优，得到优化的 k_p 、 k_i 、 k_d ，实现 PID 的整定。

2.9.2 仿真实例

被控对象为

$$G(s) = \frac{50s + 50}{s^3 + s^2 + s}$$

控制输入限制在 $[-5, 5]$ ，分两步来实现 PID 的整定：首先运行主程序 `chap2_11main.m`，初始化的参数为 $k_p = 0$ 、 $k_i = 0$ 、 $k_d = 0$ ，上下界分别取为 $LB = [0 \ 0 \ 0]$ 、 $UB = [100 \ 100 \ 100]$ ，在运行过程中可通过 Simulink 程序中的 `scope` 窗口观察到动态优化过程，优化结果为 $k_p = 1.3814$ 、 $k_i = 0.0014$ 、 $k_d = 0.1787$ ；然后采用优化后的 PID 参数运行 Simulink 程序



chap2_11sim.mdl，优化后的阶跃响应如图 2-34 所示。

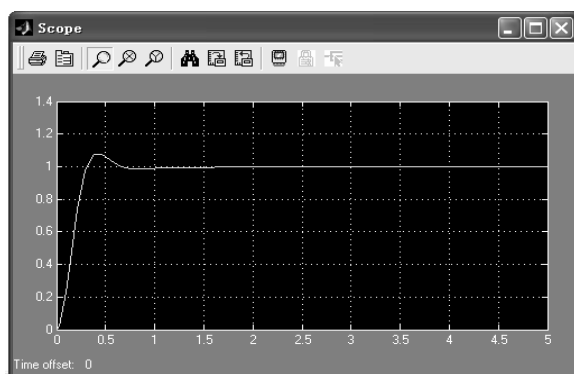


图 2-34 采用优化函数整定的阶跃响应

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：chap2_11main.m

```
clear all;
close all;
K_pid0=[0 0 0];
LB=[0 0 0];
UB=[100 100 100];
K_pid=lsqnonlin('chap2_11plant',K_pid0,LB,UB)
chap2_11sim
```

M 函数子程序：chap2_11plant.m

```
function e=pid_eq(K_pid)
assignin('base','kp',K_pid(1));
assignin('base','ki',K_pid(2));
assignin('base','kd',K_pid(3));
opt=simset('solver','ode5');
[tout,xout,y]=sim('chap2_11sim',[0 10],opt);
r=1.0;
e=r-y;
```

Simulink 子程序：chap2_11sim.mdl（见图 2-35）

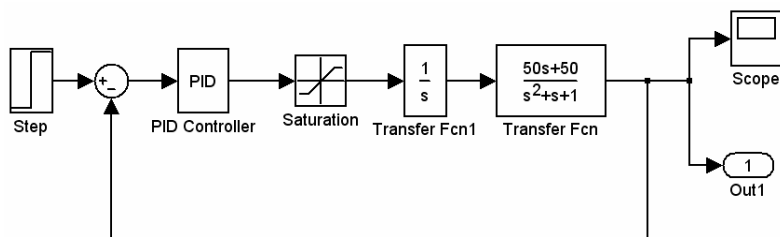
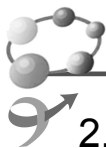


图 2-35 Simulink 子程序



2.10 基于 NCD 优化的 PID 整定

2.10.1 基本原理

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 Matlab 6.5 版本下的 NCD 工具箱可实现 PID 参数的优化设计^[69]。

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。

2.10.2 仿真实例

被控对象传递函数为

$$G(s) = \frac{1.5}{50s^2 + a_2s^2 + a_1s + 1}$$

式中, $a_2 = 43, a_1 = 3$ 。

系统包含控制输入饱和环节 ± 1.0 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 $40 \sim 50$ 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 20s;
- (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。

基于 NCD 的 PID 控制器优化步骤如下:

第一步, 参数初始化, 首先运行初始化程序 chap2_12int.m, 实现 PID 控制算法中 k_p, k_i, k_d 和被控对象 a_2, a_1 的初始化;

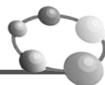
第二步, 运行 Simulink 主程序 chap2_12sim.mdl, 可得到初始 k_p, k_i, k_d 下的 PID 控制响应结果。

第三步, 打开 NCD 环境, 如果是第一次 NCD 优化, 则需要进行参数的初始化, 否则可调用以前的优化参数文件 *.mat;

第四步, 单击 Start 功能, 实现 k_p, k_i, k_d 的优化, 优化完成后, 可在 Matlab 环境下得到 k_p, k_i, k_d 的优化结果;

第五步, 再运行 Simulink 主程序 chap2_12sim.mdl, 可得到优化 k_p, k_i, k_d 下的 PID 控制响应结果。每次优化结果需要通过 NCD 环境下的 Save 功能保存在 *.mat 文件中,

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

① 通过 Step Response 命令定义阶跃响应性能指标，如图 2-36 所示。

Settling time: 调整时间，选 30s
 Rise time: 上升时间，选 20s
 Percent settling: 稳态误差百分数，取 5
 Percent overshoot: 超调量百分数，取 20
 Percent 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

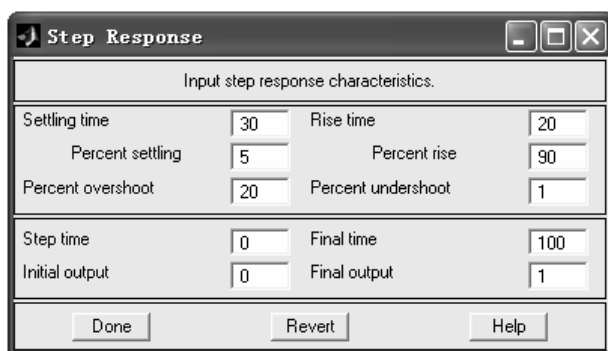


图 2-36 Step Response 设置图

② 通过 Time range 命令，设置优化时间，取 0~100s。

③ 通过选择 Y-Axis 命令，设置阶跃响应范围，取 $[-0.131 \ 1.321]$ 。

（2）Optimization 菜单的使用

① 选择 Parameters 项，定义调整变量及有关参数，如图 2-37 所示。

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

下限：0 0 0

上限：100 100 100

变量允差：0.001 约束允差：0.001

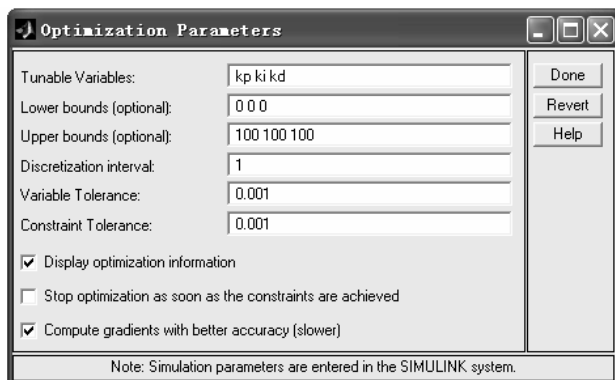


图 2-37 Optimization Parameters 设置图



② 选择 Uncertainty 项, 定义不确定变量及有关参数, 如图 2-38 所示。

输入: 不确定变量 a_1, a_2 的上下限, 可取为

下限: 1.5 40

上限: 6.0 50

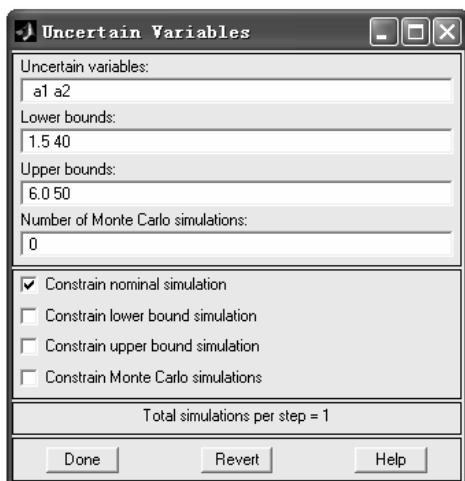


图 2-38 Uncertain Variables 设置图

③ 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。在 NCD 优化过程中, 如果将性能指标设计得过高, 如上升时间取得过小 (比如取 10s), 可能会造成死机。通过调整 NCD 优化边界, 可实现优化指标的调整, 从而可得到更好或更合理的优化结果。阶跃响应性能限制可以直接由鼠标在 NCD Blockset 约束窗口设置。

每次优化结果需要通过 NCD 环境下的 Save 功能保存在*.mat 文件中, 例如可命名为 chap2_12save.mat, 以供下一次调用。

初始 PID 参数为 $k_p=1.0, k_i=0.10, k_d=10$, NCD 优化参数如下:
 $k_p=1.477, k_i=0.0858, k_d=9.4283$, 优化过程及优化前后的响应曲线如图 2-39 至图 2-41 所示。

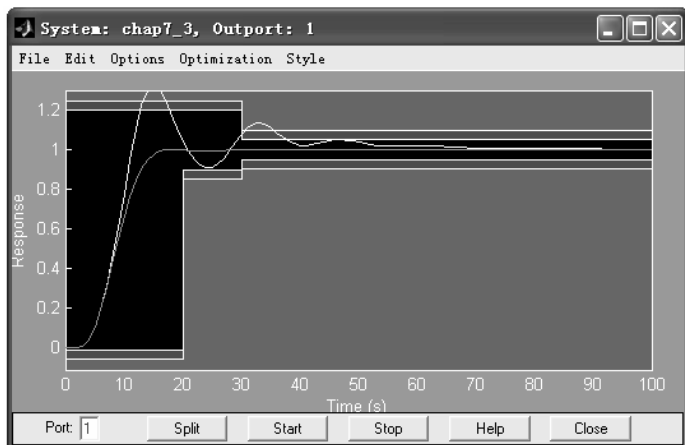


图 2-39 NCD 优化过程界面

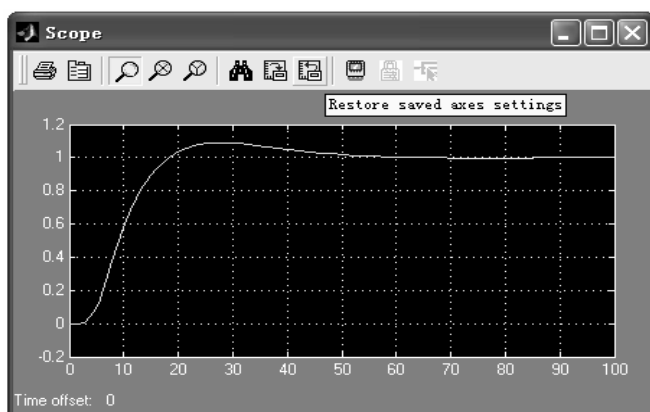
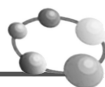


图 2-40 NCD 优化前阶跃响应曲线

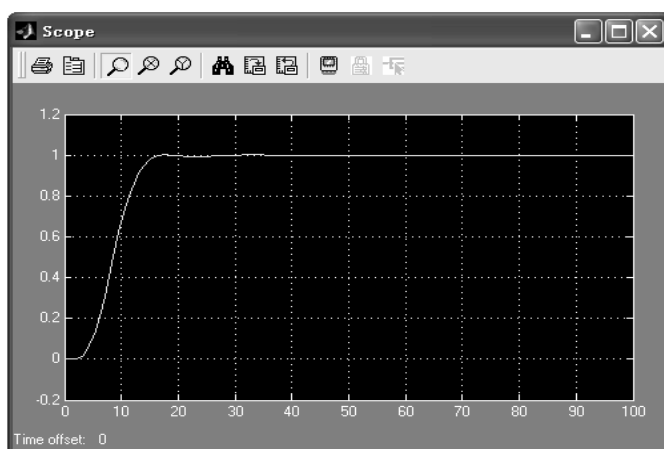


图 2-41 优化后阶跃响应曲线

(1) 初始化程序: chap2_12int.m

```
clear all;
close all;

kp=1;ki=0.10;kd=10;
a2=43;
a1=3;
```

(2) Simulink 主程序: chap2_12sim.mdl (见图 2-42)

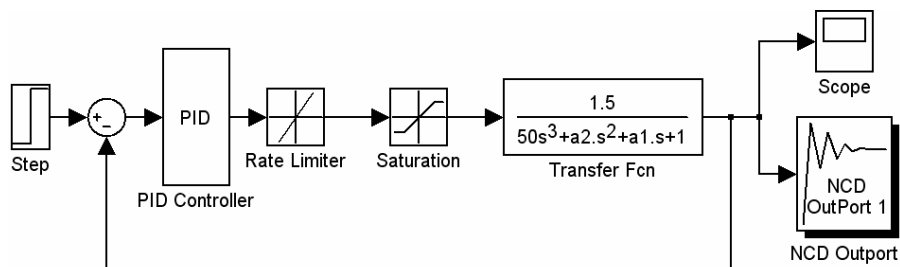
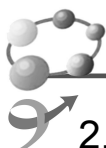


图 2-42 基于 NCD 优化的 Simulink 主程序



2.11 基于 NCD 与优化函数结合的 PID 整定

2.11.1 基本原理

单纯采用 NCD 优化 PID 控制参数,如果初始值选择不当,会造成 NCD 无法运行。可采用其他优化方法先确定 PID 控制器的参数初始值,然后再用 NCD 优化。本节采用 MATLAB 6.5 版本下的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。

分两步来实现 PID 的整定^[70]:首先,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,得到较优的 k_p, k_i, k_d ;然后,在此 k_p, k_i, k_d 基础上再进行 NCD 优化,从而得到最终的 k_p, k_i, k_d ,实现 PID 的整定。

2.11.2 仿真实例

被控对象为

$$G(s) = \frac{50s + 50}{s^3 + s^2 + s}$$

分两步来实现 PID 的整定:首先运行主程序 chap2_13main.m,初始化的参数为 $k_p = 0, k_i = 0, k_d = 0$,上下界分别取为 $LB = [0 \ 0 \ 0], UB = [100 \ 100 \ 100]$ 。优化结果为 $k_p = 1.0592, k_i = 0.0185, k_d = 0.1427$,优化后的阶跃响应如图 2-43 所示;然后采用优化参数运行 Simulink 程序 chap2_13sim.mdl,在 NCD 环境下进行 PID 再优化, k_p, k_i, k_d 优化范围设置如图 2-44 所示。优化过程中通过鼠标不断调整 NCD 响应指标,使阶跃响应性能指标尽量优化,经过多次调整得优化结果为 $k_p = 9.7254, k_i = 0.0011, k_d = 4.7124$,优化前后的阶跃响应如图 2-45 和图 2-46 所示。

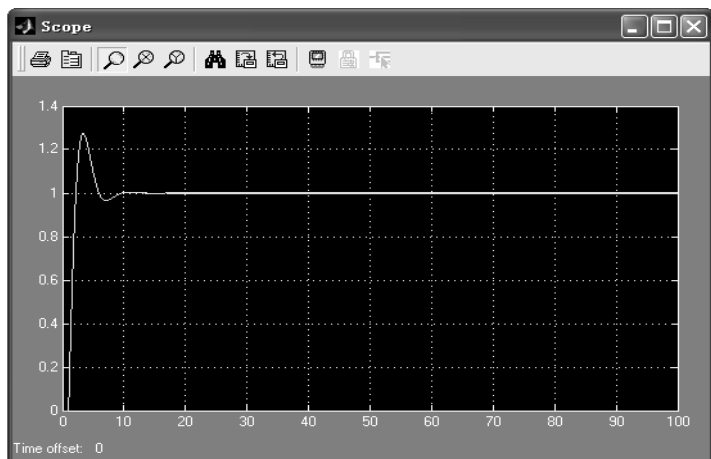


图 2-43 采用优化后函数的阶跃响应

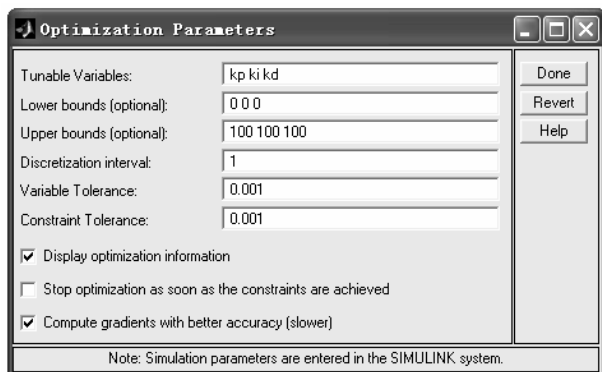


图 2-44 Optimization Parameters 设置图

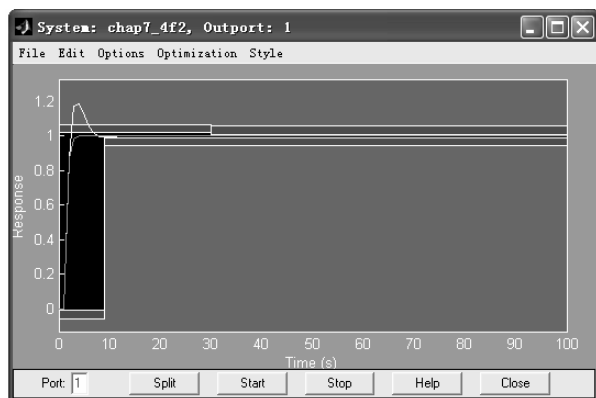


图 2-45 NCD 优化前的界面

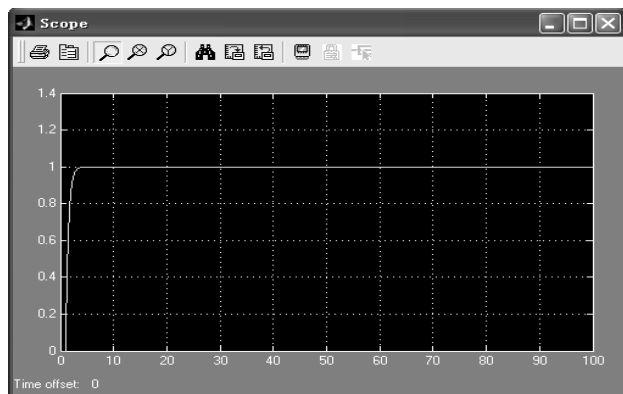


图 2-46 NCD 优化后的阶跃响应

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：chap2_13main.m

```
clear all;
close all;
K_pid0=[0 0 0];
LB=[0.1 0.0 0.0];
UB=[100 100 100];
K_pid=lsqnonlin('chap2_13eq',K_pid0,LB,UB)
```



```

chap2_13sim
M 函数子程序: chap2_13eq.m
function e=pid_eq(K_pid)
assignin('base','kp',K_pid(1));
assignin('base','ki',K_pid(2));
assignin('base','kd',K_pid(3));
opt=simset('solver','ode5');
[tout,xout,y]=sim('chap2_13sim',[0 10],opt);
r=1.0;
e=r-y;

```

Simulink 子程序: chap2_13sim.mdl (见图 2-47)

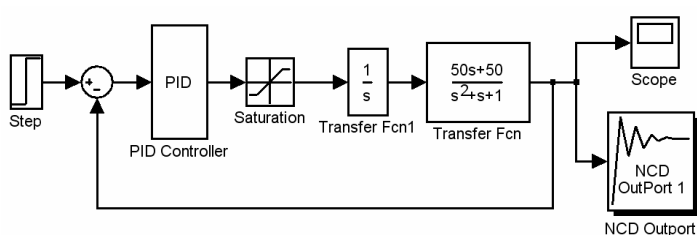


图 2-47 基于 NCD 优化的 Simulink 子程序

2.12 传递函数的频域测试

本章中有几种 PID 整定算法需要对象的精确模型, 如基于极点配置 PD 整定方法。下面介绍一种工程上常用的对象精确模型测试法—频率测试方法。

2.12.1 基本原理

可通过扫频测试得到对象的传递函数, 假设模型为 $G_p(s)$, 如图 2-48 所示。

设开环系统输入指令信号为

$$u(t) = A_m \sin(\omega t) \quad (2.32)$$

式中, A_m 、 ω 分别为输入信号的幅度和角速度。

假设开环系统是线性的, 则其位置输出可表示为

$$\begin{aligned}
 y(t) &= A_f \sin(\omega t + \varphi) \\
 &= A_f \sin(\omega t) \cos(\varphi) + A_f \cos(\omega t) \sin \varphi \\
 &= [\sin(\omega t) \quad \cos(\omega t)] \begin{bmatrix} A_f \cos \varphi \\ A_f \sin \varphi \end{bmatrix}
 \end{aligned} \quad (2.33)$$

式中, A_f 、 φ 分别为开环系统输出的幅度和相位。

在时间域上取 $t = 0, h, 2h, \dots, nh$, 并设

$$Y^T = [y(0) \quad y(h) \quad \dots \quad y(nh)]$$

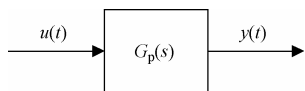
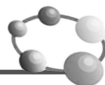


图 2-48 开环传递函数测试框图



$$\Psi^T = \begin{bmatrix} \sin(w0) & \sin(wh) & \cdots & \sin(wnh) \\ \cos(w0) & \cos(wh) & \cdots & \cos(wnh) \end{bmatrix} \quad (2.34)$$

$$c_1 = A_f \cos \varphi, \quad c_2 = A_f \sin \varphi$$

由式 (2.32) 和式 (2.33) 得

$$Y = \Psi \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \quad (2.35)$$

由式 (2.35), 根据最小二乘原理, 可求出 c_1 、 c_2 的最小二乘解为

$$\begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \end{bmatrix} = (\Psi^T \Psi)^{-1} \Psi^T Y \quad (2.36)$$

根据测得的 $\hat{c}_1$ 和 $\hat{c}_2$, 输出信号的振幅和相位估计值如下

$$\hat{A}_f = \sqrt{\hat{c}_1^2 + \hat{c}_2^2} \quad (2.37)$$

$$\hat{\varphi} = \text{tg}^{-1} \left(\frac{\hat{c}_2}{\hat{c}_1} \right) \quad (2.38)$$

相频为输出信号与输入信号相位之差, 幅频为稳态输出振幅与输入振幅之比的分贝表示。由于输入信号 $u(t) = A_m \sin(\omega t)$ 的相移为零, 则开环传递函数的相频和幅频估计值为

$$\hat{\varphi}_e = \varphi_{\text{out}} - \varphi_{\text{in}} = \hat{\varphi} - 0 = \arctan \left(\frac{\hat{c}_2}{\hat{c}_1} \right) \quad (2.39)$$

$$\hat{M} = 20Lg \left(\frac{\hat{A}_f}{A_m} \right) = 20Lg \left(\frac{\sqrt{\hat{c}_1^2 + \hat{c}_2^2}}{A_m} \right) \quad (2.40)$$

在待测量的频率段取角频率序列 $\{\omega_i\} \quad i = 0, 1, \dots, n$, 对每个角频率点, 用上面方法计算相频和幅频, 就可得到开环传递函数的频率特性数据, 从而得到模型的传递函数。

2.12.2 仿真实例

取模型为

$$G_p(s) = \frac{133}{s^2 + 25s + 10}$$

采样周期取 1ms, 即 $h = 0.001$ 。输入信号为正弦信号 $u(t) = 0.5 \sin(2\pi Ft)$, 频率 F 的起始频率为 1Hz, 终止频率为 10Hz, 步长为 0.5Hz, 对每个频率点, 运行 20000 个采样时间, 并记录采样区间为 [10000, 15000] 的数据。

求出实际模型在各个频率点的相频和幅频后, 可写出开环传递函数频率特性的复数表示, 即 $h_p = M(\cos \varphi_e + j \sin \varphi_e)$ 。由于 $w = 2\pi F$, 利用 MATLAB 函数 $\text{invfreqs}(h_p, w, nb, na)$, 可得到与复频特性 h_p 相对应的、分子分母阶数分别为 nb 和 na 的传递函数的分子分母系数 bb 和 aa , 从而得到逼近的开环传递函数, 表示为

$$\hat{G}_p(s) = \frac{131.3}{s^2 + 24.28s + 10.08}$$

拟合传递函数与实际传递函数的 Bode 图比较, 如图 2-49 所示。可见, 采用该算法能精确地求出对象的传递函数。

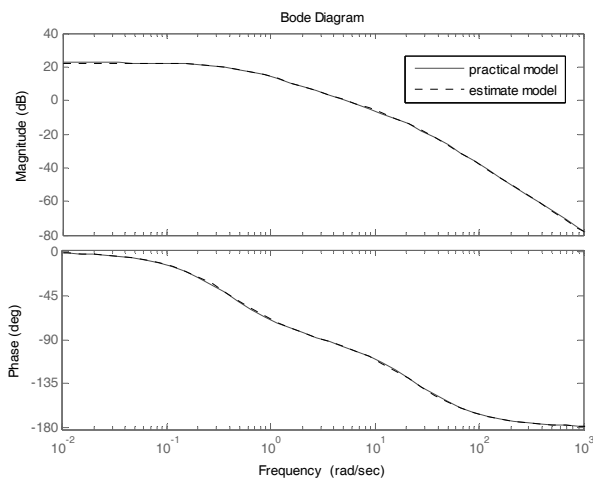


图 2-49 拟合传递函数与实际传递函数的 Bode 图比较

仿真程序: chap2_14.m

```
%Transfer function identification with frequency test
clear all;
close all;

ts=0.001;
a=25;b=133;c=10;
sys=tf(b,[1,a,c]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

Am=0.5;
kk=0;

for F=1:0.5:10
    kk=kk+1;
    FF(kk)=F;

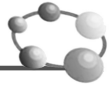
    u_1=0.0;u_2=0.0;
    y_1=0;y_2=0;

    for k=1:1:20000
        time(k)=k*ts;

        u(k)=Am*sin(1*2*pi*F*k*ts);          % Sine Signal with different frequency
        y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

        u_2=u_1;u_1=u(k);
        y_2=y_1;y_1=y(k);
    end

    plot(time,u,'r',time,y,'b');    %Dynamic Simulation
    pause(0.2);
```



```

for i=10001:1:15000
    fai(1,i-10000) = sin(2*pi*F*i*ts);
    fai(2,i-10000) = cos(2*pi*F*i*ts);
end
Fai=fai';

fai_in(kk)=0;

Y_out=y(10001:1:15000)';
cout=inv(Fai'*Fai)*Fai'*Y_out;
fai_out(kk)=atan(cout(2)/cout(1));    % Phase Frequency(Deg.)

if fai_out(kk)>0
    fai_out(kk)=fai_out(kk)-pi;
end

Af(kk)=sqrt(cout(1)^2+cout(2)^2);    % Magnitude Frequency(dB)
mag_e(kk)=20*log10(Af(kk)/Am);    % in dB.
ph_e(kk)=(fai_out(kk)-fai_in(kk))*180/pi; % in Deg.

if ph_e(kk)>0
    ph_e(kk)=ph_e(kk)-360;
end
end

FF=FF';
%%%%%%%%%%%%%% Closed system modelling
mag_e1=Af/Am;    %From dB.to ratio
ph_e1=fai_out'-fai_in'; %From Deg. to rad

hp=mag_e1.*(cos(ph_e1)+j*sin(ph_e1)); %Practical frequency response vector

na=2;    % Second order transfer function
nb=0;

w=2*pi*FF;    % in rad./s
% bb and aa gives real numerator and denominator of transfer function
[bb,aa]=invfreqs(hp,w,nb,na);    % w(in rad./s) contains the frequency values

G=tf(bb,aa)    % Transfer function fitting

figure(1);
bode(sys,'r','G','k');
legend('practical model','estimate model');

```

第3章 时滞系统的PID控制

3.1 单回路PID控制系统

系统只有一个PID控制器，如图3-1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。

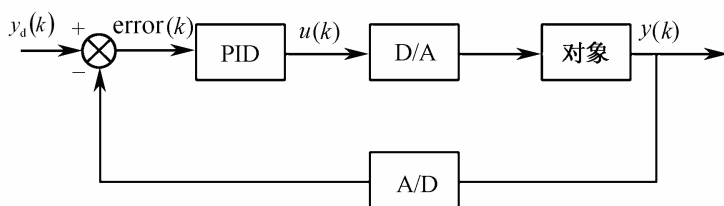


图3-1 单回路PID控制系统

单回路PID控制系统的MATLAB仿真见第1章。

3.2 串级PID控制

3.2.1 串级PID控制原理

串级计算机控制系统的典型结构如图3-2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 $R_2(s)$ 。

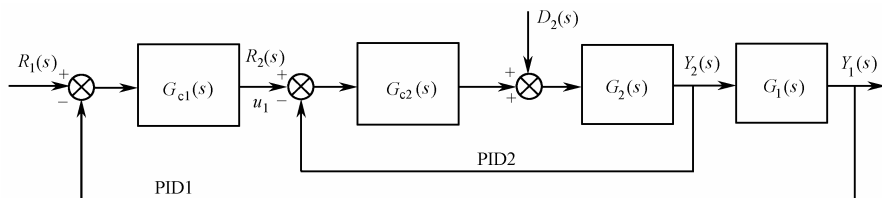
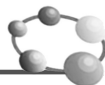


图3-2 串级计算机控制系统的典型结构

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：^[68]

- (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_1 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

$$\varphi_2(z) = \frac{Y_2(z)}{U_1(z)} = \frac{G_{c2}(z)G_2(z)}{1 + G_{c2}(z)G_2(z)} \quad (3.1)$$

可得副调节器控制律

$$G_{c2}(z) = \frac{\varphi_2(z)}{G_2(z)(1 - \varphi_2(z))} \quad (3.2)$$

一般选择

$$\varphi_2(z) = z^{-n} \quad (3.3)$$

式中， n 为 $G_2(z)$ 有理多项式分母最高次幂。

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

仿真方法一

在离散方式下进行仿真，采用 M 语言进行编程。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 (3.3) 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 (3.2) 设计。副回路输入、输出，阶跃响应结果及外加干扰信号如图 3-3～图 3-5 所示。

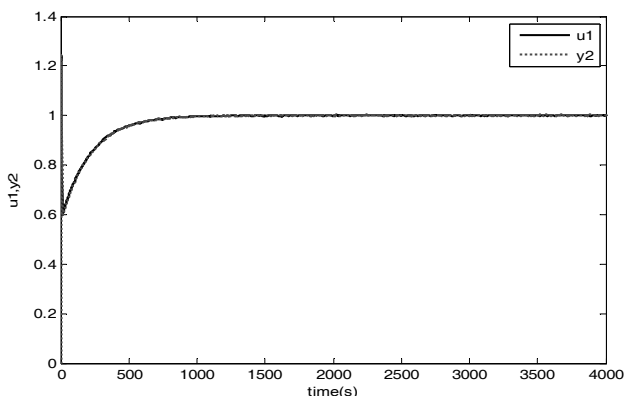


图 3-3 副回路输入、输出

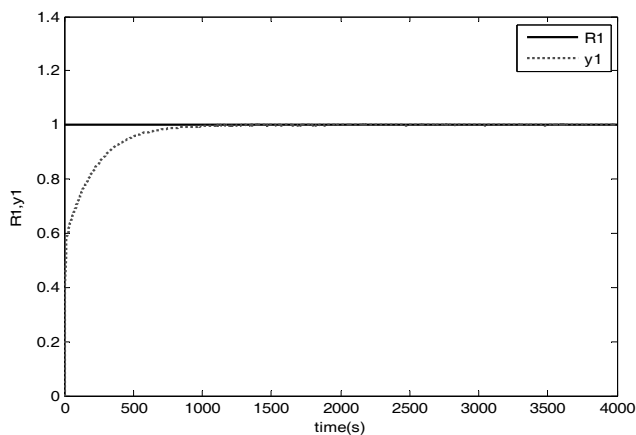


图 3-4 副回路跃阶响应

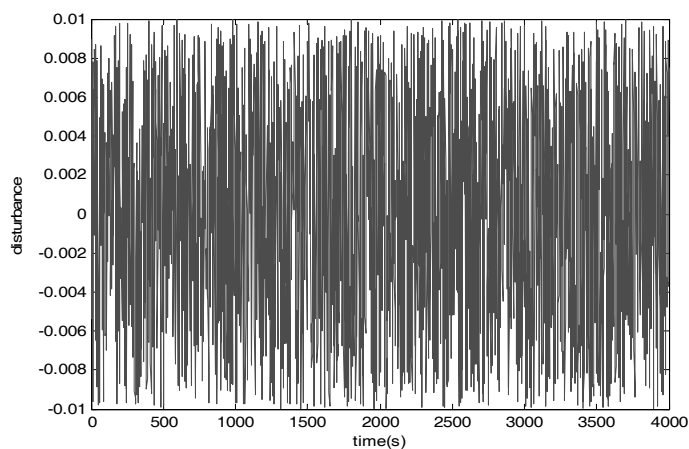


图 3-5 外加干扰信号

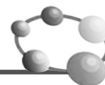
仿真程序: chap3_1.m

```
%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,'k',time,y2,'r','linewidth',2);
xlabel('time(s)');ylabel('u1,y2');
legend('u1','y2');

figure(2); %Main Control
plot(time,R1,'k',time,y1,'r','linewidth',2);
xlabel('time(s)');ylabel('R1,y1');
legend('R1','y1');

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

```



仿真方法二

按串级控制的基本原理, 采用 Simulink 进行编程, 在连续方式下进行仿真。在串级控制中, 主调节器采用 PI 控制, 取 $k_p = 50, k_i = 5$, 副调节器采用 P 控制, $k_p = 200$ 。外加干扰为正弦信号 $\sin(50t)$, 通过切换开关的切换, 分别实现常规 PID 控制及串级控制, 它们的阶跃响应结果如图 3-6 和图 3-7 所示。

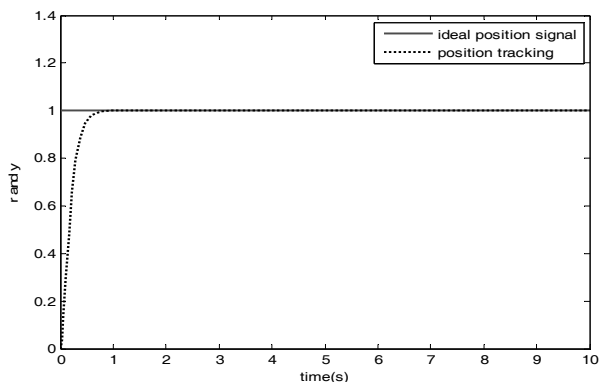


图 3-6 串级控制的阶跃响应

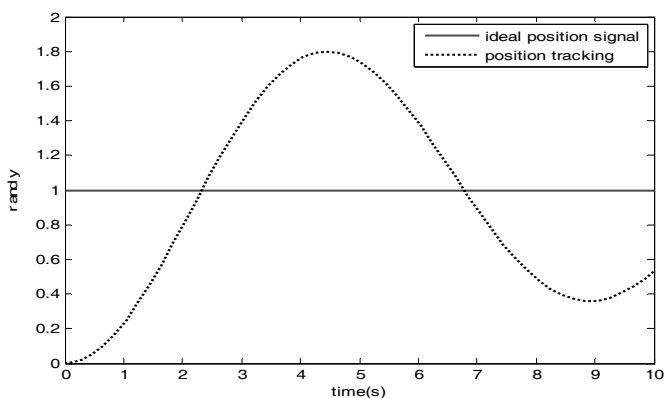


图 3-7 传统 PID 控制的阶跃响应

仿真程序

Simulink 主程序: chap3_2sim.mdl (见图 3-8)

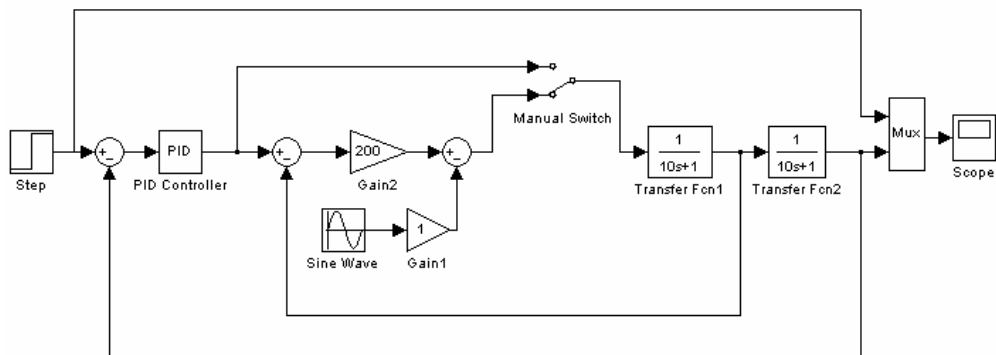
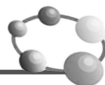


图 3-8 串级控制的 Simulink 仿真程序



作图程序: chap3_2plot.m

```
close all;
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('r and y');
legend('ideal position signal','position tracking');
```

3.3 纯滞后系统的大林控制算法

3.3.1 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。^[68]

对于如图 3-9 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

$$\phi(z) = \frac{Y(z)}{R(z)} = \frac{G_c(z)G_p(z)}{1 + G_c(z)G_p(z)} \quad (3.4)$$

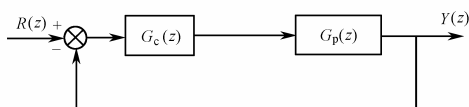


图 3-9 单回路控制系统框图

则

$$G_c(z) = \frac{U(z)}{E(z)} = \frac{1}{G_p(z)} \frac{\phi(z)}{1 - \phi(z)} \quad (3.5)$$

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

$$\phi(s) = \frac{Y(s)}{R(s)} = \frac{e^{-\tau s}}{T_\phi s + 1} \quad (3.6)$$

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。

3.3.2 仿真实例

设被控对象为

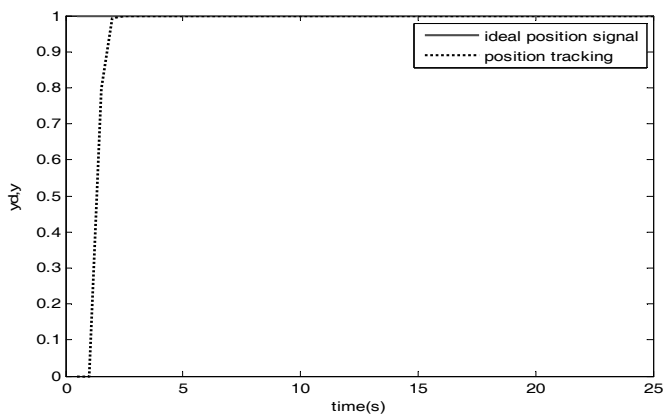
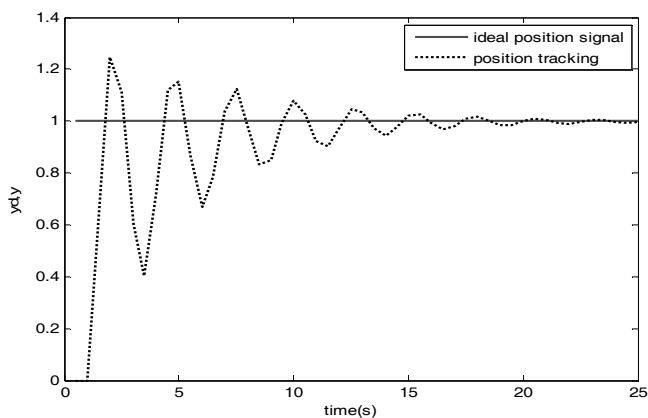
$$G_p(s) = \frac{e^{-0.76s}}{0.4s + 1}$$

采样时间为 0.5s, 期望的闭环响应设计为



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

位置指令为 $y_d = 1.0$ ， $M = 1$ 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 3-10 和图 3-11 所示。

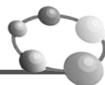
图 3-10 大林控制算法阶跃响应结果 ($M=1$)图 3-11 普通 PID 控制算法阶跃响应结果 ($M=2$)

仿真程序: chap3_3.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;

yd(k)=1.0; %Tracing Step Signal

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

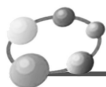
M=2;
if M==1 %Using Dalin Method
u(k)=(num(1)*error(k)+num(2)*error_1+num(3)*error_2+num(4)*error_3...
-den(3)*u_1-den(4)*u_2-den(5)*u_3-den(6)*u_4-den(7)*u_5)/den(2);
elseif M==2 %Using PID Method
ei=ei+error(k)*ts;
u(k)=1.0*error(k)+0.10*(error(k)-error_1)/ts+0.50*ei;
end
%-----Return of dalin parameters-----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

error_3=error_2;error_2=error_1;error_1=error(k);
end
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position signal','position tracking');
```



3.4 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。



3.4.1 连续 Smith 预估控制

带有纯延迟的单回路控制系统如图 3-12 所示，其闭环传递函数为

$$\phi(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G_0(s)e^{-\tau s}}{1 + G_c(s)G_0(s)e^{-\tau s}} \quad (3.7)$$

其特征方程为

$$1 + G_c(s)G_0(s)e^{-\tau s} = 0 \quad (3.8)$$

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 3-13 所示，经过 τ 的延迟时间后，被调量 Y 将重复 X 同样的变化。

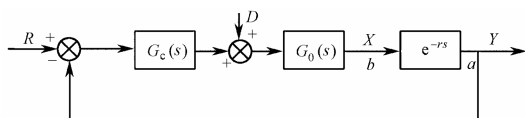


图 3-12 有纯延迟的单回路控制系统

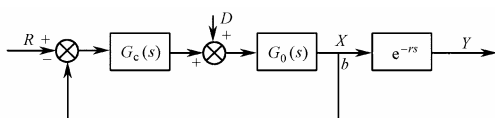


图 3-13 改进的有纯延迟的单回路控制系统

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 3-14 所示的控制系统。^[68]

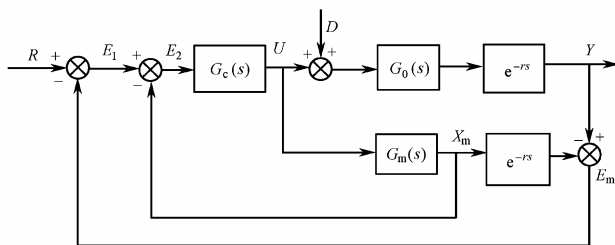


图 3-14 Smith 预估控制系统

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 3-14 变换可得到 Smith 预估控制系统等效图，如图 3-15 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。

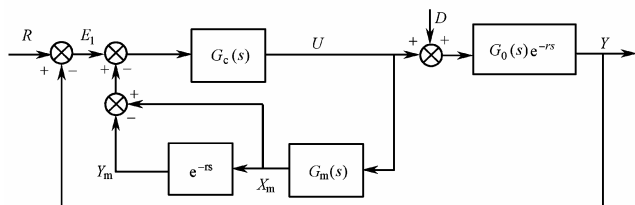
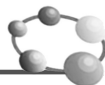


图 3-15 Smith 预估控制系统等效图

3.4.2 仿真实例

被控对象为

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

采用 Smith 控制方法，按图 3-14 的结构进行设计。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 3-16 和图 3-17 所示，仿真结果表明，Smith 控制方法具有很好控制效果。

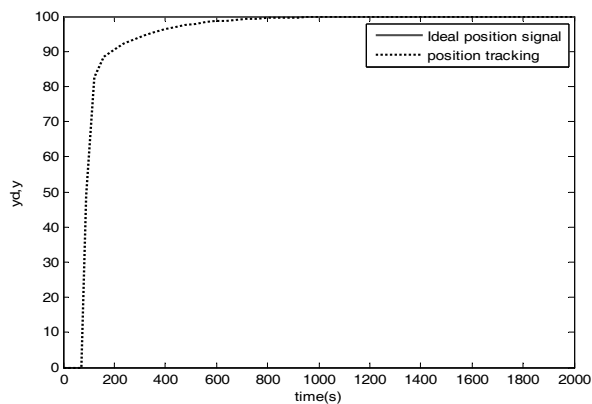


图 3-16 采用 Smith 补偿的阶跃响应

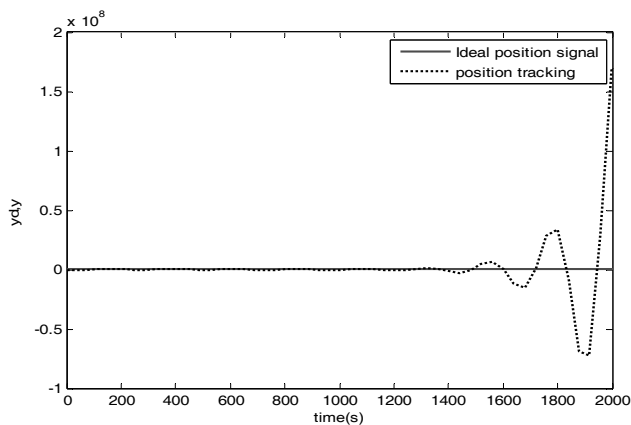
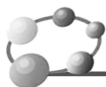


图 3-17 不采用 Smith 补偿的阶跃响应



仿真程序:

Simulink 主程序: chap3_4sim.mdl (见图 3-18)

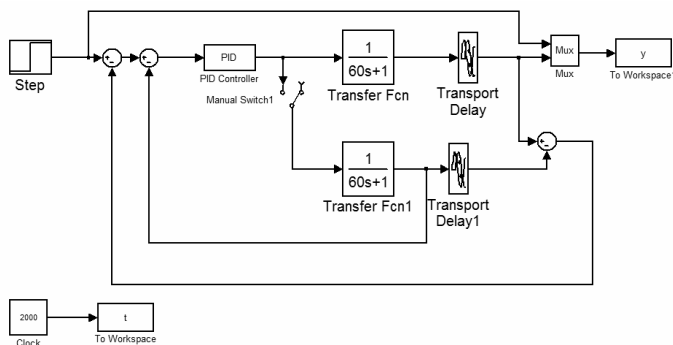


图 3-18 按图 3-14 设计的 Smith 控制系统

作图程序: chap3_4plot.m

```
close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','position tracking');
```

采用 Smith 控制方法, 按图 3-15 的结构进行设计。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。仿真结果如图 3-19 和图 3-20 所示, 仿真结果表明, Smith 控制方法具有很好控制效果。

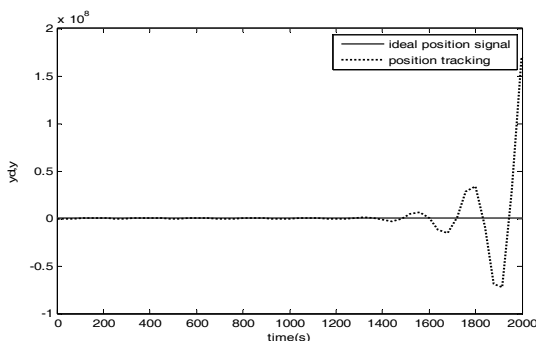


图 3-19 不采用 Smith 补偿的阶跃响应

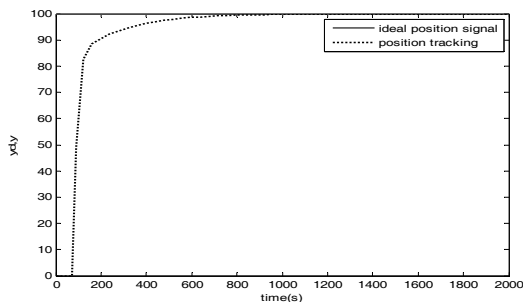


图 3-20 采用 Smith 补偿的阶跃响应



仿真程序

Simulink 主程序: chap3_5sim.mdl (见图 3-21)

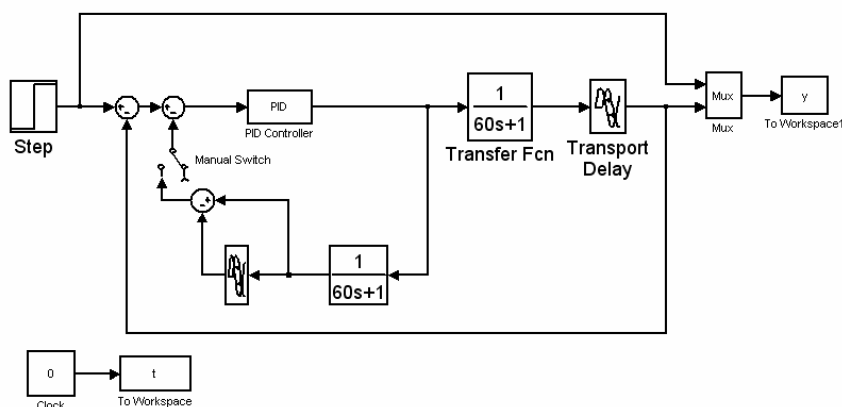


图 3-21 按图 3-15 设计的 Smith 控制系统

作图程序: chap3_5plot.m

```
close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position signal','position tracking');
```

3.4.3 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

$$G_p(s) = \frac{k_p e^{-\tau s}}{T_p s + 1} = G_0(s) e^{-\tau s} \quad (3.9)$$

被控对象离散化分别为 $G_p(z)$ 和 $G_0(z)$ ，将 Smith 预估控制系统等效图 3-15 离散化，得到数字 Smith 预估控制系统框图，如图 3-22 所示。其中 $G_{HP}(z)$ 和 $G_{HO}(z)$ 分别为 $G_p(z)$ 和 $G_0(z)$ 的估计模型。

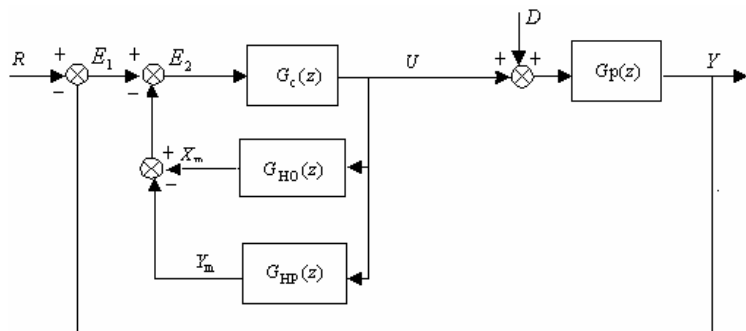


图 3-22 数字 Smith 预估控制系统框图



由图 3-22 可得

$$e_2(k) = e_1(k) - x_m(k) + y_m(k) = r(k) - y(k) - x_m(k) + y_m(k) \quad (3.10)$$

若模型是精确的, 则

$$y(k) = y_m(k) \quad (3.11)$$

$$e_2(k) = r(k) - x_m(k) \quad (3.12)$$

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

3.4.4 仿真实例

设被控对象为

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

其中采样时间为 20s。

仿真方法一

采用 M 语言进行数字化仿真。按 Smith 算法设计控制器。取位置指令为方波信号, 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 3-23 至图 3-25 所示。

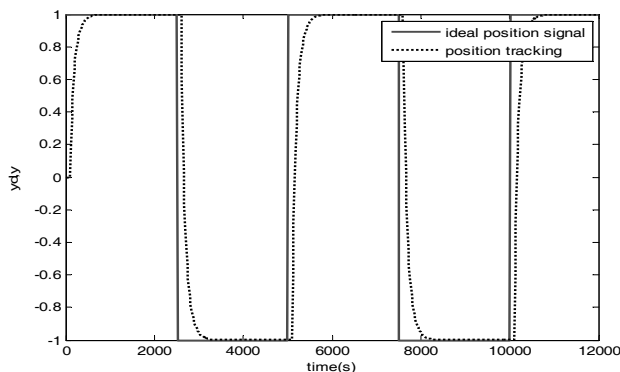


图 3-23 模型不精确时方波响应 ($M=1$)

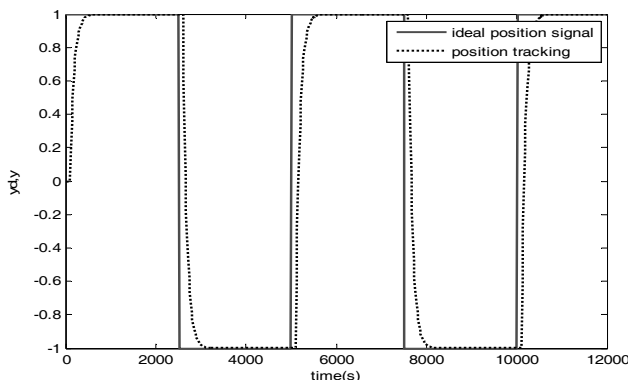
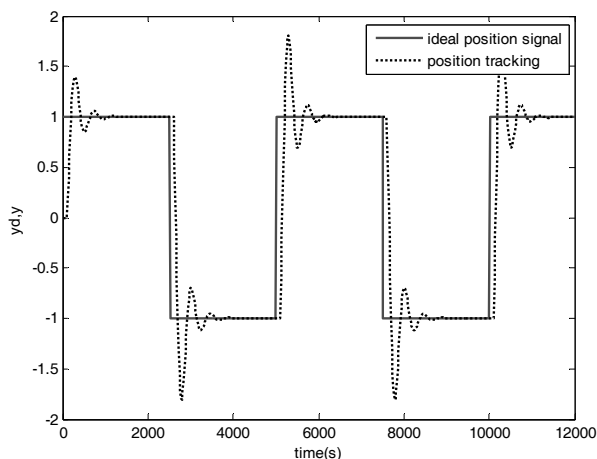
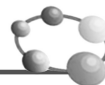


图 3-24 模型精确时方波响应 ($M=2$)

图 3-25 PI 控制时方波响应 ($M=3$)

仿真程序清单: chap3_6.m

```
%Big Delay PID Control with Smith Algorithm
clear all;close all;
Ts=20;

%Delay plant
kp=1;
Tp=60;
tol=80;
sysP=tf([kp],[Tp,1],'inputdelay',tol); %Plant
dsysP=c2d(sysP,Ts,'zoh');
[numP,denP]=tfdata(dsyp,'v');

M=1;
%Prediction model
if M==1 %No Precise Model: PI+Smith
    kp1=kp*1.10;
    Tp1=Tp*1.10;
    tol1=tol*1.0;
elseif M==2|M==3 %Precise Model: PI+Smith
    kp1=kp;
    Tp1=Tp;
    tol1=tol;
end

sysHO=tf([kp1],[Tp1,1]); %Model without delay
dsysHO=c2d(sysHO,Ts,'zoh');
[numHO,denHO]=tfdata(dsypHO,'v');

sysHP=tf([kp1],[Tp1,1],'inputdelay',tol1); %Model with delay
dsysHP=c2d(sysHP,Ts,'zoh');
```



```
[numHP,denHP]=tfdata(dsHP,'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;

    yd(k)=sign(sin(0.0002*2*pi*k*Ts)); %Tracing Square Wave Signal

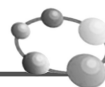
    y(k)=-denP(2)*y_1+numP(2)*u_5; %GP(z):Practical Plant

    %Prediction model
    xm(k)=-denHO(2)*xm_1+numHO(2)*u_1; %GHO(z):Without Delay
    ym(k)=-denHP(2)*ym_1+numHP(2)*u_5; %GHP(z):With Delay

    if M==1 %No Precise Model: PI+Smith
        e1(k)=yd(k)-y(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)=yd(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)=yd(k)-y(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=y(k);
end
figure(1);
plot(time,yd,'r','time,y','k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position signal','position tracking');

```

仿真方法二

采用 Simulink 进行数字化仿真，按 Smith 算法设计 Simulink 模块。在 PI 控制中， $k_p = 0.5, k_i = 0.01$ 。其响应结果如图 3-26 和图 3-27 所示。

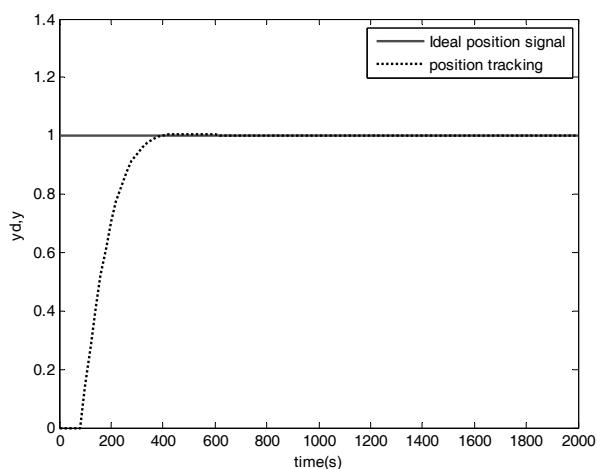


图 3-26 Smith 阶跃响应结果

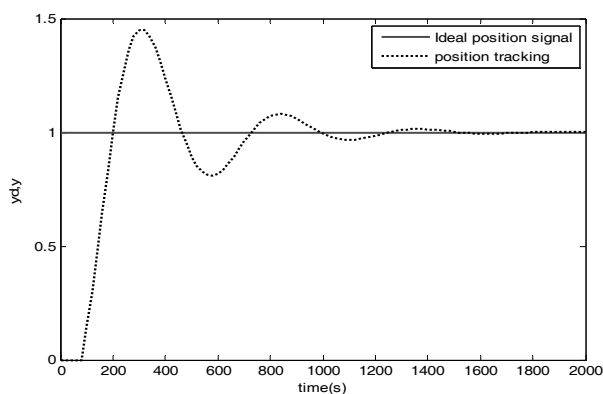


图 3-27 只采用 PI 控制时的阶跃响应结果

仿真程序：

初始化程序: chap3_7int.m

```

%Big Delay PID Control with Smith Algorithm
clear all;close all;
Ts=20;

%Delay plant

```



```
kp=1;  
Tp=60;  
tol=80;  
sysP=tf([kp],[Tp,1],'inputdelay',tol); %Plant  
dsysP=c2d(sysP,Ts,'zoh');  
[numP,denP]=tfdata(dsyp,'v');
```

仿真主程序: chap3_7sim.mdl (见图 3-28)

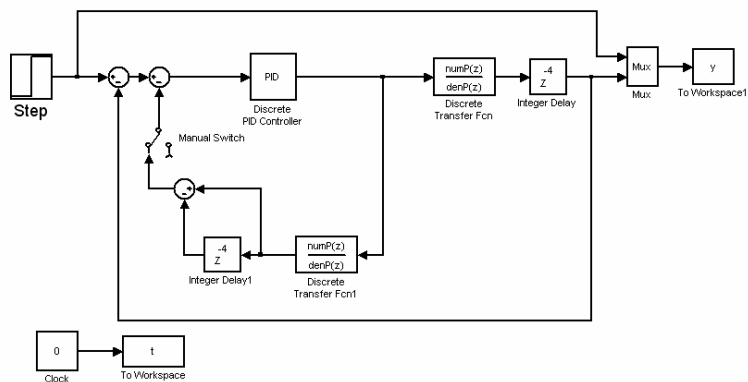


图 3-28 离散 Smith 系统的 Simulink 仿真

作图程序: chap3_7plot.m

```
close all;  
figure(1);  
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('yd,y');  
legend('Ideal position signal','position tracking');
```

第4章 基于微分器的PID控制

信号微分的求取是PID控制的关键问题,迅速精确地获取信号的速度对于控制系统至关重要,由传感器测量到的位置信息估计其速度在工程上是困难的任务和具有挑战性的问题。

本章采用微分器实现带有噪声信号的提取并求导,从而实现无需速度测量的控制。



4.1 基于全程快速微分器的PID控制

4.1.1 全程快速微分器

王新华等^[25, 26]提出了全程快速微分器,其表达式为:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= R^2 \left(-a_0 (x_1 - v(t)) - a_1 (x_1 - v(t))^{\frac{m}{n}} - b_0 \frac{x_2}{R} - b_1 \left(\frac{x_2}{R} \right)^{\frac{m}{n}} \right) \\ y &= x_2(t)\end{aligned}\quad (4.1)$$

式中, $R > 0$, $a_0, a_1, b_0, b_1 \geq 0$, m, n 均为大于0的奇数,且 $m < n$ 。系统输出 $y = x_2(t)$ 跟踪信号 $v(t)$ 的一阶导数 $\dot{v}(t)$ 。当取 $a_1 = b_1 = 0$ 时,线性微分器起主导作用:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= R^2 \left(-a_0 (x_1 - v(t)) - b_0 \frac{x_2}{R} \right) \\ y &= x_2(t)\end{aligned}\quad (4.2)$$

该种形式微分器可直接通过差分或高精度数值迭代方法来离散化。

4.1.2 仿真实例

取 $R = \frac{1}{0.01}$, $a_0 = 0.1$, $b_0 = 0.1$, 采用微分器式(4.2), 输入信号为 $v(t) = \sin t$, 采用连

续微分器,数值求解器中取固定步长0.001,扰动为幅值为0.05的随机信号,信号跟踪及导数估计曲线如图4-1和图4-2所示。图4-1中,上图为带有噪声的单位正弦信号,下图为理想单位正弦信号和线性微分器输出 x_1 。图4-2为信号理想导数 $\cos t$ 与线性微分器式的输出 x_2 。

见仿真程序 chap4_1sim.mdl。

采用离散化微分器,扰动为幅值为0.01的随机信号,信号跟踪及导数估计曲线如图4-3和图4-4所示。图4-3中,上图为带有噪声的单位正弦信号,下图为理想单位正弦信号和线性微分器输出 x_1 。图4-4中,上图为采用差分方法求导的导数输出,下图为信号理想导数 $\cos t$



与线性微分器式的输出 x_2 。见仿真程序 chap4_2.m。

将微分器用于闭环控制中, 位置指令为 $y_d(t) = \sin t$, 被控对象为 $\frac{133}{s^2 + 25s}$, 取 $R = \frac{1}{0.01}$, $a_0 = 2$, $b_0 = 1$, 采用微分器式 4.2, 对象输出叠加幅值为 0.5 的随机毛刺信号, 采用 PID 控制律, 取 $k_p = 10$, $k_d = 0.5$, 仿真结果如图 4-5 至图 4-8 所示, 其中图 4-5 和图 4-8 分别为采用和不采用微分器的正弦位置跟踪。见仿真程序 chap4_3sim.mdl。

取采样时间为 $T = 0.001$, 可实现相应的数字微分器的信号处理和闭环控制, 见仿真程序 chap4_4.m。

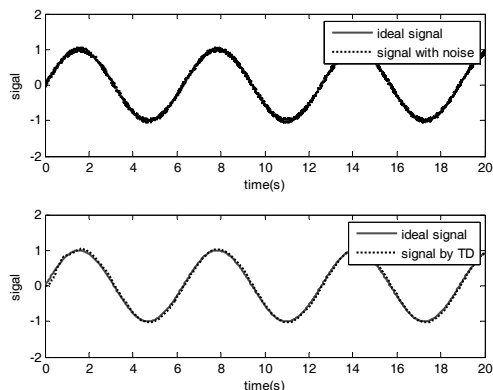


图 4-1 位置信号的提取

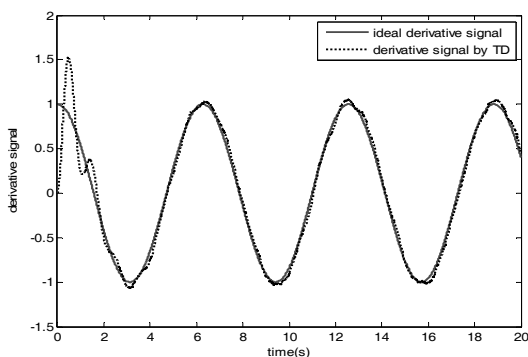


图 4-2 位置信号的求导

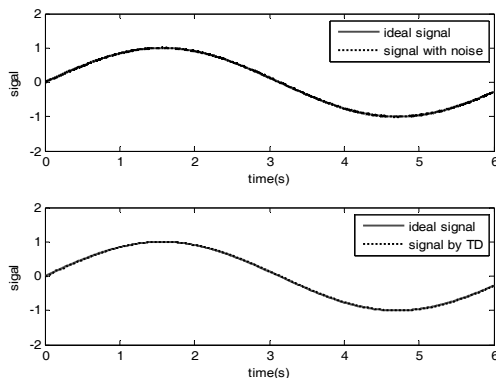


图 4-3 位置信号的提取

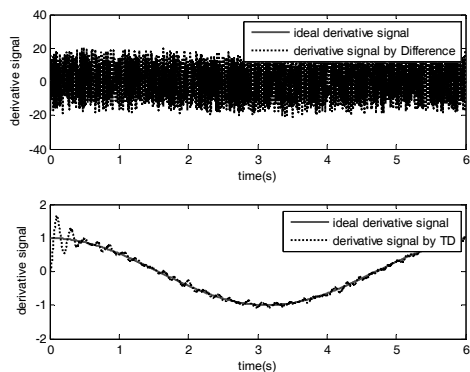


图 4-4 位置信号的求导

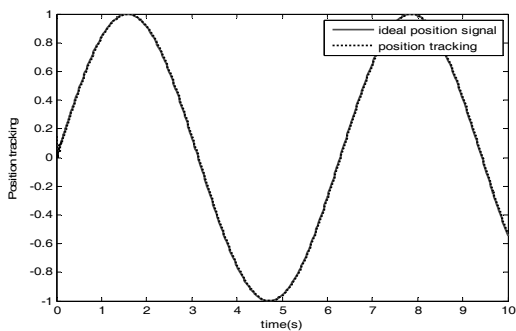


图 4-5 采用微分器的正弦位置跟踪

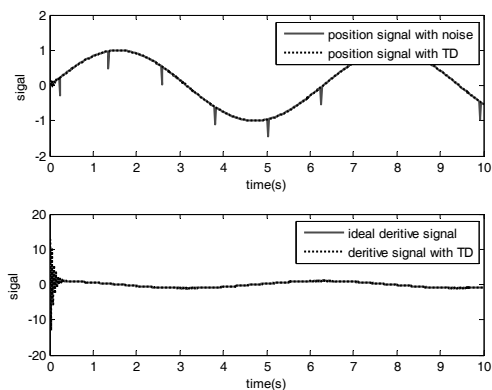


图 4-6 位置信号的提取及求导

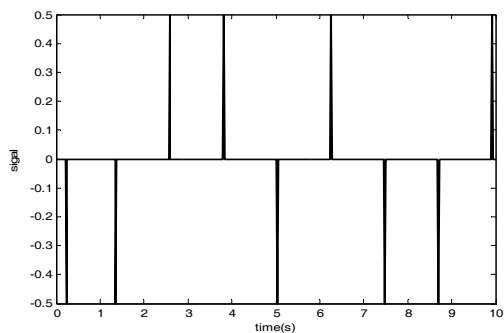


图 4-7 加在对象输出端的随机毛刺信号

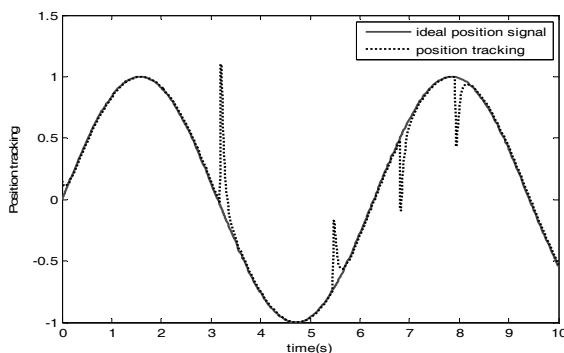


图 4-8 不采用微分器的正弦位置跟踪

仿真程序

1. 微分器信号处理

(1) 连续系统仿真

主程序: chap4_1sim.mdl (见图 4-9)

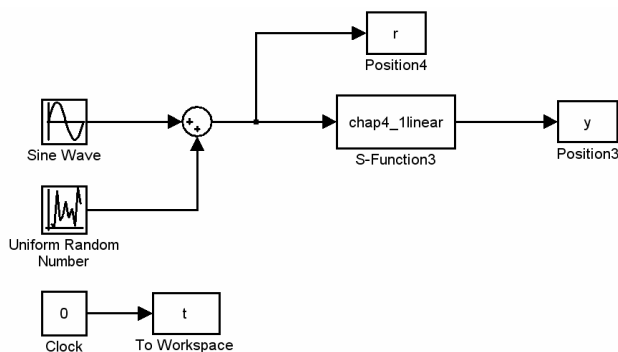
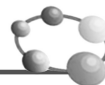


图 4-9 微分器信号处理主程序

微分器 S 函数: chap4_1linear.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
```



```

sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0 0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
vt=u(1);
e=x(1)-vt;
R=1/0.05;a0=0.1;b0=0.1;

sys(1)=x(2);
sys(2)=R^2*(-a0*e-b0*x(2)/R);
function sys=mdlOutputs(t,x,u)
sys = x;

```

作图程序: chap4_1plot.m

```

close all;

figure(1);
subplot(211);
plot(t,sin(t),'r',t,r,'k','linewidth',2);
xlabel('time(s)');ylabel('sigal');
legend('ideal signal','signal with noise');
subplot(212);
plot(t,sin(t),'r',t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('sigal');
legend('ideal signal','signal by TD');

figure(2);
plot(t,cos(t),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('derivative signal');
legend('ideal derivative signal','derivative signal by TD');

```

(2) 数字仿真

离散微分器程序: chap4_2.m

```

close all;
clear all;
T=0.001;
y_1=0;dy_1=0;
yv_1=0;
v_1=0;

```



```
for k=1:1:6000
    t=k*T;
    time(k)=t;

    v(k)=sin(t);
    dv(k)=cos(t);

    d(k)=0.01*rands(1);    %Noise
    yv(k)=v(k)+d(k);      %Practical signal

    R=1/0.01;a0=0.1;b0=0.1;
    y(k)=y_1+T*dy_1;
    dy(k)=dy_1+T*R^2*(-a0*(y(k)-yv(k))-b0*dy_1/R);

    dyv(k)=(yv(k)-yv_1)/T;    %Speed by Difference

    y_1=y(k);
    v_1=v(k);
    yv_1=yv(k);
    dy_1=dy(k);
end
figure(1);
subplot(211);
plot(time,v,'r',time,yv,'k:','linewidth',2);
xlabel('time(s)');ylabel('sigal');
legend('ideal signal','signal with noise');
subplot(212);
plot(time,v,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('sigal');
legend('ideal signal','signal by TD');

figure(2);
subplot(211);
plot(time,dv,'r',time,dyv,'k:','linewidth',2);
xlabel('time(s)');ylabel('derivative signal');
legend('ideal derivative signal','derivative signal by Difference');
subplot(212);
plot(time,dv,'r',time,dy,'k:','linewidth',2);
xlabel('time(s)');ylabel('derivative signal');
legend('ideal derivative signal','derivative signal by TD');
```

2. 基于微分器的PID控制

(1) 连续系统仿真

主程序: chap4_3sim.mdl (见图 4-10)

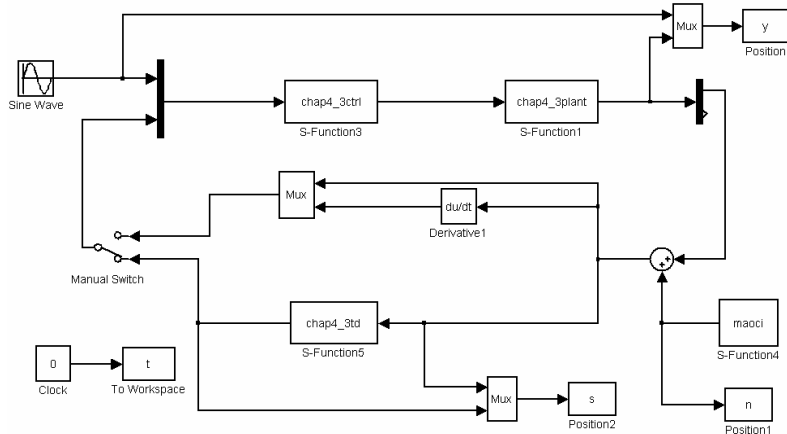
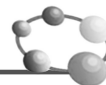


图 4-10 基于微分器 PID 控制主程序

控制器 S 函数: chap4_3ctrl.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
yd=u(1);
dyd=cos(t);
y=u(2);
dy=u(3);
e=yd-y;
de=dyd-dy;

kp=10;kd=0.5;
ut=kp*e+kd*de;
sys(1)=ut;
```



微分器 S 函数: chap4_3td.m

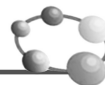
```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0 0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
vt=u(1);
e=x(1)-vt;
R=1/0.01;a0=2;b0=1;
sys(1)=x(2);
sys(2)=R^2*(-a0*e-b0*x(2)/R);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

作图程序: chap4_3plot.m

```
close all;

figure(1);
plot(t,y(:,1),'k',t,y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','tracking signal');

figure(2);
subplot(211);
plot(t,s(:,1),'k',t,s(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('signal');
```



```

legend('practical position signal','position signal with TD');
subplot(212);
plot(t,y(:,3),'k',t,s(:,3),'r','linewidth',2);
xlabel('time(s)');ylabel('sigal');
legend('ideal deritive signal','deritive signal with TD');

figure(3);
plot(t,n(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('sigal');

```

（2）数字仿真

离散微分器控制程序：chap4_4.m

```

close all;
clear all;

T=0.001;
y_1=0;yp_1=0;
dy_1=0;

%Plant
a=25;b=133;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,T,'z');
[num,den]=tfdata(dsys,'v');
u_1=0;u_2=0;
p_1=0;p_2=0;
for k=1:1:5000
t=k*T;
time(k)=t;

p(k)=-den(2)*p_1-den(3)*p_2+num(2)*u_1+num(3)*u_2;
dp(k)=(p(k)-p_1)/T;

yd(k)=sin(t);
dyd(k)=cos(t);
d(k)=1.5*sign(rands(1)); %Noise

if mod(k,100)==1|mod(k,100)==2
    yp(k)=p(k)+d(k);    %Practical signal
else
    yp(k)=p(k);
end
yp(k)=yp(k)+0.1*rands(1);

M=2;
if M==1    %By Difference
    y(k)=yp(k);
    dy(k)=(yp(k)-yp_1)/T;
elseif M==2    %By TD

```




```

R=100;a0=2;b0=1;
y(k)=y_1+T*dy_1;
dy(k)=dy_1+T*R^2*(-a0*(y(k)-yp(k))-b0*dy_1/R);
end
kp=10;kd=0.2;
u(k)=kp*(yd(k)-y(k))+kd*(dyd(k)-dy(k));

if M==3      %Using ideal plant
    u(k)=kp*(yd(k)-p(k))+kd*(dyd(k)-dp(k));
end

y_1=y(k);
yp_1=yp(k);
dy_1=dy(k);

u_2=u_1;u_1=u(k);
p_2=p_1;p_1=p(k);
end
figure(1);
plot(time,p,'k',time,yp,'r-',time,y,'b-', 'linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('ideal position signal','position signal with noise','position signal by TD');
figure(2);
plot(time,yd,'r',time,p,'k-', 'linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');

```



4.2 基于 Levant 微分器的 PID 控制

4.2.1 Levant 微分器

微分器中需要注意的问题是既要尽量准确求导，又要对信号的测量误差和输入噪声具有鲁棒性。Levant^[27]提出了一种基于滑模技术的非线性微分器，其中二阶滑模微分器表达式为

$$\begin{aligned}
 \dot{x} &= u \\
 u &= u_1 - \lambda |x - v(t)|^{1/2} \text{sign}(x - v(t)) \\
 \dot{u}_1 &= -\alpha \text{sign}(x - v(t))
 \end{aligned} \tag{4.3}$$

其中

$$\alpha > C, \lambda^2 \geq 4C \frac{\alpha + C}{\alpha - C} \tag{4.4}$$

并且 $C > 0$ 是输入信号 $v(t)$ 导数的 Lipschitz 常数上界。

采用二阶 Levant 微分器，可实现 x 跟踪 $v(t)$ ， u_1 跟踪 $\dot{v}(t)$ 。对于 Levant 微分器，需要事先知道输入信号 $v(t)$ 导数的 Lipschitz 常数上界，才能设计微分器参数，这就限制了输入信号的类型。而且，对于这种微分器，抖振现象不可避免。



4.2.2 仿真实例

4.2.2.1. 微分器频域分析

通过采用扫频测试的方法对 Levant 微分器进行频域分析, 可得该微分器的频域特性。

采样时间为 $T=0.001\text{s}$, 令输入信号为 $v(t)=\sin(2\pi t)$, $v(t)$ 导数的上界为 2π , 取 $C=2\pi$, 采用 0Hz 至 30Hz 的正弦信号进行扫频。则按式 (4.4), 取 Levant 参数为 $\alpha=15$, $\lambda=10$, 频域特性如图 4-11 (a) 所示, 取 Levant 参数为 $\alpha=15$, $\lambda=20$, 频域特性如图 4-11 (b) 所示。可见, Levant 微分器具有很好低频跟踪效果, 以及较强的抑制噪声能力, 如果增大 λ 值, 可以拓宽频带, 但同时增加了 Levant 微分器切换增益, 加大了抖振。

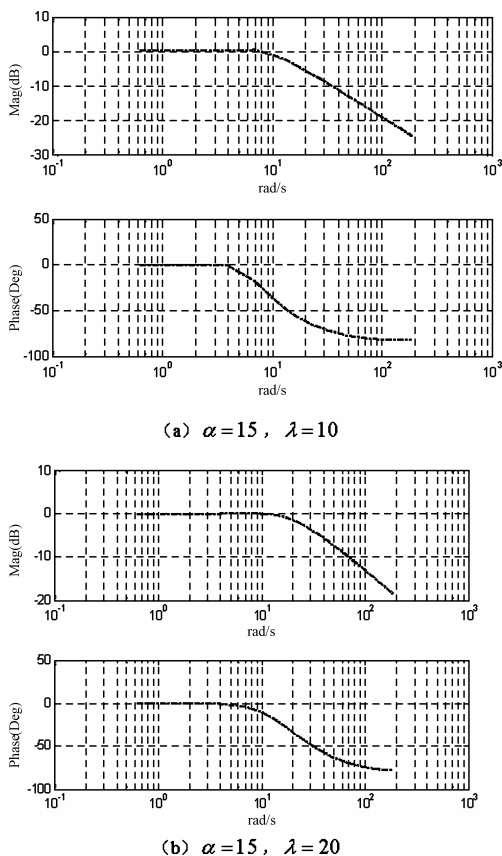


图 4-11 不同参数下 Levant 微分器的频域特性

4.2.2.2. 微分器信号处理

输入信号取 $v(t)=\sin t$, 则可得 $C=1.0$, 按式 (4.4), 取 $\alpha=18$, 则 $\lambda^2 \geq 4C \frac{\alpha+C}{\alpha-C} = 4 \frac{18+1}{18-1} = 4.4706$, 则可取 $\lambda=6$ 。扰动为随机信号, 采用连续系统仿真, 信号跟踪及导数估计如图 4-12 和图 4-13 所示。图 4-12 中, 上图为带有噪声的单位正弦信号, 下图为 Levant 微分器输出 x_1 和理想的正弦信号。图 4-13 中, 上图为采用 MATLAB 工具对信号求导的输出, 下图为 Levant 微分器导数估计输出 x_2 。采用离散系统仿真可得到同样结果,



见仿真程序 chap4_7.m。

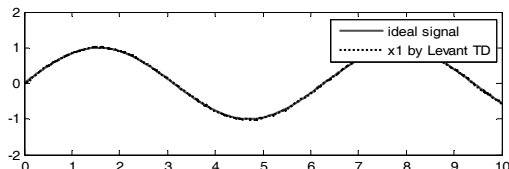
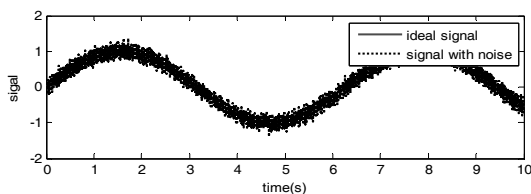


图 4-12 信号跟踪

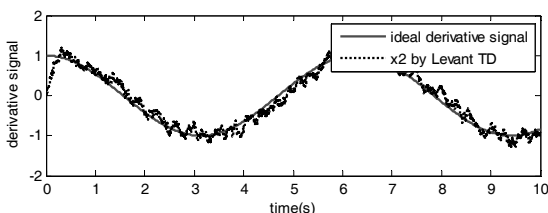
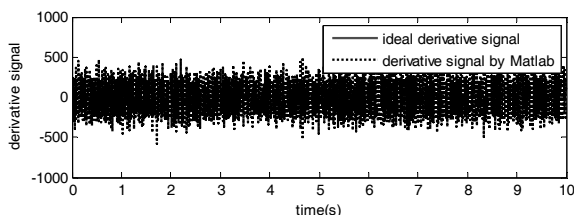


图 4-13 导数求取

4.2.2.3. 基于微分器的 PID 控制

输入信号为 $v(t) = \sin(t)$ ， $v(t)$ 导数的上界为 1，取 $C=1$ ，则按条件 (4.4)，取 Levant 参数为 $\alpha=15$ ， $\lambda=10$ 。采用 PID 控制律，取 $k_p=10$ ， $k_d=0.5$ 。采用连续系统仿真，通过 Levant 微分器式 (4.3) 来获得位置和速度，位置跟踪如图 4-14 所示，如果采用传统的微分方式获得位置和速度，位置跟踪如图 4-15 所示。离散系统仿真程序见 chap4_9.m。

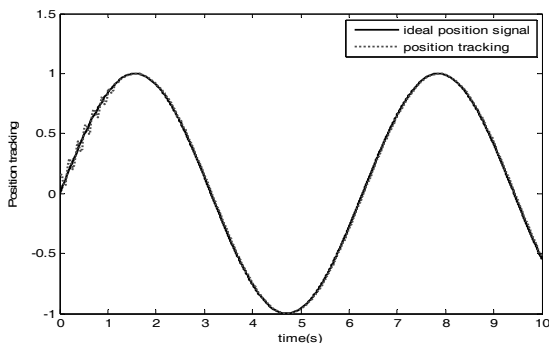


图 4-14 基于 Levant 的 PID 控制

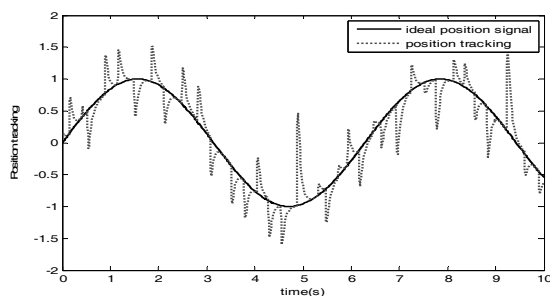
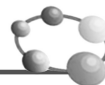


图 4-15 基于传统差分方式的 PID 控制

仿真程序：

(1) 微分器扫频分析

扫频测试程序：chap4_5a.m

```
clear all;
close all;

T=0.001;
Am=1;

f=1;
for F=0.1:0.5:30
    u_1=0;y_1=0;dy_1=0;
    for k=1:1:10000
        time(k)=k*T;
        u(f,k)=Am*sin(1*2*pi*F*k*T);           % Sine Signal with different frequency

        %Levant TD
        afa=15;nbd=10;           %From Levant paper
        afa=15;nbd=20;           %From Levant paper
        y(f,k)=y_1+T*(dy_1-nbd*sqrt(abs(y_1-u(f,k)))*sign(y_1-u(f,k)));
        dy(k)=dy_1-T*afa*sign(y_1-u(f,k));
        dy_1=dy(k);

        uk(k)=u(f,k);
        yk(k)=y(f,k);

        y_1=yk(k);
        u_1=uk(k);
    end
    f=f+1;
end

save TDfile y;
```

微分器频域特性分析程序：chap4_5b.m



```
close all;
clear all;
T=0.001;
Am=1;

load TDfile;
kk=0;
f=1;
for F=0.1:0.5:30
    kk=kk+1;
    FF(kk)=F;
    w=FF*2*pi; % in rad./s

    for i=5001:1:10000
        fai(1,i-5000) = sin(2*pi*F*i*T);
        fai(2,i-5000) = cos(2*pi*F*i*T);
    end
    Fai=fai';

    fai_in(kk)=0;

    Y_out=y(f,5001:1:10000)';
    cout=inv(Fai'*Fai)*Fai'*Y_out;
    fai_out(kk)=atan(cout(2)/cout(1)); % Phase Frequency(Deg.)

    Af(kk)=sqrt(cout(1)^2+cout(2)^2); % Magnitude Frequency(dB)
    mag_e(kk)=20*log10(Af(kk)/Am); % in dB
    ph_e(kk)=(fai_out(kk)-fai_in(kk))*180/pi; % in Deg.

    f=f+1;
end
figure(1);
hold on;
subplot(2,1,1);
semilogx(w,mag_e,'r-','linewidth',2);grid on;
xlabel('rad./s');ylabel('Mag.(dB)');
hold on;
subplot(2,1,2);
semilogx(w,ph_e,'r-','linewidth',2);grid on;
xlabel('rad./s');ylabel('Phase(Deg.)');
```

(2) 微分器信号处理分连续系统和离散系统两种情况

(2a) 连续系统仿真

Simulink 仿真主程序: chap4_6sim.mdl (见图 4-16)

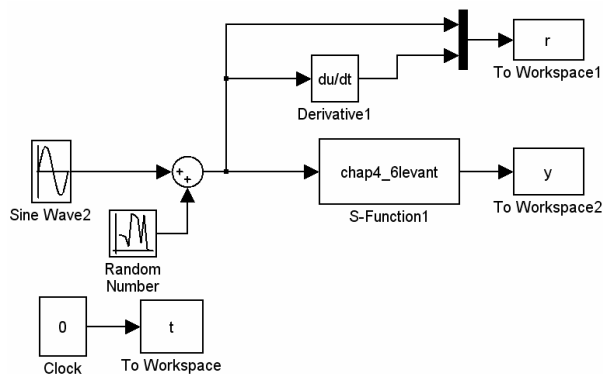
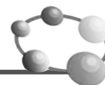


图 4-16 微分器信号处理主程序

微分器 S 函数: chap4_6levant.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0 0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
vt=u(1);
e=x(1)-vt;

nmn=6;
alfa=18;
```



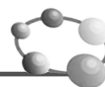
```
sys(1)=x(2)-nmn*(abs(e))^0.5*sign(e);  
sys(2)=-alfa*sign(e);  
function sys=mdlOutputs(t,x,u)  
sys = x;
```

作图程序: chap4_6plot.m

```
close all;  
  
figure(1);  
subplot(211);  
plot(t,sin(t),'r',t,r(:,1),'k','linewidth',2);  
xlabel('time(s)');ylabel('signal');  
legend('ideal signal','signal with noise');  
subplot(212);  
plot(t,sin(t),'r',t,y(:,1),'k','linewidth',2);  
legend('ideal signal','x1 by Levant TD');  
  
figure(2);  
subplot(211);  
plot(t,cos(t),'r',t,r(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('derivative signal');  
legend('ideal derivative signal','derivative signal by Matlab');  
subplot(212);  
plot(t,cos(t),'r',t,y(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('derivative signal');  
legend('ideal derivative signal','x2 by Levant TD');
```

(2b) 离散系统仿真: chap4_7.m

```
%Discrete Levant TD  
close all;  
clear all;  
T=0.001;  
y_1=0;dy_1=0;  
for k=1:1:10000  
    t=k*T;  
    time(k)=t;  
  
    u(k)=sin(k*T);  
    du(k)=cos(k*T);  
  
    d(k)=0.5; %Noise  
    d(k)=-0.5; %Noise  
    d(k)=0.5*sign(rands(1)); %Noise  
    if mod(k,100)==1  
        up(k)=u(k)+1*d(k); %Practical signal  
    else
```



```

        up(k)=u(k);
    end
    up(k)=up(k)+0.15*rands(1);

    alfa=8;nmna=6;        %M=1    Low Frequency

    y(k)=y_1+T*(dy_1-nmna*sqrt(abs(y_1-up(k)))*sign(y_1-up(k)));
    dy(k)=dy_1-T*alfa*sign(y_1-up(k));

    y_1=y(k); dy_1=dy(k);
end
figure(1);
subplot(211);
plot(time,u,'r',time,up,'k','linewidth',2);
xlabel('time(s)'),ylabel('input signal');
legend('sin(t)','signal with noises');
subplot(212);
plot(time,u,'r',time,y,'k','linewidth',2);
xlabel('time(s)'),ylabel('input signal');
legend('sin(t)','signal with TD');

figure(2);
plot(time,du,'r',time,dy,'k','linewidth',2);
xlabel('time(s)'),ylabel('derivative estimation');
legend('cos(t)','x2 by Levant differentiator');

```

(3) 基于微分器的 PID 控制：分连续系统和离散系统两种情况

(3a) 连续系统仿真

Simulink 仿真主程序：chap4_8sim.mdl（见图 4-17）

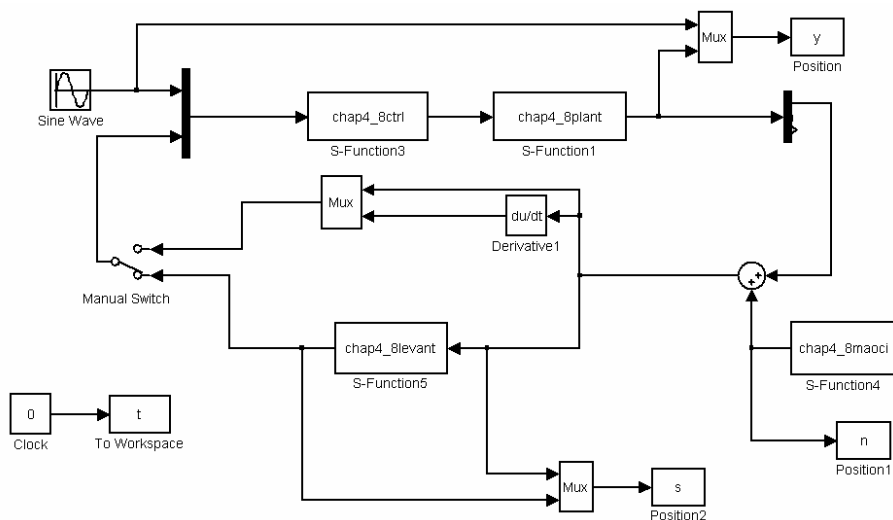


图 4-17 基于微分器的 PID 控制主程序

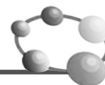


被控对象 S 函数: chap4_8plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0.15 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-25*x(2)+133*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

控制器 S 函数: chap4_8ctrl.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```



```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
yd=u(1);
dyd=cos(t);
y=u(2);
dy=u(3);
e=yd-y;
de=dyd-dy;

kp=10;kd=0.5;
ut=kp*e+kd*de;
sys(1)=ut;
```

微分器 S 函数: chap4_8levant.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
```



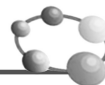
```
x0 = [0 0];  
str = [];  
ts = [0 0];  
function sys=mdlDerivatives(t,x,u)  
vt=u(1);  
e=x(1)-vt;  
alfa=15;nmna=10;  
  
sys(1)=x(2)-nmna*sqrt(abs(e))*sign(e);  
sys(2)=-alfa*sign(e);  
function sys=mdlOutputs(t,x,u)  
sys(1)=x(1);  
sys(2)=x(2);
```

作图程序: chap4_8plot.m

```
close all;  
  
figure(1);  
plot(t,y(:,1),'k',t,y(:,2),'r','linewidth',2);  
xlabel('time(s)');ylabel('Position tracking');  
legend('ideal position signal','position tracking');  
  
figure(2);  
subplot(211);  
plot(t,s(:,1),'k',t,s(:,2),'r','linewidth',2);  
xlabel('time(s)');ylabel('sigal');  
legend('position signal with noise','position signal with TD');  
subplot(212);  
plot(t,y(:,3),'r',t,s(:,3),'k','linewidth',2);  
xlabel('time(s)');ylabel('sigal');  
legend('practical deritive signal','deritive signal with TD');  
  
figure(3);  
plot(t,n(:,1),'k','linewidth',2);  
xlabel('time(s)');ylabel('sigal');
```

(3b) 离散系统仿真: chap4_9.m

```
%PID based on Discrete Levant TD  
close all;  
clear all;  
  
T=0.001;  
y_1=0;yp_1=0;  
dy_1=0;
```



```

%Plant
a=25;b=133;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,T,'z');
[num,den]=tfdata(dsys,'v');
u_1=0;u_2=0;
p_1=0;p_2=0;
for k=1:1:5000
t=k*T;
time(k)=t;

yd(k)=sin(t);
dyd(k)=cos(t);
p(k)=-den(2)*p_1-den(3)*p_2+num(2)*u_1+num(3)*u_2;

d(k)=0.5*sign(rands(1));
if mod(k,100)==1|mod(k,100)==2
    yp(k)=p(k)+d(k);    %Practical signal
else
    yp(k)=p(k);
end

M=2;
if M==1    %By Difference
    y(k)=yp(k);
    dy(k)=(yp(k)-yp_1)/T;
elseif M==2    %By TD
    alfa=8;nmna=6;
    y(k)=y_1+T*(dy_1-nmna*sqrt(abs(y_1-yp(k))))*sign(y_1-yp(k));
    dy(k)=dy_1-T*alfa*sign(y_1-yp(k));
end
kp=10;kd=0.1;
u(k)=kp*(yd(k)-y(k))+kd*(dyd(k)-dy(k));

y_1=y(k);
yp_1=yp(k);
dy_1=dy(k);

u_2=u_1;u_1=u(k);
p_2=p_1;p_1=p(k);
end
if M==1    %By Difference
figure(1);
plot(time,yd,'k',time,p,'r','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
```



```
elseif M==2      %By TD
figure(1);
subplot(211);
plot(time,p,'k',time,yp,'r',time,y,'b','linewidth',2);
xlabel('time(s)');ylabel('position signal');
legend('ideal position signal','position signal with noise','position signal by TD');
subplot(212);
plot(time,dy,'k','linewidth',2);
xlabel('time(s)');ylabel('speed signal');
legend('speed signal by TD');
figure(2);
plot(time,yd,'r',time,p,'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
end
```

第5章 基于观测器的PID控制



5.1 基于慢干扰观测器补偿的PID控制

5.1.1 系统描述

考虑带有慢干扰的二阶系统

$$\ddot{\theta} = -b\dot{\theta} + au - d \quad (5.1)$$

式中, $b > 0$, $a > 0$, a 和 b 为已知值, d 为慢干扰时变信号。

5.1.2 观测器设计

Atsuo K. 等^[28]针对慢干扰系统, 设计了如下观测器:

$$\dot{\hat{d}} = k_1(\hat{\omega} - \dot{\theta}) \quad (5.2)$$

$$\dot{\hat{\omega}} = -\hat{d} + au - k_2(\hat{\omega} - \dot{\theta}) - b\dot{\theta} \quad (5.3)$$

式中, $\hat{d}$ 为对 d 项的估计, $\hat{\omega}$ 为对 $\dot{\theta}$ 的估计, $k_1 > 0$, $k_2 > 0$ 。

稳定性分析

定义 Lyapunov 函数为

$$V = \frac{1}{2k_1}\tilde{d}^2 + \frac{1}{2}\tilde{\omega}^2 \quad (5.4)$$

式中, $\tilde{d} = d - \hat{d}$, $\tilde{\omega} = \dot{\theta} - \hat{\omega}$ 。

则

$$\dot{V} = \frac{1}{k_1}\tilde{d}\dot{\tilde{d}} + \tilde{\omega}\dot{\tilde{\omega}} = \frac{1}{k_1}\tilde{d}\left(\dot{d} - \dot{\hat{d}}\right) + \tilde{\omega}\left(\ddot{\theta} - \dot{\hat{\omega}}\right) \quad (5.5)$$

假设干扰 d 为慢时变信号, $\dot{d}$ 很小, 当取 k_1 较大值时, 有

$$\frac{1}{k_1}\dot{d} \approx 0 \quad (5.6)$$

将式 (5.2)、式 (5.3)、式 (5.6) 代入式 (5.5), 得

$$\begin{aligned} \dot{V} &= \frac{1}{k_1}\tilde{d}\dot{\tilde{d}} - \frac{1}{k_1}\tilde{d}\dot{\hat{d}} + \tilde{\omega}\left(\ddot{\theta} - \left(-\hat{d} + au - k_2(\hat{\omega} - \dot{\theta}) - b\dot{\theta}\right)\right) \\ &= \frac{1}{k_1}\tilde{d}\dot{\tilde{d}} - \frac{1}{k_1}\tilde{d}k_1(\hat{\omega} - \dot{\theta}) + \tilde{\omega}\left(-b\dot{\theta} + au - d - \left(-\hat{d} + au - k_2(\hat{\omega} - \dot{\theta}) - b\dot{\theta}\right)\right) \end{aligned}$$



$$\begin{aligned}
 &= \frac{1}{k_1} \tilde{d} \dot{d} - \tilde{d} (\hat{\omega} - \dot{\theta}) + \tilde{\omega} (-d + \hat{d} + k_2 (\hat{\omega} - \dot{\theta})) \\
 &= \frac{1}{k_1} \tilde{d} \dot{d} + \tilde{d} \tilde{\omega} + \tilde{\omega} (-\tilde{d} - k_2 \tilde{\omega}) = \frac{1}{k_1} \tilde{d} \dot{d} - k_2 \tilde{\omega}^2 \leq 0
 \end{aligned}$$

通过采用本观测器，对 d 项进行有效的观测，从而实现补偿。根据式 (5.1)，加入补偿后的控制律为

$$u = u_0 + \frac{1}{a} \hat{d} \quad (5.7)$$

其中， u_0 为 PID 控制。

5.1.3 仿真实例

对象动态方程为

$$\ddot{\theta} = -b\dot{\theta} + au - d$$

式中， $a=5$ ， $b=0.15$ ， d 为干扰，取 $d=150\text{sign}(\sin(0.1t))$ ， $\text{sgn}(\cdot)$ 为符号函数。

开环测试，输入为 $u = \sin t$ ，闭环控制，位置指令为 $\theta_d = \sin t$ 。观测器取式 (5.2) 和式 (5.3)，取 $k_1=500$ ， $k_2=200$ ，仿真结果如图 5-1 所示。分别采用加补偿和不加补偿的 PD 控制，取 $k_p=100$ ， $k_d=10$ ，位置跟踪效果如图 5-2 和图 5-3 所示。

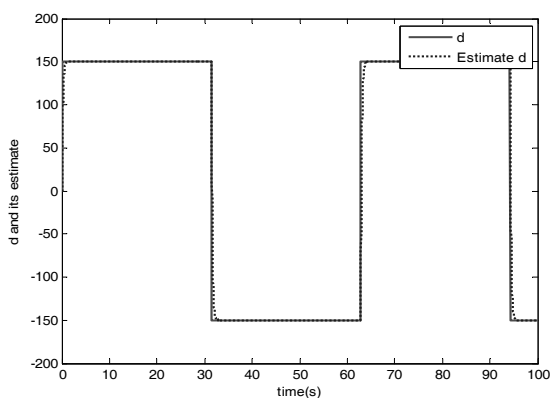


图 5-1 干扰及其观测结果

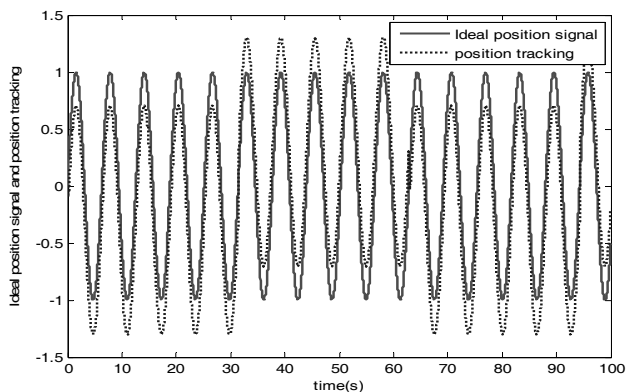


图 5-2 未加补偿时位置跟踪

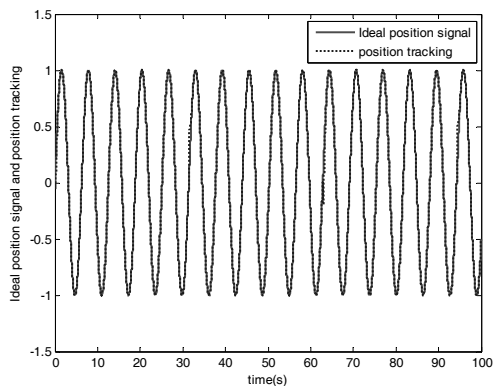
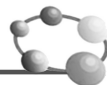


图 5-3 加补偿时位置跟踪

仿真程序

(1) 观测器仿真程序

Simulink 主程序: chap5_1sim.mdl (见图 5-4)

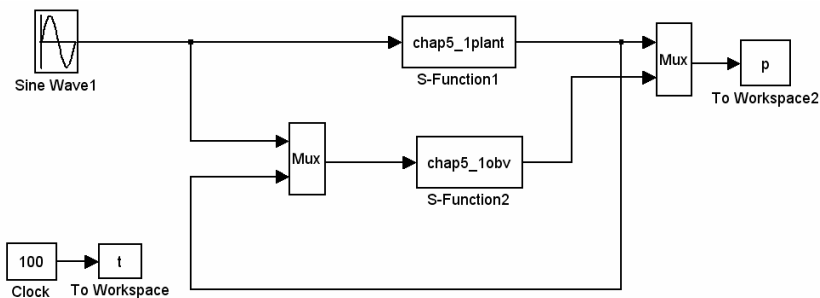


图 5-4 观测器开环测试主程序

观测器 S 函数: chap5_1obv.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
```



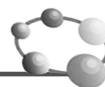

```
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
dth=u(3);

k1=1000;
k2=200;

a=5;b=0.15;
sys(1)=k1*(x(2)-dth);
sys(2)=-x(1)+a*ut-k2*(x(2)-dth)-b*dth;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);      %d estimate
sys(2)=x(2);      %speed estimate
```

被控对象 S 函数: chap5_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0];
str=[];
```



```

ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
b=0.15;
a=5;

d=150*sign(sin(0.1*t));
ddth=-b*x(2)+a*ut-d;

sys(1)=x(2);
sys(2)=ddth;
function sys=mdlOutputs(t,x,u)
d=150*sign(sin(0.1*t));
sys(1)=x(1);
sys(2)=x(2);
sys(3)=d;

```

作图程序: chap5_1plot.m

```

close all;

figure(1);
plot(t,p(:,3),'r','t,p(:,4),'b:','linewidth',2);
xlabel('time(s)');ylabel('d and its estimate');
legend('d','Estimate d');

```

(2) PID 控制仿真程序

Simulink 主程序: chap5_2sim.mdl (见图 5-5)

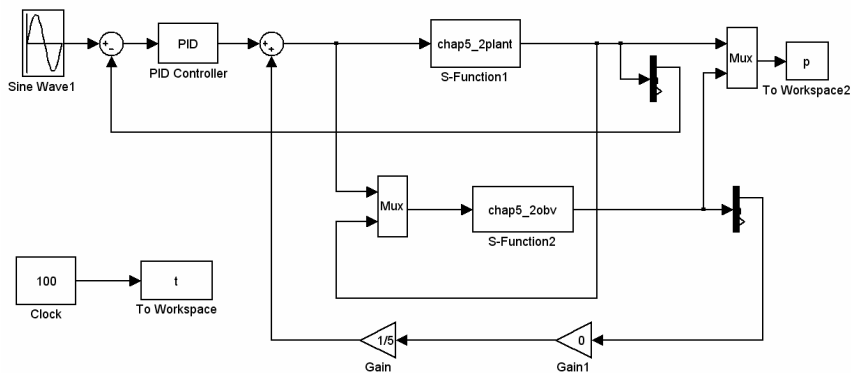


图 5-5 观测器闭环控制主程序

观测器 S 函数: chap5_2obv.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,

```



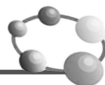
```
sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
dth=u(3);

k1=1000;
k2=200;

a=5;b=0.15;
sys(1)=k1*(x(2)-dth);
sys(2)=-x(1)+a*ut-k2*(x(2)-dth)-b*dth;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);    %d estimate
sys(2)=x(2);    %speed estimate
```

被控对象 S 函数: chap5_2plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
```



```
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
b=0.15;
a=5;

d=150*sign(sin(0.1*t));
ddth=-b*x(2)+a*ut-d;

sys(1)=x(2);
sys(2)=ddth;
function sys=mdlOutputs(t,x,u)
d=150*sign(sin(0.1*t));
sys(1)=x(1);
sys(2)=x(2);
sys(3)=d;
```

作图程序: chap5_2plot.m

```
close all;

figure(1);
plot(t,p(:,3),'r',t,p(:,4),'b:','linewidth',2);
xlabel('time(s)');ylabel('d and its estimate');
legend('d','Estimate d');

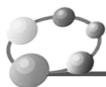
figure(2);
plot(t,sin(t),'r',t,p(:,1),'b:','linewidth',2);
xlabel('time(s)');ylabel('Ideal position signal and position tracking');
legend('Ideal position signal','position tracking');
```



5.2 基于干扰观测器的 PID 控制

5.2.1 干扰观测器基本原理

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，统统等效的控制输入端，即观测出等效干扰，在控制中引入等量的补偿，实现对干扰完全抑制。由 C.J.Kempf 等提出的干扰观测器的基本思想如图 5-6 所示。^[29]

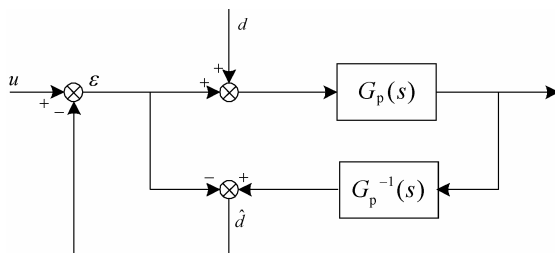


图 5-6 干扰观测器的基本思想

图 5-6 中的 $G_p(s)$ 为对象的传递函数， d 为等效干扰， $\hat{d}$ 为观测干扰， u 为控制输入。由图 5-6，求出等效干扰的估计值 $\hat{d}$ 为

$$\hat{d} = (\varepsilon + d)G_p(s)G_p^{-1}(s) - \varepsilon = d \quad (5.8)$$

式 (5.8) 说明，用上述方法可以实现对干扰的准确估计。

但对于实际的物理系统，观测器式 (5.8) 的实现存在如下问题

- (1) 通常情况下， $G_p(s)$ 的相对阶不为 0，其逆物理上不可实现；
- (2) 对象 $G_p(s)$ 的精确数学模型无法得到；
- (3) 考虑测量噪声的影响，上述方法的观测精度将下降。

C.J.Kempf 等^[29]提出 $\hat{d}$ 的后面串入低通滤波器 $Q(s)$ ，并用名义模型 $G_n(s)$ 的逆 $G_n^{-1}(s)$ 来替代 $G_p^{-1}(s)$ ，得到如图 5-7 所示的框图，其中虚线框内部分为干扰观测器， ξ 为测量噪声， u, d, ξ 为输入信号。图 5-7 的干扰观测器等效框图如图 5-8 所示。

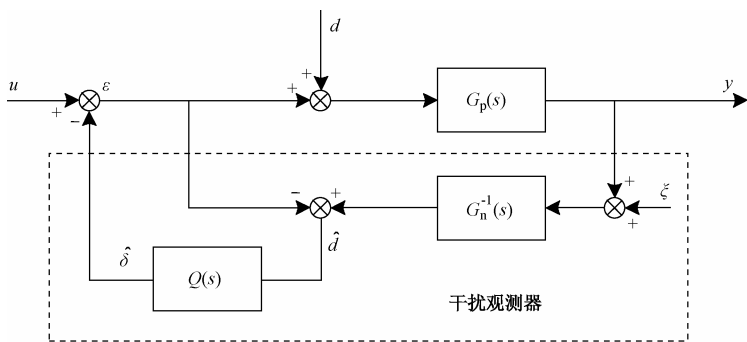


图 5-7 干扰观测器原理框图

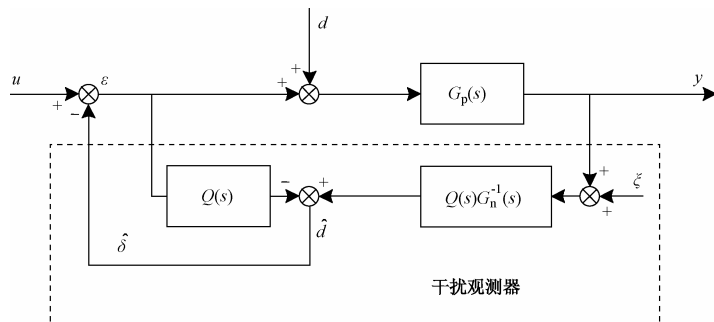
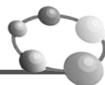


图 5-8 干扰观测器等效框图



5.2.2 干扰观测器的性能分析

C.J.Kempf 等^[29]通过等效变换对观测器的抗干扰性能和滤波性能进行了分析，简要说明如下：

用梅森公式求传递函数。梅森公式的一般形式为

$$G(s) = \frac{\sum_{k=1}^n P_k \Delta_k}{\Delta}$$

式中， $G(s)$ ——待求传递函数；

Δ ——特征式，且 $\Delta = 1 - \sum L_i + \sum L_i L_j - \sum L_i L_j L_k \cdots$ ；

P_k ——从输入端到输出端第 k 条前向通道的总传递函数；

Δ_k ——在特征式 Δ 中，将其与第 k 条前向通道接触的回路所在项除去后余下部分，并称代数余子式。

$\sum L_i$ ——所有各回路的“回路传递函数”之和；

$\sum L_i L_j$ ——两两互不接触的回路，其“回路传递函数”乘积之和；

$\sum L_i L_j L_k$ ——所有三个互不接触的回路，其“回路传递函数”乘积之和。

根据梅森公式，传递函数可由下式求得

$$G(s) = \frac{\sum_{k=1}^n P_k \Delta_k}{\Delta} \quad (5.9)$$

式中， $\sum_{k=1}^n P_k \Delta_k$ 和 Δ 的含义及求取表达式如下。

由图 5-8 可求得

$$\sum_{k=1}^n P_k \Delta_k = G_p(s), \quad \sum L_i = Q(s) - G_n^{-1}(s) Q(s) G_p(s)$$

从而可得到从 u 到 y 的传递函数

$$\begin{aligned} G_{UY}(s) &= \frac{G_p(s)}{1 - [Q(s) - G_n^{-1}(s) Q(s) G_p(s)]} = \frac{G_p(s) G_n(s)}{Q(s) G_p(s) + G_n(s) [1 - Q(s)]} \\ &= \frac{\frac{G_p(s)}{1 - Q(s)}}{1 + \frac{Q(s)}{G_n(s)} \frac{G_p(s)}{1 - Q(s)}} \end{aligned} \quad (5.10)$$

即：

$$G_{UY}(s) = \frac{G_p(s) G_n(s)}{G_n(s) + [G_p(s) - G_n(s)] Q(s)} \quad (5.11)$$

根据式 (5.10)，对图 5-7 做等效变换，可得到干扰观测器原理的简化框图，如图 5-9 所示。

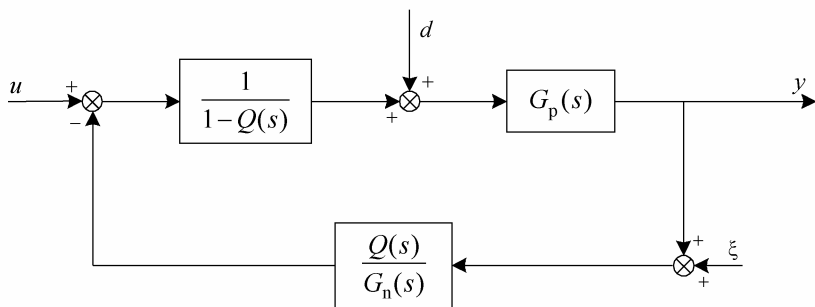


图 5-9 图 5-7 的等效变换

利用梅森公式, 根据图 5-9, 可得

$$G_{DY}(s) = \frac{G_p(s)G_n(s)[1-Q(s)]}{G_n(s) + [G_p(s) - G_n(s)]Q(s)} \quad (5.12)$$

$$G_{\xi Y}(s) = \frac{G_p(s)Q(s)}{G_n(s) + [G_p(s) - G_n(s)]Q(s)} \quad (5.13)$$

$Q(s)$ 是干扰观测器设计中一个非常重要的环节。由图 5-9 可知, $Q(s)$ 的设计必须满足 $Q(s)G_n^{-1}(s)$ 为正则, 即 $Q(s)$ 的相对阶应不小于 $G_n(s)$ 的相对阶; 其次, $Q(s)$ 带宽的设计, 必须同时满足干扰观测器的鲁棒稳定性和干扰抑制能力。

$Q(s)$ 的设计原则为: 在低频段, $Q(s)=1$; 在高频段, $Q(s)=0$ 。具体分析如下。

(1) 在低频时, $Q(s)=1$, 由式 (5.10) 至式 (5.12), 有

$$G_{UY}(s) = G_n(s), \quad G_{DY}(s) = 0, \quad G_{\xi Y}(s) = 1 \quad (5.14)$$

上式说明, 在低频段, 干扰观测器仍使得实际对象的响应与名义模型的响应一致, 即可以实现对低频干扰的有效观测, 从而保证较好的鲁棒性。 $G_{DY}(s)=0$ 说明干扰观测器对于 $Q(s)$ 频带内的低频干扰具有完全的抑制能力, $G_{\xi Y}(s)=1$ 说明干扰观测器对于低频测量噪声非常敏感, 因此, 在实际应用中, 必须考虑采取适当的措施, 减小运动状态测量中的低频噪声。

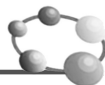
(2) 在高频段, $Q(s)=0$, 由式 (5.11) 和 (5.13), 有

$$G_{UY}(s) = G_p(s), \quad G_{DY}(s) = G_p(s), \quad G_{\xi Y}(s) = 0 \quad (5.15)$$

式 5.15 说明, 在高频时, 干扰观测器对测量噪声不敏感, 可以实现对高频噪声的有效滤除, 但对于对象参数的摄动及外部扰动没有任何抑制作用。

通过上述分析可见, 通过采用低通滤波器设计 $Q(s)$, 可以实现对低频干扰的有效观测和高频噪声的有效滤除, 是一种很有效的工程设计方法。

由简化框图 5-9 可以从另一个角度来理解干扰观测器的作用。在低频段, $Q(s)=1$, 则 $\frac{1}{1-Q(s)} = \infty$, $\frac{Q(s)}{G_n(s)} = G_n^{-1}(s)$, 显然, 加入干扰观测器后, 系统在低频段时的控制相当于高增益控制; 在高频段, $Q(s)=0$, 则 $\frac{1}{1-Q(s)} = 1$, $\frac{Q(s)}{G_n(s)} = 0$, 即前向通道的控制增益为 1, 反馈系数为 0, 则从 u 到 y 之间相当于开环, 其传递函数等于对象的开环传递函数 $G_p(s)$, 干扰观测器的作用消失。



5.2.3 干扰观测器鲁棒稳定性

C.J.Kempf 等^[29]利用补灵敏度函数及鲁棒稳定性定理, 对干扰观测器鲁棒稳定性进行了说明, 下面给出主要分析过程并加以补充。

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

$$G_p(s) = G_n(s)(1 + \Delta(s)) \quad (5.16)$$

式中, $\Delta(s)$ 表明了实际对象频率特性对名义模型的摄动, 通常情况下, 频率增加时, 对象的不确定性也增大, 因此 $|\Delta(j\omega)|$ 表现为 ω 的增函数。

由图 5-9 可得

$$\begin{aligned} \Delta G_{UY}(s) &= \frac{(G_p(s) + \Delta G_p(s))G_n(s)}{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)} - \frac{G_p(s)G_n(s)}{G_n(s) + [G_p(s) - G_n(s)]Q(s)} \\ &= \frac{G_n(s)\Delta G_p(s)G_n(s)(1 - Q(s))}{\{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)\}\{G_n(s) + [G_p(s) - G_n(s)]Q(s)\}} \end{aligned}$$

则

$$\begin{aligned} \frac{\Delta G_{UY}(s)}{G_{UY}(s)} &= \frac{G_n(s)\Delta G_p(s)G_n(s)(1 - Q(s))}{\{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)\}\{G_n(s) + [G_p(s) - G_n(s)]Q(s)\}} \\ &\quad \times \frac{G_n(s) + [G_p(s) - G_n(s)]Q(s)}{G_p(s)G_n(s)} = \frac{\Delta G_p(s)G_n(s)(1 - Q(s))}{\{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)\}G_p(s)} \end{aligned}$$

则

$$\begin{aligned} \frac{\Delta G_{UY}(s)/G_{UY}(s)}{\Delta G_p(s)/G_p(s)} &= \frac{\Delta G_p(s)G_n(s)(1 - Q(s))}{\{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)\}G_p(s)} \times \frac{G_p(s)}{\Delta G_p(s)} \\ &= \frac{G_n(s)(1 - Q(s))}{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)} \end{aligned}$$

则灵敏度函数 $S(s)$ 为

$$\begin{aligned} S(s) &= \lim_{\Delta G_p(s) \rightarrow 0} \frac{\Delta G_{UY}(s)/G_{UY}(s)}{\Delta G_p(s)/G_p(s)} = \lim_{\Delta G_p(s) \rightarrow 0} \frac{G_n(s)(1 - Q(s))}{G_n(s) + [G_p(s) + \Delta G_p(s) - G_n(s)]Q(s)} \\ &= \frac{G_n(s)(1 - Q(s))}{G_n(s) + [G_p(s) - G_n(s)]Q(s)} \end{aligned} \quad (5.17)$$

在低频段, 认为 $G_p(s) = G_n(s)$, 将上式中的 $G_p(s)$ 用 $G_n(s)$ 来替代, 则有

$$S(s) = 1 - Q(s) \quad (5.18)$$

则补灵敏度函数 $T(s)$ 为

$$T(s) = 1 - S(s) = Q(s) \quad (5.19)$$

则由鲁棒稳定性定理^[30], 系统鲁棒稳定的充分必要条件是

$$\|\Delta(j\omega)T(j\omega)\|_\infty = \|\Delta(j\omega)Q(j\omega)\|_\infty \leq 1 \quad (5.20)$$

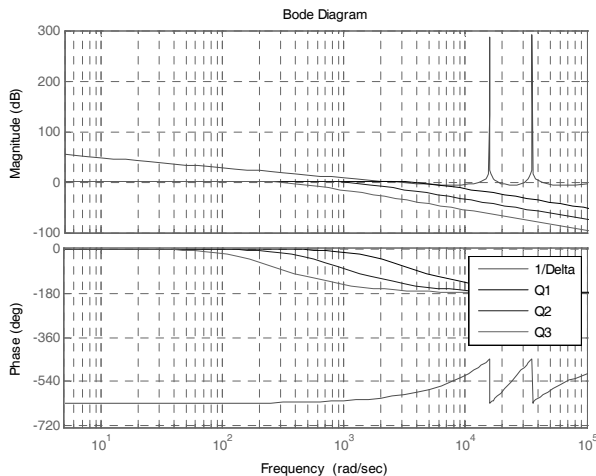
式中, $\|\cdot\|_\infty$ 为 H_∞ 范数。

通过 $Q(s)$ 的设计, 可实现鲁棒性要求。式子 (5.20) 可为

$$\|Q(j\omega)\|_\infty \leq \frac{1}{\|\Delta(j\omega)\|_\infty} \quad (5.21)$$



式 (5.21) 是 $Q(s)$ 设计的基础。以 $Q(s) = \frac{3\tau s + 1}{\tau^3 s^3 + 3\tau^2 s^2 + 3\tau s + 1}$ 为例, 假设延迟是唯一不确定部分, 此时取 $G_p(s) = \frac{B(s)}{A(s)} e^{-\tau s}$, $G_n(s) = \frac{B(s)}{A(s)}$, $\Delta(s) = e^{-\tau s} - 1$, 分别取 $\tau_1 = 0.00035$, $\tau_2 = 0.00125$, $\tau_3 = 0.00425$, 对应的低通滤波器分别为 $Q_1(s)$ 、 $Q_2(s)$ 和 $Q_3(s)$, 仿真程序见 chap5_3Q.m, 仿真结果见图 5-10。可见 $Q_2(s)$ 和 $Q_3(s)$ 满足式 (5.21) 要求。


图 5-10 $Q(s)$ 的选择

5.2.4 低通滤波器 $Q(s)$ 的设计

$Q(s)$ 为干扰观测器设计中的一个非常重要的环节, 目前较为流行的设计方法是由 H. S. Lee^[31]提出的, $Q(s)$ 的表达式为

$$Q(s) = \frac{\sum_{k=0}^M \alpha_k (\tau s)^k}{(\tau s + 1)^N} \quad (5.22)$$

式中, $\alpha_k = \frac{N!}{(N-k)!k!}$ 为系数, N 为分母的阶数, M 为分子的阶数, $N-M$ 为相对阶。

$Q(s)$ 滤波器的设计归结为确定参数 N 、 M 和 τ , 应从下面 4 个方面来考虑:^[31]

(1) N 和 M 的选择首先要满足使 $Q(s)G_n^{-1}(s)$ 正则, 物理可实现。

(2) $Q(s)$ 滤波器的阶数不应太高。随着 $Q(s)$ 阶数的升高, $\|Q(s)\|_{\infty}$ 的值也不断增大, 由鲁棒稳定条件 (见式 (5.20)) 可知, 干扰观测器稳定性将变差。另外, $Q(s)$ 阶数的升高, 还会使控制器的运算量加大, 对实时控制不利。

(3) 参数 τ 的取值决定了 $Q(s)$ 的带宽。 τ 越小, $Q(s)$ 的频带越宽, 系统抑制外干扰的能力越强, 但对测量噪声的敏感性增大。反之, τ 越大, $Q(s)$ 的频带越窄, 干扰观测器对测量噪声越不敏感, 但对外干扰的抑制能力也越弱。因此, 参数 τ 的选择实际上是在干扰观测器的对外干扰的抑制能力及对测量噪声的敏感性两者之间的折中。



5.2.5 仿真实例

设实际被控对象为 $G_p(s) = \frac{1}{0.003s^2 + 0.067s}$ ，其名义模型取 $G_n(s) = \frac{1}{0.0033s^2 + 0.0673s}$ 。

如图 5-11 所示为某伺服系统的实测频率特性 $G_p(s)$ 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。

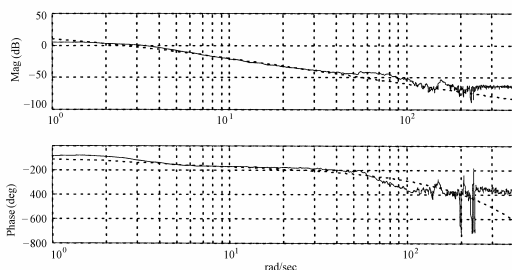


图 5-11 实测被控对象 $G_p(s)$ 与名义模型 $G_n(s)$ 频率特性

忽略非建模动态和不确定性的影响，名义模型可描述为

$$G_n(s) = \frac{1}{s(J_n s + b_n)} \quad (5.23)$$

式中， J_n 为等效惯性力矩， b_n 为等效阻尼系数。

采用分母为三阶，分子为一阶的低通滤波器，即 $N=3$ ， $M=1$ ， $k=0,1$ ，则根据式 (5.22)

得 $\alpha_0 = \frac{3!}{(3-0)!0!} = 1$ ， $\alpha_1 = \frac{3!}{(3-1)!1!} = 3$ ，则低通滤波器表达式为

$$Q(s) = \frac{\sum_{k=0}^M \alpha_k (\tau s)^k}{(\tau s + 1)^N} = \frac{\alpha_0 + \alpha_1 \tau s}{(\tau s + 1)^3} = \frac{3\tau s + 1}{\tau^3 s^3 + 3\tau^2 s^2 + 3\tau s + 1} \quad (5.24)$$

先运行参数初始化程序，实现对象参数、名义模型参数和观测器参数的初始化。干扰观测器采用式 (5.24)，取 $\tau = 0.001$ ，干扰信号为 $d(t) = 3\sin(2\pi t)$ ，对于干扰信号的观测结果如图 5-12 所示。取指令信号为 $y_d(t) = \sin t$ ，PD 控制器中取 $k_p = 10$ ， $k_d = 5$ 。通过选择增益模块 Gain 为 1 或 0，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 5-13 和图 5-14 所示。

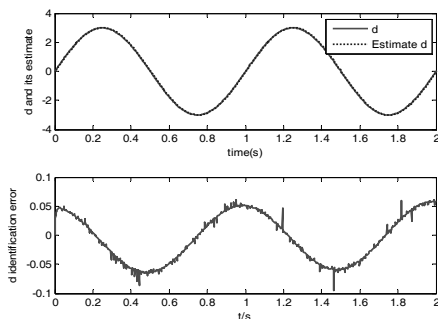


图 5-12 干扰信号观测结果及误差

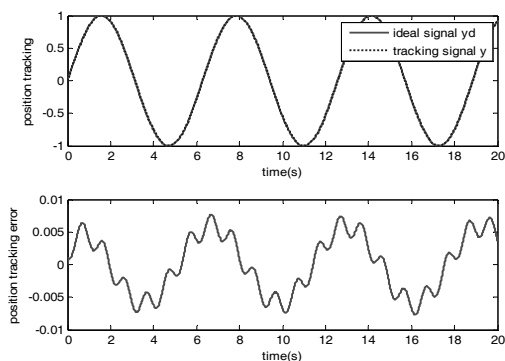


图 5-13 采用干扰观测器的 PID 控制

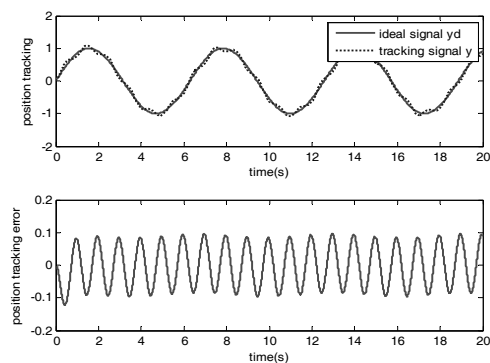


图 5-14 未采用干扰观测器的 PID 控制

仿真程序

(1) $Q(s)$ 的选择程序: chap5_3Q.m

```
clear all;
close all;

tol=400*10^(-6);

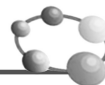
[np,dp]=pade(tol,6);

delay=tf(np,dp);
delta=tf(np,dp)-1;
sys=1/delta;

figure(1);
bode(1/delta,'r',{5,10^5});grid on;

tol1=0.00035;
Q1=tf([3*tol1,1],[tol1^3,3*tol1^2,3*tol1,1]);
hold on;
bode(Q1,'k');

tol3=0.00125;
```



```

Q3=tf([3*tol3,1],[tol3^3,3*tol3^2,3*tol3,1]);
hold on;
bode(Q3,'b');

tol4=0.00425;
Q4=tf([3*tol4,1],[tol4^3,3*tol4^2,3*tol4,1]);
hold on;
bode(Q4,'g');
legend('1/Delta','Q1','Q2','Q3');

```

(2) 干扰观测器仿真程序

初始化程序: chap5_3int.m

```

clear all;
close all;
Jp=0.0030;bp=0.067;
Jn=0.0033;bn=0.0673;
Gp=tf([1],[Jp,bp,0]); %Practical plant
Gn=tf([1],[Jn,bn,0]); %Nominal plant

tol=0.001;
Q=tf([3*tol,1],[tol^3,3*tol^2,3*tol,1]);
bode(Q);
dcgain(Q)
QGn=Q/Gn;
[num,den]=tfdata(QGn,'v');

```

Simulink 主程序: chap5_3sim.mdl (见图 5-15)

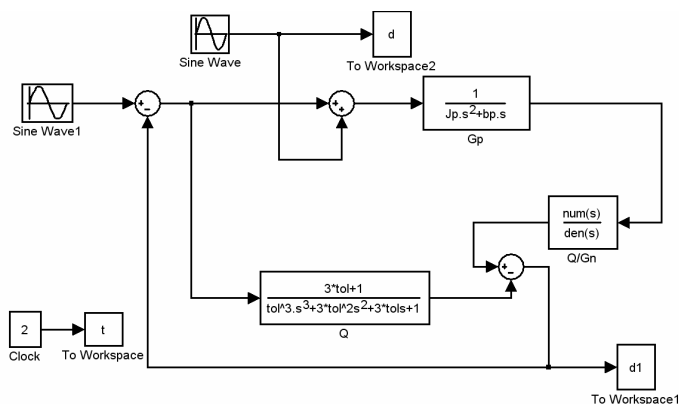


图 5-15 干扰观测器闭环测试主程序

作图程序: chap5_3plot.m

```

close all;
figure(1);
subplot(211);
plot(t,d(:,1),'r',t,d1(:,1),'b','linewidth',2);

```



```
xlabel('time(s)');ylabel('d and its estimate');  
legend('d','Estimate d');  
subplot(212);  
plot(t,d(:,1)-d1(:,1),'r','linewidth',2);  
xlabel('t/s');ylabel('d identification error');
```

(3) 基于干扰观测器的PID控制仿真程序

初始化程序: chap5_4int.m

```
clear all;  
close all;  
Jp=0.0030;bp=0.067;  
Jn=0.0033;bn=0.0673;  
Gp=tf([1],[Jp,bp,0]); %Practical plant  
Gn=tf([1],[Jn,bn,0]); %Nominal plant  
  
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;  
  
OD3 = Q*Gn;  
  
[num,den]=tfdata(OD2,'v');  
[num1,den1]=tfdata(OD1,'v');  
[num2,den2]=tfdata(OD3,'v');
```

Simulink 主程序: chap5_4sim.mdl (见图 5-16)

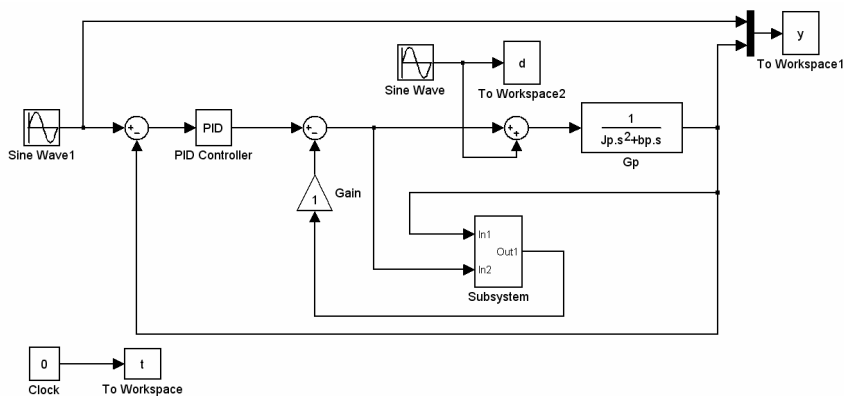
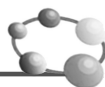


图 5-16 干扰观测器闭环测试主程序

作图程序: chap5_4plot.m

```
close all;  
figure(1);
```



```
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'b','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('ideal signal yd','tracking signal y');
subplot(212);
plot(t,y(:,1)-y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('position tracking error');

figure(2);
subplot(211);
plot(t,d(:,1),'r',t,d1(:,1),'b','linewidth',2);
xlabel('time(s)');ylabel('d and its estimate');
legend('d','Estimate d');
subplot(212);
plot(t,d(:,1)-d1(:,1),'r','linewidth',2);
xlabel('t/s');ylabel('d identification error');
```

5.3 基于扩张观测器的PID控制

5.3.1 扩张观测器的设计

考虑如下对象

$$J\ddot{\theta} = u(t) - d(t) \quad (5.25)$$

式中， J 为转动惯量， u 为控制输入， θ 为实际角度， $d(t)$ 为外加干扰。

式(5.25)可写为

$$\ddot{\theta} = bu(t) + f(t) \quad (5.26)$$

式中， $b = \frac{1}{J}$ 为已知， $f(t) = -\frac{1}{J}d(t)$ 为未知， $f(\cdot)$ 的导数存在且有界。

式(5.26)可写为

$$\dot{x} = Ax + B(bu + f(t)) \quad (5.27)$$

$$y = Cx \quad (5.28)$$

式中， $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$ ， $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ ， $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ， $C = [1 \quad 0]$ ， $|\dot{f}(\cdot)| \leq L$ 。

王新华等^[32]，将扩张观测器设计为

$$\dot{\hat{x}}_1 = \hat{x}_2 + \frac{\alpha_1}{\varepsilon}(y - \hat{x}_1) \quad (5.29)$$

$$\dot{\hat{x}}_2 = bu + \hat{\sigma} + \frac{\alpha_2}{\varepsilon^2}(y - \hat{x}_1) \quad (5.30)$$

$$\dot{\hat{\sigma}} = \frac{\alpha_3}{\varepsilon^3}(y - \hat{x}_1) \quad (5.31)$$

采用该扩张观测器，可实现



当 $t \rightarrow \infty$ 时, $\hat{x}_1(t) \rightarrow x_1(t), \hat{x}_2(t) \rightarrow x_2(t), \hat{\sigma}(t) \rightarrow f(\theta, \dot{\theta}, t)$ 。

式中, $\hat{x}_1, \hat{x}_2$ 和 $\hat{\sigma}$ 为观测器状态, $\varepsilon > 0$, α_1, α_2 和 α_3 为正实数, 多项式 $s^3 + \alpha_1 s^2 + \alpha_2 s + \alpha_3$ 满足 Hurwitz 条件。

5.3.2 扩张观测器的分析

H.K.Khalil 等^[33]对扩张观测器的收敛性进行了分析, 主要证明思路如下: 定义

$$\eta = [\eta_1 \quad \eta_2 \quad \eta_3]^T$$

式中

$$\eta_1 = \frac{x_1 - \hat{x}_1}{\varepsilon^2}, \quad \eta_2 = \frac{x_2 - \hat{x}_2}{\varepsilon}, \quad \eta_3 = f - \hat{\sigma}$$

由于

$$\begin{aligned} \varepsilon \dot{\eta}_1 &= \frac{\dot{x}_1 - \dot{\hat{x}}_1}{\varepsilon} = \frac{1}{\varepsilon} \left(x_2 - \left(\hat{x}_2 + \frac{\alpha_1}{\varepsilon} (y - \hat{x}_1) \right) \right) \\ &= \frac{1}{\varepsilon} \left(x_2 - \hat{x}_2 - \frac{\alpha_1}{\varepsilon} (y - \hat{x}_1) \right) = -\frac{\alpha_1}{\varepsilon^2} (x_1 - \hat{x}_1) + \frac{1}{\varepsilon} (x_2 - \hat{x}_2) = -\alpha_1 \eta_1 + \eta_2 \\ \varepsilon \dot{\eta}_2 &= \varepsilon \frac{\dot{x}_2 - \dot{\hat{x}}_2}{\varepsilon} = \left(bu + f(\cdot) - \left(bu + \hat{\sigma} + \frac{\alpha_2}{\varepsilon^2} (y - \hat{x}_1) \right) \right) \\ &= \left(f(\cdot) - \hat{\sigma} - \frac{\alpha_2}{\varepsilon^2} (y - \hat{x}_1) \right) = -\frac{\alpha_2}{\varepsilon^2} (x_1 - \hat{x}_1) + (f - \hat{\sigma}) = -\alpha_2 \eta_1 + \eta_3 \\ \varepsilon \dot{\eta}_3 &= \varepsilon (\dot{f} - \dot{\hat{\sigma}}) = \varepsilon \left(\dot{f} - \frac{\alpha_3}{\varepsilon^3} (y - \hat{x}_1) \right) = \varepsilon \dot{f} - \frac{\alpha_3}{\varepsilon^2} (y - \hat{x}_1) = -\alpha_3 \eta_1 + \varepsilon \dot{f} \end{aligned}$$

则观测误差状态方程可写为

$$\varepsilon \dot{\eta} = \bar{A} \eta + \varepsilon \bar{B} \dot{f} \quad (5.32)$$

式中

$$\bar{A} = \begin{bmatrix} -\alpha_1 & 1 & 0 \\ -\alpha_2 & 0 & 1 \\ -\alpha_3 & 0 & 0 \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

矩阵 $\bar{A}$ 的特征方程为

$$|\lambda I - \bar{A}| = \begin{vmatrix} \lambda + \alpha_1 & -1 & 0 \\ \alpha_2 & \lambda & -1 \\ \alpha_3 & 0 & \lambda \end{vmatrix} = 0$$

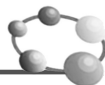
则

$$(\lambda + \alpha_1) \lambda^2 + \alpha_3 + \alpha_2 \lambda = 0$$

且

$$\lambda^3 + \alpha_1 \lambda^2 + \alpha_2 \lambda + \alpha_3 = 0 \quad (5.33)$$

通过选择 $\alpha_i (i=1,2,3)$ 使 $\bar{A}$ 为 Hurwitz, 则对于任意给定的对称正定阵 Q , 存在对称正定阵 P 满足如下 Lyapunov 方程:



$$\bar{A}^T P + P \bar{A} + Q = 0 \quad (5.34)$$

定义观测器的 Lyapunov 函数为

$$V_o = \varepsilon \eta^T P \eta \quad (5.35)$$

则

$$\begin{aligned} \dot{V}_o &= \varepsilon \dot{\eta}^T P \eta + \varepsilon \eta^T P \dot{\eta} \\ &= (\bar{A} \eta + \varepsilon \bar{B} \dot{f})^T P \eta + \eta^T P (\bar{A} \eta + \varepsilon \bar{B} \dot{f}) \\ &= \eta^T \bar{A}^T P \eta + \varepsilon (\bar{B} \dot{f})^T P \eta + \eta^T P \bar{A} \eta + \varepsilon \eta^T P \bar{B} \dot{f} \\ &= \eta^T (\bar{A}^T P + P \bar{A}) \eta + 2\varepsilon \eta^T P \bar{B} \dot{f} \\ &\leq -\eta^T Q \eta + 2\varepsilon \|P \bar{B}\| \cdot \|\eta\| \cdot |\dot{f}| \end{aligned}$$

且

$$\dot{V}_o \leq -\lambda_{\min}(Q) \|\eta\|^2 + 2\varepsilon L \|P \bar{B}\| \|\eta\|$$

式中, $\lambda_{\min}(Q)$ 为 Q 的最小特征值。

由 $\dot{V}_o \leq 0$ 可得观测器的收敛条件为

$$\|\eta\| \leq \frac{2\varepsilon L \|P \bar{B}\|}{\lambda_{\min}(Q)} \quad (5.36)$$

由式 (5.36) 可见, η 与 ε 有关, 如果取 ε 很小, 可使观测器的收敛误差减小。

注意:

(1) 如果扩张观测器的初始值与对象的初值不同, 对于很小的 ε , 将产生峰值现象, 造成观测器的收敛效果差。为了防止峰值现象, 设计 ε 为^[25]

$$\frac{1}{\varepsilon} = R = \begin{cases} 100t^3, & 0 \leq t \leq 1 \\ 100, & t > 1 \end{cases} \quad (5.37)$$

或

$$\frac{1}{\varepsilon} = R = \begin{cases} \mu \frac{1 - e^{-\lambda_1 t}}{1 + e^{-\lambda_2 t}}, & 0 \leq t \leq t_{\max} \\ \mu, & t > t_{\max} \end{cases} \quad (5.38)$$

式中 μ , λ_1 和 λ_2 为正实数。

例如, 取 $\lambda_1 = \lambda_2 = 50$, $\mu = 100$, 运行程序 chap5_5sim.mdl, R 和 ε 的变化如图 5-17 所示。

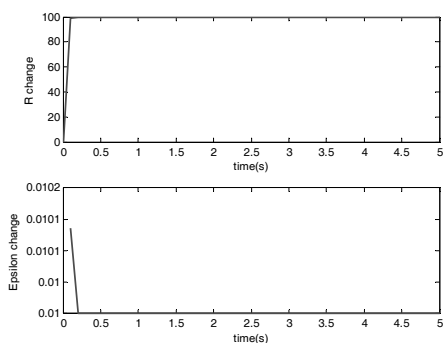
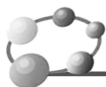


图 5-17 R 和 ε 的变化



(2) 如果实测信号含有噪声, 对于非常小的 ε , 将会产生很大的观测误差。为了防止这种现象, 参考文献[34]提出了切换增益 ε 的设计方法。

(3) $\alpha_i (i=1,2,3)$ 的设计。对于 $\lambda^3 + \alpha_1 \lambda^2 + \alpha_2 \lambda + \alpha_3 = 0$, 通过选择 $(\lambda+1)(\lambda+2)(\lambda+3)=0$, 则 $\lambda^3 + 6\lambda^2 + 11\lambda + 6 = 0$, 从而 $\alpha_1 = 6$, $\alpha_2 = 11$, $\alpha_3 = 6$ 。

5.3.3 仿真实例

考虑如下对象

$$J\ddot{\theta} = u(t) - d(t) \quad (5.39)$$

式中, J 为转动惯量, u 为控制输入, θ 为实际角度, $d(t)$ 为外加干扰, θ 为角度信号。

在扩张观测器仿真中, 对象信息取 $d(t) = 3\sin(t)$, $J = 10$, $u(t) = 0.1\sin t$, 取观测器参数为 $\alpha_1 = 6$, $\alpha_2 = 11$, $\alpha_3 = 6$, 为了防止峰值现象, 按式 (5.37) 或式 (5.38) 设计 ε , 以连续系统仿真为例, 仿真结果如图 5-18 至图 5-20 所示。

为了显示扩张观测器的控制补偿效果, 以连续系统仿真为例, 考虑如下对象

$$\ddot{\theta} = 100u(t) - 25\dot{\theta} - 100\text{sgn}(\dot{\theta}) \quad (5.40)$$

对应于式 (2.26), $b = 100$, $f(t) = -25\dot{\theta} - 100\text{sgn}(\dot{\theta})$ 。基于扩张观测器实现状态和干扰的观测及补偿, 采用 PID 控制, 取 $k_p = 10$, $k_d = 10$, 为了防止峰值现象, 按式 (5.37) 或式 (5.38) 设计 ε , 仿真结果如图 5-21 至图 5-23 所示。以式 (5.39) 为被控对象, 离散形式的控制系统仿真见程序 chap5_9.m。

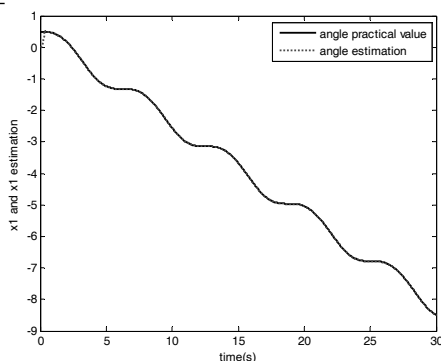


图 5-18 θ 及其观测信号

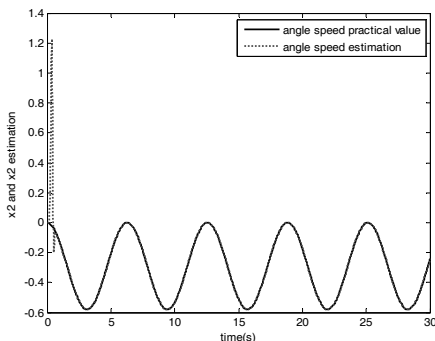


图 5-19 $\dot{\theta}$ 及其观测信号

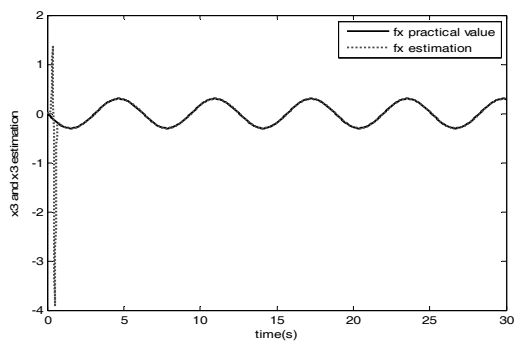
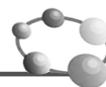


图 5-20 不确定性及其观测结果

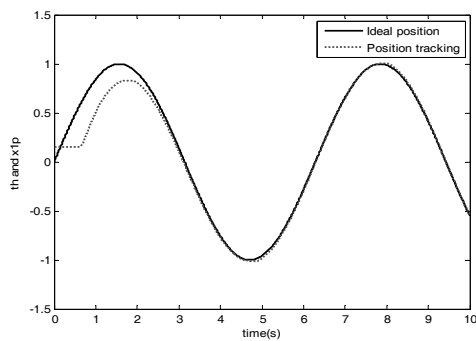


图 5-21 正弦位置跟踪

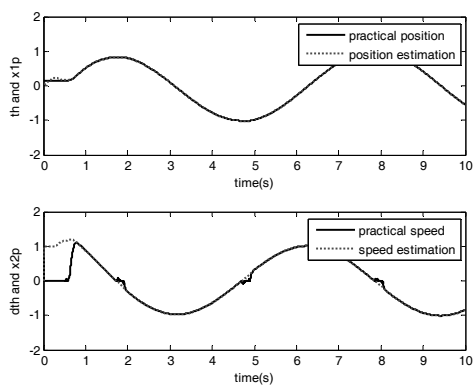


图 5-22 位置、速度及其观测

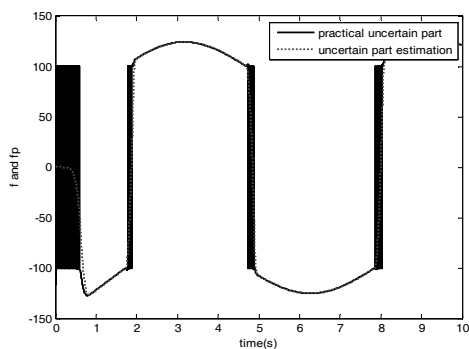


图 5-23 不确定部分及其观测



仿真程序

1. 峰值抑制

主程序: chap5_5.sim.mdl (见图 5-24)

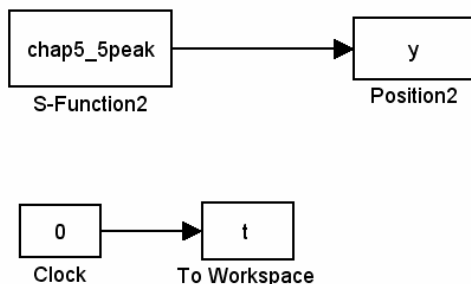
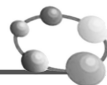


图 5-24 峰值抑制主程序

峰值抑制 S 函数: chap5_5peak.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1, 2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
Lambda=50;
R=100*(1-exp(-Lambda*t))/(1+exp(-Lambda*t));
Epsilon=1/R;
sys(1)=R;
sys(2)=Epsilon;
```



作图程序: chap5_5plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('R change');
subplot(212);
plot(t,y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('Epsilon change');
```

2. 扩张观测器

(1) 连续系统仿真

主程序: chap5_6sim.mdl (见图 5-25)

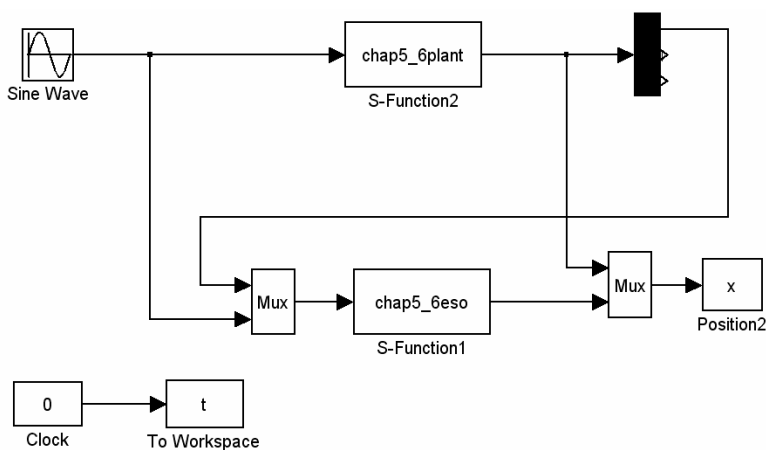


图 5-25 扩张观测器开环测试主程序

扩张观测器 S 函数: chap5_6eso.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
```



```
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
y=u(1);
ut=u(2);

J=10;
b=1/J;

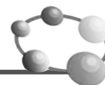
alfa1=6;alfa2=11;alfa3=6;

M=2;
if M==1
    epc=0.01;
elseif M==2
    if t<=1;
        R=100*t^3;
    elseif t>1;
        R=100;
    end
    epc=1/R;
elseif M==3
    nmn=0.1;
    R=100*(1-exp(-nmn*t))/(1+exp(-nmn*t));
    epc=1/R;
end

e=y-x(1);
sys(1)=x(2)+alfa1/epc*e;
sys(2)=b*ut+x(3)+alfa2/epc^2*e;
sys(3)=alfa3/epc^3*e;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
```

对象 S 函数: chap5_6plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
```



```

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.5;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
J=10;
ut=u(1);

d=3.0*sin(t);
sys(1)=x(2);
sys(2)=1/J*(ut-d);
function sys=mdlOutputs(t,x,u)
J=10;
d=3.0*sin(t);
f=-d/J;
sys(1)=x(1);
sys(2)=x(2);
sys(3)=f;

```

作图程序: chap5_6plot.m

```

close all;

figure(1);
plot(t,x(:,1),'k',t,x(:,4),'r','linewidth',2);
xlabel('time(s)');ylabel('x1 and x1 estimation');
legend('angle practical value','angle estimation');

```



```
figure(2);
plot(t,x(:,2),'k',t,x(:,5),'r','linewidth',2);
xlabel('time(s)');ylabel('x2 and x2 estimation');
legend('angle speed practical value','angle speed estimation');

figure(3);
plot(t,x(:,3),'k',t,x(:,6),'r','linewidth',2);
xlabel('time(s)');ylabel('x3 and x3 estimation');
legend('fx practical value','fx estimation');
```

(2) 离散系统仿真

主程序: chap5_7.m

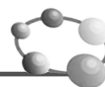
```
%Discrete ESO
clear all;
close all;
ts=0.001; %Sampling time
xk=[0.15 0];
x1p_1=0;x2p_1=0;x3p_1=0;
u_1=0;x1_1=0;
for k=1:1:3000
time(k) = k*ts;
u(k)=sin(2*pi*k*ts);

tSpan=[0 ts];
para=[u(k) time(k)]; %D/A
[t,xx]=ode45('chap5_7plant',tSpan,xk,[],para); %Plant
xk=xx(length(xx),:); %A/D
x1(k)=xk(1);
x2(k)=xk(2);
th(k)=x1(k);

J=10;
dt(k)=3.0*sin(time(k));
fx(k)=-1/J*dt(k);
b=1/J;

h1=6;h2=11;h3=6;

M=3;
if M==1
epc=0.01;
elseif M==2
if time(k)<=1;
R=100*time(k)^3;
elseif time(k)>1;
```



```

        R=100;
    end
    epc=1/R;
elseif M==3
    nmn=1.0;
    R=100*(1-exp(-nmn*time(k)))/(1+exp(-nmn*time(k)));
    epc=1/R;
end
%Extended observer
x1p(k)=x1p_1+ts*(x2p_1-h1/epc*(x1p_1-th(k)));
x2p(k)=x2p_1+ts*(x3p_1-h2/epc^2*(x1p_1-th(k))+b*u(k));
x3p(k)=x3p_1+ts*(-h3/epc^3*(x1p_1-th(k)));

fxp(k)=x3p(k);

u_1=u(k);
x1_1=x1(k);
x1p_1=x1p(k);
x2p_1=x2p(k);
x3p_1=x3p(k);
end
figure(1);
subplot(211);
plot(time,th,'r',time,x1p,':');
xlabel('time(s)');ylabel('x1 and x1p');
subplot(212);
plot(time,x2,'r',time,x2p,':');
xlabel('time(s)');ylabel('x2 and x2p');

figure(2);
plot(time,fx,'r',time,fxp,'b');
xlabel('time(s)');ylabel('f and fp');

```

对象 S 函数: chap5_7plant.m

```

function dx=Plant(t,x,flag,para)
dx=zeros(2,1);
J=10;
ut=para(1);
t=para(2);

dt=3.0*sin(t);
dx(1)=x(2);
dx(2)=1/J*(ut-dt);

```

3. 基于扩张观测器的 PID 控制

(1) 连续系统仿真

主程序: chap5_8sim.mdl (见图 5-26)

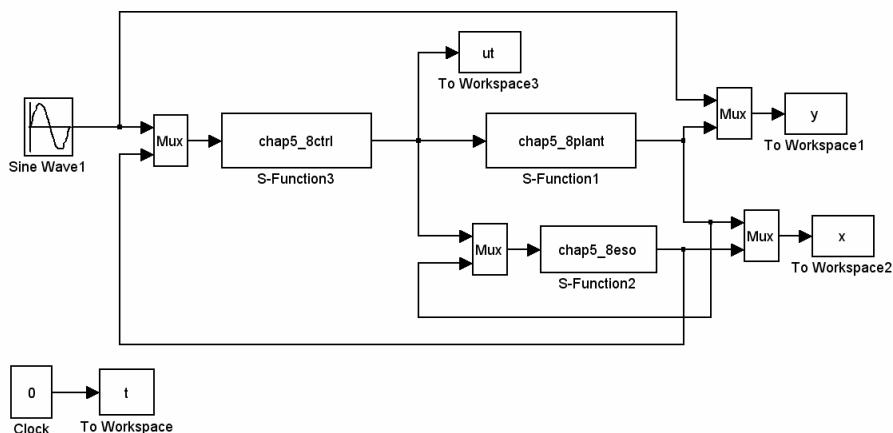


图 5-26 扩张观测器闭环控制主程序

控制器 S 函数: chap5_8ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
yd=u(1);
dyd=cos(t);
yp=u(2);
dyp=u(3);
fp=u(4);
```



```
e=yd-yp;
de=dyd-dyp;

kp=10;kd=10;
M=1;
if M==1           %With Compensation
    b=100;
    ut=kp*e+kd*de-1/b*fp;
elseif M==2       %Without Compensation
    ut=kp*e+kd*de;
end
sys(1)=ut;
```

扩张观测器 S 函数: chap5_8eso.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
th=u(2);

h1=6;h2=11;h3=6;
M=2;
if M==1
```



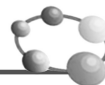
```
    epc=0.01;
elseif M==2
    if t<=1;
        R=100*t^3;
    elseif t>1;
        R=100;
    end
    epc=1/R;
elseif M==3
    nmn=0.1;
    R=100*(1-exp(-nmn*t))/(1+exp(-nmn*t));
    epc=1/R;
end

sys(1)=x(2)-h1/epc*(x(1)-th);
sys(2)=x(3)-h2/epc^2*(x(1)-th)+100*ut;
sys(3)=-h3/epc^3*(x(1)-th);
function sys=mdlOutputs(t,x,u)
fp=x(3);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fp;
```

对象 S 函数: chap5_8plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
```



```
sys=simsize(sizes);
x0=[0.15;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
b=100;

dt=100*sign(x(2));
fx=-25*x(2)-dt;

sys(1)=x(2);
sys(2)=fx+b*ut;
function sys=mdlOutputs(t,x,u)
dt=100*sign(x(2));
fx=-25*x(2)-dt;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

作图程序: chap5_8plot.m

```
close all;

figure(1);
plot(t,y(:,1),'k',t,y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('th and x1p');
legend('Ideal position','Position tracking');

figure(2);
subplot(211);
plot(t,x(:,1),'k',t,x(:,4),'r','linewidth',2);
xlabel('time(s)');ylabel('th and x1p');
legend('practical position','position estimation');
subplot(212);
plot(t,x(:,2),'k',t,x(:,5),'r','linewidth',2);
xlabel('time(s)');ylabel('dth and x2p');
legend('practical speed','speed estimation');

figure(3);
plot(t,x(:,3),'k',t,x(:,6),'r','linewidth',2);
xlabel('time(s)');ylabel('f and fp');
legend('practical uncertain part','uncertain part estimation');

figure(4);
plot(t,ut(:,1),'r','linewidth',2);
```



```
xlabel('time(s)');ylabel('Control input');
```

(2) 离散系统仿真

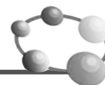
主程序: chap5_9.m

```
%PID Control Based on Discrete ESO
clear all;
close all;
ts=0.001; %Sampling time
xk=[0.15 0];
x1p_1=0;x2p_1=0;x3p_1=0;
u_1=0;x1_1=0;
for k=1:1:3000
time(k) = k*ts;

r(k)=sin(2*pi*k*ts);
dr(k)=2*pi*cos(2*pi*k*ts);
e(k)=r(k)-x1_1;
de(k)=dr(k)-x2p_1;

kp=1500;kd=100;
fxp(k)=x3p_1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
J=0.10;
dt(k)=3.0*sin(time(k));
b=1/J;
fx(k)=-1/J*dt(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M=1;
if M==1 %With Compensation
    u(k)=kp*e(k)+kd*de(k)-1/b*fxp(k);
elseif M==2 %Without Compensation
    u(k)=kp*e(k)+kd*de(k);
end

tSpan=[0 ts];
para=[u(k) time(k)]; %D/A
[t,xx]=ode45('chap5_9plant',tSpan,xk,[],para); %Plant
xk=xx(length(xx),:); %A/D
x1(k)=xk(1);
x2(k)=xk(2);
th(k)=x1(k);
%Extended observer
h1=6;h2=11;h3=6;
M=2;
if M==1
    epc=0.01;
```



```

elseif M==2
    if time(k)<=1;
        R=100*time(k)^3;
    elseif time(k)>1;
        R=100;
    end
    epc=1/R;
elseif M==3
    nmn=1.0;
    R=100*(1-exp(-nmn*time(k)))/(1+exp(-nmn*time(k)));
    epc=1/R;
end

x1p(k)=x1p_1+ts*(x2p_1-h1/epc*(x1p_1-th(k)));
x2p(k)=x2p_1+ts*(x3p_1-h2/epc^2*(x1p_1-th(k))+b*u(k));
x3p(k)=x3p_1+ts*(-h3/epc^3*(x1p_1-th(k)));

fxp(k)=x3p(k);

u_1=u(k);
x1_1=x1(k);
x1p_1=x1p(k);
x2p_1=x2p(k);
x3p_1=x3p(k);
end
figure(1);
plot(time,r,'k',time,th,'r','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('Ideal position','Position tracking');

figure(2);
subplot(211);
plot(time,th,'k',time,x1p,'r','linewidth',2);
xlabel('time(s)');ylabel('x1 and x1p');
subplot(212);
plot(time,x2,'k',time,x2p,'r','linewidth',2);
xlabel('time(s)');ylabel('x2 and x2p');
figure(3);
plot(time,fx,'k',time,fxp,'r','linewidth',2);
xlabel('time(s)');ylabel('f and fp');

```

对象 S 函数: chap5_9plant.m

```

function dx=Plant(t,x,flag,para)
dx=zeros(2,1);
J=0.10;
ut=para(1);

```



```

t=para(2);

dt=3.0*sin(t);

b=1/J;
fx=-1/J*dt;
dx(1)=x(2);
dx(2)=fx+b*ut;

```

5.4 基于输出延迟观测器的PID控制

5.4.1 系统描述

考虑对象

$$G(s) = \frac{1}{s^2 + 10s + 1} \quad (5.41)$$

式(5-60)可表示为

$$\ddot{\theta} = -10\dot{\theta} - \theta + u(t) \quad (5.42)$$

式中, $\theta(t)$ 为位置信号, u 为控制输入。

取 $z = [\theta \quad \dot{\theta}]^T$, 式(5.42)可表示为

$$\dot{z}(t) = Az(t) + Hu(t) \quad (5.43)$$

式中, $A = \begin{bmatrix} 0 & 1 \\ -1 & -10 \end{bmatrix}$, $H = [0 \quad 1]^T$ 。

观测的目标为: 当 $t \rightarrow \infty$ 时, $\hat{\theta}(t) \rightarrow \theta(t)$ 。假设输出信号有延迟, Δ 为输出的位置时间延迟, 则实际输出可表示为

$$y(t) = \theta(t - \Delta) \quad (5.44)$$

5.4.2 输出延迟观测器的设计

假设输出信号有延迟, Δ 为输出的位置时间延迟, 则实际输出可表示为

$$\bar{y}(t) = \theta(t - \Delta) = Cz(t - \Delta) \quad (5.45)$$

式中, $C = [1, 0]$ 。

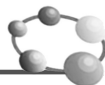
针对系统(5.43), X.H.Wang 等设计了如下观测器^[35]:

$$\dot{\hat{z}}(t) = A\hat{z}(t) + Hu(t) + e^{A\Delta}K[\bar{y}(t) - C\hat{z}(t - \Delta)] \quad (5.46)$$

$$\hat{z}(t - \Delta) = A\hat{z}(t - \Delta) + Hu(t - \Delta) + K[\bar{y}(t) - C\hat{z}(t - \Delta)] \quad (5.47)$$

式中, $A - KC$ 满足 Hurwitz 条件, 即 $A - KC$ 的特征根实部为负。

观测器中, 式(5.47)针对实测的延迟信号进行观测, 式(5.46)对所观测的延迟信号



进行校正。

5.4.3 延迟观测器的分析

针对输出延迟观测器 (5.46) 和 (5.47), X.H.Wang 等^[35]进行了如下收敛性分析:
定义观测误差为

$$\delta(t) = [\delta_1(t) \ \delta_2(t)]^T = [\hat{\theta}(t) - \theta(t) \ \hat{\omega}(t) - \omega(t)]^T = \hat{z}(t) - z(t) \quad (5.48)$$

其中 $\hat{\omega}(t) = \hat{\dot{\theta}}(t)$, $\omega(t) = \dot{\theta}(t)$ 。

由式 (5.43) 得

$$\dot{z}(t - \Delta) = Az(t - \Delta) + Hu(t - \Delta) \quad (5.49)$$

由式 (5.48) 得

$$\delta(t - \Delta) = \hat{z}(t - \Delta) - z(t - \Delta) \quad (5.50)$$

由式 (5.47) 和式 (5.49) 相减, 得

$$\dot{\delta}(t - \Delta) = (A - KC)\delta(t - \Delta) \quad (5.51)$$

式 (5.51) 的解为

$$\delta(t - \Delta) = e^{(A - KC)(t - t_0)} \delta(t_0 - \Delta) \quad (5.52)$$

式中, t_0 为初始时间。

式 (5.52) 中, 由于 $A - KC$ 满足 Hurwitz 条件, 则存在正常数 λ , 有

$$\|e^{A - KC}\| \leq le^{-\lambda}$$

即

$$\|\delta(t - \Delta)\| \leq \|\delta(t_0 - \Delta)\| e^{-\lambda(t - t_0)} \quad (5.53)$$

由式 (5.53) 可得出 $\delta(t - \Delta)$ 的收敛性, 即当 $t \rightarrow \infty$ 时, $\delta(t - \Delta) \rightarrow 0$ 。下面分析 $\delta(t)$ 的收敛性。

由式 (5.46) 减式 (5.43), 得

$$\dot{\delta}(t) = A\delta(t) + e^{A\Delta}K[\bar{y}(t) - C\hat{z}(t - \Delta)] = A\delta(t) - e^{A\Delta}KC\delta(t - \Delta) \quad (5.54)$$

根据线性系统理论, 方程 $\dot{x}(t) = Ax(t) + Bu(t)$ 的解为 $x(t) = e^{A(t - t_0)}x(t_0) + \int_{t_0}^t e^{A(t - \tau)}Bu(\tau)d\tau$ 。则解式 (5.54), 得:

$$\delta(t) = e^{A\Delta}\delta(t - \Delta) - \int_{t - \Delta}^t e^{A(t - \tau)}e^{A\Delta}KC\delta(t - \Delta)d\tau \quad (5.55)$$

由上式并结合积分中值定理, 可得:

$$\int_{t - \Delta}^t e^{A(t - \tau)}e^{A\Delta}KC\delta(t - \Delta)d\tau = e^{A(t - \xi)}e^{A\Delta}KC\delta(\xi - \Delta) \cdot \Delta$$

式中, $\xi \in [t - \Delta, t]$ 。

则

$$\begin{aligned} \|\delta(t)\| &\leq \|e^{A\Delta}\delta(t - \Delta)\| + \|e^{A(t - \xi)}e^{A\Delta}KC\delta(\xi - \Delta)\| \cdot \Delta \\ &\leq \|e^{A\Delta}\| \|\delta(t - \Delta)\| + \|e^{A(t - \xi)}e^{A\Delta}KC\delta(\xi - \Delta)\| \cdot \Delta \end{aligned}$$



$$\leq \|e^{A(t-\xi)}\| \cdot \|e^{A\Delta}KC\| \cdot \Delta \cdot \|\delta(\xi - \Delta)\| \quad (5.56)$$

由已知 $0 \leq t - \xi \leq \Delta$ ，则 $\|e^{A(t-\xi)}\| \cdot \|e^{A\Delta}KC\| \cdot \Delta$ 有界。由式 (5.53) 可知，当 $t \rightarrow \infty$ 时， $\delta(t - \Delta) \rightarrow 0$ ， $\delta(\xi - \Delta) \rightarrow 0$ ，从而由式 (5.56) 可得： $\delta(t) \rightarrow 0$ 。

5.4.4 仿真实例

考虑被控对象式 (5.41)，位置延迟时间为 $\Delta = 3.0$ 。首先，选择 K 使 $A - KC$ 为 Hurwitz。由于

$$A - KC = \begin{bmatrix} 0 & 1 \\ -1 & -10 \end{bmatrix} - \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} -k_1 & 1 \\ -1-k_2 & -10 \end{bmatrix}$$

根据 $|\lambda I - (A - KC)| = 0$ ，有 $\begin{vmatrix} \lambda + k_1 & -1 \\ k_2 + 1 & \lambda + 10 \end{vmatrix} = 0$ ，即 $\lambda^2 + (k_1 + 10)\lambda + k_2 + 1 + 10k_1 = 0$ 。

由 $(\lambda + a)^2 = 0$ 得 $\lambda^2 + 2\lambda a + a^2 = 0$ ，取 $k_1 + 10 = 2a$ ， $k_2 + 1 + 10k_1 = a^2$ 。为了保证 $A - KC$ 为 Hurwitz，通过取 $a > 0$ 使 $\lambda < 0$ 。不妨取 $a = 10$ ，则 $k_1 = 10$ ， $k_2 = -1$ 。

取 $u(t) = \sin t$ ，被控对象式 (5.42)，初始值为 $\theta(0) = 0.20$ ， $\omega(0) = 0$ ，观测器式 (5.46) 和式 (5.47) 的初始值 $\hat{z}(t - \Delta) = [0 \ 0 \ 0 \ 0]^T$ 。延迟观测器的观测结果如图 5-27 和图 5-28 所示。分别对加入和不加入延迟观测器进行 PID 控制，取 $k_p = 10$ ， $k_d = 5$ ，控制效果如图 5-29 至图 5-31 所示。可见，通过采用输出延迟观测器，可获得很好的跟踪性能。

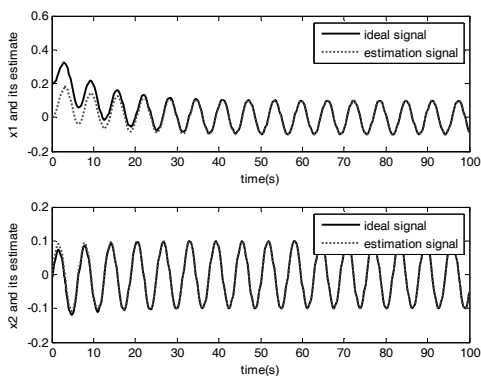


图 5-27 位置和速度的观测

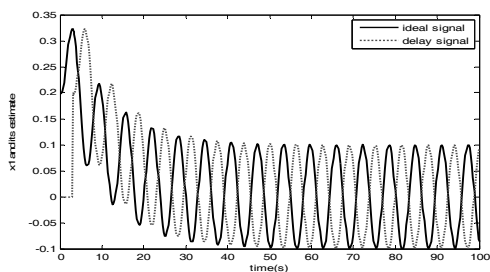


图 5-28 实际位置信号及其延迟信号

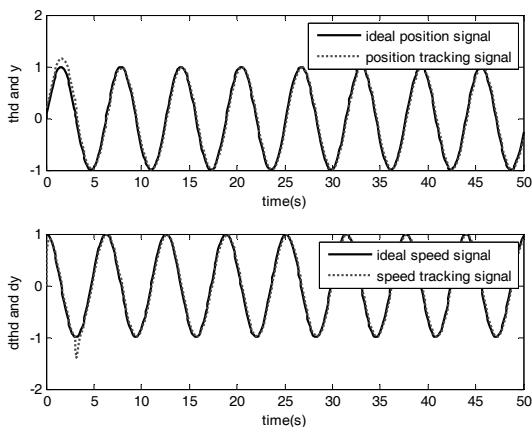


图 5-29 采用延迟观测器的 PD 控制位置、速度跟踪 ($M=1$)

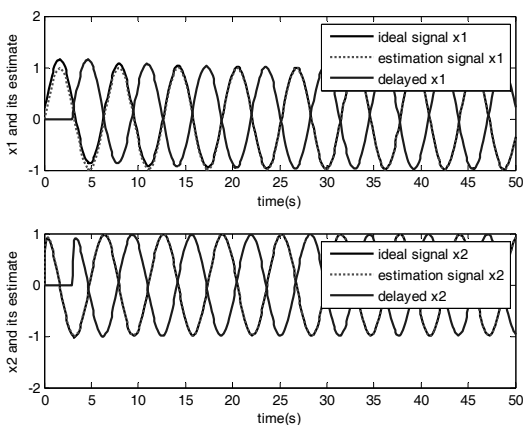


图 5-30 延迟观测器的观测结果 ($M=1$)

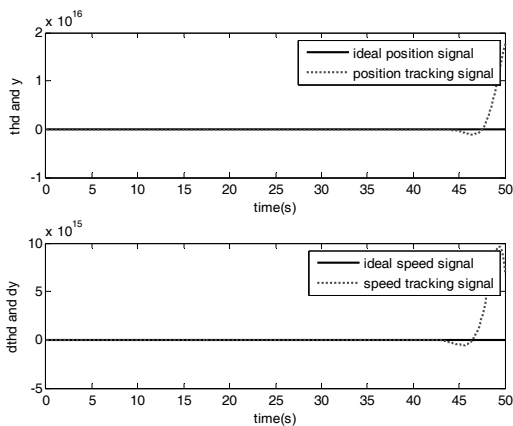


图 5-31 不采用延迟观测器的位置速度跟踪 ($M=2$)

仿真程序

1. 延迟观测器

主程序: chap5_10sim.mdl (见图 5-32)

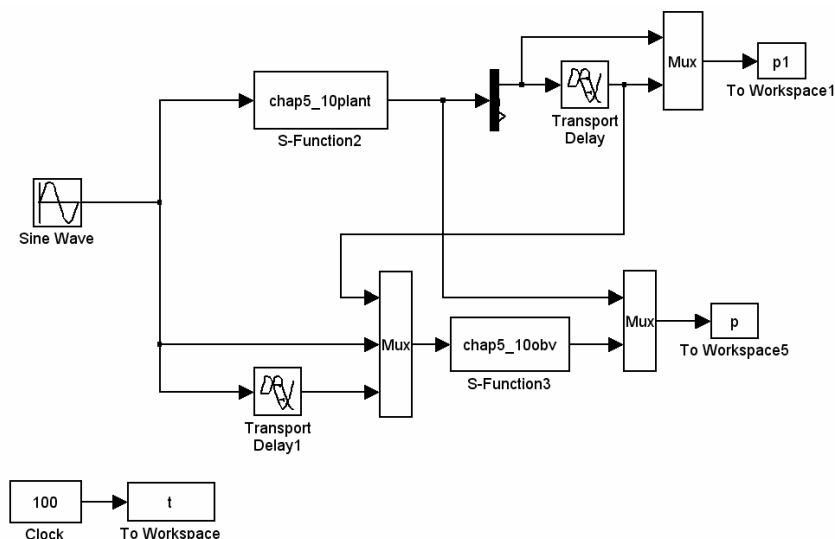
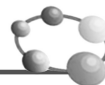


图 5-32 延迟观测器开环测试主程序

观测器程序: chap5_10obv.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0 0 0 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
tol=3.0;
th_tol=u(1);
```



```

yp=th_tol;

ut=u(2);
ut_tol=u(3);
z_tol=[x(3);x(4)];
thp_tol=x(3);
thp=x(1);wp=x(2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[0 1;-1 -10];
C=[1 0];

H=[0;1];

k1=10;k2=-1;
K=[k1 k2]';
z=[thp wp]';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E=expm(A*tol); %EXPM Matrix exponential

dz=A*z+H*ut+E*K*(yp-C*z_tol);
dz_tol=A*z_tol+H*ut_tol+K*(yp-C*z_tol);

for i=1:2
    sys(i)=dz(i);
    sys(i+2)=dz_tol(i);
end
function sys=mdlOutputs(t,x,u)
thp=x(1);wp=x(2);
thp_tol=x(3);wp_tol=x(4);

sys(1)=thp;
sys(2)=wp;
sys(3)=thp_tol;
sys(4)=wp_tol;

```

对象程序: chap5_10plant.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];

```



```
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0.2 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-10*x(2)-x(1)+u(1);
function sys=mdlOutputs(t,x,u)
th=x(1);w=x(2);

sys(1)=th;
sys(2)=w;
```

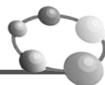
作图程序: chap5_10plot.m

```
close all;

figure(1);
subplot(211);
plot(t,p(:,1),'k',t,p(:,3),'r','linewidth',2);
xlabel('time(s)');ylabel('x1 and its estimate');
legend('ideal signal','estimation signal');
subplot(212);
plot(t,p(:,2),'k',t,p(:,4),'r','linewidth',2);
xlabel('time(s)');ylabel('x2 and its estimate');
legend('ideal signal','estimation signal');

figure(2);
subplot(211);
plot(t,p(:,1)-p(:,3),'r','linewidth',2);
xlabel('time(s)');ylabel('error of x1 and its estimate');
subplot(212);
plot(t,p(:,2)-p(:,4),'r','linewidth',2);
xlabel('time(s)');ylabel('error of x2 and its estimate');

figure(3);
```



```
plot(t,p1(:,1),'k',t,p1(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('x1 and its estimate');
legend('ideal signal','delay signal');
```

2. 控制

主程序：chap5_11sim.mdl（见图 5-33）

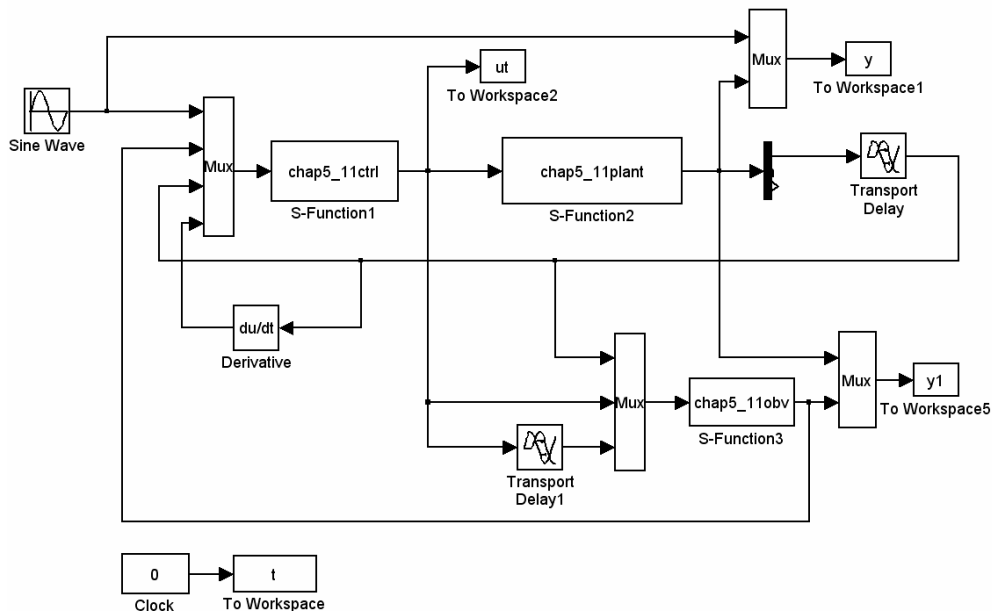


图 5-33 延迟观测器闭环控制主程序

控制器程序：chap5_11ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9},
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 7;
sizes.DirFeedthrough = 1;
```



```
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[-1 0];
function sys=mdlOutputs(t,x,u)
tol=3;
thd=sin(t);
wd=cos(t);
ddthd=-sin(t);

thp=u(2);
wp=u(3);
thp_tol=u(4);
th_tol=u(6);

%Estimated error
e1p=thd-thp;
e2p=wd-wp;

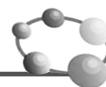
%Practical error
e1=thd-th_tol;
w_tol=u(7);
e2=wd-w_tol;

kp=100;kd=10;

M=2;
if M==1
    ut=kp*e1p+kd*e2p; %With delay observer
elseif M==2
    ut=kp*e1+kd*e2; %Without delay observer
end
sys(1)=ut;
```

观测器程序: chap5_11obv.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
```



```

otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0 0 0 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
tol=3.0;
th_tol=u(1);
yp=th_tol;

ut=u(2);
ut_tol=u(3);
z_tol=[x(3);x(4)];
thp_tol=x(3);
thp=x(1);wp=x(2);
%%%%%%%%%%%%%%
A=[0 1;-1 -10];
C=[1 0];

H=[0;1];

k1=10;k2=-1;
K=[k1 k2]';
z=[thp wp]';
%%%%%%%%%%%%%%
E=expm(A*tol); %EXPM Matrix exponential

dz=A*z+H*ut+E*K*(yp-C*z_tol);
dz_tol=A*z_tol+H*ut_tol+K*(yp-C*z_tol);

for i=1:2
    sys(i)=dz(i);
    sys(i+2)=dz_tol(i);
end
function sys=mdlOutputs(t,x,u)
thp=x(1);wp=x(2);

```




```
thp_tol=x(3);wp_tol=x(4);
```

```
sys(1)=thp;  
sys(2)=wp;  
sys(3)=thp_tol;  
sys(4)=wp_tol;
```

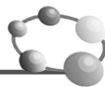
被控对象程序: chap5_11plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,  
    [sys,x0,str,ts]=mdlInitializeSizes;  
case 1,  
    sys=mdlDerivatives(t,x,u);  
case 3,  
    sys=mdlOutputs(t,x,u);  
case {2, 4, 9 }  
    sys = [];  
otherwise  
    error(['Unhandled flag = ',num2str(flag)]);  
end  
function [sys,x0,str,ts]=mdlInitializeSizes  
sizes = simsizes;  
sizes.NumContStates = 2;  
sizes.NumDiscStates = 0;  
sizes.NumOutputs = 2;  
sizes.NumInputs = 1;  
sizes.DirFeedthrough = 1;  
sizes.NumSampleTimes = 1;  
sys=simsizes(sizes);  
x0=[0.2 0];  
str=[];  
ts=[-1 0];  
function sys=mdlDerivatives(t,x,u)  
sys(1)=x(2);  
sys(2)=-10*x(2)-x(1)+u(1);  
function sys=mdlOutputs(t,x,u)  
th=x(1);w=x(2);  
  
sys(1)=th;  
sys(2)=w;
```

作图程序: chap5_11plot.m

```
close all;
```

```
figure(1);
```



```
subplot(211);
plot(t,y(:,1),'k',t,y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('thd and y');
legend('ideal position signal','position tracking signal');
subplot(212);
plot(t,cos(t),'k',t,y(:,3),'r','linewidth',2);
xlabel('time(s)');ylabel('dthd and dy');
legend('ideal speed signal','speed tracking signal');

figure(2);
subplot(211);
plot(t,y1(:,1),'k',t,y1(:,3),'r',t,y1(:,5),'b','linewidth',2);
xlabel('time(s)');ylabel('x1 and its estimate');
legend('ideal signal x1','estimation signal x1','delayed x1');
subplot(212);
plot(t,y1(:,2),'k',t,y1(:,4),'r',t,y1(:,6),'b','linewidth',2);
xlabel('time(s)');ylabel('x2 and its estimate');
legend('ideal signal x2','estimation signal x2','delayed x2');

figure(3);
plot(t,ut(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

第6章 自抗扰控制器及其PID控制

6.1 非线性跟踪微分器

6.1.1 微分器描述

韩京清^[36]利用二阶最速开关系统构造出跟踪不连续输入信号并提取近似微分信号的机构,提出了非线性跟踪-微分器的概念。韩京清所提出的一种离散形式的非线性微分跟踪器在一些运动控制系统中得到了应用。离散形式的非线性微分跟踪器为

$$\begin{cases} r_1(k+1) = r_1(k) + hr_2(k) \\ r_2(k+1) = r_2(k) + hfst(r_1(k) - v(k), r_2(k), \delta, h) \end{cases} \quad (6.1)$$

式中, h 为采样周期, $v(k)$ 为第 k 时刻的输入信号, δ 为决定跟踪快慢的参数。 $fst(\cdot)$ 函数为最速控制综合函数, 描述如下:

$$\begin{aligned} fst(x_1, x_2, \delta, h) &= \begin{cases} -\delta \text{sign}(a) & |a| > d \\ -\delta \frac{a}{d} & |a| \leq d \end{cases} \\ a &= \begin{cases} x_2 + \frac{a_0 - d}{2} \text{sign}(y) & |y| > d_0 \\ x_2 + y/h & |y| \leq d_0 \end{cases} \end{aligned} \quad (6.2)$$

式中, $d = \delta h$, $d_0 = hd$, $y = x_1 + hx_2$, $a_0 = \sqrt{d^2 + 8\delta|y|}$ 。

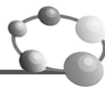
输入信号为 $v(k)$, 则采用微分器 (6.1), 可实现 $r_1(k) \rightarrow v(k)$, $r_2(k) \rightarrow \dot{v}(k)$ 。并且如果 $v(k)$ 是带有噪声的信号, 微分器可同时实现滤波。

6.1.2 仿真实例

输入信号 $v(t) = \sin(2\pi t)$, 采样周期为 $h = 0.001$, $\delta = 150$, 扰动为幅值为 0.05 的随机信号。信号跟踪及导数估计曲线如图 6-1 和图 6-2 所示。图 6-1 中, 上图为带有噪声的正弦信号, 下图为理想正弦信号和微分器输出。图 6-2 中, 上图为信号理想导数与差分方法求得的导数值, 下图为信号理想导数与微分器导数估计值。

将微分器用于 PD 控制中, 位置指令为 $y_d(t) = \sin t$, 被控对象为 $\frac{133}{s^2 + 25s}$, 取 $\delta = 1000$, 对象输出叠加幅值为 0.5 的随机毛刺信号。

此时微分器输入信号 $v(t)$ 为对象输出信号, 采用微分器式 (6.1) 来获得位置和速度, 采



用PD控制律，取 $k_p=10$ ， $k_d=0.5$ ，如图6-3和图6-4所示分别为采用和不采用微分器的正弦位置跟踪。

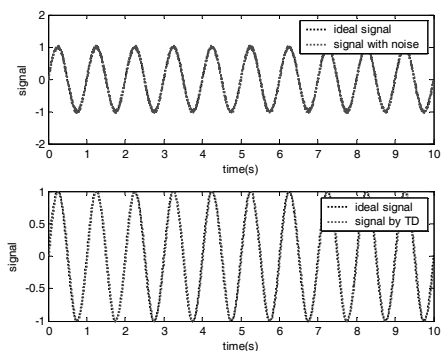


图 6-1 信号跟踪

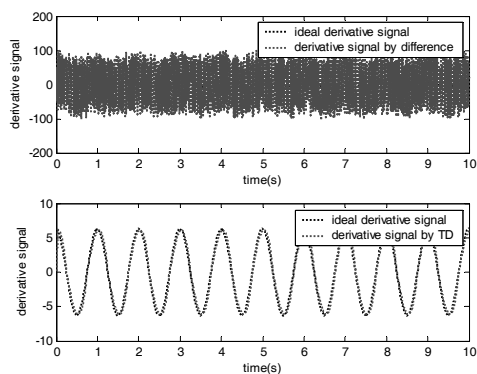


图 6-2 导数求取

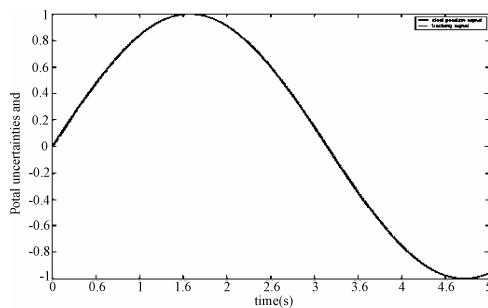


图 6-3 基于微分器下PD控制位置跟踪

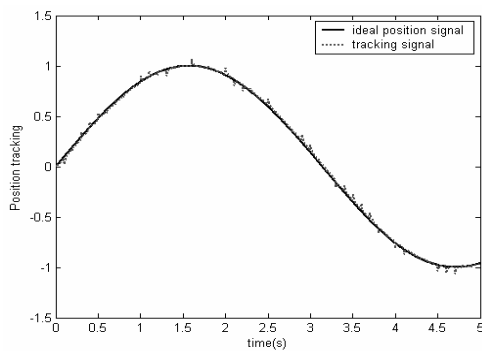


图 6-4 无微分器下PD控制位置跟踪



仿真程序

(1) 信号处理

非线性跟踪微分器主程序: chap6_1.m

```
clear all;
close all;

h=0.001; %Sampling time
delta=150;
r1_1=0;r2_1=0;
vn_1=0;
for k=1:1:10000
time(k)=k*h;

v(k)=sin(2*pi*k*h);
n(k)=0.05*rands(1);
vn(k)=v(k)+n(k);
dv(k)=2*pi*cos(2*pi*k*h);

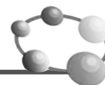
r1(k)=r1_1+h*r2_1;
r2(k)=r2_1+h*chap6_1fst(r1_1-v(k),r2_1,delta,h);

dvn(k)=(vn(k)-vn_1)/h; %By difference

vn_1=vn(k);

r1_1=r1(k);
r2_1=r2(k);
end
figure(1);
subplot(211);
plot(time,v,'k:',time,vn,'r:', 'linewidth',2);
xlabel('time(s)'),ylabel('signal');
legend('ideal signal','signal with noise');
subplot(212);
plot(time,v,'k:',time,r1,'r:', 'linewidth',2);
xlabel('time(s)'),ylabel('signal');
legend('ideal signal','signal by TD');

figure(2);
subplot(211);
plot(time,dv,'k:',time,dvn,'r:', 'linewidth',2);
xlabel('time(s)'),ylabel('derivative signal');
legend('ideal derivative signal','derivative signal by difference');
subplot(212);
plot(time,dv,'k:',time,r2,'r:', 'linewidth',2);
```



```
xlabel('time(s)'),ylabel('derivative signal');  
legend('ideal derivative signal','derivative signal by TD');
```

fst(·) 函数程序: fst.m

```
function f=fst(x1,x2,delta,h)  
d=delta*h;  
d0=h*d;  
y=x1+h*x2;  
a0=sqrt(d^2+8*delta*abs(y));  
  
if abs(y)>d0  
    a=x2+(a0-d)/2*sign(y);  
else  
    a=x2+y/h;  
end  
  
if abs(a)>d  
    f=-delta*sign(a);  
else  
    f=-delta*a/d;  
end
```

(2) 基于微分器的 PD 控制

程序: chap6_2.m

```
close all;  
clear all;  
  
h=0.001;  
y_1=0;yp_1=0;  
dy_1=0;  
  
%Plant  
a=25;b=133;  
sys=tf(b,[1,a,0]);  
dsys=c2d(sys,h,'z');  
[num,den]=tfdata(dsys,'v');  
u_1=0;u_2=0;  
p_1=0;p_2=0;  
for k=1:1:5000  
    t=k*h;  
    time(k)=t;  
  
    p(k)=-den(2)*p_1-den(3)*p_2+num(2)*u_1+num(3)*u_2;  
    dp(k)=(p(k)-p_1)/h;
```



```
yd(k)=sin(t);
dyd(k)=cos(t);
d(k)=0.5*sign(rands(1)); %Noise
if mod(k,100)==1|mod(k,100)==2
    yp(k)=p(k)+d(k);    %Practical signal
else
    yp(k)=p(k);
end

M=1;
if M==1    %By Difference
    y(k)=yp(k);
    dy(k)=(yp(k)-yp_1)/h;
elseif M==2    %By TD
    delta=1000;
    y(k)=y_1+h*dy_1;
    dy(k)=dy_1+h*fst(y_1-yp(k),dy_1,delta,h);
end

kp=10;kd=0.5;
u(k)=kp*(yd(k)-y(k))+kd*(dyd(k)-dy(k));

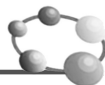
y_1=y(k);
yp_1=yp(k);
dy_1=dy(k);

u_2=u_1;u_1=u(k);
p_2=p_1;p_1=p(k);
end
figure(1);
plot(time,p,'k',time,yp,'r',time,y,'b','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('ideal position signal','position signal with noise','position signal by TD');
figure(2);
plot(time,yd,'k',time,p,'r','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','tracking signal');
```

6.2 安排过渡过程及 PID 控制

6.2.1 安排过渡过程

在阶跃和方波跟踪时，由于被控对象的输出是动态环节的输出，有一定的惯性，其变化不可能是跳变的，而指令信号是跳变的，这意味着让一个不可能跳变的量来跟踪跳变的量，



这是一个不合理的要求^[37]。

当初始误差较大时，为了加快跟踪效果，势必要加大控制增益，这就必然产生较大的超调，从而造成很大的初始冲击。为了降低初始误差，需要设计一个合适的过渡过程。由于跟踪微分器能实现真实信号的提取及求导，故可采用跟踪微分器实现阶跃指令信号的过渡过程。微分器不仅给出过渡过程本身，同时给出过渡过程的微分信号。本节采用 6.1 节介绍的离散非线性跟踪微分器^[36]对方波指令信号进行处理，实现方波信号的安排过渡过程。

6.2.2 仿真实例

输入信号为方波信号，采样周期为 $h=0.01$ ， $\delta=50$ 。

位置指令为方波，被控对象为 $G(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$ 。采用 6.1 节介绍的离散跟踪

微分器来获得指令信号的位置和速度，运行安排过渡过程程序 chap6_3.m，经过微分器处理过的方波信号及其导数如图 6-5 所示。

在离散方式下，将微分器处理过的方波信号及其导数作为位置和速度指令，采用 PD 控制律，取 $k_p=1.0$ ， $k_d=0.02$ ，如图 6-6 和图 6-7 所示分别为采用和不采用微分器的正弦位置跟踪。可见，采用微分器实现方波的过渡过程，可实现对象的平稳跟踪。还可以采用连续系统仿真实现安排过渡过程，如图 6-8 所示为基于 Levant 微分器的方波过渡过程，其中 Levant 微分器算法见 4.2 节，参数取 $\alpha=1$ ， $\lambda=5$ 。

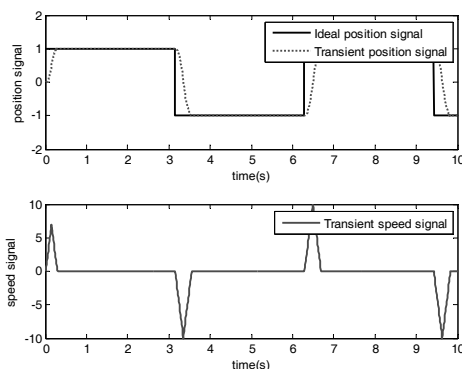


图 6-5 方波信号及其过渡过程

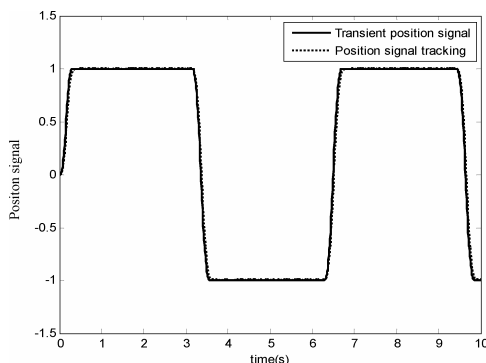


图 6-6 基于安排过渡过程的方波跟踪 ($S=2$)

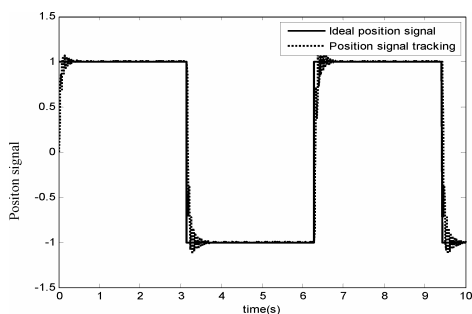
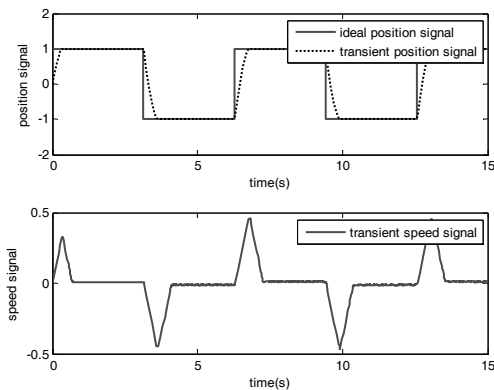
图 6-7 不采用过渡过程的方波跟踪 ($S=1$)

图 6-8 基于 Levant 微分器的方波过渡过程

仿真程序

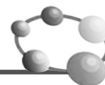
1. 离散系统仿真：采用离散跟踪微分器

(1) 安排过渡过程程序：chap6_3.m

```
clear all;
close all;
h=0.01; %Sampling time

delta=50;
xk=zeros(3,1);
u_1=0;
r_1=0;
r1_1=0;r2_1=0;
for k=1:1:1000
time(k)=k*h;

r(k)=sign(sin(k*h));
dr(k)=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%TD: Transient process
x1=r1_1-r_1;
x2=r2_1;
```



```

r1(k)=r1_1+h*r2_1;
r2(k)=r2_1+h*fst(x1,x2,delta,h);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r_1=r(k);
r1_1=r1(k);
r2_1=r2(k);
end
figure(1);
subplot(211);
plot(time,r,'k',time,r1,'r','linewidth',2);
legend('Ideal position signal','Transient position signal');
xlabel('time(s)'),ylabel('position signal');
subplot(212);
plot(time,r2,'r','linewidth',2);
legend('Transient speed signal');
xlabel('time(s)'),ylabel('speed signal');

```

(2) 采用安排过渡过程的 PID 控制主程序：chap6_4.m

```

%PD Control with TD Transient
clear all;
close all;

M=2;
if M==1
    h=0.001; %Sampling time
elseif M==2
    h=0.01; %Sampling time
end

delta=50;
xk=zeros(3,1);
u_1=0;
r_1=0;
r1_1=0;r2_1=0;
for k=1:1:1000
    time(k)=k*h;

    p1=u_1;
    p2=time(k);
    tSpan=[0 h];
    [tt,xx]=ode45('chap6_4plant',tSpan,xk,[],p1,p2);
    xk = xx(length(xx),:);
    y(k)=xk(1);
    dy(k)=xk(2);

    r(k)=sign(sin(k*h));

```



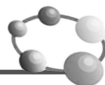
```
dr(k)=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%TD Transient
x1=r1_1-r_1;
x2=r2_1;

r1(k)=r1_1+h*r2_1;          %Transient position signal
r2(k)=r2_1+h*fst(x1,x2,delta,h); %Transient speed signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
kp=1.0;kd=0.02;
S=2;
if S==1
    u(k)=kp*(r(k)-y(k))+kd*(dr(k)-dy(k));    %Ordinary PD
elseif S==2    %PD with TD
    u(k)=kp*(r1(k)-y(k))+kd*(r2(k)-dy(k));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r_1=r(k);
r1_1=r1(k);
r2_1=r2(k);

u_1=u(k);
end

if S==1
    figure(1);
    plot(time,r,'k',time,y,'r','linewidth',2);
    legend('Ideal position signal','Position signal tracking');
    xlabel('time(s)'),ylabel('r,y');
elseif S==2
    figure(1);
    subplot(211);
    plot(time,r,'k',time,r1,'r','linewidth',2);
    legend('Ideal position signal','Transient position signal');
    xlabel('time(s)'),ylabel('position signal');
    subplot(212);
    plot(time,r2,'r','linewidth',2);
    legend('Transient speed signal');
    xlabel('time(s)'),ylabel('speed signal');
    figure(2);
    plot(time,r1,'r',time,y,'b','linewidth',2);
    legend('Transient position signal','Position signal tracking');
    xlabel('time(s)'),ylabel('r,r1,y');
end
```

被控对象程序: chap6_4plant.m



```
function dy = PlantModel(t,y,flag,p1,p2)
ut=p1;
time=p2;
dy=zeros(3,1);
dy(1) = y(2);
dy(2) = y(3);
dy(3)=-87.35*y(3)-10470*y(2)+523500*ut;
```

2. 连续系统仿真：采用 Levant 跟踪微分器实现安排过渡过程

采用安排过渡过程主程序：chap6_5sim.mdl（见图 6-9）

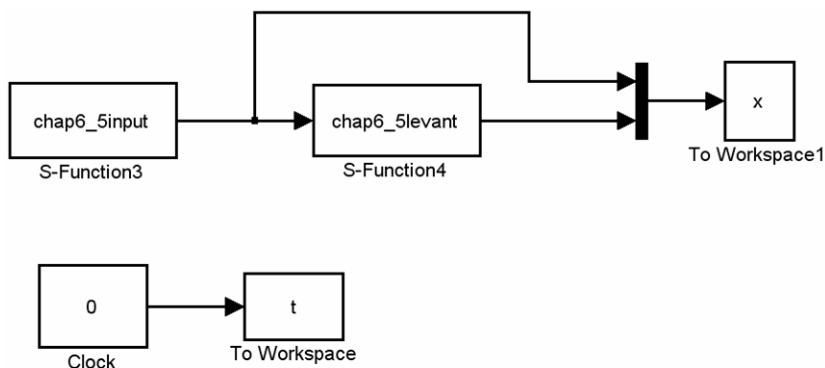


图 6-9 采用 Levant 微分器实现安排过渡过程主程序

输入指令程序：chap6_5input.m

```
function [sys,x0,str,ts] = input(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumOutputs = 1;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
```



```
function sys=mdlOutputs(t,x,u)
yd=1.0*sign(sin(t));
sys(1)=yd;
```

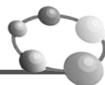
安排过渡过程程序: chap6_5levant.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0 0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
vt=u(1);
e=x(1)-vt;
alfa=1;
nmn=5;

sys(1)=x(2)-nmn*(abs(e))^0.5*sign(e);
sys(2)=-alfa*sign(e);
function sys=mdlOutputs(t,x,u)
sys = x;
```

作图程序: chap6_5plot.m

```
close all;
figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'k','linewidth',2);
```



```
xlabel('time(s)'),ylabel('position signal');
legend('ideal position signal', 'transient position signal');
subplot(212);
plot(t,x(:,3),'r','linewidth',2);
xlabel('time(s)'),ylabel('speed signal');
legend('transient speed signal');
```



6.3 基于非线性扩张观测器的 PID 控制

6.3.1 系统描述

对象表示为

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + bu \\ y &= x_1\end{aligned}\quad (6.3)$$

式中, $f(x_1, x_2)$ 为未知, bu 为已知。

取对象中未知部分为 $x_3(t) = f(x_1, x_2)$, 则对象表示为

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= bu + x_3 \\ y &= x_1\end{aligned}\quad (6.4)$$

通过设计扩张观测器来实现速度的估计, 并实现未知不确定性和外加干扰的估计。将其应用于闭环 PID 控制中, 可实现无需速度测量的控制, 并实现对未知不确定性和外加干扰的补偿。

6.3.2 非线性扩张观测器

将被扩张的系统的状态观测器称为扩张观测器, 韩京清所设计的非线性扩张观测器表示为^[38]

$$\begin{aligned}e &= z_1 - y \\ \dot{z}_1 &= z_2 - \beta_1 e \\ \dot{z}_2 &= z_3 - \beta_2 \text{fal}(e, \alpha_1, \delta) + bu \\ \dot{z}_3 &= -\beta_3 \text{fal}(e, \alpha_2, \delta)\end{aligned}\quad (6.5)$$

式中, $\beta_i > 0 (i=1, 2, 3)$, $\alpha_1 = 0.5$, $\alpha_2 = 0.25$ 。饱和函数 $\text{fal}(e, \alpha, \delta)$ 的作用为抑制信号抖振, 表示为

$$\text{fal}(e, \alpha, \delta) = \begin{cases} \frac{e}{\delta^{1-\alpha}}, & |e| \leq \delta \\ |e|^\alpha \text{sgn}(e), & |e| > \delta \end{cases}\quad (6.6)$$

则有



$$z_1(t) \rightarrow x_1(t), \quad z_2(t) \rightarrow x_2(t), \quad z_3(t) \rightarrow x_3(t) = f_1(x_1, x_2) + (b - b_0)u(t)。$$

观测器式(6.5)中, 变量 $z_3(t)$ 称为被扩张的状态。可见, 通过非线性扩张观测器式(6.5), 可实现对被控对象(6.3)的位置、速度和未知部分的观测。在实际控制工程中, 可采用本观测器实现无需速度测量的控制, 并可实现对未知不确定性和外加干扰的补偿。

由于扩张观测器只用到对象的名义模型信息, 而没有用到描述对象的函数 $f(\cdot)$ 信息, 因此, 该观测器具有很好的工程应用价值。由非线性扩张观测器的表达式中的非线性切换部分可见, 当误差较大时, 通过对其绝对值进行开方使其切换增益降低, 防止产生超调, 当误差较小时, 通过对其绝对值进行开方使其切换增益增大, 加快收敛过程。

6.3.3 仿真实例

被观测对象为

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + bu \end{aligned}$$

仿真之一: 扩张观测器

取 $f(x_1, x_2) = -25x_2$, $b = 133$ 。假设 $f(x_1, x_2)$ 为未知部分, bu 为已知部分。采用离散扩张观测器, 观测器参数参考文献^[37]中4.3节的例1, 取值为: 采样时间 $h = 0.01$, $\beta_1 = 100$, $\beta_2 = 300$, $\beta_3 = 1000$, $\delta = h$, $\alpha_1 = 0.5$, $\alpha_2 = 0.25$, 观测器的输入信号为 $\sin t$, 对象的初始位置和速度取值为零, 扩张观测器仿真结果如图6-10和图6-11所示。采用连续扩张观测器, 运行程序chap6_6sim.mdl, 也可以得到同样的效果。

仿真之二: 基于扩张观测器的PD控制

取 $f(x_1, x_2) = -25x_2 + 33\sin(\pi t)$ 为未知部分, 已知部分为 $b = 133$, 采用扩张观测器观测未知部分。采用离散控制系统仿真, 取值与扩张观测器的相同。采用离散方式进行仿真, 采用PD控制, 取 $k_p = 10$, $k_d = 0.3$, 位置跟踪、扩张观测器仿真结果如图6-12至图6-14所示。采用连续系统仿真, 运行程序chap6_8sim.mdl, 也可以得到同样的效果。

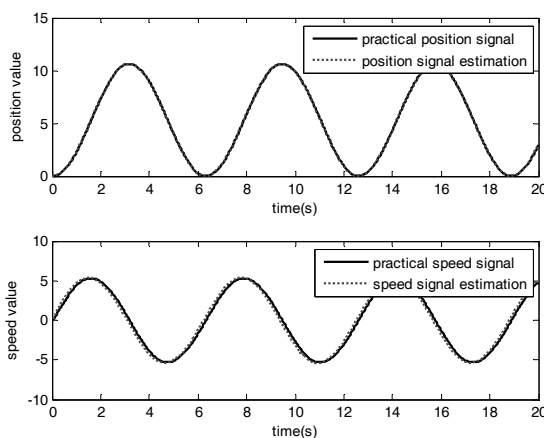


图 6-10 位置、速度的观测

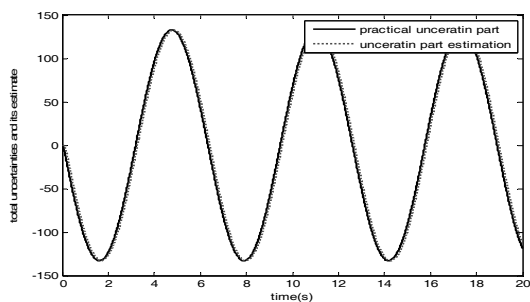
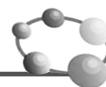


图 6-11 不确定性的观测

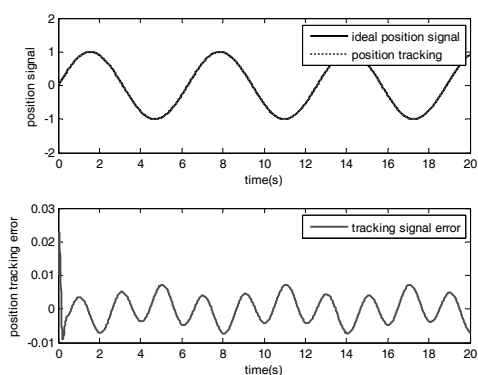
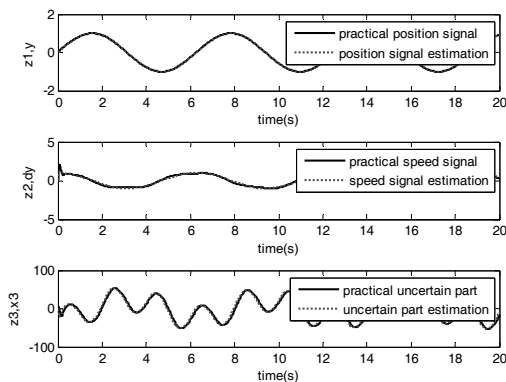
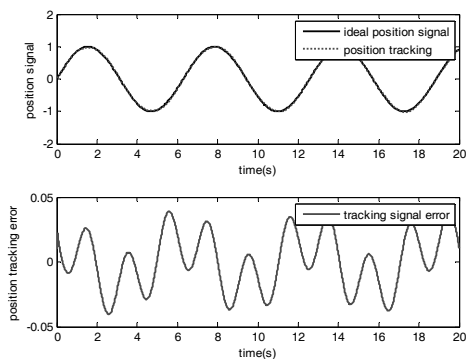
图 6-12 正弦位置跟踪（基于扩张观测器 $M=1$ ）

图 6-13 位置、速度和不确定性的观测

图 6-14 正弦位置跟踪（无扩张观测器， $M=2$ ）



仿真程序：

仿真程序之一：非线性扩张观测器仿真程序

(1) 连续程序

主程序：chap6_6sim.mdl (见图 6-15)

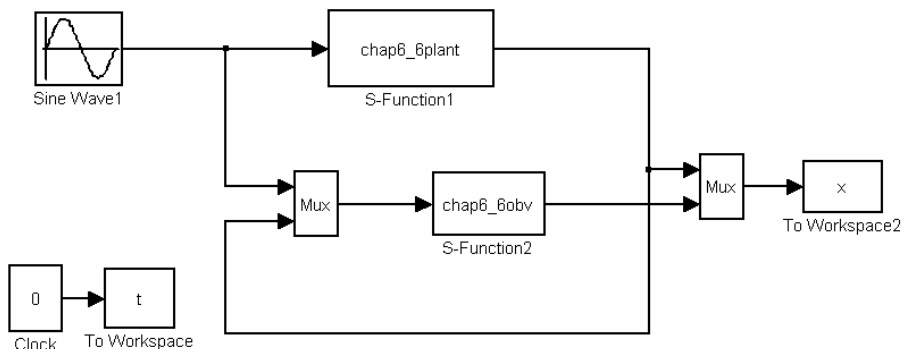


图 6-15 非线性扩张观测器开环测试主程序

观测器程序：chap6_6obv.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
th=u(2);
```



```

b=133;

e=x(1)-th;
beta1=100;beta2=300;beta3=1000;
delta=0.01;
alfa1=0.5;alfa2=0.25;

if abs(e)>delta
    fal1=abs(e)^alfa1*sign(e);
else
    fal1=e/(delta^(1-alfa1));
end

if abs(e)>delta
    fal2=abs(e)^alfa2*sign(e);
else
    fal2=e/(delta^(1-alfa2));
end

sys(1)=x(2)-beta1*e;
sys(2)=x(3)-beta2*fal1+b*ut;
sys(3)=-beta3*fal2;
function sys=mdlOutputs(t,x,u)
fp=x(3);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fp;

```

被控对象: chap6_6plant.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;

```



```
sizes.NumDiscStates = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.15;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

fx=-25*x(2);
sys(1)=x(2);
sys(2)=fx+133*ut;
function sys=mdlOutputs(t,x,u)
fx=-25*x(2);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

作图程序: chap6_6plot.m

```
close all;

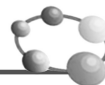
figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,4),'k','linewidth',2);
xlabel('time(s)');ylabel('th and x1p');
legend('practical position signal', 'position signal estimation');
subplot(212);
plot(t,x(:,2),'r',t,x(:,5),'k','linewidth',2);
xlabel('time(s)');ylabel('dth and x2p');
legend('practical speed signal', 'speed signal estimation');
figure(2);
plot(t,x(:,3),'r',t,x(:,6),'k','linewidth',2);
xlabel('time(s)');ylabel('f and fp');
legend('practical unceratin part', 'unceratin part estimation');
```

(2) 离散程序

主程序: chap6_7.m

```
clear all;
close all;

T=0.01; %Sampling time
beta1=100;beta2=300;beta3=1000;
```



```

delta=T;
alfa1=0.5;alfa2=0.25;

xk=zeros(2,1);
e1_1=0;
u_1=0;
r_1=0;
z1_1=0;z2_1=0;z3_1=0;
for k=1:1:2000
time(k)=k*T;

p=u_1;
tSpan=[0 T];
[tt,xx]=ode45('chap6_7plant',tSpan,xk,[],p);
xk = xx(length(xx),:);
y(k)=xk(1);
dy(k)=xk(2);

u(k)=sin(k*T);
dr(k)=0;

f(k)=-25*dy(k);    %Unknown part
b=133;

x3(k)=f(k);
%ESO
epc0=z1_1-y(k);
z1(k)=z1_1+T*(z2_1-beta1*epc0);
z2(k)=z2_1+T*(z3_1-beta2*fal(epc0,alfa1,delta)+b*u(k));
z3(k)=z3_1-T*beta3*fal(epc0,alfa2,delta);

z1_1=z1(k);
z2_1=z2(k);
z3_1=z3(k);

z1_1=z1(k);z2_1=z2(k);z3_1=z3(k);
u_1=u(k);
end
figure(1);
subplot(211);
plot(time,y,'k',time,z1,'r','linewidth',2);
xlabel('time(s)');ylabel('position value');
legend('practical position signal', 'position signal estimation');
subplot(212);
plot(time,dy,'k',time,z2,'r','linewidth',2);
xlabel('time(s)');ylabel('speed value');

```



```
legend('practical speed signal', 'speed signal estimation');  
figure(2);  
plot(time,x3,'k',time,z3,'r','linewidth',2);  
xlabel('time(s)');ylabel('total uncertainties and its estimate');  
legend('practical unceratin part', 'unceratin part estimation');
```

被控对象: chap6_7plant.m

```
function dy = PlantModel(t,y,flag,p)  
ut=p;  
dy=zeros(2,1);  
  
f=-25*y(2);    %Unknown part  
b=133;  
  
dy(1)=y(2);  
dy(2)=f+b*ut;
```

饱和函数: fal.m

```
function y=fal(epec,alfa,delta)  
if abs(epec)>delta  
    y=abs(epec)^alfa*sign(epec);  
else  
    y=epec/(delta^(1-alfa));  
end
```

仿真程序之二: 基于非线性扩张观测器的PID控制

(1) 连续程序

Simulink 主程序: chap6_8sim.mdl (见图 6-16)

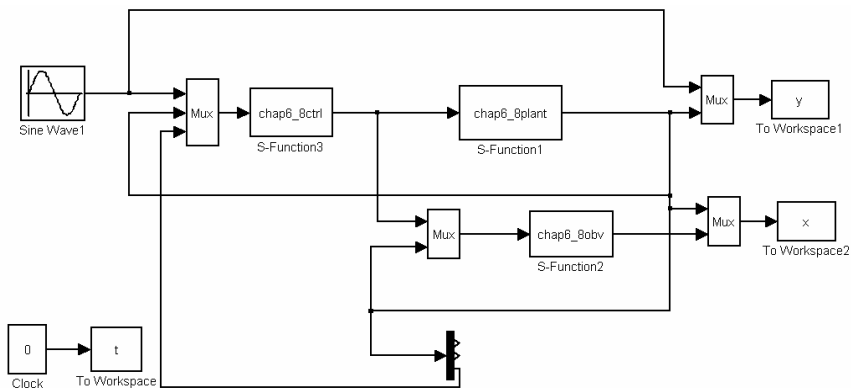
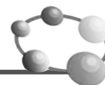


图 6-16 基于非线性扩张观测器的PID控制主程序

观测器程序: chap6_8obv.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,
```



```

        [sys,x0,str,ts]=mdlInitializeSizes;
    case 1,
        sys=mdlDerivatives(t,x,u);
    case 3,
        sys=mdlOutputs(t,x,u);
    case {2, 4, 9 }
        sys = [];
    otherwise
        error(['Unhandled flag = ',num2str(flag)]);
    end
function [sys,x0,str,ts]=mdlInitializeSizes
    sizes = simsizes;
    sizes.NumContStates = 3;
    sizes.NumDiscStates = 0;
    sizes.NumOutputs = 3;
    sizes.NumInputs = 4;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 0;
    sys=simsizes(sizes);
    x0=[0 0 0];
    str=[];
    ts=[];
    function sys=mdlDerivatives(t,x,u)
        ut=u(1);
        th=u(2);
        b0=133;

        epc0=x(1)-th;
        beta1=100;beta2=300;beta3=1000;
        delta1=0.0025;delta0=0.01;
        delta=10;
        alfa1=0.5;alfa2=0.25;

        if abs(epc0)>delta1
            fal1=abs(epc0)^alfa1*sign(epc0);
        else
            fal1=epc0/(delta1^(1-alfa1));
        end

        if abs(epc0)>delta1
            fal2=abs(epc0)^alfa2*sign(epc0);
        else
            fal2=epc0/(delta1^(1-alfa2));
        end

        sys(1)=x(2)-beta1*epc0;

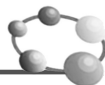
```



```
sys(2)=x(3)-beta2*fal1+b0*ut;  
sys(3)=-beta3*fal2;  
function sys=mdlOutputs(t,x,u)  
fp=x(3);  
  
sys(1)=x(1);  
sys(2)=x(2);  
sys(3)=fp;
```

控制器程序: chap6_8ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,  
    [sys,x0,str,ts]=mdlInitializeSizes;  
case 1,  
    sys=mdlDerivatives(t,x,u);  
case 3,  
    sys=mdlOutputs(t,x,u);  
case {2, 4, 9 }  
    sys = [];  
otherwise  
    error(['Unhandled flag = ',num2str(flag)]);  
end  
function [sys,x0,str,ts]=mdlInitializeSizes  
sizes = simsizes;  
sizes.NumContStates = 0;  
sizes.NumDiscStates = 0;  
sizes.NumOutputs = 1;  
sizes.NumInputs = 5;  
sizes.DirFeedthrough = 1;  
sizes.NumSampleTimes = 0;  
sys=simsizes(sizes);  
x0=[];  
str=[];  
ts=[];  
function sys=mdlOutputs(t,x,u)  
yd=u(1);  
dyd=cos(t);  
y=u(2);  
dy=u(3);  
fp=u(5);  
  
M=1;  
if M==1 %Without compensation  
    ut=10*(yd-y)+10*(dyd-dy);  
elseif M==2 %With compensation
```



```
ut=10*(yd-y)+10*(dyd-dy)-1/133*fp;  
end  
  
sys(1)=ut;
```

被控对象: chap6_8plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,  
    [sys,x0,str,ts]=mdlInitializeSizes;  
case 1,  
    sys=mdlDerivatives(t,x,u);  
case 3,  
    sys=mdlOutputs(t,x,u);  
case {2, 4, 9 }  
    sys = [];  
otherwise  
    error(['Unhandled flag = ',num2str(flag)]);  
end  
function [sys,x0,str,ts]=mdlInitializeSizes  
sizes = simsizes;  
sizes.NumContStates = 2;  
sizes.NumDiscStates = 0;  
sizes.NumOutputs = 3;  
sizes.NumInputs = 1;  
sizes.DirFeedthrough = 0;  
sizes.NumSampleTimes = 0;  
sys=simsizes(sizes);  
x0=[0.15;0];  
str=[];  
ts=[];  
function sys=mdlDerivatives(t,x,u)  
ut=u(1);  
  
fx=-25*x(2)-100*sign(x(2));  
sys(1)=x(2);  
sys(2)=fx+133*ut;  
function sys=mdlOutputs(t,x,u)  
fx=-25*x(2)-100*sign(x(2));  
  
sys(1)=x(1);  
sys(2)=x(2);  
sys(3)=fx;
```

作图程序: chap6_8plot.m

```
close all;
```




```
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k-','linewidth',2);
xlabel('time(s)');ylabel('yd and y');
legend('ideal position signal', 'tracking signal');

figure(2);
subplot(211);
plot(t,x(:,1),'r',t,x(:,4),'k','linewidth',2);
xlabel('time(s)');ylabel('th and x1p');
subplot(212);
plot(t,x(:,2),'r',t,x(:,5),'k','linewidth',2);
xlabel('time(s)');ylabel('dth and x2p');
figure(3);
plot(t,x(:,3),'r',t,x(:,6),'k','linewidth',2);
xlabel('time(s)');ylabel('f and fp');
```

(2) 离散程序

主程序: chap6_9.m

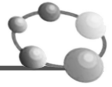
```
clear all;
close all;
h=0.01; %Sampling time
%ESO Parameters
beta1=100;beta2=300;beta3=1000;
delta1=0.0025;
alfa1=0.5;alfa2=0.25;

kp=10;kd=0.3;

xk=zeros(2,1);
u_1=0;
z1_1=0;z2_1=0;z3_1=0;
for k=1:1:2000
time(k) = k*h;

p1=u_1;
p2=k*h;
tSpan=[0 h];
[tt,xx]=ode45('chap6_9plant',tSpan,xk,[],p1,p2);
xk = xx(length(xx),:);
y(k)=xk(1);
dy(k)=xk(2);

yd(k)=sin(k*h);
dyd(k)=cos(k*h);
```



```

f(k)=-25*dy(k)+33*sin(pi*p2);    %Unknown part
b=133;
x3(k)=f(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ESO
e=z1_1-y(k);
z1(k)=z1_1+h*(z2_1-beta1*e);
z2(k)=z2_1+h*(z3_1-beta2*fal(e,alfa1,delta1)+b*u_1);
z3(k)=z3_1-h*beta3*fal(e,alfa2,delta1);

z1_1=z1(k);
z2_1=z2(k);
z3_1=z3(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%disturbance compensation
e1(k)=yd(k)-z1(k);
e2(k)=dyd(k)-z2(k);

M=1;
if M==1    %With ESO Compensation
    u(k)=kp*(yd(k)-y(k))+kd*(dyd(k)-dy(k))-z3(k)/b;
elseif M==2 %Without ESO Compensation
    u(k)=kp*(yd(k)-y(k))+kd*(dyd(k)-dy(k));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
z1_1=z1(k);z2_1=z2(k);z3_1=z3(k);
u_1=u(k);
end
figure(1);
subplot(211);
plot(time,yd,'k',time,y,'r','linewidth',2);
xlabel('time(s)'),ylabel('position signal');
legend('ideal position signal','position tracking');
subplot(212);
plot(time,yd-y,'r','linewidth',2);
xlabel('time(s)'),ylabel('position tracking error');
legend('tracking signal error');
figure(2);
subplot(311);
plot(time,z1,'k',time,y,'r','linewidth',2);
xlabel('time(s)'),ylabel('z1,y');
legend('practical position signal', 'position signal estimation');
subplot(312);
plot(time,z2,'k',time,dy,'r','linewidth',2);
xlabel('time(s)'),ylabel('z2,dy');
legend('practical speed signal', 'speed signal estimation');

```



```
subplot(313);
plot(time,z3,'k',time,x3,'r','linewidth',2);
xlabel('time(s)'),ylabel('z3,x3');
legend('practical uncertain part', 'uncertain part estimation');
```

被控对象: chap6_9plant.m

```
function dy = PlantModel(t,y,flag,p1,p2)
ut=p1;
time=p2;
dy=zeros(2,1);

f=-25*y(2)+33*sin(pi*p2);    %Unknown part
b=133;
dy(1)=y(2);
dy(2)=f+b*ut;
```



6.4 非线性 PID 控制

6.4.1 非线性 PID 控制算法

传统的 PID 控制形式为误差的现在 (P)、过去 (I) 和将来 (变化趋势 D) 的线性组合, 显然这种线性组合不是最佳的组合形式, 可以在非线性范围内寻求更合适、更有效的组合形式^[22]。

韩京清教授推荐了三种非线性组合形式的 PID 控制器, 其中的一种 PD 形式的非线性组合表示为^[37]

$$u = \beta_1 \text{fal}(e_1, \alpha_1, \delta) + \beta_2 \text{fal}(e_2, \alpha_2, \delta) \quad (6.7)$$

式中, $0 < \alpha_1 < 1 < \alpha_2$, $k_p = \beta_1$, $k_d = \beta_2$, e_1 为指令信号与被控对象位置输出之差, e_2 为指令信号微分与被控对象速度输出之差。

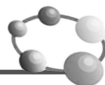
为了避免高频振荡现象, 将幂函数 $|e|^\alpha \text{sign}(e)$ 改造成原点附近具有线性段的连续的幂次函数, 即饱和函数, 表示为

$$\text{fal}(e, \alpha, \delta) = \begin{cases} \frac{e}{\delta^{\alpha-1}}, & |e| \leq \delta \\ |e|^\alpha \text{sign}(e), & |e| > \delta \end{cases} \quad (6.8)$$

式中, δ 为线性段的区间长度。

6.4.2 仿真实例

被控对象为



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

取位置指令为1.0，采用控制律式(6.7)，控制律参数按参考文献^[22,37]，取采样时间为 $h=0.001$ ， $\alpha_1 = \frac{3}{4}$ ， $\alpha_2 = \frac{3}{2}$ ， $\delta = 2h$ ， $\beta_1 = 150$ ， $\beta_2 = 1.0$ ，仿真结果如图 6-17 所示。如果采用线性 PD 控制， k_p 和 k_d 不变，仿真结果如图 6-18 所示。

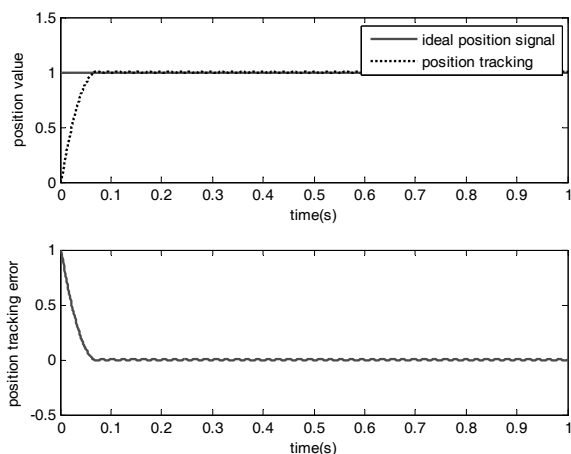


图 6-17 非线性 PID 控制阶跃响应 ($M=1$)

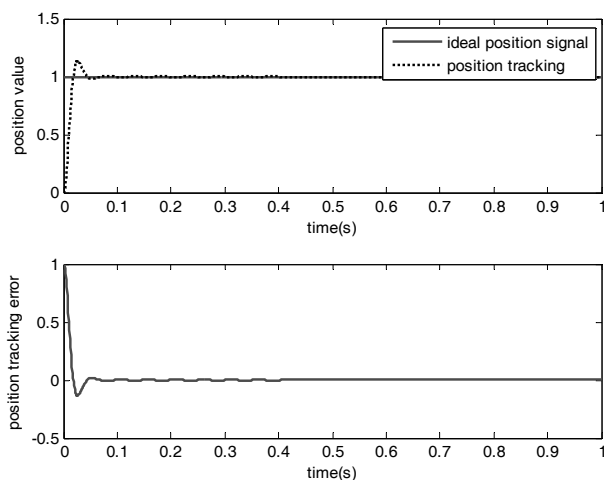


图 6-18 线性 PID 控制阶跃响应 ($M=2$)

仿真程序:

控制程序: chap6_10.m

```
clear all;
close all;

h=0.001; %Sampling time

beta1=150;beta2=1.0;
kp=beta1;kd=beta2;
```



```
alfa1=0.75;alfa2=1.5;
delta=2*h;

xk=zeros(2,1);
u_1=0;
for k=1:1:1000
time(k)=k*h;

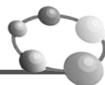
p1=u_1;
p2=time(k);
tSpan=[0 h];
[tt,xx]=ode45('chap6_10plant',tSpan,xk,[],p1,p2);
xk=xx(length(xx),:);
y(k)=xk(1);
dy(k)=xk(2);

yd(k)=1.0;
dyd(k)=0;

e1(k)=yd(k)-y(k);
e2(k)=dyd(k)-dy(k);

M=1;
if M==1
    u(k)=kp*fal(e1(k),alfa1,delta)+kd*fal(e2(k),alfa2,delta);    % NPD
elseif M==2
    u(k)=kp*e1(k)+kd*e2(k); % PD
end

u_1=u(k);
end
figure(1);
subplot(211);
plot(time,yd,'r',time,y,'k','linewidth',2);
legend('ideal position signal','position tracking');
xlabel('time(s)');ylabel('position value');
subplot(212);
plot(time,yd-y,'r','linewidth',2);
xlabel('time(s)');ylabel('position tracking error');
被控对象程序: chap6_10plant.m
function dx = PlantModel(t,x,flag,p1,p2)
ut=p1;
time=p2;
dx=zeros(2,1);
```



```
dx(1)=x(2);
dx(2)=-25*x(2)+133*ut;
```

6.5 自抗扰控制

6.5.1 自抗扰控制结构

自抗扰控制（Active Disturbance Rejection Control, ADRC）由韩京清教授提出^[4,37,38]，该控制策略对经典PID控制进行了4个方面的改进：

- （1）安排过渡过程；
- （2）采用跟踪微分器对被控对象提取微分信号；
- （3）由非线性扩张观测器实现扰动估计和补偿；
- （4）由误差的P、I、D的非线性组合构成非线性PID控制器。

采用微分器实现安排过渡过程，由非线性扩张观测器实现扰动估计和补偿，控制器采用非线性PID控制，自抗扰控制系统结构如图6-19所示。^[37]

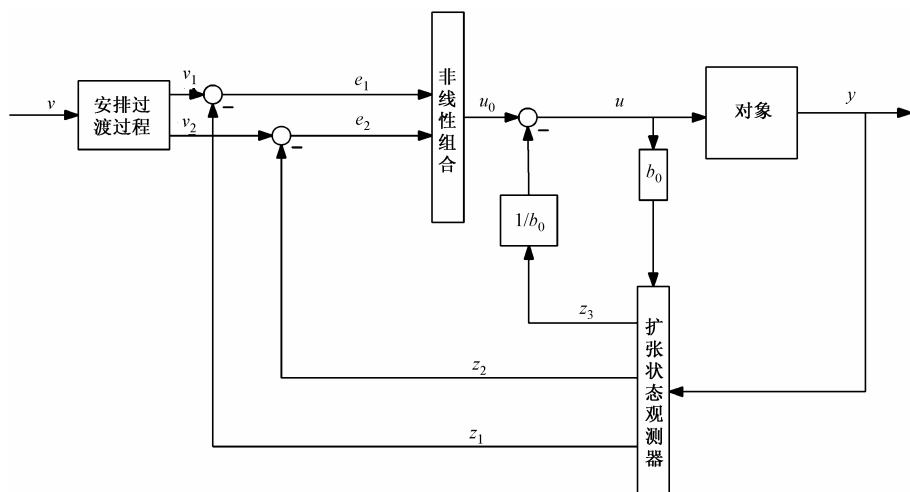


图 6-19 自抗扰控制系统结构

自抗扰控制策略具体的设计方法如下：^[37]采用6.1节介绍的非线性跟踪微分器实现安排过渡过程，非线性扩张观测器采用6.3节介绍的方法，根据6.4节设计非线性PID控制器。

6.5.2 仿真实例

被控对象为

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -25x_2 + 33\sin(\pi t) + 133u$$

式中， $f(x_1, x_2) = -25x_2 + 33\sin(\pi t)$ 为未知， $b = 133$ 为已知。



取位置指令为幅值为1.0的方波信号。按离散和连续两种方式进行仿真。

(1) 连续系统仿真

用于安排过渡过程的微分器参数取 $\alpha = 1.0$, $\lambda = 5.0$ 。扩张观测器中, 取 $\beta_1 = 100$, $\beta_2 = 300$, $\beta_3 = 1000$, $\delta = 0.0025$, $\alpha_1 = 0.5$, $\alpha_2 = 0.25$ 。非线性 PID 控制器中, 取 $\alpha_1 = \frac{3}{4}$, $\alpha_2 = \frac{3}{2}$, $\delta = 0.02$, $\beta_1 = 6.0$, $\beta_2 = 1.5$ 。采用自抗扰控制方法的方波跟踪及扩张观测器仿真结果如图 6-20 和图 6-21 所示。

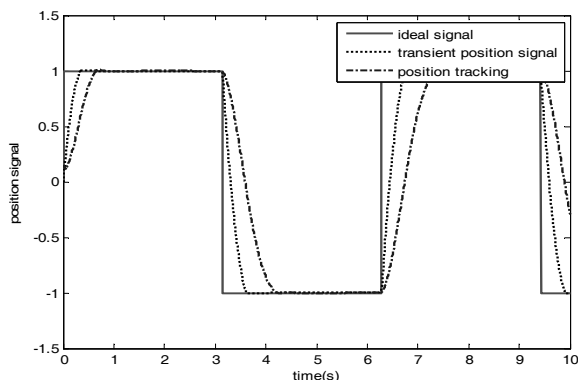


图 6-20 采用自抗扰控制的方波响应

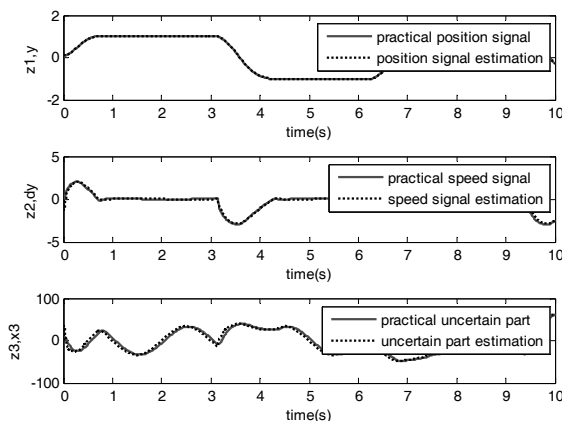
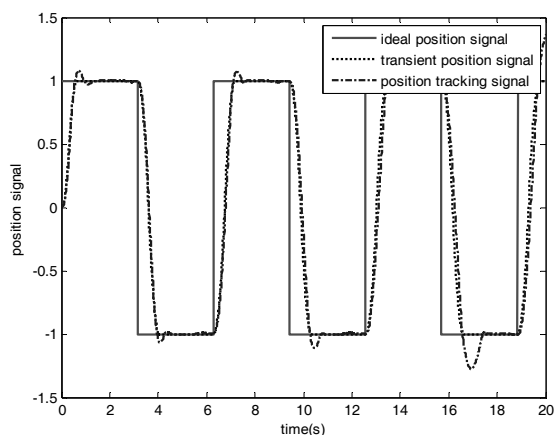
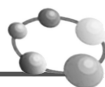
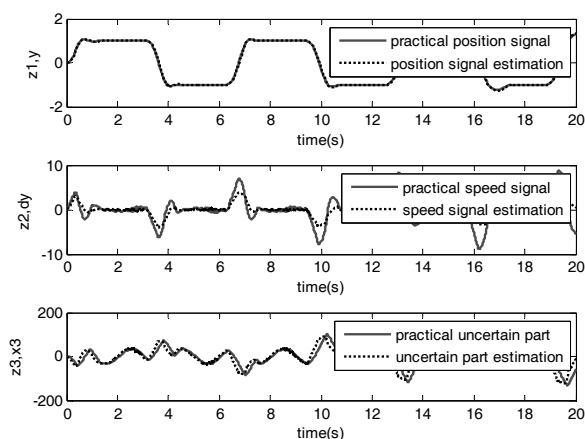
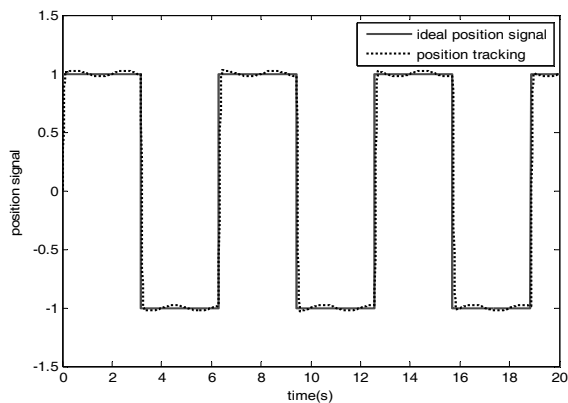


图 6-21 扩张观测器的观测结果

(2) 离散系统仿真

取采样时间为 $h = 0.01$ 。用于安排过渡过程的微分器参数取 $\delta = 10$ 。扩张观测器中, 取 $\beta_1 = 100$, $\beta_2 = 300$, $\beta_3 = 1000$, $\delta = 0.0025$, $\alpha_1 = 0.5$, $\alpha_2 = 0.25$ 。非线性 PID 控制器中, 取 $\alpha_1 = \frac{3}{4}$, $\alpha_2 = \frac{3}{2}$, $\delta = 2h$, $\beta_1 = 3.0$, $\beta_2 = 0.3$ 。采用自抗扰控制方法的方波跟踪及扩张观测器仿真结果如图 6-22 和图 6-23 所示。如果采用线性 PD 控制, k_p 和 k_d 不变, 仿真结果如图 6-24 所示。

图 6-22 基于自抗扰控制的方波响应 ($M=1$)图 6-23 扩张观测器的观测结果 ($M=1$)图 6-24 传统线性 PID 下的方波响应 ($M=2$)

仿真程序

1. 连续系统仿真

Simulink 主程序: chap6_11sim.mdl (见图 6-25)

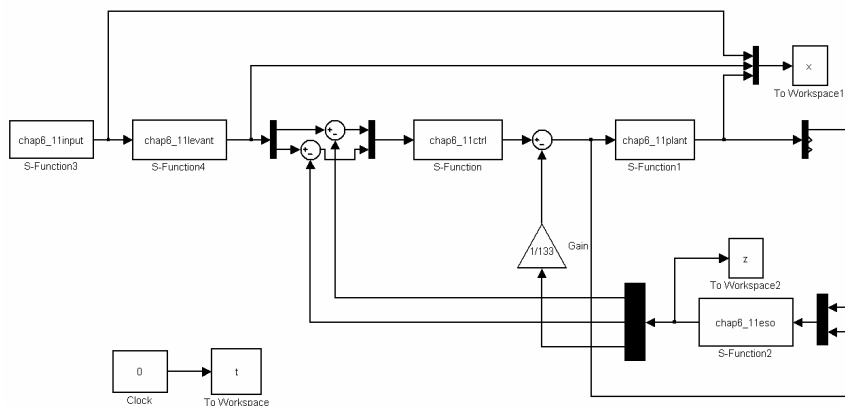


图 6-25 自抗扰控制 Simulink 主程序

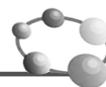
方波输入指令 S 函数: chap6_11input.m

```
function [sys,x0,str,ts] = input(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumOutputs = 1;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
yd=1.0*sign(sin(t));
sys(1)=yd;
```

微分器 S 函数: chap6_11levant.m

```
function [sys,x0,str,ts] = Differentiator(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
```



```

case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0 0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
vt=u(1);
e=x(1)-vt;
alfa=1;
nmn=5;

sys(1)=x(2)-nmn*(abs(e))^0.5*sign(e);
sys(2)=-alfa*sign(e);
function sys=mdlOutputs(t,x,u)
sys = x;

```

控制器 S 函数: chap6_11ctrl.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;

```



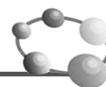
```
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[0 0];
function sys=mdlOutputs(t,x,u)
e1=u(1);
e2=u(2);
%NPID Parameters
delta0=0.02;
alfa01=3/4;alfa02=3/2;    %0<alfa1<1<alfa2
beta01=6.0;beta02=1.5;
kp=beta01;kd=beta02;

if abs(e1)>delta0
    fal1=abs(e1)^alfa01*sign(e1);
else
    fal1=e1/(delta0^(1-alfa01));
end
if abs(e2)>delta0
    fal2=abs(e2)^alfa02*sign(e2);
else
    fal2=e2/(delta0^(1-alfa02));
end

ut=kp*fal1+kd*fal2;    %NPD
sys(1)=ut;
```

扩张观测器 S 函数: chap6_11eso.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```



```

end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
%ESO Parameters
beta1=100;beta2=300;beta3=1000;
delta1=0.0025;
alfa1=0.5;alfa2=0.25;
x1=u(1);
ut=u(2);
e=x(1)-x1;

if abs(e)>delta1
    fal1=abs(e)^alfa1*sign(e);
else
    fal1=e/(delta1^(1-alfa1));
end
if abs(e)>delta1
    fal2=abs(e)^alfa2*sign(e);
else
    fal2=e/(delta1^(1-alfa2));
end

b=133;
sys(1)=x(2)-beta1*e;
sys(2)=x(3)-beta2*fal1+b*ut;
sys(3)=-beta3*fal2;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);

```

被控对象 S 函数: chap6_11plant.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,

```

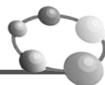


```
[sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.15;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

f=-25*x(2)+33*sin(pi*t);    %Unknown part
b=133;
sys(1)=x(2);
sys(2)=f+b*ut;
function sys=mdlOutputs(t,x,u)
f=-25*x(2)+33*sin(pi*t);    %Unknown part
sys(1)=x(1);
sys(2)=x(2);
sys(3)=f;
```

作图程序: chap6_11plot.m

```
close all;
figure(1);
plot(t,x(:,1),'r',t,x(:,2),'k:',t,x(:,4),'b-','linewidth',2);
xlabel('time(s)'),ylabel('position signal');
legend('ideal signal', 'transient position signal','position tracking');
figure(2);
subplot(311);
plot(t,z(:,1),'r',t,x(:,4),'k:', 'linewidth',2);
xlabel('time(s)'),ylabel('z1,y');
legend('practical position signal', 'position signal estimation');
```



```
subplot(312);
plot(t,z(:,2),'r',t,x(:,5),'k','linewidth',2);
xlabel('time(s)'),ylabel('z2,dy');
legend('practical speed signal', 'speed signal estimation');
subplot(313);
plot(t,z(:,3),'r',t,x(:,6),'k','linewidth',2);
xlabel('time(s)'),ylabel('z3,x3');
legend('practical uncertain part', 'uncertain part estimation');
```

2. 离散系统仿真

主程序: chap6_12.m

```
clear all;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h=0.01; %Sampling time
%Transient Parameters with TD
delta=10;
%ESO Parameters
beta1=100;beta2=200;beta3=500;
delta1=0.0025;
alfa1=0.5;alfa2=0.25;
%NPID Parameters
delta0=2*h;
alfa01=3/4;alfa02=3/2; %0<alfa1<1<alfa2
beta01=10;beta02=0.3;
kp=beta01;kd=beta02;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xk=zeros(2,1);
e1_1=0;
u_1=0;
v_1=0;
v1_1=0;v2_1=0;
z1_1=0;z2_1=0;z3_1=0;
for k=1:1:2000
time(k) = k*h;

p1=u_1;
p2=k*h;
tSpan=[0 h];
[tt,xx]=ode45('chap6_12plant',tSpan,xk,[],p1,p2);
xk = xx(length(xx),:);
y(k)=xk(1);
dy(k)=xk(2);

v(k)=sign(sin(k*h));
```



```
dv(k)=0;

f(k)=-25*dy(k)+33*sin(pi*p2);    %Unknown part
b=133;
x3(k)=f(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%TD Transient
x1=v1_1-v_1;
x2=v2_1;

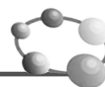
v1(k)=v1_1+h*v2_1;
v2(k)=v2_1+h*fst(x1,x2,delta,h);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ESO
e=z1_1-y(k);
z1(k)=z1_1+h*(z2_1-beta1*e);
z2(k)=z2_1+h*(z3_1-beta2*fal(e,alfa1,delta1)+b*u_1);
z3(k)=z3_1-h*beta3*fal(e,alfa2,delta1);

z1_1=z1(k);
z2_1=z2(k);
z3_1=z3(k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%N-PD and disturbance compensation
e1(k)=v1(k)-z1(k);
e2(k)=v2(k)-z2(k);

M=2;
if M==1                %ADRC
    u0(k)=kp*fal(e1(k),alfa01,delta0)+kd*fal(e2(k),alfa02,delta0);
    u(k)=u0(k)-z3(k)/b;
elseif M==2            % Ordinary PD
    u(k)=kp*(v(k)-y(k))+kd*(dv(k)-dy(k));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v_1=v(k);
v1_1=v1(k);
v2_1=v2(k);

z1_1=z1(k);z2_1=z2(k);z3_1=z3(k);
u_1=u(k);
end

if M==1
    figure(1);
    plot(time,v,'r',time,v1,'k:',time,y,'b-','linewidth',2);
```



```

xlabel('time(s)'),ylabel('position signal');
legend('ideal position signal', 'transient position signal','position tracking signal');
figure(2);
subplot(311);
plot(time,z1,'r',time,y,'k:','linewidth',2);
xlabel('time(s)'),ylabel('z1,y');
legend('practical position signal', 'position signal estimation');
subplot(312);
plot(time,z2,'r',time,dy,'k:','linewidth',2);
xlabel('time(s)'),ylabel('z2,dy');
legend('practical speed signal', 'speed signal estimation');
subplot(313);
plot(time,z3,'r',time,x3,'k:','linewidth',2);
xlabel('time(s)'),ylabel('z3,x3');
legend('practical uncertain part', 'uncertain part estimation');
elseif M==2
    figure(1);
    plot(time,v,'r',time,y,'k:','linewidth',2);
    xlabel('time(s)'),ylabel('position signal');
    legend('ideal position signal','position tracking');
end

```

被控对象程序: chap6_12plant.m

```

function dy = PlantModel(t,y,flag,p1,p2)
ut=p1;
time=p2;
dy=zeros(2,1);

f=-25*y(2)+33*sin(pi*p2);    %Unknown part
b=133;
dy(1)=y(2);
dy(2)=f+b*ut;

```


第7章 PD 鲁棒自适应控制



7.1 挠性航天器稳定 PD 鲁棒控制

7.1.1 挠性航天器建模

挠性航天器姿态运动和挠性振动方程如下^[39]

$$\begin{aligned} J\ddot{\theta} + F\ddot{q} &= u - dt \\ \ddot{q}_i + 2\xi_i\omega_i\dot{q}_i + \omega_i^2q_i + F_i\ddot{\theta} &= 0 \end{aligned} \quad (7.1)$$

式中， i 代表第 i 个挠性模态， $i=1, \dots, n$ ， θ 为姿态角度， J 为中心刚体和未变形时挠性附件的总转动惯量， u 为作用在刚体上的控制力矩， dt 为控制干扰， q_i 、 ξ_i 、 ω_i 分别为挠性附件第 i 阶模态的模态变量、阻尼比和约束模态频率， F 为耦合系数矩阵。

如图 7-1 所示为带有挠性附件的航天器模型。^[39]

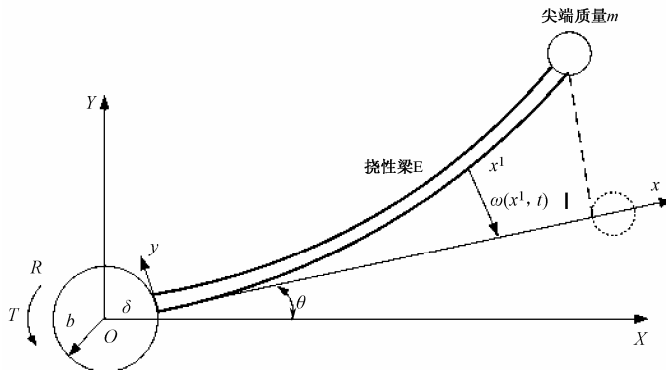


图 7-1 带有挠性附件的航天器模型

可见，挠性结构的振动和航天器的姿态运动通过耦合项互相影响。挠性结构的振动通过耦合项 $F\ddot{q}$ 影响航天器的姿态运动，控制力矩通过耦合项 $F_i\ddot{\theta}$ 影响挠性结构的振动。

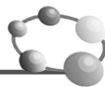
将挠性航天器的运动方程 (7.1) 改写成如下矩阵形式

$$M\ddot{X} + N\dot{X} + KX = B(u - dt) \quad (7.2)$$

式中， $X = [\theta, q^T]^T$ ， $q = [q_1, \dots, q_n]^T$ ， $M = \begin{bmatrix} J & F \\ F^T & I_{n \times n} \end{bmatrix}$ ， $F = [F_1, F_2, \dots, F_n]$ ，

$N = \begin{bmatrix} 0 & \theta_{1 \times n} \\ \theta_{n \times 1} & \Gamma \end{bmatrix}$ ， $\Gamma = \text{diag}\{2\xi_1\omega_1, 2\xi_2\omega_2, \dots, 2\xi_n\omega_n\}$ ， $K = \begin{bmatrix} 0 & \theta_{1 \times n} \\ \theta_{n \times 1} & A \end{bmatrix}$ ， $A = \text{diag}\{\omega_1^2, \omega_2^2, \dots, \omega_n^2\}$ ，

$B = [1, 0, \dots, 0]^T$ 。



挠性航天器控制的要求为: 姿态角度 θ 进行 60° 的大角度姿态机动, 只知道姿态角度 θ 和姿态角速度 $\dot{\theta}$, 其他信息如挠性附件的第 i 阶模态的模态变量 q_i 、阻尼比 ξ_i 和约束模态频率 ω_i 、耦合系数矩阵 F 未知, 要求通过控制律的设计实现姿态角度 θ 收敛到期望值 θ_d (θ_d 为固定值, $\dot{\theta}_d = 0$), 且激起的振动能得到很好的抑制, 即 $\theta \rightarrow \frac{\pi}{3}$, $\dot{\theta} \rightarrow 0$, $q \rightarrow 0$, $\dot{q} \rightarrow 0$ 。

7.1.2 PD 控制器的设计

针对挠性航天器的控制要求, 胡庆雷^[39]对其稳定性进行了详细分析。首先定义系统的 Lyapunov 函数为^[39]

$$V = \frac{1}{2} \dot{X}^T M \dot{X} + \frac{1}{2} q^T A q + \frac{1}{2} K_p e^2 \quad (7.3)$$

式中, $e = \theta - \theta_d$ 。

由式(7.3)知, M 、 A 为正定对称矩阵, $K_p > 0$, 则 $V \geq 0$, 当且仅当 $\theta = \theta_d, \dot{\theta} = 0, q = \dot{q} = 0$ 时, $V = 0$ 。

考虑外加干扰, 参考文献^[39]设计一种带有鲁棒项的 PD 控制器为

$$u = -K_p(\theta - \theta_d) - K_d \dot{\theta} - \eta \text{sign} \dot{\theta} \quad (7.4)$$

式中, $K_p > 0$, $K_d > 0$, θ_d 为航天器期望的旋转姿态角, $\eta \geq |\dot{\theta}|$ 。

将控制律式(7.4)代入式(7.2), 可得到如下闭环系统

$$M\ddot{X} + N\dot{X} + KX = \begin{bmatrix} -K_p(\theta - \theta_d) - K_d\dot{\theta} - \eta \text{sign} \dot{\theta} - d\dot{\theta} \\ 0 \end{bmatrix} \quad (7.5)$$

由于 $\dot{e} = \dot{\theta} - \dot{\theta}_d = \dot{\theta}$, $\dot{q}^T A q = \dot{X}^T K X$, $\dot{X}^T N \dot{X} = \dot{q}^T \Gamma q$, 则

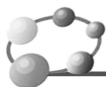
$$\begin{aligned} \dot{V} &= \dot{X}^T M \ddot{X} + \dot{q}^T A q + K_p e \dot{e} = \dot{X}^T (M\ddot{X} + KX) + K_p e \dot{e} \\ &= -\dot{X}^T N \dot{X} + \dot{X}^T (M\ddot{X} + N\dot{X} + KX) + K_p e \dot{e} \\ &= -\dot{q}^T \Gamma q - \dot{\theta} (K_p e + K_d \dot{\theta} + \eta \text{sgn} \dot{\theta} + d\dot{\theta}) + K_p e \dot{e} \\ &= -\dot{q}^T \Gamma q - \dot{\theta} K_d \dot{\theta} - \eta |\dot{\theta}| - \dot{\theta} d\dot{\theta} \leq -\eta |\dot{\theta}| - \dot{\theta} d\dot{\theta} \leq 0 \end{aligned}$$

由于 Γ 正定, 当 $K_d > 0$ 且取值较大且 $f - \hat{f} \rightarrow 0$ 时, $\dot{V} \leq 0$; 当且仅当 $\dot{\theta} = 0$, $\dot{q} = 0$ 时, $\dot{V} = 0$, 因此, 闭环系统 Lyapunov 意义稳定。

7.1.3 仿真实例

被控对象为式(7.1)所示的非线性耦合模型, 仅考虑一、二阶振动模态, 系统总的转动惯量(包括中心刚体的转动惯量、挠性梁对中心的转动惯量及尖端质量对中心的转动惯量)为 $3250 \text{ kg} \cdot \text{m}^2$, 耦合系数矩阵取 $F = [54 \quad 6]$, 一、二阶模态频率分别为 1.2 rad/s 和 3.4 rad/s , 一二阶模态阻尼系数都取为 0.01 。

为了仿真实现, 取 $x = [\theta, q_1, q_2]^T$, $x_1 = x$, $x_2 = \dot{x}$, 将式(7.2)写为以下形式



$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = M^{-1}(-Nx_2 - Kx_1 + B(u - dt))$$

式中, $M = \begin{bmatrix} J & F_1 & F_2 \\ F_1 & 1 & 0 \\ F_2 & 0 & 1 \end{bmatrix}$, $N = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2\zeta_1\omega_1 & 0 \\ 0 & 0 & 2\zeta_2\omega_2 \end{bmatrix}$, $K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \omega_1^2 & 0 \\ 0 & 0 & \omega_2^2 \end{bmatrix}$, $B = [1 \ 0 \ 0]^T$ 。

如果不考虑干扰 dt , 采用式 (7.4) 设计的控制器, 取 $\eta = 0$, 进行 60° 的大角度姿态机动。控制参数选择 $K_p = 100, K_d = 1000$, 仿真结果如图 7-2 至图 7-4 所示。

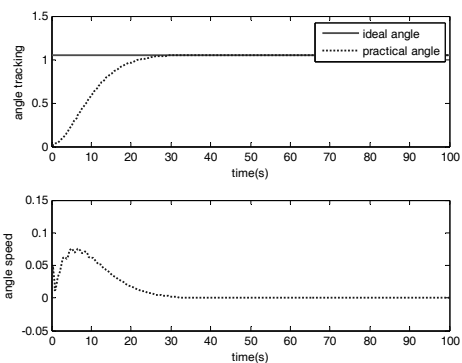


图 7-2 姿态角度和角速度响应

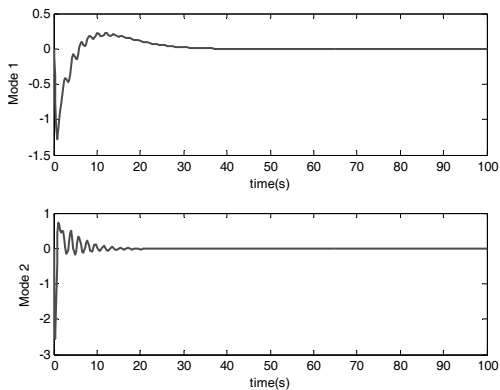


图 7-3 一阶、二阶模态响应

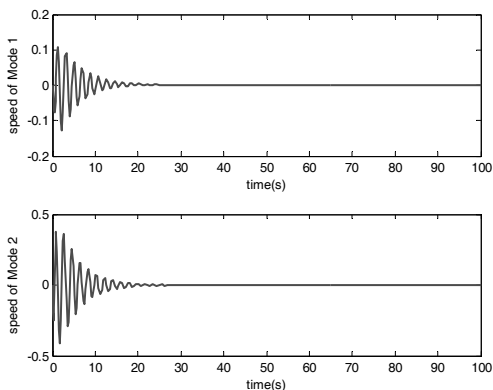
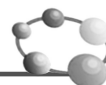


图 7-4 一阶、二阶模态变化率



仿真程序

Simulink 主程序: chap7_1sim.mdl (见图 7-5)

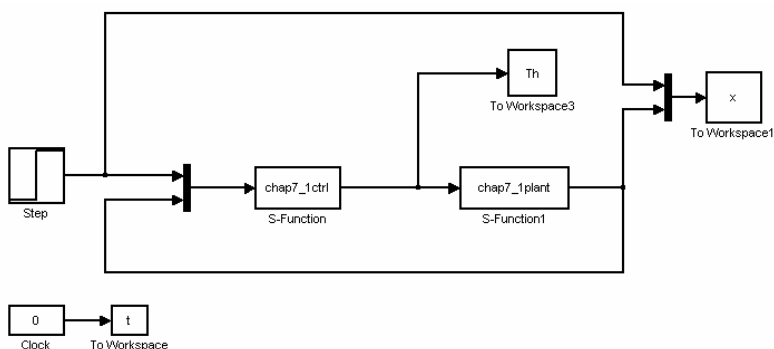


图 7-5 PD 鲁棒控制主程序

控制器子程序: chap7_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
```

```
switch flag,
```

```
case 0,
```

```
    [sys,x0,str,ts]=mdlInitializeSizes;
```

```
case 3,
```

```
    sys=mdlOutputs(t,x,u);
```

```
case {2,4,9}
```

```
    sys=[];
```

```
otherwise
```

```
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
```

```
sizes = simsizes;
```

```
sizes.NumOutputs      = 1;
```

```
sizes.NumInputs       = 7;
```

```
sizes.DirFeedthrough = 1;
```

```
sizes.NumSampleTimes = 1;
```

```
sys = simsizes(sizes);
```

```
x0  = [];
```

```
str = [];
```

```
ts  = [0 0];
```

```
function sys=mdlOutputs(t,x,u)
```

```
thd=u(1);
```

```
dthd=0;
```

```
ddthd=0;
```

```
th=u(2);
```

```
q1=u(3);
```



```
q2=u(4);

dth=u(5);
dq1=u(6);
dq2=u(7);

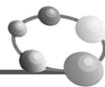
q=[q1 q2]';
dq=[dq1 dq2]';
e=th-thd;

kp=100;
kd=1000;
xite=4.0;
xite=0;
ut=-kp*e-kd*dth-xite*sign(dth);

sys(1)=ut;
```

被控对象子程序: chap7_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 6;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 6;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0 0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
thd=pi/3;qd1=0;qd2=0;
```



```

th=x(1);
q1=x(2);
q2=x(3);
ut=u(1);

J=3250;
F1=54;
F2=6;
w1=1.2;w2=3.4;
Ks1=0.01;Ks2=0.01;

M=[J F1 F2;F1 1 0;F2 0 1];
N=[0 0 0;0 2*Ks1*w1 0;0 0 2*Ks2*w2];
K=[0 0 0;0 w1^2 0;0 0 w2^2];

dt=0*3*sin(t);

x1=[x(1) x(2) x(3)]';
x2=[x(4) x(5) x(6)]';
B=[1 0 0]';

dx2=inv(M)*(-N*x2-K*x1+B*(ut-dt));
sys(1)=x2(1);    %th
sys(2)=x2(2);    %q1
sys(3)=x2(3);    %q2

sys(4)=dx2(1);    %dth
sys(5)=dx2(2);    %dq1
sys(6)=dx2(3);    %dq2
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);    %th
sys(2)=x(2);    %q1
sys(3)=x(3);    %q2
sys(4)=x(4);    %dth
sys(5)=x(5);    %dq1
sys(6)=x(6);    %dq2

```

作图子程序: chap7_1plot.m

```

close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'b:','linewidth',2);
xlabel('time(s)');ylabel('angle tracking');
legend('ideal angle','practical angle');

```



```
subplot(212);  
plot(t,x(:,5),'b:','linewidth',2);  
xlabel('time(s)');ylabel('angle speed');  
  
figure(2);  
subplot(211);  
plot(t,x(:,3),'r','linewidth',2);  
xlabel('time(s)');ylabel(' Mode 1');  
subplot(212);  
plot(t,x(:,6),'r','linewidth',2);  
xlabel('time(s)');ylabel('Mode 2');  
  
figure(3);  
subplot(211);  
plot(t,x(:,4),'r','linewidth',2);  
xlabel('time(s)');ylabel('speed of Mode 1');  
subplot(212);  
plot(t,x(:,7),'r','linewidth',2);  
xlabel('time(s)');ylabel('speed of Mode 2');  
  
figure(4);  
plot(t,Th(:,1),'r','linewidth',2);  
xlabel('time(s)');ylabel('Th');
```



7.2 基于名义模型的机械手 PI 鲁棒控制

本节介绍一种基于名义模型的机械手 PI 鲁棒控制的设计、分析和仿真方法。

7.2.1 问题的提出

设 n 关节机械手方程为

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau - d(\dot{q}) \quad (7.6)$$

式中, $D(q)$ 为 $n \times n$ 阶正定惯性矩阵, $C(q,\dot{q})$ 为 $n \times n$ 阶离心和哥氏力项, $G(q)$ 为 $n \times 1$ 阶重力项, $d(\dot{q})$ 为摩擦力矩。

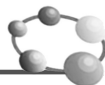
在实际控制中, $D(q)$ 、 $C(q,\dot{q})$ 和 $G(q)$ 为未知, 取

$$D(q) = D_0(q) + E_D$$

$$C(q,\dot{q}) = C_0(q,\dot{q}) + E_C$$

$$G(q) = G_0(q) + E_G$$

式中, E_D 、 E_C 和 E_G 分别为 $D(q)$ 、 $C(q,\dot{q})$ 和 $G(q)$ 的建模误差, 则



$$\begin{aligned} & D(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + G(q) \\ & = D_0(q)\ddot{q}_r + C_0(q, \dot{q})\dot{q}_r + G_0(q) + E_M \end{aligned} \quad (7.7)$$

式中, $E_M = E_D\ddot{q}_r + E_C\dot{q}_r + E_G$ 。

7.2.2 鲁棒控制律的设计

定义误差为 $e(t) = q_d(t) - q(t)$, 其中 $q_d(t)$ 为理想的位置指令, $q(t)$ 为实际位置。定义误差函数为

$$r = \dot{e} + \Lambda e \quad (7.8)$$

取 $\dot{q}_r = r(t) + \dot{q}(t)$ 和 $\ddot{q}_r = \dot{r}(t) + \ddot{q}(t)$, 则

$$\dot{q}_r = \dot{q}_d + \Lambda e$$

$$\ddot{q}_r = \ddot{q}_d + \Lambda \dot{e}$$

式中, $\Lambda > 0$ 。

则由式 (7.6) 得

$$\begin{aligned} \tau &= D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) + d(\dot{q}) \\ &= D(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + G(q) - D(q)\dot{r} - C(q, \dot{q})r + d(\dot{q}) \\ &= D_0(q)\ddot{q}_r + C_0(q, \dot{q})\dot{q}_r + G_0(q) - D(q)\dot{r} - C(q, \dot{q})r + E_M + d(\dot{q}) \end{aligned} \quad (7.9)$$

式中, $E_M = \Delta D(q)\ddot{q}_r + \Delta C(q, \dot{q})\dot{q}_r + \Delta G(q)$, $E = E_M + d(\dot{q})$ 。

采用 Ge S.S.等^[40]所提出的控制方法, 将控制律设计为

$$\tau = \tau_m + K_p r + K_i \int r dt + \tau_r \quad (7.10)$$

式中, $K_p > 0$, $K_i > 0$, τ_m 为基于名义模型的控制项, τ_r 为用于建模误差和摩擦干扰的鲁棒项。

取

$$\tau_m = D_0(q)\ddot{q}_r + C_0(q, \dot{q})\dot{q}_r + G_0(q) \quad (7.11)$$

$$\tau_r = K_r \text{sgn}(r) \quad (7.12)$$

式中, $K_r = \text{diag}[k_{r_{ii}}]$, $k_{r_{ii}} \geq |E_i|$, $i = 1, \dots, n$ 。

由式 (7.9) 和式 (7.10) 的右边得

$$\begin{aligned} & D_0(q)\ddot{q}_r + C_0(q, \dot{q})\dot{q}_r + G_0(q) - D(q)\dot{r} - C(q, \dot{q})r + E_M + d(\dot{q}) \\ &= D_0(q)\ddot{q}_r + C_0(q, \dot{q})\dot{q}_r + G_0(q) + K_p r + K_i \int_0^t r dt + K_r \text{sgn}(r) \end{aligned}$$

上式又可以写为

$$D(q)\dot{r} + C(q, \dot{q})r + K_i \int_0^t r dt = -K_p r - K_r \text{sgn}(r) + E \quad (7.13)$$

7.2.3 稳定性分析

参考 Ge S.S.等^[40]所设计的 Lyapunov 函数, 将 Lyapunov 函数取为

$$V = \frac{1}{2} r^T D r + \frac{1}{2} \left(\int_0^t r d\tau \right)^T K_i \left(\int_0^t r d\tau \right) \quad (7.14)$$



则

$$\dot{V} = \mathbf{r}^T \left[\mathbf{D}\dot{\mathbf{r}} + \frac{1}{2}\dot{\mathbf{D}}\mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} d\tau \right]$$

考虑到 $\dot{\mathbf{D}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ 的斜对称特性, 有

$$\dot{V} = \mathbf{r}^T \left[\mathbf{D}\dot{\mathbf{r}} + \mathbf{C}\mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} d\tau \right]$$

将式 (7.13) 代入式 (7.14), 得

$$\dot{V} = -\mathbf{r}^T \mathbf{K}_p \mathbf{r} + \mathbf{r}^T \mathbf{E} - \mathbf{r}^T \mathbf{K}_r \text{sign}(\mathbf{r})$$

考虑 $k_{ri} \geq |E_i|$, 得

$$\dot{V} \leq -\mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq 0$$

7.2.4 仿真实例

一个典型的双关节刚性机械手示意图如图 7-6 所示。

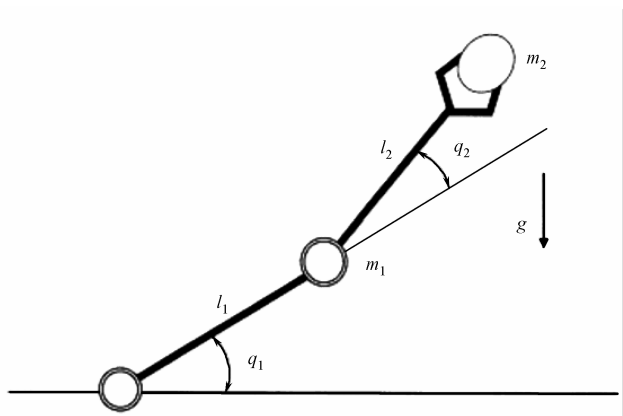


图 7-6 双关节刚性机械手示意图

取双关节机器人系统, 不考虑摩擦力和干扰, 其动力学模型为

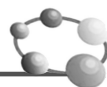
$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau}$$

式中

$$\begin{aligned} \mathbf{D}(\mathbf{q}) &= \begin{bmatrix} p_1 + p_2 + 2p_3 \cos q_2 & p_2 + p_3 \cos q_2 \\ p_2 + p_3 \cos q_2 & p_2 \end{bmatrix} \\ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \begin{bmatrix} -p_3 \dot{q}_2 \sin q_2 & -p_3 (\dot{q}_1 + \dot{q}_2) \sin q_2 \\ p_3 \dot{q}_1 \sin q_2 & 0 \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} p_4 g \cos q_1 + p_5 g \cos(q_1 + q_2) \\ p_5 g \cos(q_1 + q_2) \end{bmatrix} \end{aligned}$$

取 $\mathbf{p} = [2.90 \quad 0.76 \quad 0.87 \quad 3.04 \quad 0.87]^T$ 。

被控对象初值为 $\mathbf{q}_0 = [0.09 \quad -0.09]^T$, $\dot{\mathbf{q}}_0 = [0.0 \quad 0.0]^T$ 。仿真中, 取位置指令为



$q_{d1} = 0.5\sin(\pi t)$, $q_{d2} = \sin(\pi t)$ 。取控制器参数为 $K_p = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $K_i = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $K_r = 15$, $A = \begin{bmatrix} 5.0 & 0 \\ 0 & 5.0 \end{bmatrix}$ 。控制律取式 (7.10), 采用 Simulink 和 S 函数进行控制系统的设计, 仿真结果如图 7-7 和图 7-8 所示。

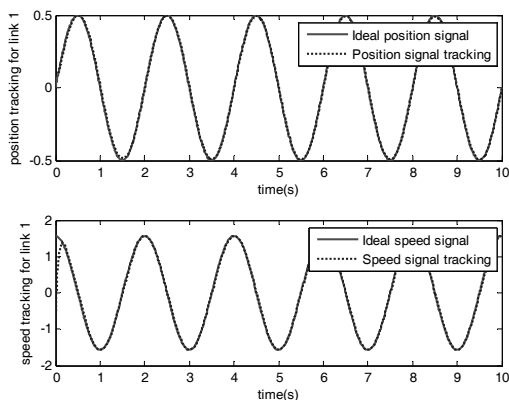


图 7-7 关节 1 的位置速度跟踪

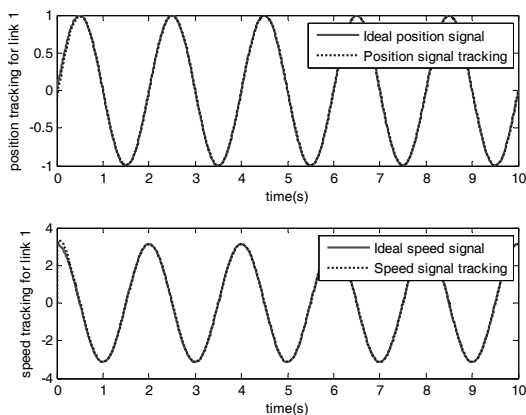


图 7-8 关节 2 的位置速度跟踪

双力臂仿真程序

Simulink 主程序: chap7_2sim.mdl (见图 7-9)

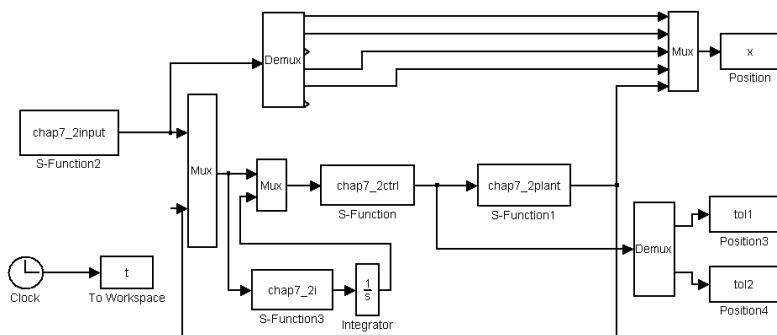


图 7-9 双力臂控制主程序



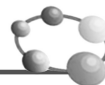
位置指令子程序: chap7_2input.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 6;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];

function sys=mdlOutputs(t,x,u)
S=2;
if S==1
    qd0=[0;0];
    qdtf=[1;2];
    td=1;
    if t<1
        qd1=qd0(1)+(-2*t.^3/td^3+3*t.^2/td^2)*(qdtf(1)-qd0(1));
        qd2=qd0(2)+(-2*t.^3/td^3+3*t.^2/td^2)*(qdtf(2)-qd0(2));
        d_qd1=(-6*t.^2/td^3+6*t./td^2)*(qdtf(1)-qd0(1));
        d_qd2=(-6*t.^2/td^3+6*t./td^2)*(qdtf(2)-qd0(2));
        dd_qd1=(-12*t/td^3+6/td^2)*(qdtf(1)-qd0(1));
        dd_qd2=(-12*t/td^3+6/td^2)*(qdtf(2)-qd0(2));
    else
        qd1=qdtf(1);
        qd2=qdtf(2);
```



```

        d_qd1=0;
        d_qd2=0;

        dd_qd1=0;
        dd_qd2=0;
    end
elseif S==2
    qd1=0.5*sin(pi*t);
    d_qd1=0.5*pi*cos(pi*t);
    dd_qd1=-0.5*pi*pi*sin(pi*t);

    qd2=sin(pi*t);
    d_qd2=pi*cos(pi*t);
    dd_qd2=-pi*pi*sin(pi*t);
end

sys(1)=qd1;
sys(2)=d_qd1;
sys(3)=dd_qd1;
sys(4)=qd2;
sys(5)=d_qd2;
sys(6)=dd_qd2;

```

控制器子程序: chap7_2ctrl.m

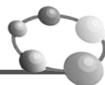
```

function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 12;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];

```



```
str = [];  
ts = [];  
function sys=mdlOutputs(t,x,u)  
qd1=u(1);  
d_qd1=u(2);  
dd_qd1=u(3);  
qd2=u(4);  
d_qd2=u(5);  
dd_qd2=u(6);  
  
q1=u(7);  
d_q1=u(8);  
q2=u(9);  
d_q2=u(10);  
  
e1=qd1-q1;  
e2=qd2-q2;  
de1=d_qd1-d_q1;  
de2=d_qd2-d_q2;  
e=[e1;e2];  
de=[de1;de2];  
Fai=5*eye(2);  
r=de+Fai*e;  
  
dqd=[d_qd1;d_qd2];  
dqr=dqd+Fai*e;  
ddqd=[dd_qd1;dd_qd2];  
ddqr=ddqd+Fai*de;  
  
p=[2.9 0.76 0.87 3.04 0.87];  
g=9.8;  
D=[p(1)+p(2)+2*p(3)*cos(q2) p(2)+p(3)*cos(q2);  
p(2)+p(3)*cos(q2) p(2)];  
C=[-p(3)*d_q2*sin(q2) -p(3)*(d_q1+d_q2)*sin(q2);  
p(3)*d_q1*sin(q2) 0];  
G=[p(4)*g*cos(q1)+p(5)*g*cos(q1+q2);  
p(5)*g*cos(q1+q2)];  
tolm=D*ddqr+C*dqr+G;  
  
Kr=15;  
tolr=Kr*sign(r);  
  
Kp=100*eye(2);  
Ki=100*eye(2);  
  
I=[u(11);u(12)];
```



```
tol=tolm+Kp*r+Ki*I+tolr;
```

```
sys(1)=tol(1);
```

```
sys(2)=tol(2);
```

积分子程序: chap7_2i.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
```

```
switch flag,
```

```
case 0,
```

```
    [sys,x0,str,ts]=mdlInitializeSizes;
```

```
case 3,
```

```
    sys=mdlOutputs(t,x,u);
```

```
case {2,4,9}
```

```
    sys=[];
```

```
otherwise
```

```
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
```

```
sizes = simsizes;
```

```
sizes.NumContStates = 0;
```

```
sizes.NumDiscStates = 0;
```

```
sizes.NumOutputs = 2;
```

```
sizes.NumInputs = 10;
```

```
sizes.DirFeedthrough = 1;
```

```
sizes.NumSampleTimes = 0;
```

```
sys = simsizes(sizes);
```

```
x0 = [];
```

```
str = [];
```

```
ts = [];
```

```
function sys=mdlOutputs(t,x,u)
```

```
qd1=u(1);
```

```
d_qd1=u(2);
```

```
dd_qd1=u(3);
```

```
qd2=u(4);
```

```
d_qd2=u(5);
```

```
dd_qd2=u(6);
```

```
q1=u(7);
```

```
d_q1=u(8);
```

```
q2=u(9);
```

```
d_q2=u(10);
```

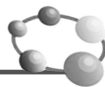
```
q=[q1;q2];
```



```
e1=qd1-q1;  
e2=qd2-q2;  
de1=d_qd1-d_q1;  
de2=d_qd2-d_q2;  
e=[e1;e2];  
de=[de1;de2];  
Hur=5*eye(2);  
r=de+Hur*e;  
  
sys(1:2)=r;
```

被控对象子程序: chap7_2plant.m

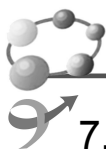
```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,  
    [sys,x0,str,ts]=mdlInitializeSizes;  
case 1,  
    sys=mdlDerivatives(t,x,u);  
case 3,  
    sys=mdlOutputs(t,x,u);  
case {2, 4, 9 }  
    sys = [];  
otherwise  
    error(['Unhandled flag = ',num2str(flag)]);  
end  
function [sys,x0,str,ts]=mdlInitializeSizes  
sizes = simsizes;  
sizes.NumContStates = 4;  
sizes.NumDiscStates = 0;  
sizes.NumOutputs = 4;  
sizes.NumInputs = 2;  
sizes.DirFeedthrough = 0;  
sizes.NumSampleTimes = 0;  
sys=simsizes(sizes);  
x0=[0.09 0 -0.09 0];  
str=[];  
ts=[];  
function sys=mdlDerivatives(t,x,u)  
p=[2.9 0.76 0.87 3.04 0.87];  
g=9.8;  
  
D0=[p(1)+p(2)+2*p(3)*cos(x(3)) p(2)+p(3)*cos(x(3));  
    p(2)+p(3)*cos(x(3)) p(2)];  
C0=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));  
    p(3)*x(2)*sin(x(3)) 0];  
G0=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));  
    p(5)*g*cos(x(1)+x(3))];  
  
tol=u(1:2);
```



```
dq=[x(2);x(4)];  
d=20*sign(dq);  
  
S=inv(D0)*(tol-C0*dq-G0-d);  
  
sys(1)=x(2);  
sys(2)=S(1);  
sys(3)=x(4);  
sys(4)=S(2);  
function sys=mdlOutputs(t,x,u)  
sys(1)=x(1);  
sys(2)=x(2);  
sys(3)=x(3);  
sys(4)=x(4);
```

绘图子程序: chap7_2plot.m

```
close all;  
  
figure(1);  
subplot(211);  
plot(t,x(:,1),'r',t,x(:,5),'b','linewidth',2);  
xlabel('time(s)');ylabel('position tracking for link 1');  
legend('Ideal position signal','Position signal tracking');  
subplot(212);  
plot(t,x(:,2),'r',t,x(:,6),'b','linewidth',2);  
xlabel('time(s)');ylabel('speed tracking for link 1');  
legend('Ideal speed signal','Speed signal tracking');  
  
figure(2);  
subplot(211);  
plot(t,x(:,3),'r',t,x(:,7),'b','linewidth',2);  
xlabel('time(s)');ylabel('position tracking for link 1');  
legend('Ideal position signal','Position signal tracking');  
subplot(212);  
plot(t,x(:,4),'r',t,x(:,8),'b','linewidth',2);  
xlabel('time(s)');ylabel('speed tracking for link 1');  
legend('Ideal speed signal','Speed signal tracking');  
  
figure(3);  
subplot(211);  
plot(t,tol1(:,1),'r','linewidth',2);  
xlabel('time(s)');ylabel('control input of link 1');  
subplot(212);  
plot(t,tol2(:,1),'r','linewidth',2);  
xlabel('time(s)');ylabel('control input of link 2');
```

7.3 基于 Anti-windup 的 PID 控制

7.3.1 Anti-windup 基本原理

控制器的 windup 问题一般被认为是当控制器输出和输入之间存在非线性特性 $N(u)$ 时, 产生于 PI/PID 控制器积分部分的一种不良现象。^[71]

任何实际的控制系统都包含饱和和非线性特性。由于饱和特性的存在, 导致控制器产生 windup 问题, 进而影响系统在较大信号输入时的性能指标, 乃至整个控制系统的稳定性,

饱和特性对实际系统的影响十分严重。由于在系统调试过程中大都以小信号作为系统的调试信号, 所以造成设计者对饱和特性非线性的认识不足而忽略了它的存在。在实际过程中, 当有大信号输入或其他情况使控制系统进入饱和状态时, 系统的性能会产生较大的降低, 不能满足性能的要求。因此, 引入适当的补偿环节, 使控制系统在出现饱和现象时仍能达到比较满意的性能指标, Anti-windup 设计技术已经成为进行具有饱和特性的控制系统设计基本思路。

7.3.2 基于 Anti-windup 的 PID 控制

针对积分 windup 现象, A.S. Hodel 等^[41]提出了抗饱和的变结构 PID 控制算法, 其结构如图 7-10 所示。

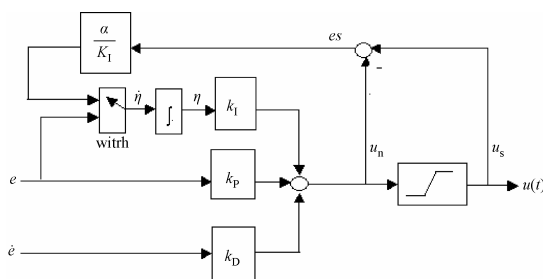


图 7-10 基于 Anti-windup 的 PID 控制器结构

图 7-10 描述的抗饱和控制思想为: 对控制输入饱和误差 $u_n - u_s$ 进行积分, 并通过自适应系数调整将其加到 PID 控制中的积分项中。

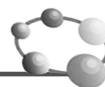
A.S. Hodel 等^[41]所提出的抗饱和变结构 PID 控制算法如下: 采用系数 η 实现积分项的自适应调整, 其自适应变化律为

$$\dot{\eta} = \begin{cases} -\alpha(u_n - u_s)/K_I, & u_n \neq u_s, e(u_n - \bar{u}) > 0 \\ e, & u_n = u_s \end{cases} \quad (7.15)$$

式中, $\bar{u} = (u_{\min} + u_{\max})/2$, $\alpha > 0$, $u_{\max}$ 和 $u_{\min}$ 为控制输入信号的最大最小值。

基于 Anti-windup 的 PID 控制算法为

$$u(t) = k_p e(t) + k_i \eta + k_d \frac{de(t)}{dt} \quad (7.16)$$



7.3.3 仿真实例

设被控对象为 $\frac{133}{s^2 + 25s}$ ，指令为一大的阶跃信号 10，控制输入限制在在 $[0, 10]$ 范围内，分别采用控制式器和传统 PID 进行仿真。仿真中，当取 $M=1$ 时为基于 Anti-windup 的 PID 控制，当 $M=2$ 时为传统 PID 控制。针对连续系统，采用 Simulink 方式进行仿真，取 $\alpha=1.0$ ， $k_p=50$ ， $k_i=10$ ， $k_d=1$ 。仿真结果如图 7-11 至图 7-14 所示。

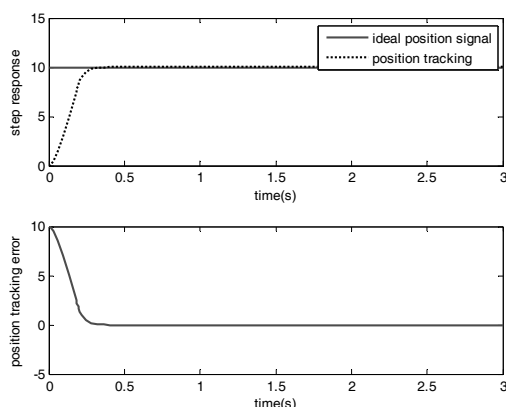


图 7-11 基于 Anti-windup 的阶跃响应

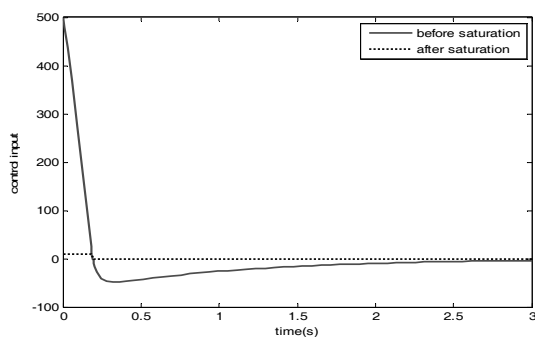


图 7-12 基于 Anti-windup 的控制器输出

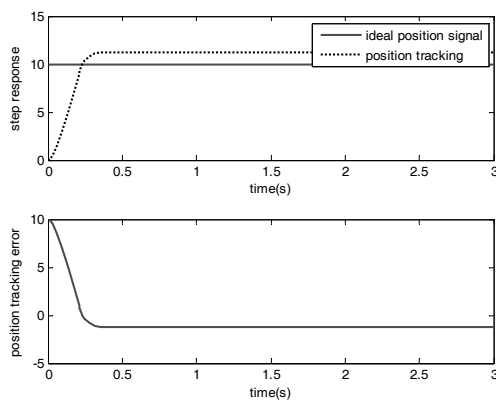


图 7-13 基于传统 PID 的阶跃响应

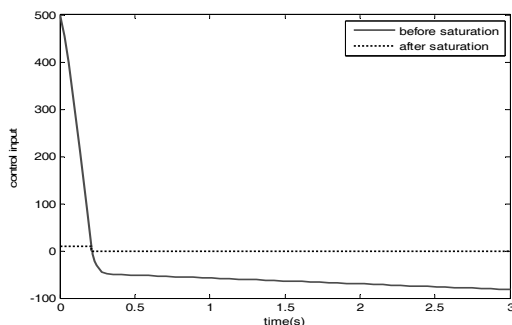


图 7-14 传统 PID 方法的控制器输出

仿真程序

Simulink 主程序: chap7_3sim.mdl (见图 7-15)

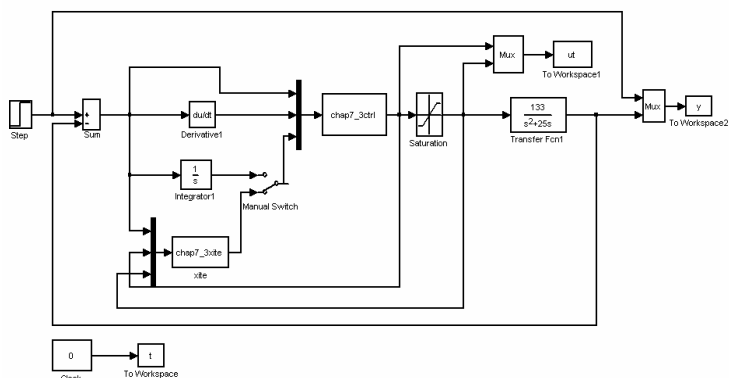
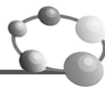


图 7-15 基于 Anti-windup 的 PID 控制主程序

控制器子程序: chap7_3ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
```



```
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
e=u(1);
de=u(2);
ei=u(3);

kp=50;ki=10;kd=1;

ut=kp*e+ki*ei+kd*de;    %Without anti-windup
sys(1)=ut;
```

自适应调整系数子程序: chap7_3xite.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
e=u(1);
un=u(2);
us=u(3);
```



```

ki=10;
alfa=1.0;

umin=0;umax=10;
ua=(umin+umax)/2;
if un~=us&&e*(un-ua)>0
    dxite=-alfa*(un-us)/ki;
else
    dxite=e;
end
sys(1)=dxite;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);    %xite

```

作图子程序: chap7_3plot.m

```

close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('step response');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,y(:,1)-y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('position tracking error');

figure(2);
plot(t,ut(:,1),'r',t,ut(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('control input');
legend('before saturation','after saturation');

```



7.4 基于 PD 增益自适应调节的模型参考自适应控制

通过对参考文献^[42]的控制方法进行详细推导及分析,探讨基于 PD 增益自适应调节的模型参考自适应控制的设计、分析和仿真方法。

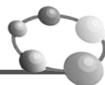
7.4.1 问题描述

设被控对象为

$$\ddot{\theta} + a_1 \dot{\theta} + a_2 \theta = au \quad (7.17)$$

式中, θ 为系统输出转角, u 为控制输入, a_1 、 a_2 为非负的实数, a 为正实数。

定义参考模型为



$$\ddot{\theta}_m + b_1 \dot{\theta}_m + b_2 \theta_m = br \quad (7.18)$$

式中, θ_m 为模型输出, r 为系统指令输入, b_1 、 b_2 、 b 为已知正实数。

定义误差信号为

$$e = \theta_m - \theta \quad (7.19)$$

由式 (7.17) 至式 (7.19) 得到误差动态方程

$$\ddot{e} + b_1 \dot{e} + b_2 e = br - au + (a_1 - b_1)\dot{\theta} + (a_2 - b_2)\theta \quad (7.20)$$

定义 $\varepsilon = [e \quad \dot{e}]^T$, 则得到误差状态方程如下

$$\dot{\varepsilon} = A\varepsilon - \begin{bmatrix} 0 \\ a \end{bmatrix} u + \begin{bmatrix} 0 \\ \Delta \end{bmatrix} \quad (7.21)$$

$$\text{式中, } \Delta = br + (a_1 - b_1)\dot{\theta} + (a_2 - b_2)\theta, \quad A = \begin{bmatrix} 0 & 1 \\ -b_2 & -b_1 \end{bmatrix}。$$

7.4.2 控制律的设计与分析

通过 b_1 和 b_2 的设计, 使矩阵 A 的特征值具有负实部, 从而保证模型式 (7.21) 是稳定的。则存在对称正定矩阵 P 和 Q , 使得下式成立

$$A^T P + PA = -Q \quad (7.22)$$

取 PD 形式定义控制项为

$$\hat{e} = p_{21}e + p_{22}\dot{e} \quad (7.23)$$

$$\text{其中 } [p_{21} \quad p_{22}] = [0 \quad 1] \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = [0 \quad 1] P。$$

刘强等^[42]采用前馈加 PD 反馈的形式设计控制律, 表达式为

$$u = k_0 r + k_1 \theta + k_2 \dot{\theta} \quad (7.24)$$

将控制律式 (7.24) 代入式 (7.20) 得

$$\begin{aligned} \ddot{e} + a_1 \dot{e} + a_2 e &= br - a(k_0 r + k_1 \theta + k_2 \dot{\theta}) + (a_1 - b_1)\dot{\theta} + (a_2 - b_2)\theta \\ &= br - ak_0 r - ak_1 \theta - ak_2 \dot{\theta} + (a_1 - b_1)\dot{\theta} + (a_2 - b_2)\theta \\ &= (b - ak_0)r + (a_2 - b_2 - ak_1)\theta + (a_1 - b_1 - ak_2)\dot{\theta} \end{aligned} \quad (7.25)$$

在式 (7.25) 中, 为保证 $\ddot{e} + b_1 \dot{e} + b_2 e \rightarrow 0$, 刘强等^[42]设计 Lyapunov 函数为:

$$V = \frac{1}{2} \varepsilon^T P \varepsilon + \frac{1}{2\lambda_0 a} (b - ak_0)^2 + \frac{1}{2\lambda_1 a} (a_2 - b_2 - ak_1)^2 + \frac{1}{2\lambda_2 a} (a_1 - b_1 - ak_2)^2$$

式中, $\lambda_i > 0, i=0,1,2$ 。

对 V 取导数, 由于

$$\left(\frac{1}{2} \varepsilon^T P \varepsilon \right)' = \frac{1}{2} \varepsilon^T (A^T P + PA) \varepsilon + \varepsilon^T P \begin{bmatrix} 0 \\ 1 \end{bmatrix} (\Delta - au)$$

考虑到式 (7.22), 且



$$\varepsilon^T P \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \quad 1] P \varepsilon = [p_{21} \quad p_{22}] \varepsilon = [p_{21} \quad p_{22}] \begin{bmatrix} e \\ \dot{e} \end{bmatrix}^T = \hat{e}$$

则

$$\dot{V} = -\frac{1}{2} \varepsilon^T Q \varepsilon + \hat{e}(\Delta - au) - \frac{\dot{k}_0}{\lambda_0}(b - ak_0) - \frac{\dot{k}_1}{\lambda_1}(a_2 - b_2 - ak_1) - \frac{\dot{k}_2}{\lambda_2}(a_1 - b_1 - ak_2)$$

将式 (7.21) 中的 Δ 项和控制律式 (7.24) 代入上式, 得

$$\hat{e}(\Delta - au) = \hat{e}r(b - ak_0) + \hat{e}\theta(a_2 - b_2 - ak_1) + \hat{e}\dot{\theta}(a_1 - b_1 - ak_2)$$

则

$$\dot{V} = -\frac{1}{2} \varepsilon^T Q \varepsilon + \left(\hat{e}r - \frac{\dot{k}_0}{\lambda_0} \right) (b - ak_0) + \left(\hat{e}\theta - \frac{\dot{k}_1}{\lambda_1} \right) (a_2 - b_2 - ak_1) + \left(\hat{e}\dot{\theta} - \frac{\dot{k}_2}{\lambda_2} \right) (a_1 - b_1 - ak_2)$$

设计自适应律为^[42]

$$\dot{k}_0 = \lambda_0 \hat{e}r \quad (7.26a)$$

$$\dot{k}_1 = \lambda_1 \hat{e}\theta \quad (7.26b)$$

$$\dot{k}_2 = \lambda_2 \hat{e}\dot{\theta} \quad (7.26c)$$

将式 (7.26) 代入 $\dot{V}$ 中, 得

$$\dot{V} = -\frac{1}{2} \varepsilon^T Q \varepsilon \leq 0$$

7.4.3 仿真实例

设被控对象为

$$\ddot{\theta} + 20\dot{\theta} + 25\theta = 133u$$

参考模型为

$$\ddot{\theta}_m + 20\dot{\theta}_m + 30\theta_m = 50r$$

则 $A = \begin{bmatrix} 0 & 1 \\ -b_2 & -b_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -30 & -20 \end{bmatrix}$, 在 MATLAB 下, 运行 `eig(A)` 可得 A 的特征值为 -1.6334

和 -18.3666 。

指令信号分两种, 当 $S=1$ 时为方波, 当 $S=2$ 时为正弦。取 $S=1$, 即指令信号为方波, MATLAB

表示为 $r = \text{sgn}(\sin(0.1\pi t))$, 控制器参数取 $\lambda_0 = \lambda_1 = \lambda_2 = 200$, 取 $Q = \begin{bmatrix} 20 & 10 \\ 10 & 20 \end{bmatrix}$ 。

采用 M 语言进行仿真, 可得 $P = \begin{bmatrix} 12.1667 & 0.3333 \\ 0.3333 & 0.5167 \end{bmatrix}$ 。仿真中, 被控对象初始状态取为

$[0.5, 0]$, 参考模型初始状态取为 $[0, 0]$, 自适应参数 k_0, k_1, k_2 的初始状态取为 $[0, 0, 0]$ 。仿真结果如图 7-16 至图 7-18 所示, 图 7-16 参考模型位置跟踪及跟踪误差, 图 7-17 为控制器输出, 图 7-18 为控制器参数 k_0, k_1, k_2 的自适应变化过程。还可以采用 Simulink 实现控制系统的仿真, 程序附后。

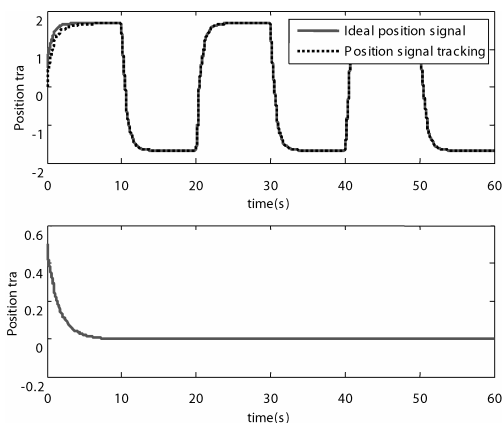
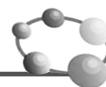


图 7-16 位置跟踪及跟踪误差

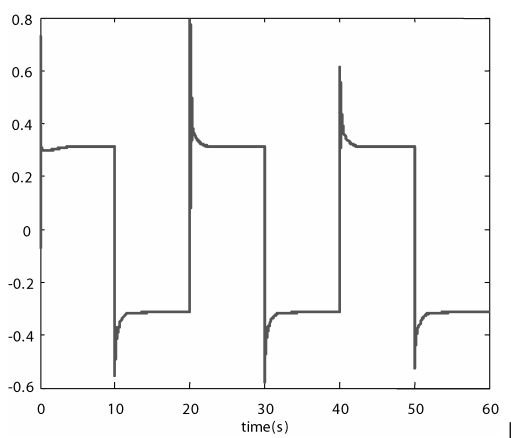
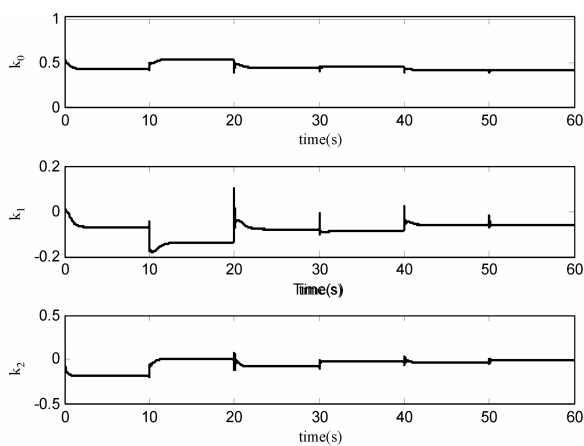


图 7-17 控制器输出信号

图 7-18 k_0, k_1, k_2 的变化过程

M 语言仿真程序

主程序: chap7_4.m

```
%Adaptive Robust Control Based on PD Term
clear all;
close all;
```




```
global S

ts=0.001;
TimeSet=[0:ts:60];
b1=20;b2=30;b=50;
Am=[0,1;-b2,-b1];
eig(Am)
Q=[20,10;10,20];

P=lyap(Am',Q);
p12=P(1,2);
p22=P(2,2);
para=[b1,b2,b,p12,p22];

[t,y]=ode45('chap7_4plant',TimeSet,[0.5 0 0 0 0 0 0],[],para);
k0=y(:,5);
k1=y(:,6);
k2=y(:,7);

switch S
case 1
    r=1.0*sign(sin(0.05*t*2*pi));    %Square Signal
case 2
    r=1.0*sin(1.0*t*2*pi);           %Sin Signal
end
u=k0.*r+k1.*y(:,3)+k2.*y(:,4);

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k','linewidth',2);
xlabel('Time(s)');ylabel('Position tracking');
legend('Ideal position signal','Position signal tracking');
subplot(212);
plot(t,y(:,1)-y(:,3),'r','linewidth',2);
xlabel('Time(s)');ylabel('Position tracking error');

figure(2);
plot(t,u,'r','linewidth',2);
xlabel('Time(s)');ylabel('Control input');

figure(3);
subplot(3,1,1);
plot(t,k0,'r','linewidth',2);
xlabel('Time(s)');ylabel('k0');
subplot(3,1,2);
plot(t,k1,'r','linewidth',2);
```



```

xlabel('Time(s)');ylabel('k1');
subplot(3,1,3);
plot(t,k2,'r','linewidth',2);
xlabel('Time(s)');ylabel('k2');

```

子程序: chap7_4plant.m

```

function dy=DynamicModel(t,y,flag,para)
global S
dy=zeros(7,1);

S=1;
switch S
case 1
    r=1.0*sign(sin(0.05*t*2*pi));    %Square Signal
case 2
    r=1.0*sin(1.0*t*2*pi);           %Sin Signal
end
p12=para(4);
p22=para(5);

e=y(1)-y(3);
de=y(2)-y(4);
eF=p12*e+p22*de;

k0=y(5);
k1=y(6);
k2=y(7);
u=k0*r+k1*y(3)+k2*y(4);

b1=para(1);b2=para(2);b=para(3);
dy(1)=y(2);    %Reference Model
dy(2)=b*r-b1*y(2)-b2*y(1);

a1=20;a2=25;a=133;
dy(3)=y(4);    %Practical Plant
dy(4)=-a2*y(3)-a1*y(4)+a*u;

dy(5)=200*eF*r;    %k0
dy(6)=200*eF*y(3); %k1
dy(7)=200*eF*y(4); %k2

```

Simulink 仿真程序

主程序: chap7_5sim.mdl (见图 7-19)

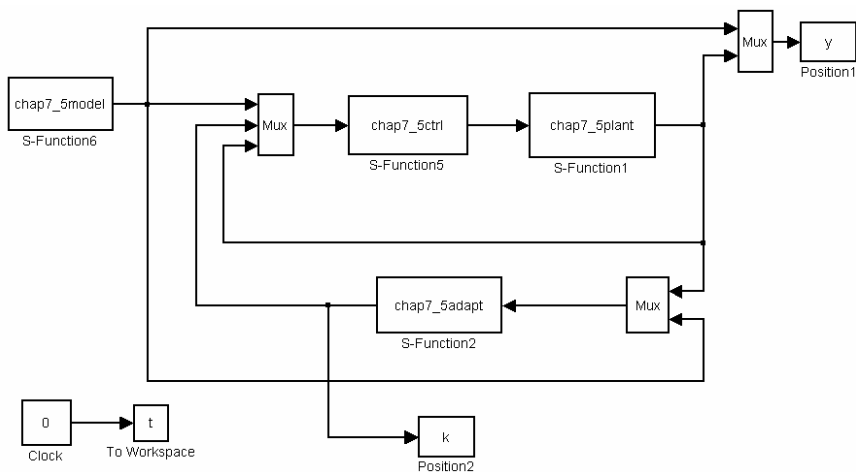
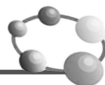


图 7-19 模型参考自适应 PD 控制主程序

参考模型子程序: chap7_5model.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
r=1.0*sign(sin(0.05*t*2*pi));
a0=30;a1=20;b=50;

sys(1)=x(2);
```



```

sys(2)=b*r-a1*x(2)-a0*x(1);
function sys=mdlOutputs(t,x,u)
r=1.0*sign(sin(0.05*t*2*pi));
sys(1)=r;
sys(2)=x(1);
sys(3)=x(2);

```

控制器子程序: chap7_5ctrl.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1, 2, 4, 9}
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 8;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[-1 0];
function sys=mdlOutputs(t,x,u)
r=u(1);
k0=u(4);k1=u(5);k2=u(6);
th=u(7);dth=u(8);

ut=k0*r+k1*th+k2*dth;

sys(1)=ut;

```

自适应律子程序: chap7_5adapt.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);

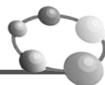
```



```
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0 0 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
th=u(1);
dth=u(2);
r=u(3);
thm=u(4);
dthm=u(5);
b1=20;b2=30;b=50;
Am=[0,1;-b2,-b1];
eig(Am);
%Q=[10 0;0,10];
Q=[20,10;10,20];
P=lyap(Am',Q);
p12=P(1,2);
p22=P(2,2);

e=thm-th;
de=dthm-dth;
eF=p12*e+p22*de;

sys(1)=30*eF*r;      %k0
sys(2)=30*eF*th;     %k1
sys(3)=30*eF*dth;    %k2
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
```



被控对象子程序: chap7_5plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[0.5 0];
str=[];
ts=[-1 0];
function sys=mdlDerivatives(t,x,u)
a1=20;a2=25;a=133;
ut=u(1);
sys(1)=x(2);
sys(2)=-a1*x(2)-a2*x(1)+a*ut;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

作图子程序: chap7_5plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,2),'r',t,y(:,4),'b:','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('Ideal position signal','Position signal tracking');
subplot(212);
plot(t,y(:,3),'r',t,y(:,5),'b:','linewidth',2);
```



```
xlabel('time(s)');ylabel('speed tracking');  
legend('Ideal speed signal','Speed signal tracking');
```

```
figure(2);  
subplot(311);  
plot(t,k(:,1),'r','linewidth',2);  
xlabel('time(s)');ylabel('k0');  
subplot(312);  
plot(t,k(:,2),'r','linewidth',2);  
xlabel('time(s)');ylabel('k1');  
subplot(313);  
plot(t,k(:,3),'r','linewidth',2);  
xlabel('time(s)');ylabel('k2');
```

第8章 模糊 PD 控制和专家 PID 控制

模糊控制是一种基于规则的控制，它直接采用语言型控制规则，出发点是现场操作人员的控制经验或相关专家的知识，在设计中不需要建立被控对象的精确的数学模型，因而使得控制机理和策略易于接受与理解，设计简单，便于应用。基于模糊原理的模糊系统具有万能逼近的特点。

本章内容包括两个部分：一是利用模糊规则调节实现 PID 的整定，二是利用模糊系统的万能逼近理论逼近被控对象的未知项，实现控制补偿，提高 PID 控制的性能。



8.1 倒立摆稳定的 PD 控制

8.1.1 系统描述

考虑如下二阶非线性系统

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u \quad (8.1)$$

式中， $\mathbf{f}$ 和 $\mathbf{g}$ 为已知非线性函数， $u \in R$ 和 $\theta \in R$ 分别为系统的输入。

8.1.2 控制律设计

设位置指令为 θ_d ，实际位置为 θ ，取误差为 $e = \theta_d - \theta$ ，取控制律为

$$u = \frac{1}{g(\mathbf{x})} [-f(\mathbf{x}) + \ddot{\theta}_d + k_p e + k_d \dot{e}] \quad (8.2)$$

式中， $\mathbf{x} = [\theta, \dot{\theta}]^T$ 。

将式 (8.2) 代入式 (8.1)，得到闭环控制系统的方程

$$\ddot{e} + k_d \dot{e} + k_p e = 0 \quad (8.3)$$

通过 k_p 和 k_d 的选取，可得 $t \rightarrow \infty$ 时， $e(t) \rightarrow 0$ ，即系统的位置输出 θ 渐进地收敛于理想输出 θ_d 。

如果非线性函数 $\mathbf{f}(\mathbf{x})$ 是已知的，则可以选择控制 u 来消除其非线性的性质，然后再根据线性控制理论设计控制器。

选择 k_p 和 k_d ，使多项式 $s^2 + k_d s + k_p$ 的所有根部都在复平面左半平面上，即需要使 $s^2 + k_d s + k_p = 0$ 的根为负。

对于任意负根为 $-\lambda (\lambda > 0)$ ，有 $(s + \lambda)^2 = 0$ ，可得 $s^2 + 2\lambda s + \lambda^2 = 0$ ，则可设计 $k_d = 2\lambda$ ，



$k_p = \lambda^2$ 。

8.1.3 仿真实例

被控对象取单级倒立摆，其动态方程如下

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = f(\mathbf{x}) + g(\mathbf{x})u$$

式中， $f(\mathbf{x}) = \frac{g \sin x_1 - m l x_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$ ， $g(\mathbf{x}) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$ ， x_1 和 x_2

分别为摆角和摆速， $g = 9.8 \text{ m/s}^2$ ， m_c 为小车质量， $m_c = 1 \text{ kg}$ ， m 为摆杆质量， $m = 0.1 \text{ kg}$ ， l 为摆长的一半， $l = 0.5 \text{ m}$ ， u 为控制输入。

位置指令为 $x_d(t) = 0.1 \sin(t)$ 。倒立摆初始状态为 $[\pi/60, 0]$ ，采用式，取 $\lambda = 5$ ，则有 $k_p = 25$ ， $k_d = 10$ ，仿真结果如图 8-1 和图 8-2 所示。

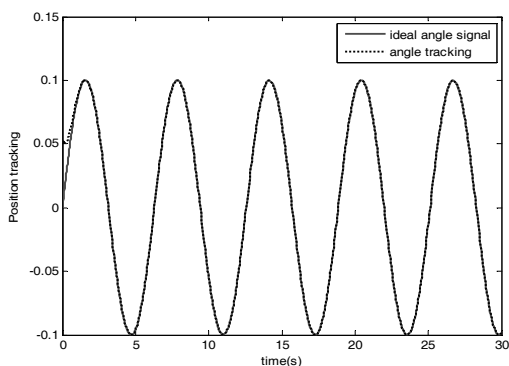


图 8-1 位置跟踪

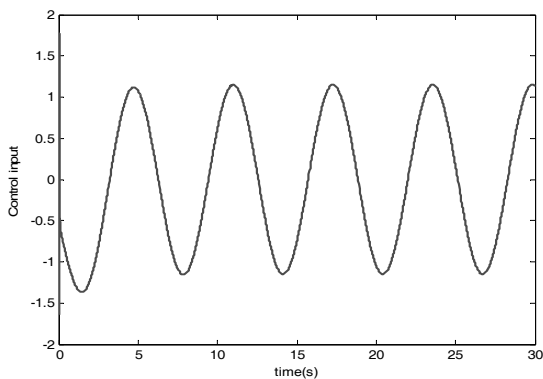


图 8-2 控制输入信号

仿真程序

(1) Simulink 主程序: chap8_1sim.mdl (见图 8-3)

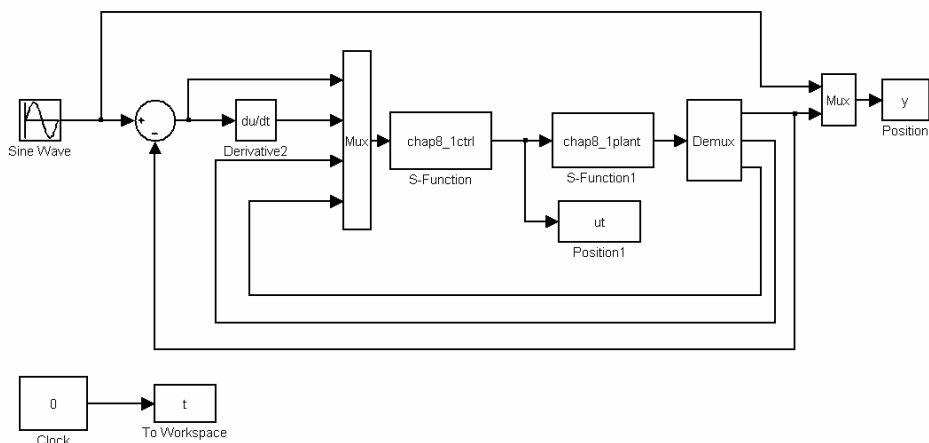
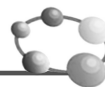


图 8-3 稳定的 PD 控制主程序

(2) 控制器 S 函数: chap8_1ctrl.m;

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c bn
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
xd=0.1*sin(t);
dxd=0.1*cos(t);
ddxd=-0.1*sin(t);
```



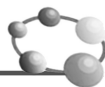
```
e=u(1);
de=u(2);
fx=u(3);
gx=u(4);

kp=25;
kd=10;
ut=1/gx*(-fx+ddxd+kp*e+kd*de);

sys(1)=ut;
```

(3) 被控对象 S 函数: chap8_1plant.m;

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;
```



```

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;
sys(1)=x(1);
sys(2)=fx;
sys(3)=gx;

```

(4) 作图程序: chap8_1plot.m。

```

close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal angle signal','angle tracking');
figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```



8.2 基于自适应模糊补偿的倒立摆 PD 控制

8.2.1 问题描述

考虑如下二阶非线性系统

$$\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u \quad (8.4)$$

式中, f 为未知非线性函数, 代表建模不确定部分或干扰, g 为已知非线性函数, $u \in R^n$ 和 $y \in R^n$ 分别为系统的输入和输出。

式(8.4)还可写为

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + g(x_1, x_2)u \\ y &= x_1 \end{aligned} \quad (8.5)$$

设位置指令为 x_d , 令

$$e = x_d - y = x_d - x_1, \quad E = (e \quad \dot{e})^T$$

选择 $K = [k_p, k_d]^T$, 使多项式 $s^2 + k_d s + k_p = 0$ 的所有根部都在复平面左半平面上。

取控制律为



$$u^* = \frac{1}{g(\mathbf{x})} \left[-f(\mathbf{x}) + \ddot{x}_d + \mathbf{K}^T \mathbf{E} \right] \quad (8.6)$$

将式(8.6)代入式(8.4), 得到闭环控制系统的方程

$$\ddot{e} + k_d \dot{e} + k_p e = 0 \quad (8.7)$$

由 $\mathbf{K}$ 的选取, 可得 $t \rightarrow \infty$ 时 $e(t) \rightarrow 0$, $\dot{e}(t) \rightarrow 0$, 即系统的输出 y 及其导数渐进地收敛于理想输出 x_d 及其导数。

如果非线性函数 $f(\mathbf{x})$ 是已知的, 则可以选择控制 u 来消除其非线性的性质, 然后再根据线性控制理论设计控制器。

8.2.2 自适应模糊控制器设计与分析

如果 $f(\mathbf{x})$ 未知, 控制律式(8.6)很难实现。可采用模糊系统 $\hat{f}(\mathbf{x})$ 代替 $f(\mathbf{x})$, 实现自适应模糊补偿。

8.2.2.1 基本的模糊系统

利用王立新提出的万能逼近定理^[59], 可实现未知函数的模糊逼近。以 $\hat{f}(\mathbf{x}|\theta_f)$ 来逼近 $f(\mathbf{x})$ 为例, 可用以下两步构造模糊系统 $\hat{f}(\mathbf{x}|\theta_f)$ 。

步骤1: 对变量 x_i ($i=1, 2$), 定义 p_i 个模糊集合 $A_i^{l_i}$ ($l_i=1, 2, \dots, p_i$)。

步骤2: 采用以下 $\prod_{i=1}^2 p_i$ 条模糊规则来构造模糊系统 $\hat{f}(\mathbf{x}|\theta_f)$ 。

$$R^{(j)}: \text{IF } x_1 \text{ is } A_1^{l_1} \text{ AND } x_2 \text{ is } A_2^{l_2} \text{ THEN } \hat{f} \text{ is } E^{l_1 l_2} \quad (8.8)$$

式中, $l_i=1, 2$, $i=1, 2$ 。

采用乘积推理机、单值模糊器和中心平均解模糊器, 则模糊系统的输出为

$$\hat{f}(\mathbf{x}|\theta_f) = \frac{\sum_{l_1=1}^{p_1} \sum_{l_2=1}^{p_2} \bar{y}_f^{l_1 l_2} \left(\prod_{i=1}^2 \mu_{A_i^{l_i}}(x_i) \right)}{\sum_{l_1=1}^{p_1} \sum_{l_2=1}^{p_2} \left(\prod_{i=1}^2 \mu_{A_i^{l_i}}(x_i) \right)} \quad (8.9)$$

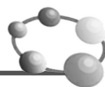
式中, $\mu_{A_i^{l_i}}(x_i)$ 为 x_i 的隶属函数。

令 $\bar{y}_f^{l_1 l_2}$ 是自由参数, 放在集合 $\theta_f \in R^{\prod_{i=1}^2 p_i}$ 中。引入列向量 $\xi(\mathbf{x})$, 式(8.9)变为

$$\hat{f}(\mathbf{x}|\theta_f) = \theta_f^T \xi(\mathbf{x}) \quad (8.10)$$

式中, $\xi(\mathbf{x})$ 为 $\prod_{i=1}^2 p_i$ 维列向量, 其中第 l_1, l_2 个元素为

$$\xi_{l_1 l_2}(\mathbf{x}) = \frac{\prod_{i=1}^2 \mu_{A_i^{l_i}}(x_i)}{\sum_{l_1=1}^{p_1} \sum_{l_2=1}^{p_2} \left(\prod_{i=1}^2 \mu_{A_i^{l_i}}(x_i) \right)} \quad (8.11)$$



8.2.2.2. 自适应控制器的设计^[59]

采用模糊系统逼近 f ，则控制律式 (8.6) 变为

$$u = \frac{1}{g(\mathbf{x})} \left[-\hat{f}(\mathbf{x}|\theta_f) + \ddot{x}_d + \mathbf{K}^T \mathbf{E} \right] \quad (8.12)$$

$$\hat{f}(\mathbf{x}|\theta_f) = \theta_f^T \xi(\mathbf{x}) \quad (8.13)$$

式中， $\xi(\mathbf{x})$ 为模糊向量，参数 θ_f 根据自适应律而变化。

设计自适应律为

$$\dot{\theta}_f = -\gamma \mathbf{E}^T \mathbf{P} \mathbf{b} \xi(\mathbf{x}) \quad (8.14)$$

8.2.3 稳定性分析

参照王立新^[59]所提出的间接自适应模糊控制方法，对本闭环系统进行稳定性分析。

由式 (8.12) 代入式 (8.4)，可得如下模糊控制系统的闭环动态方程

$$\ddot{e} = -\mathbf{K}^T \mathbf{E} + \left[\hat{f}(\mathbf{x}|\theta_f) - f(\mathbf{x}) \right] \quad (8.15)$$

令

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k_p & -k_d \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (8.16)$$

则动态方程 (8.15) 可写为向量形式

$$\dot{\mathbf{E}} = \mathbf{A} \mathbf{E} + \mathbf{b} \left[\hat{f}(\mathbf{x}|\theta_f) - f(\mathbf{x}) \right] \quad (8.17)$$

设最优参数为

$$\theta_f^* = \arg \min \left[\sup \left| \hat{f}(\mathbf{x}|\theta_f) - f(\mathbf{x}) \right| \right] \quad (8.18)$$

式中， Ω_f 为 θ_f 的集合， $\theta_f \in \Omega_f$ 。

定义最小逼近误差为

$$\omega = \hat{f}(\mathbf{x}|\theta_f^*) - f(\mathbf{x}) \quad (8.19)$$

式 (8.17) 可写为

$$\dot{\mathbf{E}} = \mathbf{A} \mathbf{E} + \mathbf{b} \left\{ \left[\hat{f}(\mathbf{x}|\theta_f) - \hat{f}(\mathbf{x}|\theta_f^*) \right] + \omega \right\} \quad (8.20)$$

将式 (8.13) 代入式 (8.20)，可得闭环动态方程

$$\dot{\mathbf{E}} = \mathbf{A} \mathbf{E} + \mathbf{b} \left[(\theta_f - \theta_f^*)^T \xi(\mathbf{x}) + \omega \right] \quad (8.21)$$

该方程清晰地描述了跟踪误差和控制参数 θ_f 之间的关系。自适应律的任务是为 θ_f 确定一个调节机理，使得跟踪误差 $\mathbf{E}$ 和参数误差 $\theta_f - \theta_f^*$ 达到最小。

定义 Lyapunov 函数

$$V = \frac{1}{2} \mathbf{E}^T \mathbf{P} \mathbf{E} + \frac{1}{2\gamma} (\theta_f - \theta_f^*)^T (\theta_f - \theta_f^*) \quad (8.22)$$

式中， γ 是正常数， $\mathbf{P}$ 为一个正定矩阵且满足 Lyapunov 方程

$$\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} = -\mathbf{Q} \quad (8.23)$$



式中, $\mathbf{Q}$ 是一个任意的 2×2 正定矩阵, $\mathbf{A}$ 由式 (8.16) 给出。

取 $V_1 = \frac{1}{2} \mathbf{E}^T \mathbf{P} \mathbf{E}$, $V_2 = \frac{1}{2\gamma} (\theta_f - \theta_f^*)^T (\theta_f - \theta_f^*)$ 。为了描述方便, 令 $M = b \left[(\theta_f - \theta_f^*)^T \xi(x) + \omega \right]$, 则式 (8.21) 变为

$$\dot{\mathbf{E}} = \mathbf{A} \mathbf{E} + M$$

则

$$\begin{aligned} \dot{V}_1 &= \frac{1}{2} \dot{\mathbf{E}}^T \mathbf{P} \mathbf{E} + \frac{1}{2} \mathbf{E}^T \mathbf{P} \dot{\mathbf{E}} = \frac{1}{2} (\mathbf{E}^T \mathbf{A}^T + \mathbf{M}^T) \mathbf{P} \mathbf{E} + \frac{1}{2} \mathbf{E}^T \mathbf{P} (\mathbf{A} \mathbf{E} + M) \\ &= \frac{1}{2} \mathbf{E}^T (\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A}) \mathbf{E} + \frac{1}{2} \mathbf{M}^T \mathbf{P} \mathbf{E} + \frac{1}{2} \mathbf{E}^T \mathbf{P} M \\ &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \frac{1}{2} (\mathbf{M}^T \mathbf{P} \mathbf{E} + \mathbf{E}^T \mathbf{P} M) = -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} M \end{aligned}$$

将 M 代入上式, 并考虑 $\mathbf{E}^T \mathbf{P} b (\theta_f - \theta_f^*)^T \xi(x) = (\theta_f - \theta_f^*)^T [\mathbf{E}^T \mathbf{P} b \xi(x)]$, 得

$$\begin{aligned} \dot{V}_1 &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} b (\theta_f - \theta_f^*)^T \xi(x) + \mathbf{E}^T \mathbf{P} b \omega \\ &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + (\theta_f - \theta_f^*)^T \mathbf{E}^T \mathbf{P} b \xi(x) + \mathbf{E}^T \mathbf{P} b \omega \\ \dot{V}_2 &= \frac{1}{\gamma} (\theta_f - \theta_f^*)^T \dot{\theta}_f \end{aligned}$$

V 的导数为

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} b \omega + \frac{1}{\gamma} (\theta_f - \theta_f^*)^T [\dot{\theta}_f + \gamma \mathbf{E}^T \mathbf{P} b \xi(x)]$$

将自适应律式 (8.14) 代入上式, 得

$$\dot{V} = -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} b \omega$$

由于 $-\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} \leq 0$, 通过选取最小逼近误差 ω 非常小的模糊系统, 可实现 $\dot{V} \leq 0$ 。

8.2.4 仿真实例

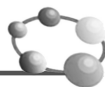
被控对象取单级倒立摆, 其动态方程如下

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + g(x)u \end{aligned}$$

式中, $f(x) = \frac{g \sin x_1 - m l x_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, $g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, x_1 和

x_2 分别为摆角和摆速, $g = 9.8 \text{ m/s}^2$, m_c 为小车质量, $m_c = 1 \text{ kg}$, m 为摆杆质量, $m = 0.1 \text{ kg}$, l 为摆长的一半, $l = 0.5 \text{ m}$, u 为控制输入。

位置指令为 $x_d(t) = 0.1 \sin(\pi t)$ 。针对变量 x_1 和 x_2 , 分别定义 5 个模糊集合, 即取



$p_1 = p_2 = 5$ ，则 $\prod_{i=1}^2 p_i = p_1 p_2 = 25$ 。

取以下 5 种隶属函数： $\mu_{\text{NM}}(x_i) = \exp\left[-\left((x_i + \pi/6)/(\pi/24)\right)^2\right]$ ， $\mu_{\text{NS}}(x_i) = \exp\left[-\left((x_i + \pi/12)/(\pi/24)\right)^2\right]$ ， $\mu_{\text{Z}}(x_i) = \exp\left[-\left(x_i/(\pi/24)\right)^2\right]$ ， $\mu_{\text{PS}}(x_i) = \exp\left[-\left((x_i - \pi/12)/(\pi/24)\right)^2\right]$ ， $\mu_{\text{PM}}(x_i) = \exp\left[-\left((x_i - \pi/6)/(\pi/24)\right)^2\right]$ 。则用于逼近 f 的模糊规则有 25 条。

根据隶属函数设计程序，可得到隶属函数图，如图 8-4 所示。

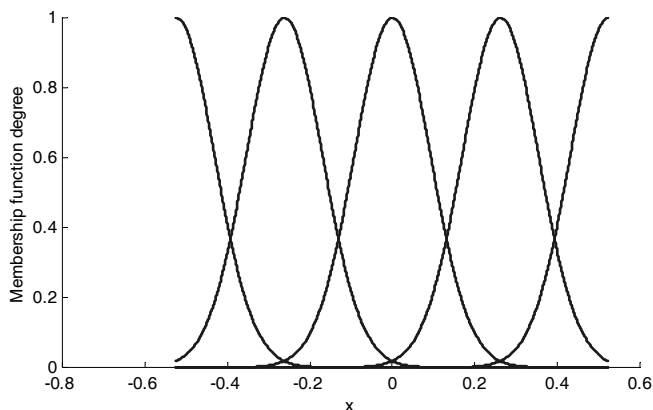


图 8-4 x_i 的隶属函数

倒立摆初始状态为 $[\pi/60, 0]$ ， θ_f 的初始值取 0.10，采用控制律式 (8.12)，自适应律取式 (8.14)，取 $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$ ， $k_d = 20$ ， $k_p = 10$ ，自适应参数取 $\gamma = 100$ 。

在程序中，分别用 FS_2 、 FS_1 和 FS 表示模糊系统 $\xi(x)$ 的分子、分母及 $\xi(x)$ ，仿真结果如图 8-5 至图 8-7 所示。

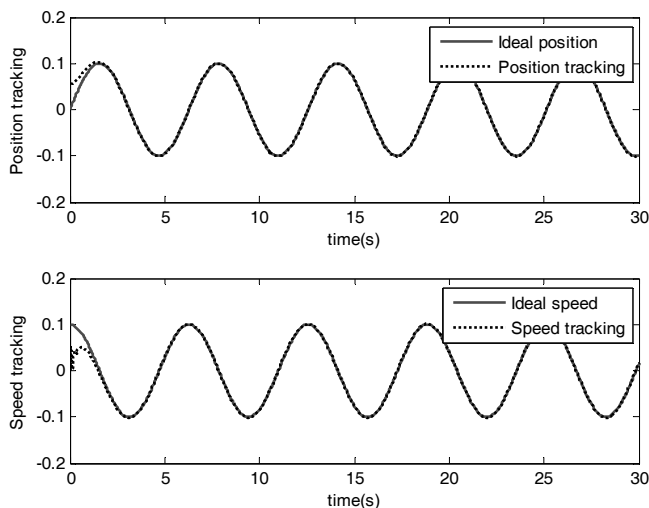


图 8-5 位置和速度跟踪

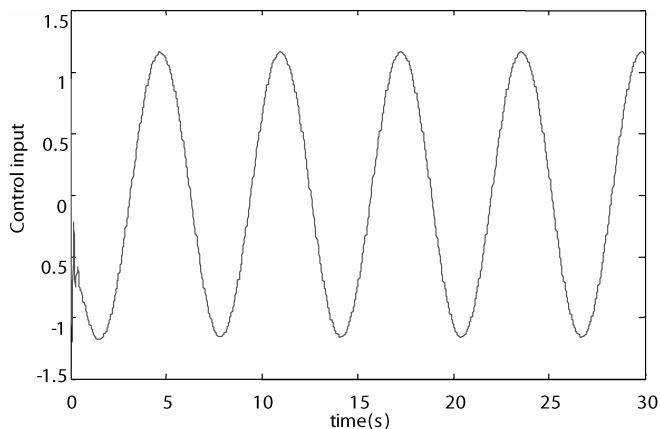


图 8-6 控制输入信号

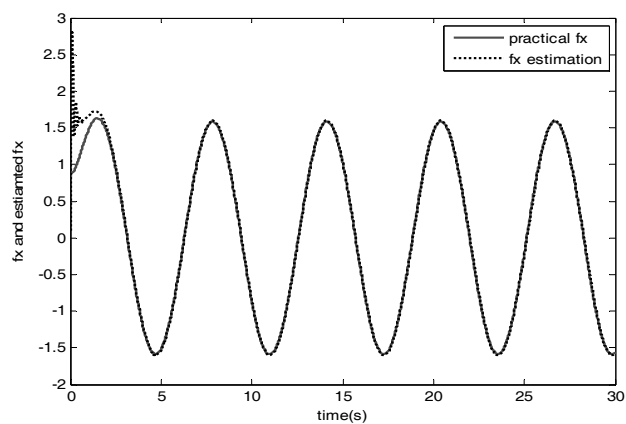


图 8-7 $f(x)$ 及 $\hat{f}(x)$ 的变化

仿真程序

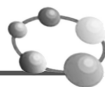
(1) 隶属函数设计程序: chap8_2mf.m;

```
clear all;
close all;

L1=-pi/6;
L2=pi/6;
L=L2-L1;

T=L*1/1000;

x=L1:T:L2;
figure(1);
for i=1:1:5
    gs=-[(x+pi/6-(i-1)*pi/12)/(pi/24)].^2;
    u=exp(gs);
    hold on;
    plot(x,u);
```



```
end
xlabel('x');ylabel('Membership function degree');
```

(2) Simulink 主程序: chap8_2sim.mdl (见图 8-8)

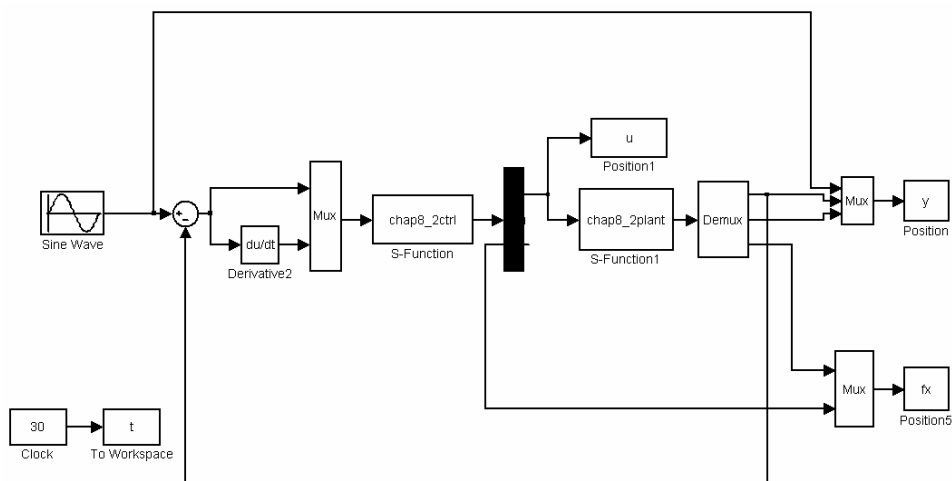


图 8-8 自适应模糊 PD 控制主程序

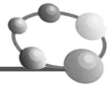
(3) 控制器 S 函数: chap8_2ctrl.m;

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 25;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0.1*ones(25,1)];
str = [];
```



```
ts = [];  
function sys=mdlDerivatives(t,x,u)  
gama=100;  
xd=0.1*sin(t);  
dxd=0.1*cos(t);  
ddxd=-0.1*sin(t);  
  
e=u(1);  
de=u(2);  
x1=xd-e;  
x2=de-dxd;  
  
kp=10;  
kd=20;  
k=[kp;kd];  
E=[e,de]';  
  
for i=1:1:25  
    thtaf(i,1)=x(i);  
end  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
A=[0 -kp;  
    1 -kd];  
Q=[500 0;0 500];  
P=lyap(A,Q);  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
FS1=0;  
for l1=1:1:5  
    gs1=-(x1+pi/6-(l1-1)*pi/12)/(pi/24)^2;  
    u1(l1)=exp(gs1);  
end  
  
for l2=1:1:5  
    gs2=-(x2+pi/6-(l2-1)*pi/12)/(pi/24)^2;  
    u2(l2)=exp(gs2);  
end  
  
for l1=1:1:5  
    for l2=1:1:5  
        FS2(5*(l1-1)+l2)=u1(l1)*u2(l2);  
        FS1=FS1+u1(l1)*u2(l2);  
    end  
end  
  
FS=FS2/(FS1+0.001);
```



```

b=[0;1];
S=-gama*E'*P*b*FS;

for i=1:1:25
    sys(i)=S(i);
end

function sys=mdlOutputs(t,x,u)
xd=0.1*sin(t);
dxd=0.1*cos(t);
ddxd=-0.1*sin(t);

e=u(1);
de=u(2);
x1=xd-e;
x2=de-dxd;

kp=10;
kd=20;
k=[kp;kd];
E=[e,de]';

for i=1:1:25
    thtaf(i,1)=x(i);
end

FS1=0;
for l1=1:1:5
    gs1=-[(x1+pi/6-(l1-1)*pi/12)/(pi/24)]^2;
    u1(l1)=exp(gs1);
end

for l2=1:1:5
    gs2=-[(x2+pi/6-(l2-1)*pi/12)/(pi/24)]^2;
    u2(l2)=exp(gs2);
end

for l1=1:1:5
    for l2=1:1:5
        FS2(5*(l1-1)+l2)=u1(l1)*u2(l2);
        FS1=FS1+u1(l1)*u2(l2);
    end
end
FS=FS2/(FS1+0.001);

fxp=thtaf'*FS';

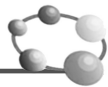
```



```
%%%%%%%%%%  
g=9.8;mc=1.0;m=0.1;l=0.5;  
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));  
gx=cos(x(1))/(mc+m);  
gx=gx/S;  
%%%%%%%%%%  
ut=1/gx*(-fxp+ddxd+k'*E);  
  
sys(1)=ut;  
sys(2)=fxp;
```

(4) 被控对象 S 函数: chap8_2plant.m;

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)  
switch flag,  
case 0,  
    [sys,x0,str,ts]=mdlInitializeSizes;  
case 1,  
    sys=mdlDerivatives(t,x,u);  
case 3,  
    sys=mdlOutputs(t,x,u);  
case {2, 4, 9 }  
    sys = [];  
otherwise  
    error(['Unhandled flag = ',num2str(flag)]);  
end  
function [sys,x0,str,ts]=mdlInitializeSizes  
sizes = simsizes;  
sizes.NumContStates = 2;  
sizes.NumDiscStates = 0;  
sizes.NumOutputs = 3;  
sizes.NumInputs = 1;  
sizes.DirFeedthrough = 0;  
sizes.NumSampleTimes = 0;  
sys=simsizes(sizes);  
x0=[pi/60 0];  
str=[];  
ts=[];  
function sys=mdlDerivatives(t,x,u)  
g=9.8;mc=1.0;m=0.1;l=0.5;  
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));  
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);  
fx=fx/S;  
gx=cos(x(1))/(mc+m);  
gx=gx/S;  
  
sys(1)=x(2);
```



```

sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
g=9.8;
mc=1.0;
m=0.1;
l=0.5;

S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;

```

(5) 作图程序: chap8_2plot.m。

```

close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position','Position tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,3),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed','Speed tracking');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('fx and estimated fx');
legend('practical fx','fx estimation');

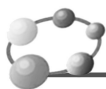
```



8.3 基于模糊规则表的模糊 PD 控制

8.3.1 基本原理

将跟踪误差和误差变化量作为模糊规则的输入，控制输入作为规则的输出，根据经验构



造模糊规则表, 可实现无需模型信息的控制。

以误差和误差变化为前提, 第 ij 条模糊控制规则的表达式为

$$\text{Rule } ij: \text{ IF } e = \mu_i \text{ and } \Delta e = \mu_j \text{ THEN } u = u_{ij} \quad (8.24)$$

采用乘积推理机, 规则前部分的隶属函数为

$$f_{ij} = \mu_i(e) \cdot \mu_j(\Delta e) \quad (8.25)$$

式中, $\mu_i(e)$ 和 $\mu_j(\Delta e)$ 分别为 e 和 Δe 的隶属度。

采用中心平均解模糊器进行反模糊化, ^[59]得到模糊控制器

$$u = \frac{\sum_{i,j} f_{ij} u_{ij}}{\sum_{i,j} f_{ij}} \quad (8.26)$$

式中, u_{ij} 的值由模糊规则表确定。

模糊规则表中每条规则的输出 u_{ij} 值可由模糊推理或根据经验确定。假设 e 和 Δe 各有三个隶属函数, 则共有 9 条规则, 模糊规则表的形式如表 8-1 所示。

表 8-1 控制规则表

u_{ij}		Δe		
		N	Z	P
e	N	u_{11}	u_{12}	u_{13}
	Z	u_{21}	u_{22}	u_{23}
	P	u_{31}	u_{32}	u_{33}

8.3.2 仿真实例

被控对象为

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

位置指令信号为 $\cos t$, 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

$$y(k) = -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) + \text{num}(2)u(k-1) + \text{num}(3)u(k-2)$$

针对误差 e 及误差变化率 Δe , 分别采用三个隶属函数进行模糊化, 即

$$\mu_1(x) = \exp\left[-\left(\frac{x + \pi/6}{\pi/12}\right)^2\right], \quad \mu_2(x) = \exp\left[-\left(\frac{x}{\pi/12}\right)^2\right], \quad \mu_3(x) = \exp\left[-\left(\frac{x - \pi/6}{\pi/12}\right)^2\right],$$

隶属函数如图 8-9 所示。

采用经验确定模糊规则表, 根据实际被控对象和指令信号, 将控制规则表设计为表 8-2 的形式。采用控制律式(8.26), 正弦位置跟踪结果如图 8-10 所示。可见, 控制器性能的好坏决定于控制规则表, 而采用经验很难确定控制性能高的规则表。可采用定性分析及遗传算法对规则表中规则的数目和规则表中的数值进行优化^[43]。

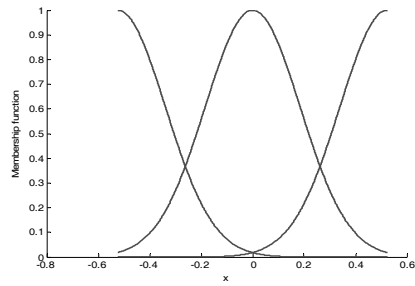
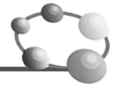


图 8-9 e 和 Δe 的隶属函数度曲线

表 8-2 控制规则表

u_{ij}		Δe		
		N	Z	P
e	N	-200	-100	0
	Z	-100	0	100
	P	0	100	200

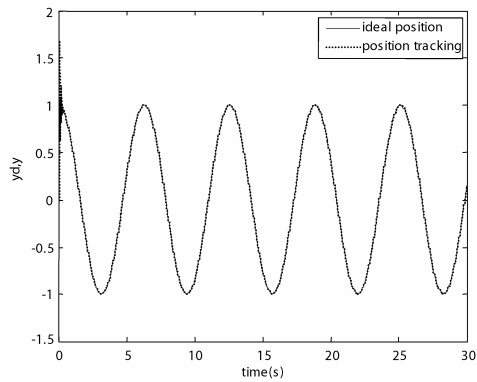


图 8-10 正弦位置跟踪

仿真程序

隶属函数设计程序: chap8_3mf.m

```
clear all;
close all;

L1=-pi/6;
L2=pi/6;
L=L2-L1;

T=L*1/1000;

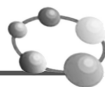
x=L1:T:L2;
figure(1);
for i=1:1:3
    gs=-[(x+pi/6-(i-1)*pi/6)/(pi/12)].^2;
    u=exp(gs);
    hold on;
```




```
plot(x,u,'r','linewidth',2);  
end  
xlabel('x');ylabel('Membership function');
```

主程序: chap8_3.m

```
%PD Type Fuzzy Controller Design  
clear all;  
close all;  
  
ts=0.001;  
  
sys=tf(133,[1,25,0]);  
dsys=c2d(sys,ts,'z');  
[num,den]=tfdata(dsys,'v');  
  
e_1=0;  
u_1=0;u_2=0;  
y_1=0;y_2=0;  
  
for k=1:1:30000  
time(k)=k*ts;  
  
yd(k)=cos(k*ts);  
  
y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;  
  
e(k)=yd(k)-y(k);  
de(k)=e(k)-e_1;  
  
for l1=1:1:3  
gs1=-[(e(k)+pi/6-(l1-1)*pi/6)/(pi/12)]^2;  
u1(l1)=exp(gs1);  
end  
  
for l2=1:1:3  
gs2=-[(de(k)+pi/6-(l2-1)*pi/6)/(pi/12)]^2;  
u2(l2)=exp(gs2);  
end  
  
U=[-200 -100 0  
-100 0 100  
0 100 200];  
  
fnum=0;  
fden=0;  
for i=1:1:3  
for j=1:1:3  
fnum=fnum+u1(i)*u2(j)*U(i,j);  
fden=fden+u1(i)*u2(j);
```



```

end
end

u(k)=fnum/(fden+0.01);

e_1=e(k);
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=y(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('ideal position','position tracking');

```



8.4 模糊自适应整定 PID 控制

8.4.1 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。

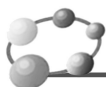
随着计算机技术的发展,人们利用人工智能的方法将操作人员的调整经验作为知识存入计算机中,根据现场实际情况,计算机能自动调整 PID 参数,这样就出现了专家 PID 控制器。这种控制器把古典的 PID 控制与先进的专家系统相结合,实现系统的最佳控制。这种控制必须精确地确定对象模型,首先将操作人员(专家)长期实践积累的经验知识用控制规则模型化,然后运用推理便可对 PID 参数实现最佳调整。

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,专家 PID 方法受到局限。模糊理论是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以及有关信息(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(利用模糊控制规则在线对 PID 参数进行修改),以满足不同时刻的 e 和 ec 对 PID 参数自整定的要求。自适应模糊 PID 控制器结构如图 8-11 所示。

离散 PID 控制算法为

$$u(k) = k_p e(k) + k_i T \sum_{j=0}^k e(j) + k_d \frac{e(k) - e(k-1)}{T} \quad (8.27)$$



式中, k 为采样序号, T 为采样时间。

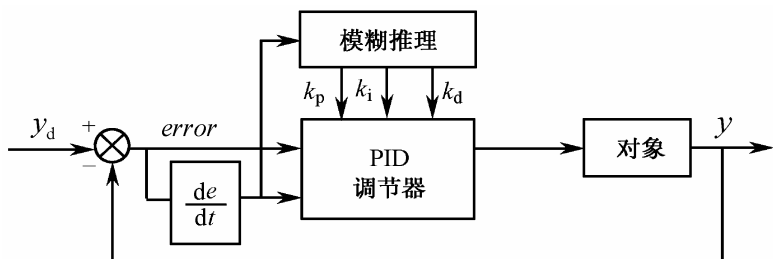


图 8-11 自适应模糊控制器结构

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。

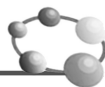
(3) 微分作用系数 k_d 的作用是改善系统的动态特性, 其作用主要是在响应过程中抑制偏差向任何方向的变化, 对偏差变化进行提前预报。但 k_d 过大, 会使响应过程提前制动, 从而延长调节时间, 而且会降低系统的抗干扰性能。

以 PI 参数整定为例, 必须考虑到在不同时刻二个参数的作用以及相互之间的互联关系。模糊自整定 PI 是在 PI 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。针对 k_p , k_i 二个参数分别整定的模糊控制表如下。

(1) k_p 整定原则: 当响应在上升过程时 (e 为 P), Δk_p 取正, 即增大 k_p ; 当超调时 (e 为 N), Δk_p 取负, 即降低 k_p 。当误差在零附近时 (e 为 Z), 分三种情况: ec 为 N 时, 超调越来越大, 此时 Δk_p 取负; ec 为 Z 时, 为了降低误差, Δk_p 取正; ec 为 P 时, 正向误差越来越大, Δk_p 取正。 k_p 整定的模糊规则表见表 8-3。

表 8-3 k_p 的模糊规则表

Δk_p $e \backslash ec$			
	N	Z	P
N	N	N	N
Z	N	P	P
P	P	P	P



(2) k_i 整定原则：采用积分分离策略，即误差在零附近时， Δk_i 取正，否则 Δk_i 取零。 k_i 整定的模糊规则见表 8-4。

表 8-4 k_i 的模糊规则表

Δk_i ec	N	Z	P
e			
N	Z	Z	Z
Z	P	P	P
P	Z	Z	Z

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。

$$e, ec = \{-1, 0, 1\} \quad (8.28)$$

其模糊子集为 $e, ec = \{N, Z, P\}$ ，子集中元素分别代表负，零，正。设 e, ec 和 k_p 、 k_i 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 PI 参数的模糊矩阵表，查出修正参数代入下式计算

$$k_p = k_{p0} + \Delta k_p, \quad k_i = k_{i0} + \Delta k_i \quad (8.29)$$

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 8-12 所示。

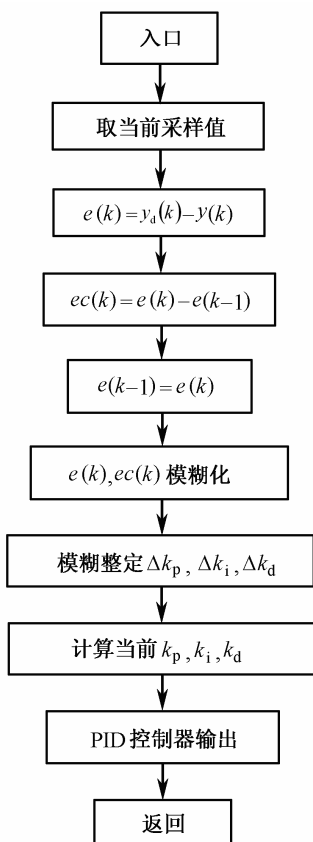
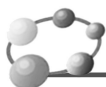


图 8-12 模糊 PID 工作流程图



8.4.2 仿真实例

被控对象为

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

采样时间为 1ms, 采用 z 变换进行离散化, 离散化后的被控对象为

$$y(k) = -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) + \text{num}(2)u(k-1) + \text{num}(3)u(k-2)$$

位置指令为幅值为 1.0 的阶跃信号, $y_d(k) = 1.0$ 。仿真时, 先运行模糊推理系统设计程序 chap8_4a.m, 实现模糊推理系统 fuzzpid.fis, 并将此模糊推理系统调入内存中, 然后运行模糊控制程序 chap8_4b.m。在程序 chap8_4a.m 中, 根据模糊规则见表 8-3 至表 8-4, 分别对 e 、 ec 、 k_p 、 k_i 进行隶属函数的设计。根据位置指令、初始误差和经验设计 e 、 ec 、 k_p 、 k_i 的范围。

在 MATLAB 环境下, 对模糊系统 a, 运行 plotmf 命令, 可得到模糊系统 e 、 ec 、 k_p 、 k_i 的隶属函数, 如图 8-13 至图 8-16 所示, 运行命令 showrule 可显示模糊规则, 可显示 9 条模糊规则, 描述如下:

1. If (e is N) and (ec is N) then (kp is N)(ki is Z) (1)
2. If (e is N) and (ec is Z) then (kp is N)(ki is Z) (1)
3. If (e is N) and (ec is P) then (kp is N)(ki is Z) (1)
4. If (e is Z) and (ec is N) then (kp is N)(ki is P) (1)
5. If (e is Z) and (ec is Z) then (kp is P)(ki is P) (1)
6. If (e is Z) and (ec is P) then (kp is P)(ki is P) (1)
7. If (e is P) and (ec is N) then (kp is P)(ki is Z) (1)
8. If (e is P) and (ec is Z) then (kp is P)(ki is Z) (1)
9. If (e is P) and (ec is P) then (kp is P)(ki is Z) (1)

另外, 针对模糊推理系统 fuzzpid.fis, 运行命令 fuzzy 可进行规则库和隶属函数的编辑, 如图 8-17 所示, 运行命令 ruleview 可实现模糊系统的动态仿真, 如图 8-18 所示。

在程序 chap8_4b.m 中, 利用所设计的模糊系统 fuzzpid.fis 进行 PI 控制参数的整定。为了显示模糊规则调整效果, 取 k_p 、 k_i 的初始值为零, 响应结果及 PI 控制参数的自适应变化如图 8-19 至图 8-20 所示。

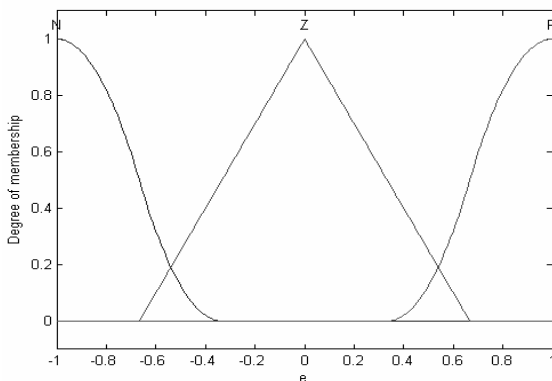


图 8-13 误差的隶属函数

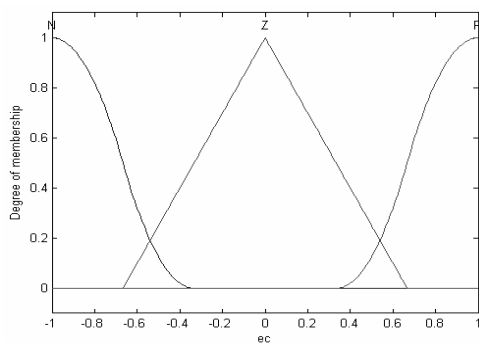
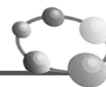
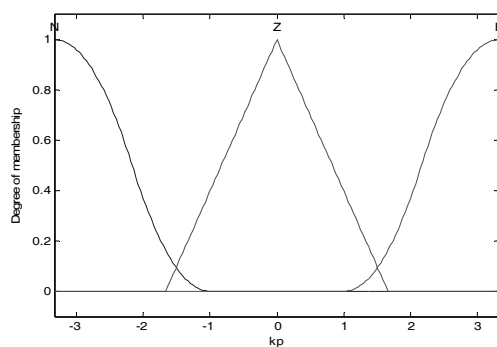
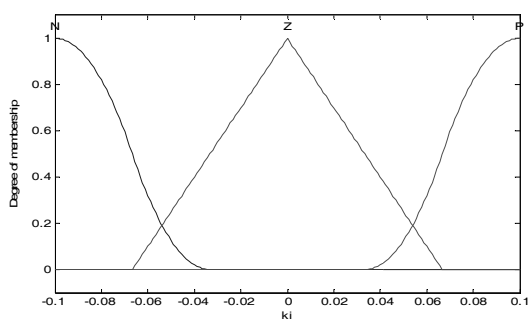
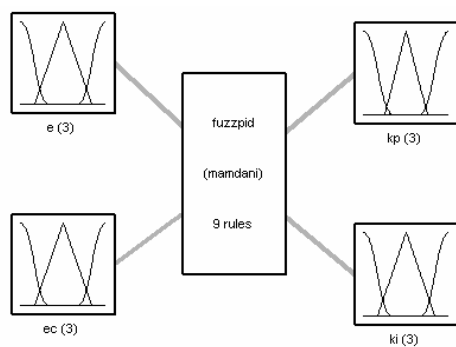


图 8-14 误差变化率的隶属函数

图 8-15 k_p 的隶属函数图 8-16 k_i 的隶属函数

System fuzzpid: 2 inputs, 2 outputs, 9 rules

图 8-17 模糊系统 fuzzpid.fis 的结构

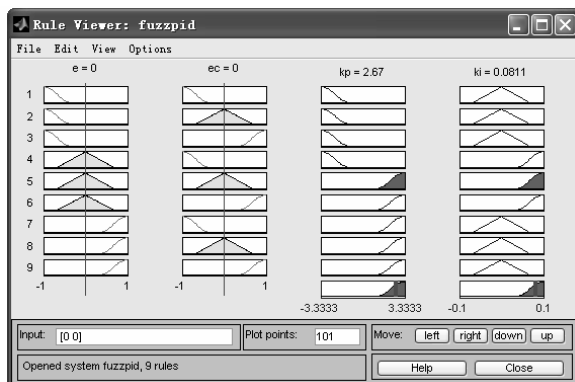
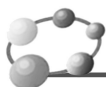


图 8-18 模糊推理系统的动态仿真环境

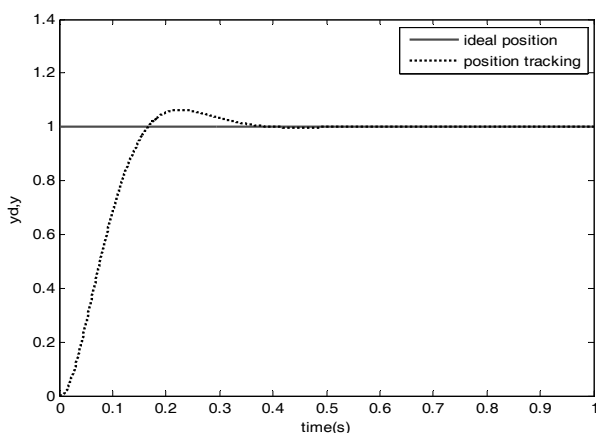


图 8-19 模糊 PI 控制阶跃响应

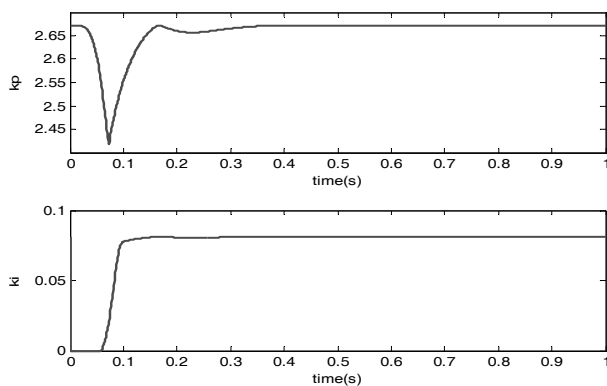
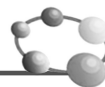


图 8-20 k_p 和 k_d 的模糊自适应调整

仿真程序

(1) 模糊系统程序: chap8_4a.m

```
%Fuzzy Tunning PI Control
clear all;
close all;
a=newfis('fuzzpid');
```



```

a=addvar(a,'input','e',[-1,1]);                                %Parameter e
a=addmf(a,'input',1,'N','zmf',[-1,-1/3]);
a=addmf(a,'input',1,'Z','trimf',[-2/3,0,2/3]);
a=addmf(a,'input',1,'P','smf',[1/3,1]);

a=addvar(a,'input','ec',[-1,1]);                                %Parameter ec
a=addmf(a,'input',2,'N','zmf',[-1,-1/3]);
a=addmf(a,'input',2,'Z','trimf',[-2/3,0,2/3]);
a=addmf(a,'input',2,'P','smf',[1/3,1]);

a=addvar(a,'output','kp',1/3*[-10,10]);                        %Parameter kp
a=addmf(a,'output',1,'N','zmf',1/3*[-10,-3]);
a=addmf(a,'output',1,'Z','trimf',1/3*[-5,0,5]);
a=addmf(a,'output',1,'P','smf',1/3*[3,10]);

a=addvar(a,'output','ki',1/30*[-3,3]);                          %Parameter ki
a=addmf(a,'output',2,'N','zmf',1/30*[-3,-1]);
a=addmf(a,'output',2,'Z','trimf',1/30*[-2,0,2]);
a=addmf(a,'output',2,'P','smf',1/30*[1,3]);

rulelist=[1 1 1 2 1 1;
          1 2 1 2 1 1;
          1 3 1 2 1 1;

          2 1 1 3 1 1;
          2 2 3 3 1 1;
          2 3 3 3 1 1;

          3 1 3 2 1 1;
          3 2 3 2 1 1;
          3 3 3 2 1 1];
a=addrule(a,rulelist);
a=setfis(a,'DefuzzMethod','centroid');
writefis(a,'fuzzpid');

a=readfis('fuzzpid');
figure(1);
plotmf(a,'input',1);
figure(2);
plotmf(a,'input',2);
figure(3);
plotmf(a,'output',1);
figure(4);
plotmf(a,'output',2);
figure(5);

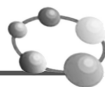
```




```
plotfis(a);  
  
fuzzy fuzzpid;  
showrule(a)  
ruleview fuzzpid;
```

(2) 模糊控制程序: chap8_4b.m

```
%Fuzzy PI Control  
close all;  
clear all;  
  
warning off;  
a=readfis('fuzzpid');    %Load fuzzpid.fis  
  
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;  
e_1=0;ec_1=0;ei=0;  
  
kp0=0;ki0=0;  
for k=1:1:1000  
time(k)=k*ts;  
  
yd(k)=1;  
%Using fuzzy inference to tuning PI  
k_pid=evalfis([e_1,ec_1],a);  
kp(k)=kp0+k_pid(1);  
ki(k)=ki0+k_pid(2);  
u(k)=kp(k)*e_1+ki(k)*ei;  
  
y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;  
e(k)=yd(k)-y(k);  
%%%%%%%%%%%%Return of parameters%%%%%%%%%%%%  
u_2=u_1;u_1=u(k);  
y_2=y_1;y_1=y(k);  
  
ei=ei+e(k)*ts;    % Calculating I  
  
ec(k)=e(k)-e_1;  
e_1=e(k);  
ec_1=ec(k);  
end
```



```
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
figure(2);
subplot(211);
plot(time,kp,'r','linewidth',2);
xlabel('time(s)');ylabel('kp');
subplot(212);
plot(time,ki,'r','linewidth',2);
xlabel('time(s)');ylabel('ki');
figure(3);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```



8.5 专家 PID 控制

8.5.1 专家 PID 控制原理

PID 专家控制的实质是，基于受控对象和控制规律的各种知识，无需知道被控对象的精确模型，利用专家经验来设计 PID 参数。专家 PID 控制是一种直接型专家控制器。

典型的线性系统单位阶跃响应误差曲线如图 8-21 所示。

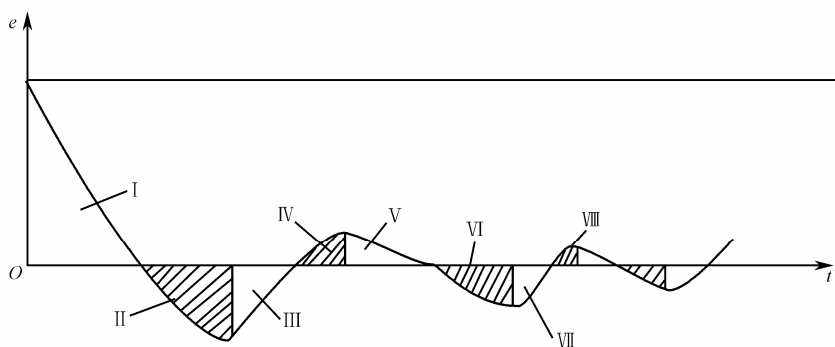


图 8-21 典型线性系统单位阶跃响应误差曲线

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

$$\begin{aligned}\Delta e(k) &= e(k) - e(k-1) \\ \Delta e(k-1) &= e(k-1) - e(k-2)\end{aligned}\quad (8.30)$$

王耀南在著作^[62]中对由误差曲线获取专家控制规则进行了分析。参考文献^[62]，根据误差及其变化，对如图 8-21 所示的单位阶跃响应误差曲线进行如下定性分析。

(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。

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

$$u(k) = u(k-1) + k_1 \{k_p [e(k) - e(k-1)] + k_i e(k) + k_d [e(k) - 2e(k-1) + e(k-2)]\} \quad (8.31)$$

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，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

$$u(k) = u(k-1) + k_p [e(k) - e(k-1)] + k_i e(k) + k_d [e(k) - 2e(k-1) + e(k-2)] \quad (8.32)$$

(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

$$u(k) = u(k-1) + k_1 k_p e_m(k) \quad (8.33)$$

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

$$u(k) = u(k-1) + k_2 k_p e_m(k) \quad (8.34)$$

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

以上各式中， $e_m(k)$ 为误差 e 的第 k 个极值； $u(k)$ 为第 k 次控制器的输出； $u(k-1)$ 为第 $k-1$ 次控制器的输出； k_1 为增益放大系数， $k_1 > 1$ ； k_2 为抑制系数， $0 < k_2 < 1$ ； M_1 、 M_2 为设定的误差界限， $M_1 > M_2 > 0$ ； k 为控制周期的序号（自然数）； ε 为任意小的正实数。

在图 8-21 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。

8.5.2 仿真实例

求三阶传递函数的阶跃响应

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

对象采样时间取 0.001，采用 z 变换进行离散化，经过 z 变换后的离散化对象为

$$\begin{aligned} y(k) = & -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) - \text{den}(4)y(k-3) \\ & + \text{num}(2)u(k-1) + \text{num}(3)u(k-2) + \text{num}(4)u(k-3) \end{aligned}$$

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。仿真结果如图 8-22 所示。

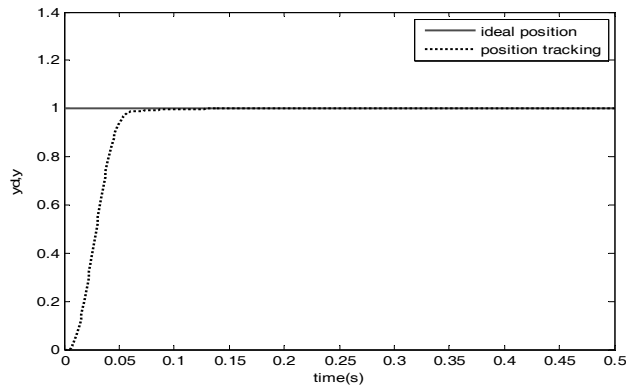
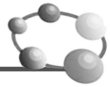


图 8-22 PID 控制阶跃响应曲线

仿真程序: chap8_5.m

```
%Expert PID Controller
clear all;
close all;
ts=0.001;

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

u_1=0;u_2=0;u_3=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_1=0;
for k=1:1:500
time(k)=k*ts;

yd(k)=1.0; %Tracing Step 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 rule
    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
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

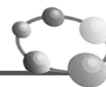
if abs(x(1))<=0.001    %Rule5: Integration separation PI control
    u(k)=0.5*x(1)+0.010*x(3);
end

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

%Linear model
y(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)=yd(k)-y(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=y(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,yd,'r',time,y,'k:','linewidth',2);  
xlabel('time(s)');ylabel('yd,y');  
legend('ideal position','position tracking');
```

第9章 神经PID控制

神经网络（Neural Network）是模拟人脑思维方式的数学模型。神经网络控制是20世纪80年代末期发展起来的自动控制领域的前沿学科之一。它是智能控制的一个重要分支，为解决复杂的非线性、不确定、不确定系统的控制问题开辟了新途径。本章内容包括两个部分：一是利用神经网络调节实现PID的整定，二是利用神经网络逼近被控对象的未知项，实现控制补偿，提高PID控制的性能。



9.1 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。^[66]

9.1.1 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，则它们之间的连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

$$\Delta w_{ij}(k) = \eta o_j(k) o_i(k) \quad (9.1)$$

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

$$\Delta w_{ij}(k) = \eta (d_j(k) - o_j(k)) o_i(k) \quad (9.2)$$

（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

$$\Delta w_{ij}(k) = \eta (d_j(k) - o_j(k)) o_j(k) o_i(k) \quad (9.3)$$

9.1.2 单神经元自适应PID控制

王宁等^[60,61]针对单神经网络控制问题进行了深入研究单神经元自适应PID控制结构如图9-1所示。^[60,61]

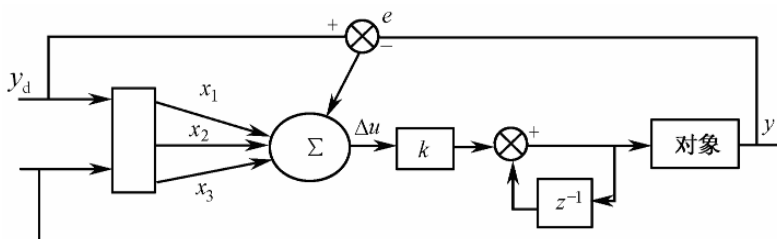
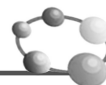


图 9-1 单神经元自适应 PID 控制结构

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^[64]

$$u(k) = u(k-1) + K \sum_{i=1}^3 w'_i(k) x_i(k) \quad (9.4)$$

$$w'_i(k) = w_i(k) / \sum_{i=1}^3 |w_i(k)| \quad (9.5)$$

$$\begin{aligned} w_1(k) &= w_1(k-1) + \eta_I z(k) u(k) x_1(k) \\ w_2(k) &= w_2(k-1) + \eta_P z(k) u(k) x_2(k) \\ w_3(k) &= w_3(k-1) + \eta_D z(k) u(k) x_3(k) \end{aligned} \quad (9.6)$$

式中， $x_1(k) = e(k)$ ， $x_2(k) = e(k) - e(k-1)$ ， $x_3(k) = \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2)$ ， $z(k) = e(k)$ ， η_I, η_P, η_D 分别为积分、比例、微分的 learning rate， 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。

9.1.3 改进的单神经元自适应 PID 控制

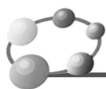
单神经元自适应控制有许多改进方法^[65]。例如可将单神经元自适应 PID 控制算法中的加权系数学习修正部分进行修改，即将其中的 $x_i(k)$ 改为 $e(k) + \Delta e(k)$ ，改进后的算法由文献^[67]给出，表达如下：

$$u(k) = u(k-1) + K \sum_{i=1}^3 w_i(k) x_i(k) \quad (9.7)$$

$$\begin{aligned} w_i(k) &= w_j(k) / \sum_{j=1}^3 |w_j(k)| \\ w_1(k) &= w_1(k-1) + \eta_I z(k) u(k) (e(k) + \Delta e(k)) \\ w_2(k) &= w_2(k-1) + \eta_P z(k) u(k) (e(k) + \Delta e(k)) \\ w_3(k) &= w_3(k-1) + \eta_D z(k) u(k) (e(k) + \Delta e(k)) \end{aligned}$$

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。



9.1.4 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： $y_d(k) = 0.5\text{sgn}(\sin(4\pi t))$ ，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 9-2 至图 9-5 所示。

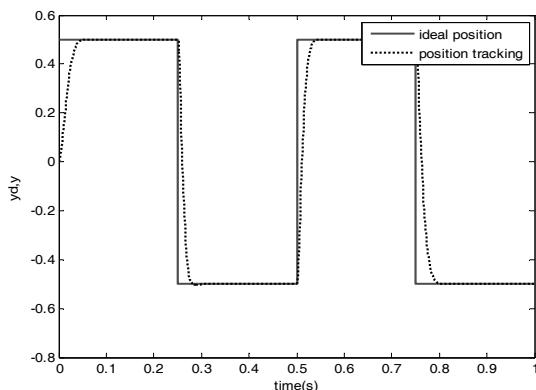


图 9-2 基于无监督 Hebb 学习规则的位置跟踪($M=1$)

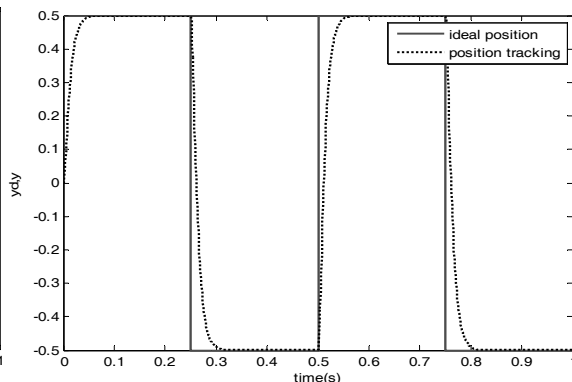


图 9-3 基于有监督的 Delta 学习规则的位置跟踪($M=2$)

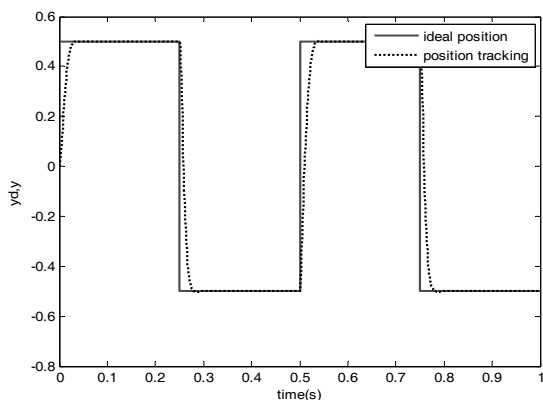


图 9-4 基于有监督 Hebb 学习规则的位置跟踪($M=3$)

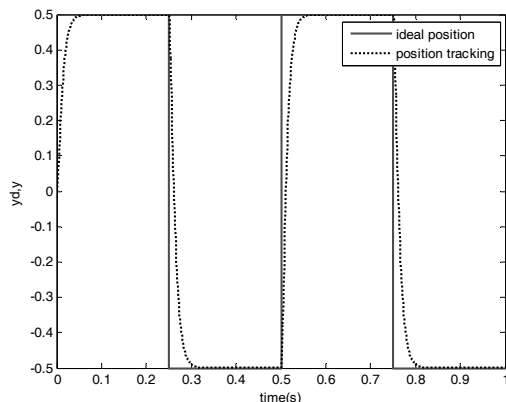


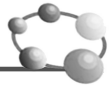
图 9-5 基于改进学习规则的位置跟踪($M=4$)

仿真程序：chap9_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;
    yd(k)=0.5*sign(sin(2*pi*k*ts));
    y(k)=0.368*y_1+0.26*y_2+0.1*u_1+0.632*u_2;
    error(k)=yd(k)-y(k);

    %Adjusting Weight Value by hebb learning algorithm
    M=1;
    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
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

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=y(k);

wkp_1=wkp(k);
wkd_1=wkd(k);
wki_1=wki(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

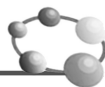
9.1.5 基于二次型性能指标学习算法的单神经元自适应PID控制

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。王顺晃^[67]设计了一种采用二次型性能指标学习算法的单神经元自适应PID控制器,简要描述如下:

设性能指标为

$$E(k) = \frac{1}{2} \left(P(y_d(k) - y(k))^2 + Q\Delta^2 u(k) \right) \quad (9.8)$$

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



神经元的输出为

$$u(k) = u(k-1) + K \sum_{i=1}^3 w'_i(k) x_i(k) \quad (9.9)$$

$$\begin{aligned} w'_i(k) &= w_i(k) / \sum_{i=1}^3 |w_i(k)| \quad (i=1,2,3) \\ w_1(k) &= w_1(k-1) + \eta_1 K \left[P b_0 z(k) x_1(k) - Q K \sum_{i=1}^3 (w_i(k) x_i(k)) x_1(k) \right] \\ w_2(k) &= w_2(k-1) + \eta_p K \left[P b_0 z(k) x_2(k) - Q K \sum_{i=1}^3 (w_i(k) x_i(k)) x_2(k) \right] \\ w_3(k) &= w_3(k-1) + \eta_D K \left[P b_0 z(k) x_3(k) - Q K \sum_{i=1}^3 (w_i(k) x_i(k)) x_3(k) \right] \end{aligned} \quad (9.10)$$

式中, b_0 为输出响应的第一个值, 且

$$\begin{aligned} x_1(k) &= e(k) \\ x_2(k) &= e(k) - e(k-1) \\ x_3(k) &= \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2) \\ z(k) &= e(k) \end{aligned} \quad (9.11)$$

9.1.6 仿真实例

设被控对象过程模型为

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

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 $y_d(k) = 1.0$ 。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 $\eta_1 = 4, \eta_p = 120, \eta_D = 159, w_1(0) = 0.34, w_2(0) = 0.32, w_3(0) = 0.33$, 神经元自适应 PID 跟踪及中权值变化结果如图 9-6 和图 9-7 所示。

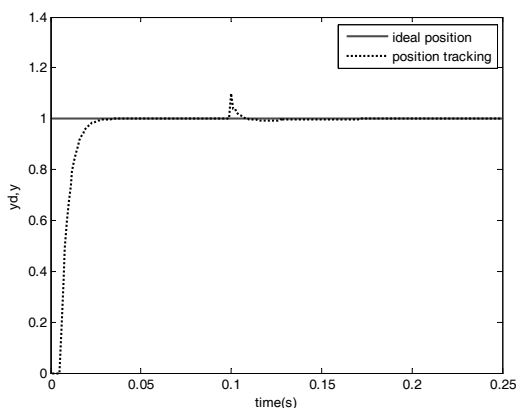


图 9-6 二次型性能指标学习单神经元自适应 PID 位置跟踪

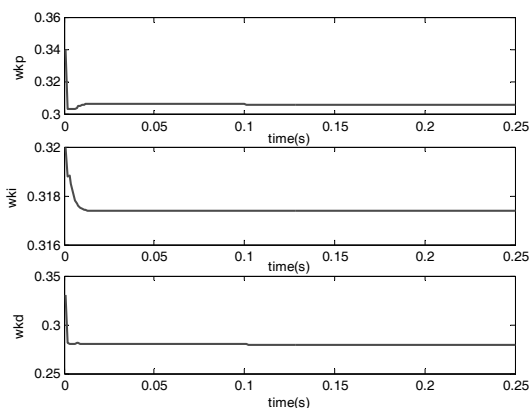


图 9-7 单神经元 PID 控制过程中权值变化

仿真程序: chap9_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;

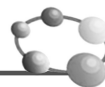
%Initilizing kp,ki and kd
wkp_1=rand;
wki_1=rand;
wkd_1=rand;

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;
    yd(k)=1.0;                                %Tracing Step Signal

    ym(k)=0;
    if k==100
        ym(k)=0.10; %Disturbance
```



```

end
y(k)=0.368*y_1+0.26*y_2+u_6+0.632*u_7+ym(k);
error(k)=yd(k)-y(k);

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

b0=y(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

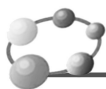
error_2=error_1;
error_1=error(k);

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=y(k);
end
figure(1);
plot(time,yd,'r',time,y,'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
figure(2);

```



```

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

```



9.2 基于 RBF 神经网络整定的 PID 控制

9.2.1 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 9-8 所示。

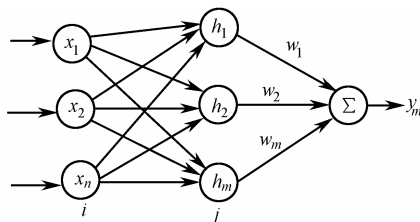


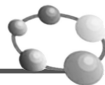
图 9-8 RBF 神经网络结构

(2) 被控对象 Jacobian 信息的辨识算法

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

$$h_j = \exp\left(-\frac{\|x - c_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \dots, m \quad (9.12)$$

其中网络的第 j 个节点的中心矢量为 $C_j = [c_{j1}, c_{j2}, \dots, c_{ji}, \dots, c_{jn}]^T$, $i = 1, 2, \dots, n$ 。



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

$$\mathbf{W} = [[w_1, w_2, \dots, w_m]]^T \quad (9.13)$$

辨识网络的输出为

$$y_m(k) = w_1 h_1 + w_2 h_2 + \dots + w_m h_m \quad (9.14)$$

辨识器的性能指标函数为

$$J = \frac{1}{2} (y(k) - y_m(k))^2 \quad (9.15)$$

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

$$\Delta w_j(k) = \eta (y(k) - y_m(k)) h_j$$

$$w_j(k) = w_j(k-1) + \Delta w_j(k) + \alpha (w_j(k-1) - w_j(k-2))$$

$$\Delta b_j(k) = \eta (y(k) - y_m(k)) w_j h_j \frac{\|\mathbf{X} - \mathbf{C}_j\|^2}{b_j^3}$$

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

$$\Delta c_{ji}(k) = \eta (y(k) - y_m(k)) w_j \frac{x_j - c_{ji}}{b_j^2}$$

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

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变化的灵敏度信息) 算法为

$$\frac{\partial y(k)}{\partial \Delta u(k)} \approx \frac{\partial y_m(k)}{\partial \Delta u(k)} = \sum_{j=1}^m w_j h_j \frac{c_{ji} - x_1}{b_j^2} \quad (9.16)$$

式中, $x_1 = \Delta u(k)$ 。

9.2.2 RBF 网络 PID 整定原理

采用增量式 PID 控制器, 控制误差为

$$e(k) = y_d(k) - y(k)$$

PID 三项输入为

$$\begin{aligned} xc(1) &= e(k) - e(k-1) \\ xc(2) &= e(k) \\ xc(3) &= e(k) - 2e(k-1) + e(k-2) \end{aligned} \quad (9.17)$$

控制算法为

$$\begin{aligned} u(k) &= u(k-1) + \Delta u(k) \\ \Delta u(k) &= k_p (e(k) - e(k-1)) + k_i e(k) + k_d (e(k) - 2e(k-1) + e(k-2)) \end{aligned} \quad (9.18)$$

神经网络整定指标为



$$E(k) = \frac{1}{2} e(k)^2 \quad (9.19)$$

k_p, k_i, k_d 的调整采用梯度下降法

$$\Delta k_p = -\eta \frac{\partial E}{\partial k_p} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial \Delta u} \frac{\partial \Delta u}{\partial k_p} = \eta e(k) \frac{\partial y}{\partial \Delta u} xc(1) \quad (9.20)$$

$$\Delta k_i = -\eta \frac{\partial E}{\partial k_i} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial \Delta u} \frac{\partial \Delta u}{\partial k_i} = \eta e(k) \frac{\partial y}{\partial \Delta u} xc(2) \quad (9.21)$$

$$\Delta k_d = -\eta \frac{\partial E}{\partial k_d} = -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial \Delta u} \frac{\partial \Delta u}{\partial k_d} = \eta e(k) \frac{\partial y}{\partial \Delta u} xc(3) \quad (9.22)$$

式中, $\frac{\partial y}{\partial \Delta u}$ 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 9-9 所示。

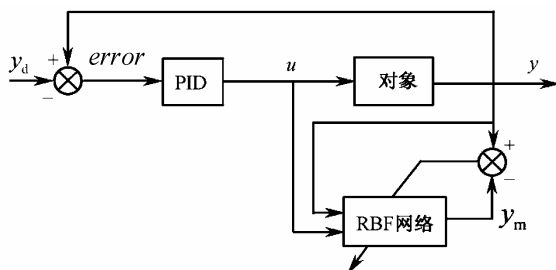


图 9-9 RBF 网络整定 PID 控制框图

9.2.3 仿真实例

被控对象为

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

输入指令信号为 1.0, RBF 网络结构选为 3-6-1, 网络辨识的三个输入为: $\Delta u(k)$, $y(k)$, $y(k-1)$ 。 $M=1$ 时为 RBF 整定的 PID 控制, 其结果如图 9-10 至图 9-12 所示, $M=2$ 时为未加整定的 PID 控制, 其结果如图 9-13 所示。

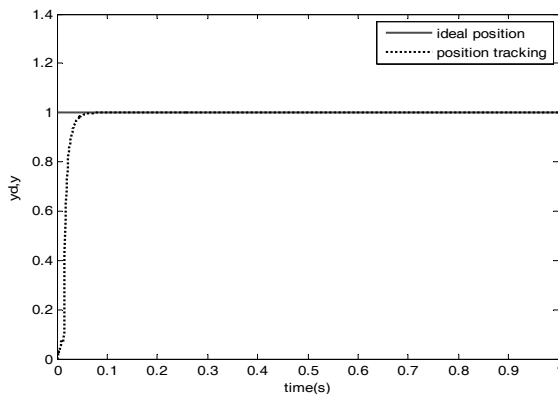
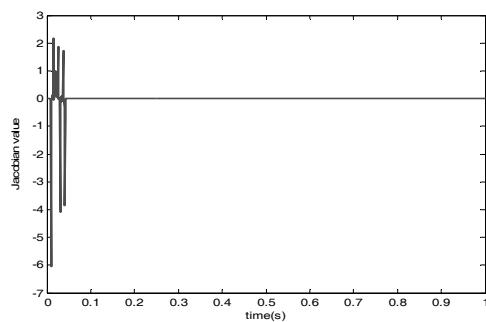
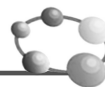
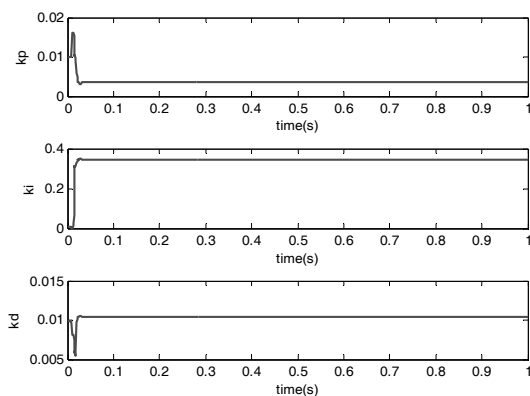
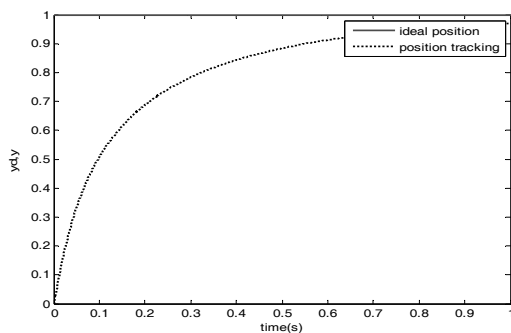


图 9-10 RBF 整定 PID 控制响应 ($M=1$)

图 9-11 对象 Jacobian 信息的辨识结果 ($M=1$)图 9-12 参数自适应整定曲线 ($M=1$)图 9-13 未整定的 PID 控制方波响应 ($M=2$)

仿真程序: chap9_3.m

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

xite=0.5;
alfa=0.05;
beta=0.01;
x=[0,0,0]';

ci=zeros(3,6);
```



```
bi=10*ones(6,1);
w=0.10*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;
kp0=0.01;ki0=0.01;kd0=0.01;

kp_1=kp0;
kd_1=kd0;
ki_1=ki0;

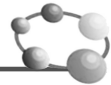
xitekp=0.15;
xitek=0.15;
xiteki=0.15;

ts=0.001;
for k=1:1:1000
    time(k)=k*ts;
    yd(k)=1.0;
    y(k)=(-0.1*y_1+u_1)/(1+y_1^2); %Nonlinear plant

    for j=1:1:6
        h(j)=exp(-norm(x-ci(:,j))^2/(2*bi(j)*bi(j)));
    end
    ym(k)=w'*h;

    d_w=0*w;
    for j=1:1:6
        d_w(j)=xite*(y(k)-ym(k))*h(j);
    end
    w=w_1+d_w+alfa*(w_1-w_2);

    d_bi=0*bi;
    for j=1:1:6
        d_bi(j)=xite*(y(k)-ym(k))*w(j)*h(j)*(bi(j)^-3)*norm(x-ci(:,j))^2;
    end
    bi=bi_1+d_bi+alfa*(bi_1-bi_2);
    for j=1:1:6
        for i=1:1:3
```



```

        d_ci(i,j)=xite*(y(k)-ym(k))*w(j)*h(j)*(x(i)-ci(i,j))*(bi(j)^-2);
    end
end
ci=ci_1+d_ci+alfa*(ci_1-ci_2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yu=0;
for j=1:1:6
    yu=yu+w(j)*h(j)*(-x(1)+ci(1,j))/bi(j)^2;
end
dyu(k)=yu;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Start of Control system%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
error(k)=yd(k)-y(k);
kp(k)=kp_1+xitekp*error(k)*dyu(k)*xc(1);
kd(k)=kd_1+xitekd*error(k)*dyu(k)*xc(2);
ki(k)=ki_1+xiteki*error(k)*dyu(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;
switch M
case 1
case 2 %Only PID Control
    kp(k)=kp0;
    ki(k)=ki0;
    kd(k)=kd0;
end
du(k)=kp(k)*xc(1)+kd(k)*xc(2)+ki(k)*xc(3);
u(k)=u_1+du(k);
%Return of parameters
x(1)=du(k);
x(2)=y(k);
x(3)=y_1;

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

ci_3=ci_2;
ci_2=ci_1;
ci_1=ci;

```



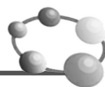
```
bi_3=bi_2;
bi_2=bi_1;
bi_1=bi;

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 I

error_2=error_1;
error_1=error(k);

kp_1=kp(k);
kd_1=kd(k);
ki_1=ki(k);
end
if M==1
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
figure(2);
plot(time,y,'r',time,ym,'b','linewidth',2);
xlabel('time(s)');ylabel('y,ym');
figure(3);
plot(time,dyu,'r','linewidth',2);
xlabel('time(s)');ylabel('Jacobian value');
figure(4);
subplot(311);
plot(time,kp,'r','linewidth',2);
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'r','linewidth',2);
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'r','linewidth',2);
xlabel('time(s)');ylabel('kd');
elseif M==2
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
```



```
legend('ideal position','position tracking');
end
```

需要注意的是,采用 Hebb 学习规则或梯度下降法来设计神经网络权值调节律,神经网络参数或控制参数都是按经验选取或试凑,只能保证闭环系统在局部得到优化,闭环系统的稳定性得不到保障,如果神经网络参数或控制参数选择不当,闭环系统控制很容易发散。针对这一问题,出现了在线自适应神经网络控制方法,它是基于 Lyapunov 稳定性理论获得权值自适应律,从而获得闭环系统的稳定性。



9.3 基于自适应神经网络补偿的倒立摆 PD 控制

9.3.1 问题描述

考虑如下二阶非线性系统

$$\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u \quad (9.23)$$

式中, f 为未知非线性函数, g 为已知非线性函数, $u \in R^n$ 和 $y \in R^n$ 分别为系统的输入和输出。

式 (9.23) 还可写为

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + g(x_1, x_2)u \\ y &= x_1 \end{aligned} \quad (9.24)$$

设位置指令为 y_d , 令

$$e = y_d - y = y_d - x_1, \quad E = (e \quad \dot{e})^T$$

选择 $K = (k_p, k_d)^T$, 使多项式 $s^2 + k_d s + k_p = 0$ 的所有根部都在复平面左半平面上。

取控制律为^[59]

$$u^* = \frac{1}{g(x)} [-f(x) + \ddot{y}_d + K^T E] \quad (9.25)$$

将式 (9.25) 代入式 (9.23), 得到闭环控制系统的方程

$$\ddot{e} + k_p e + k_d \dot{e} = 0 \quad (9.26)$$

由 K 的选取, 可得 $t \rightarrow \infty$ 时 $e(t) \rightarrow 0$, $\dot{e}(t) \rightarrow 0$, 即系统的输出 y 及其导数渐进地收敛于理想输出 y_d 及其导数。

如果非线性函数 $f(x)$ 是已知的, 则可以选择控制 u 来消除其非线性的性质, 然后再根据线性控制理论设计控制器。

9.3.2 自适应神经网络设计与分析

如果 $f(x)$ 未知, 控制律式 (9.25) 很难实现。可采用神经网络系统 $\hat{f}(x)$ 代替 $f(x)$, 实现自适应神经网络补偿。



9.3.2.1 基本的神经网络系统

RBF 神经网络具有万能逼近的特性^[44], 采用 RBF 网络可实现对不确定项 f 进行自适应逼近。RBF 网络算法为

$$h_j = g\left(\left\|\mathbf{x} - \mathbf{c}_{ij}\right\|^2 / b_j^2\right)$$

$$f = \mathbf{W}^T \mathbf{h}(\mathbf{x}) + \varepsilon$$

式中, $\mathbf{x}$ 为网络的输入信号, i 为网络输入个数, j 为网络隐含层节点的个数, $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ 为高斯基函数的输出, $\mathbf{W}$ 为神经网络权值, ε 为神经网络逼近误差, $\varepsilon \leq \varepsilon_N$ 。

采用 RBF 网络逼近 f , 根据 f 的表达式, 网络输入取 $\mathbf{x} = [e \quad \dot{e}]^T$, RBF 神经网络的输出为

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) \quad (9.27)$$

9.3.2.2 自适应神经网络控制器的设计与分析

采用神经网络逼近 f , 则控制律式 (9.25) 变为

$$u = \frac{1}{g(\mathbf{x})} \left[-\hat{f}(\mathbf{x}) + \ddot{y}_d + \mathbf{K}^T \mathbf{E} \right] \quad (9.28)$$

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) \quad (9.29)$$

式中, $\mathbf{h}(\mathbf{x})$ 为神经网络高斯基函数, 神经网络权值 $\hat{\mathbf{W}}$ 根据自适应律而变化。

设计自适应律为

$$\dot{\hat{\mathbf{W}}} = -\gamma \mathbf{E}^T \mathbf{P} \mathbf{b} \mathbf{h}(\mathbf{x}) \quad (9.30)$$

9.3.2.3 稳定性分析

参考王立新^[59]所提出的间接自适应模糊控制方法, 对本闭环系统进行稳定性分析如下: 由式 (9.28) 代入式 (9.23), 可得如下系统的闭环动态方程

$$\ddot{e} = -\mathbf{K}^T \mathbf{E} + \left[\hat{f}(\mathbf{x}) - f(\mathbf{x}) \right] \quad (9.31)$$

令:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k_p & -k_d \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (9.32)$$

则动态方程 (9.31) 可写为向量形式:

$$\dot{\mathbf{E}} = \mathbf{A} \mathbf{E} + \mathbf{b} \left[\hat{f}(\mathbf{x}) - f(\mathbf{x}) \right] \quad (9.33)$$

设最优参数为

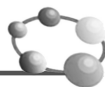
$$\mathbf{W}^* = \arg \min_{\mathbf{W} \in \Omega} \left[\sup \left| \hat{f}(\mathbf{x}) - f(\mathbf{x}) \right| \right] \quad (9.34)$$

式中, Ω 为 $\mathbf{W}$ 的集合。

定义最小逼近误差为

$$\omega = \hat{f}(\mathbf{x} | \mathbf{W}^*) - f(\mathbf{x}) \quad (9.35)$$

式 (9.33) 可写为



$$\dot{E} = \Lambda E + b \left\{ \left[\hat{f}(x) - \hat{f}(x|W^*) \right] + \omega \right\} \quad (9.36)$$

将式 (9.29) 代入式 (9.36), 可得闭环动态方程:

$$\dot{E} = \Lambda E + b \left[(\hat{W} - W^*)^T h(x) + \omega \right] \quad (9.37)$$

该方程清晰地描述了跟踪误差和权值 $\hat{W}$ 之间的关系。自适应律的任务是为 $\hat{W}$ 确定一个调节机理, 使得跟踪误差 E 和参数误差 $\hat{W} - W^*$ 达到最小。

定义 Lyapunov 函数

$$V = \frac{1}{2} E^T P E + \frac{1}{2\gamma} (\hat{W} - W^*)^T (\hat{W} - W^*) \quad (9.38)$$

式中, γ 是正常数, P 为一个正定矩阵且满足 Lyapunov 方程

$$\Lambda^T P + P \Lambda = -Q \quad (9.39)$$

式中, Q 是一个任意的 2×2 正定矩阵, Λ 由式 (9.32) 给出。

取 $V_1 = \frac{1}{2} E^T P E$, $V_2 = \frac{1}{2\gamma} (\hat{W} - W^*)^T (\hat{W} - W^*)$, 令 $M = b \left[(\hat{W} - W^*)^T h(x) + \omega \right]$, 则 (9.37) 式变为

$$\dot{E} = \Lambda E + M$$

则

$$\begin{aligned} \dot{V}_1 &= \frac{1}{2} \dot{E}^T P E + \frac{1}{2} E^T P \dot{E} = \frac{1}{2} (E^T \Lambda^T + M^T) P E + \frac{1}{2} E^T P (\Lambda E + M) \\ &= \frac{1}{2} E^T (\Lambda^T P + P \Lambda) E + \frac{1}{2} M^T P E + \frac{1}{2} E^T P M \\ &= -\frac{1}{2} E^T Q E + \frac{1}{2} (M^T P E + E^T P M) = -\frac{1}{2} E^T Q E + E^T P M \end{aligned}$$

将 M 代入上式, 并考虑 $E^T P b (\hat{W} - W^*)^T h(x) = (\hat{W} - W^*)^T [E^T P b h(x)]$, 得

$$\begin{aligned} \dot{V}_1 &= -\frac{1}{2} E^T Q E + E^T P b (\hat{W} - W^*)^T h(x) + E^T P b \omega \\ &= -\frac{1}{2} E^T Q E + (\hat{W} - W^*)^T E^T P b h(x) + E^T P b \omega \\ \dot{V}_2 &= \frac{1}{\gamma} (\hat{W} - W^*)^T \dot{\hat{W}} \end{aligned}$$

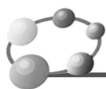
V 的导数为

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\frac{1}{2} E^T Q E + E^T P b \omega + \frac{1}{\gamma} (\hat{W} - W^*)^T \left[\dot{\hat{W}} + \gamma E^T P b h(x) \right]$$

将自适应律式 (9.30) 代入上式, 得

$$\dot{V} = -\frac{1}{2} E^T Q E + E^T P b \omega$$

由于 $-\frac{1}{2} E^T Q E \leq 0$, 通过选取最小逼近误差 ω 非常小的神经网络, 可实现 $\dot{V} \leq 0$ 。



9.3.3 仿真实例

被控对象取单级倒立摆，其动态方程如下

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + g(x)u\end{aligned}$$

式中， $f(x) = \frac{g \sin x_1 - m l x_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$ ， $g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$ ， x_1 和

x_2 分别为摆角和摆速， $g = 9.8 \text{ m/s}^2$ ， m_c 为小车质量， $m_c = 1 \text{ kg}$ ， m 为摆杆质量， $m = 0.1 \text{ kg}$ ， l 为摆长的一半， $l = 0.5 \text{ m}$ ， u 为控制输入。

位置指令为 $y_d(t) = 0.1 \sin t$ 。倒立摆初始状态为 $[\pi/60, 0]$ 。取 RBF 网络结构为 2-5-1。RBF 网络高斯基函数参数的取值对神经网络控制的作用很重要，如果参数取值不合适，将使高斯基函数无法得到有效的映射，从而导致 RBF 网络无效。故 c_{ij} 按网络输入值的范围取值，取 $c_{ij} = 0.10$ ， $\sigma_j = 0.50$ ， $i = 2, j = 5$ ，神经网络权值初始值取 0。

采用控制律式 (9.28)，自适应律取式 (9.30)，取 $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$ ， $k_d = 20$ ， $k_p = 10$ ，自适应参数取 $\gamma = 100$ 。仿真结果如图 9-14 至图 9-16 所示。

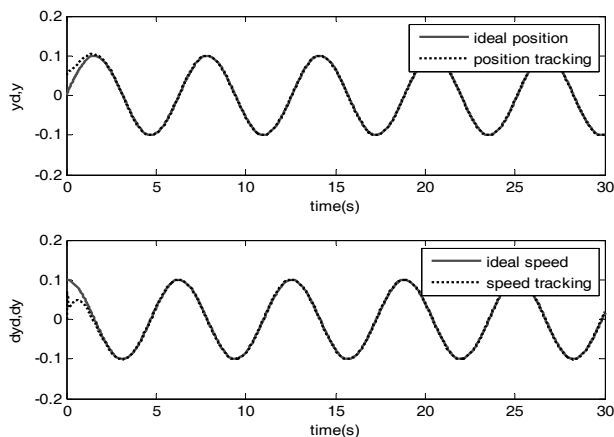


图 9-14 位置和速度跟踪

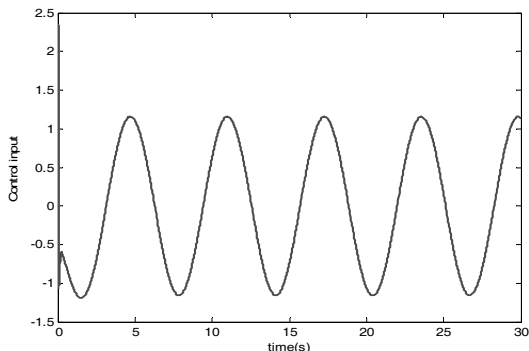
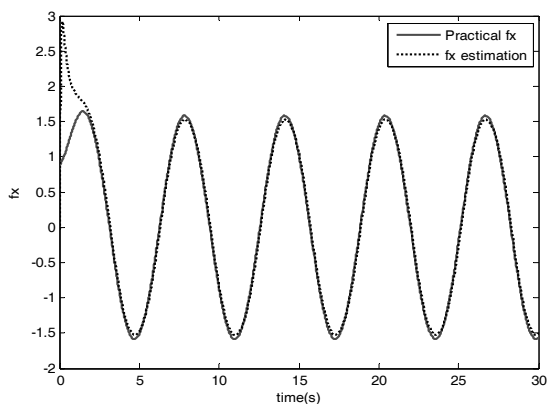
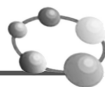


图 9-15 控制输入信号

图 9-16 $f(x)$ 及 $\hat{f}(x)$ 的变化

仿真程序：

(1) Simulink 主程序：chap9_4sim.mdl（见图 9-17）

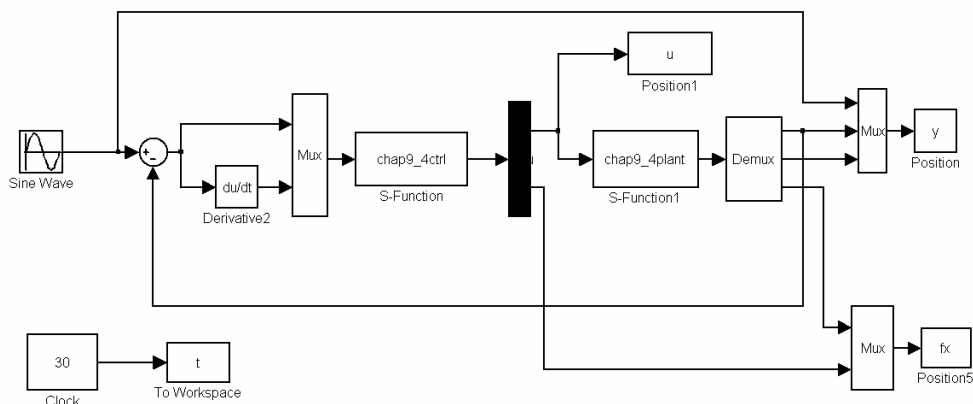


图 9-17 自适应神经网络 PD 控制主程序

(2) 控制器 S 函数：chap9_4ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c bn
sizes = simsizes;
```



```
sizes.NumContStates = 5;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0*ones(5,1)];
c=0.10*ones(2,5);
bn=0.5*ones(5,1);
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global c bn
gama=30;
yd=0.1*sin(t);
dyd=0.1*cos(t);
ddy=-0.1*sin(t);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

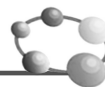
k1=50;
k2=30;
k=[k2;k1];
E=[e,de]';

A=[0 -k2;
    1 -k1];
Q=[500 0;0 500];
P=lyap(A,Q);

xi=[e,de];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*bn(j)*bn(j)));
end
W=[x(1) x(2) x(3) x(4) x(5)]';

b=[0;1];
S=-gama*E'*P*b*h;

for i=1:1:5
    sys(i)=S(i);
```



```

end

function sys=mdlOutputs(t,x,u)
global c bn
yd=0.1*sin(t);
dyd=0.1*cos(t);
ddyd=-0.1*sin(t);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

k1=50;
k2=30;
k=[k2;k1];
E=[e,de]';

W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[e;de];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*bn(j)*bn(j)));
end
fxp=W'*h;

%%%%%%%%%%%%%
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
gx=cos(x(1))/(mc+m);
gx=gx/S;
%%%%%%%%%%%%%
ut=1/gx*(-fxp+ddyd+k'*E);

sys(1)=ut;
sys(2)=fxp;

```

(3) 被控对象 S 函数: chap9_4plant.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);

```



```
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

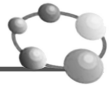
sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
g=9.8;
mc=1.0;
m=0.1;
l=0.5;

S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

(4) 作图程序: chap9_4plot.m

```
close all;
```



```
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,3),'k','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('fx');
legend('Practical fx','fx estimation');
```

第 10 章 基于遗传算法整定的 PID 控制



10.1 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。

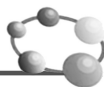
遗传算法的基本操作如下^[45]。

(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。根据位串的适配值复制，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 0.40~1.0 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。



10.2 遗传算法的优化设计

10.2.1 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是多少。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。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。

10.2.2 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 $J(x)$ 到个体适应度函数 $F(x)$ 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。



10.3 遗传算法求函数极大值

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。

10.3.1 二进制编码遗传算法求函数极大值

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, $x:0000110111 \quad 1101110001$ 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

$$x_i = 4.096 \times \frac{y_i}{1023} - 2.048 \quad (i=1, 2) \quad (10.1)$$

例如, 对个体 $x:0000110111 \quad 1101110001$, 它由两个代码所组成

$$y_1 = 55, y_2 = 881$$



仿真程序: chap10_1.m

```
%Generic Algorithm for function f(x1,x2) optimum
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;

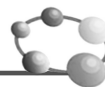
for s=1:1:Size
m=E(s,:);
y1=0;y2=0;

%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
```

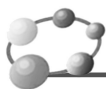
```
%***** Step 3 : Crossover Operation *****
```

```
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
```



```
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,'r','linewidth',2);
xlabel('Times');ylabel('Best J');
figure(2);
plot(time,bfi,'r','linewidth',2);
xlabel('times');ylabel('Best F');
```

10.3.2 实数编码遗传算法求函数极大值

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 2 个实数分别表示两个决策变量 x_1, x_2 , 分别将 x_1, x_2 的定义域离散化为从离散点 -2.048 到离散点 2.048 的 Size 个实数。
- (4) 确定个体评价方法: 个体的适应度直接取为对应的目标函数值, 即

$$F(x) = f(x_1, x_2) \quad (10.4)$$

选个体适应度的倒数作为目标函数

$$J(x) = \frac{1}{F(x)} \quad (10.5)$$

(5) 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。

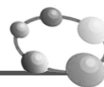
(6) 确定遗传算法的运行参数: 群体大小 $M = 500$, 终止进化代数 $G = 200$, 交叉概率 $P_c = 0.90$, 采用自适应变异概率 $P_m = 0.10 - [1:1:\text{Size}] \times 0.01/\text{Size}$, 即变异概率与适应度有关, 适应度越小, 变异概率越大。

上述六个步骤构成了用于求 Rosenbrock 函数极大值的优化计算的实数编码遗传算法。

采用上述方法进行仿真, 经过 200 步迭代, 最佳样本为

$$\text{BestS} = [-2.0438 \quad -2.044]$$

即当 $x_1 = -2.0438$, $x_2 = -2.044$ 时, Rosenbrock 函数具有极大值, 极大值为 3880.3。



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 10-3 和图 10-4 所示，由仿真结果可知，采用实数编码的遗传算法搜索效率低于二进制遗传算法。

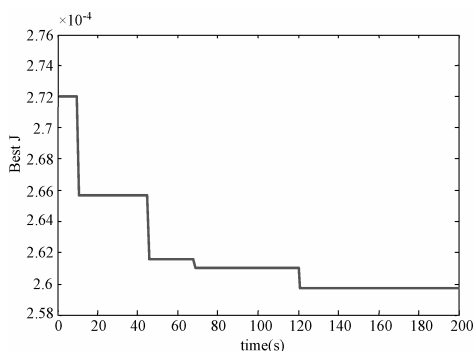


图 10-3 目标函数 J 的优化过程

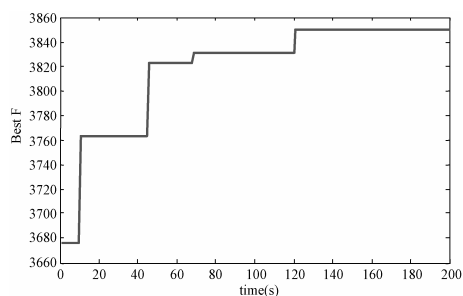


图 10-4 适应度 F 的优化过程

仿真程序: chap10_2.m

```
%Generic Algorithm for function f(x1,x2) optimum
clear all;
close all;

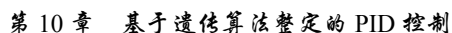
Size=500;
CodeL=2;

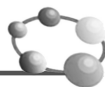
MinX(1)=-2.048;
MaxX(1)=2.048;
MinX(2)=-2.048;
MaxX(2)=2.048;

E(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
E(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);

G=200;
BsJ=0;

%***** Start Running *****
for kg=1:1:G
    time(kg)=kg;
```





```

        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*E(i+1,:)+(1-alfa)*E(i,:);
            TempE(i+1,:)=alfa*E(i,:)+(1-alfa)*E(i+1,:);
        end
    end
    TempE(Size,:)=BestS;
    E=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;
    E=TempE;

    end
    BestS
    Bestfi

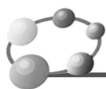
    figure(1);
    plot(time,BestJ,'r','linewidth',2);
    xlabel('Times');ylabel('Best J');
    figure(2);
    plot(time,bfi,'r','linewidth',2);
    xlabel('times');ylabel('Best F');

```



10.4 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,只通过字符串进行简单地复制、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。

10.4.1 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。

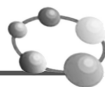
(2) 选取初始种群

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,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 个字串，每个字串 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 10-5 来表示。

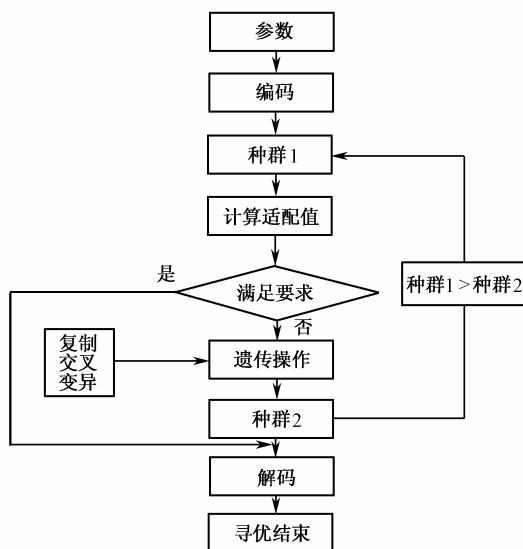


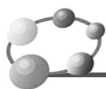
图 10-5 遗传算法流程图

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 ，

$$\text{取 } f = \frac{1}{J} ;$$

- （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。



10.4.2 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

$$J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t)) dt + w_3 \cdot t_u \quad (10.6)$$

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

$$\text{if } e(t) < 0 \quad J = \int_0^{\infty} (w_1 |e(t)| + w_2 u^2(t) + w_4 |e(t)|) dt + w_3 \cdot t_u \quad (10.7)$$

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 $w_1 = 0.999, w_2 = 0.001, w_3 = 2.0, w_4 = 100$ 。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 10-6 和图 10-7 所示。

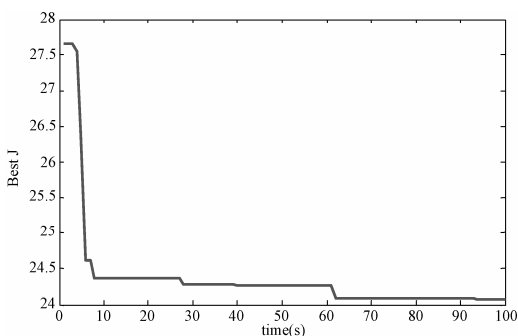


图 10-6 代价函数值 J 的优化过程

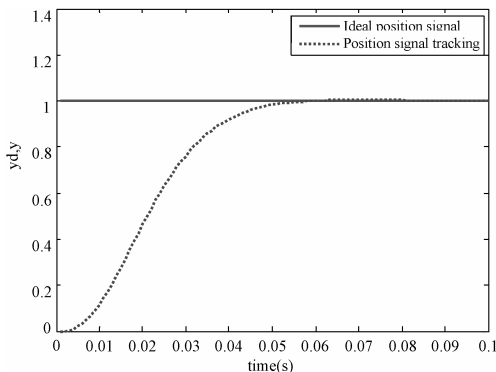
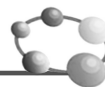


图 10-7 整定后的 PID 阶跃响应



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。

仿真程序：

主程序：chap10_3.m

```
%GA(Generic Algorithm) Program to optimize PID Parameters
clear all;
close all;
global yd y timef

Size=30;
CodeL=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]=chap10_3plant(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);

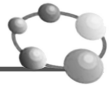
Rest=fi_Size-fi_S;
[RestValue,Index]=sort(Rest);

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(i,:)=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
end
%Guarantee TempE(Size,:) belong to the best individual
TempE(Size,:)=BestS;
Kpid=TempE;
end
Bestfi
BestS
Best_J=BestJ(G)
figure(1);
plot(time,BestJ,'r','linewidth',2);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,yd,'r',timef,y,'b:','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('Ideal position signal','Position signal tracking');
被控对象子程序: chap10_3plant.m
function [Kpidi,BsJ]=pid_gaf(Kpidi,BsJ)
global yd y timef

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;
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;
    yd(k)=1.0;

```



```
u(k)=Kpdi(1)*x(1)+Kpdi(2)*x(2)+Kpdi(3)*x(3);

if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=yd(k)-y(k);
%----- Return of PID parameters -----
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=y(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

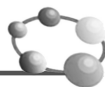
error_2=error_1;
error_1=error(k);
if s==0
    if y(k)>0.95&y(k)<1.05
        tu=timef(k);
        s=1;
    end
end
end
end

for i=1:1:P
    Ji(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
    B=B+Ji(i);
    if i>1
        erry(i)=y(i)-y(i-1);
        if erry(i)<0
            B=B+100*abs(erry(i));
        end
    end
end
end
BsJ=B+0.2*tu*10;
```

10.4.3 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，输入指令为单位阶跃信号。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 $\text{BestS}=[0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$ 。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 10-8 所示。采用整定后的二进制遗传算法优化 PID 阶跃响应如图 10-9 所示。

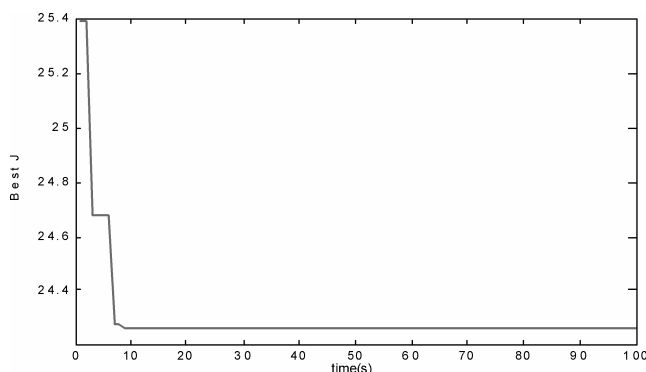


图 10-8 代价函数值 J 的优化过程

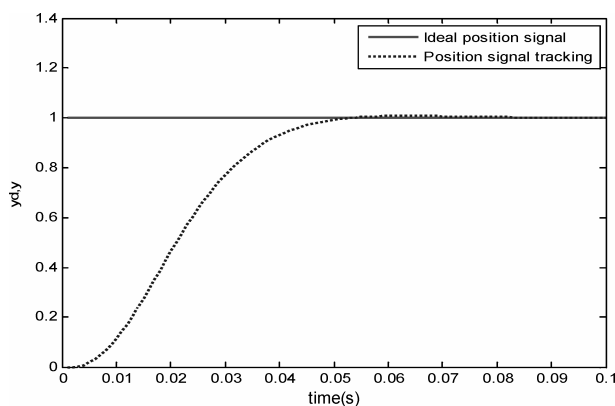


图 10-9 二进制遗传算法优化 PID 阶跃响应

仿真程序

主程序：chap10_4.m

```
%GA(Generic Algorithm) Program to optimize Parameters of PID
clear all;
close all;
global yd y 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
time(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);

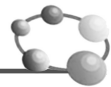
m2=m(CodeL+1:1:2*CodeL);
for i=1:1:CodeL
    y2=y2+m2(i)*2^(i-1);
end
Kpid(s,2)=(MaxX(2)-MinX(2))*y2/1023+MinX(2);

m3=m(2*CodeL+1:1:3*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]=chap10_4plant(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);
            TempE(i+1,j)=E(i,j);
        end
    end
end
end

```



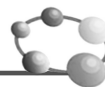
```
TempE(Size,:)=BestS;
E=TempE;

%***** 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=1:1:3*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 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,'r','linewidth',2);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,yd,'r',timef,y,'b:','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('Ideal position signal','Position signal tracking');
子程序: chap10_4plant.m
function [Kpidi,BsJ]=pid_gaf(Kpidi,BsJ)
global yd y timef

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;
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;
    yd(k)=1.0;

    u(k)=Kpdi(1)*x(1)+Kpdi(2)*x(2)+Kpdi(3)*x(3);

    if u(k)>=10
        u(k)=10;
    end
    if u(k)<=-10
        u(k)=-10;
    end
    y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
    error(k)=yd(k)-y(k);
%----- Return of PID parameters -----
    u_2=u_1;u_1=u(k);
    y_2=y_1;y_1=y(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);
    if s==0
        if y(k)>0.95&y(k)<1.05
            tu=timef(k);
            s=1;
        end
    end
end
end

for i=1:1:P
    Ji(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
    B=B+Ji(i);
    if i>1
        erry(i)=y(i)-y(i-1);
    end
end

```



```

if erry(i)<0
    B=B+100*abs(erry(i));
end
end
end
BsJ=B+0.2*tu*10;

```

10.4.4 基于自适应在线遗传算法整定的 PD 控制

所谓在线 PD 整定，即在每个采样时间分别对 PD 参数进行整定。采用遗传算法在线整定 PID，就是针对每个采样时间实现 PD 控制参数的遗传算法优化。在采样时间 k ，选取足够多的个体，计算不同个体的自适应度，通过遗传算法的优化，选择自适应度大的个体所对应的 PD 控制参数作为该采样时间下 PD 的控制参数。

设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，并防止产生超调，采用误差绝对值及误差和误差变化率的加权和作为第 k 个采样时间第 i 个个体的参数选择最小目标函数。

$$J(i) = \alpha_p |\text{error}(i)| + \beta_p |de(i)| \quad (10.8)$$

式中， $\text{error}(i)$ 为第 k 个采样时间第 i 个个体的位置跟踪误差， $de(i)$ 为第 k 个采样时间第 i 个个体的位置跟踪误差变化率。

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

$$\text{if error}(i) < 0 \quad J(i) = J(i) + 100 |\text{error}(i)| \quad (10.9)$$

针对每个采样时间进行 PD 参数的遗传算法优化。仿真程序中， $M=1$ 时为采用遗传算法，否则为未整定的 PID 控制。在最小目标函数中，取 $\alpha_p = 0.95$ ， $\beta_p = 0.05$ 。遗传算法中使用的个体数为 120，进化代数为 10 代。交叉概率为 $P_c = 0.9$ ，采用自适应变异概率方法，即自适应度越大，变异概率越小，变异概率为 $P_m = 0.20 - [1:1:\text{Size}] \times 0.01/\text{Size}$ 。采用实数编码方式，参数 k_p 的取值范围为 $[9.0, 12.0]$ ， k_d 的取值范围为 $[0.2, 0.3]$ 。

PD 的阶跃响应、PD 整定过程中控制器 $u(k)$ 的变化及参数 k_p 、 k_d 的整定过程如图 10-10 至图 10-12 所示。由仿真结果可见，在控制的初始阶段（误差小于 0.50 时），为了尽快降低误差， k_p 上升、 k_d 下降；当上升到一定程度时（误差大于 0.50 时），为了防止误差变化太快而产生超调， k_p 下降、 k_d 上升；当上升到指令值而产生超调时（ $t=0.40s$ 时），为了尽快降低误差， k_p 上升、 k_d 下降。

为了避免参数选取范围过大，先按经验选取一组 k_p 、 k_d 参数，然后再在这组参数的周围利用遗传算法进行设计，从而减少初始寻优的盲目性，节约计算量。

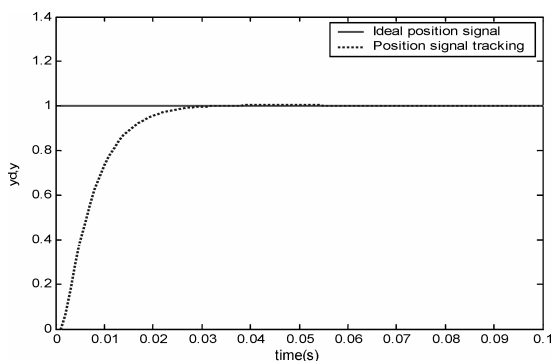
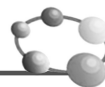


图 10-10 阶跃响应

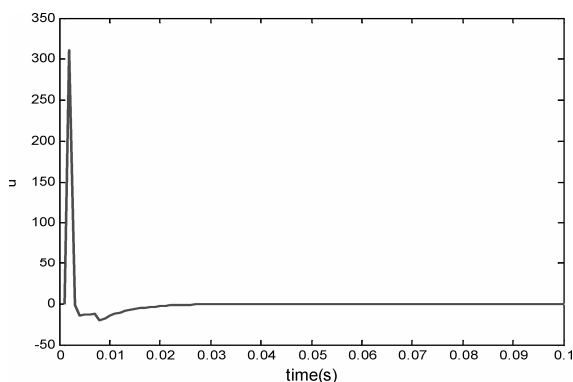
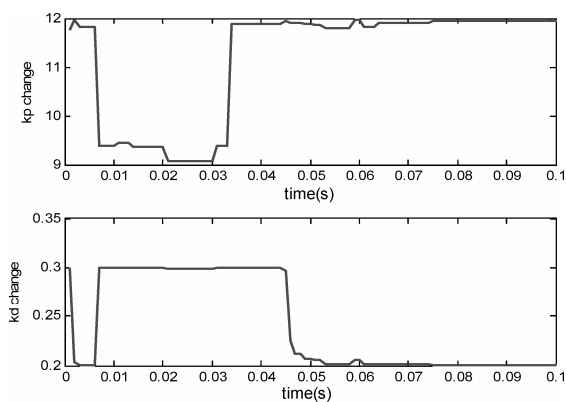


图 10-11 控制输入

图 10-12 PD 整定过程中 k_p 和 k_d 的变化

仿真程序

主程序: chap10_5.m

```
%GA(Generic Algorithm) to Optimize Online PID Control
```

```
clear all;
```

```
close all;
```

```
Size=120;
```

```
CodeL=2;
```

```
MinX(1)=9*ones(1);MaxX(1)=12*ones(1);
```



```
MinX(2)=0.20*ones(1);MaxX(2)=0.30*ones(1);

Kpid(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
Kpid(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);

BsJ=0;
J=0;
x=zeros(1,2);
xi=zeros(1,2);
xk=zeros(1,2);
ts=0.001;

error_1=0;

BestS=zeros(2,1);
for k=1:1:100
    k
    time(k)=k*ts;
    yd(k)=1;

    %u(k)=10*x(1)+0.2*x(2);    %Test PI: good results
    u(k)=BestS(1)*x(1)+BestS(2)*x(2);
    para=u(k);
    tSpan=[0 ts];

    [tt,xx]=ode45('chap10_5plant',tSpan,xk,[],para);
    xk=xx(length(xx),:);
    y(k)=xk(1);

    error(k)=yd(k)-y(k);
    x(1)=error(k);                % Calculating P
    x(2)=(error(k)-error_1)/ts;    % Calculating D

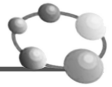
    error_1=error(k);

    B=0;
    G=10;
    for kg=1:1:G    %Era evolution

    %***** Step 1 : Evaluate BestJ *****
    for i=1:1:Size

        Kpidi=Kpid(i,:);
        ui(i)=Kpidi(1)*x(1)+Kpidi(2)*x(2);

        para=ui(i);
```



```

[tt,xxi]=ode45('chap10_5plant',tSpan,xk,[],para);
yi(i)=xxi(length(xxi),1);

errori(i)=yd(k)-yi(i);
du(i)=ui(i)-u(k);
de(i)=(errori(i)-error(k))/ts;
alfap=0.95;
betap=0;

if abs(error(k))<=0.50
    betap=0.05;
end
J=alfap*abs(errori(i))+betap*abs(de(i));

B=J;
if errori(i)<0
    B=B+100*abs(errori(i));
end

BsJi(i)=B;

[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;
end    %End of a Size!!!!!!!!!!

[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)
%***** 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

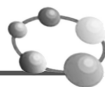
kr=1;
for i=Size:-1:1      % Select the Sizeth and Reproduce firstly
    for j=1:1:fi_S(i)
        TempE(kr,:)=Kpid(Indexfi(i,:),:);      % Select and Reproduce
        kr=kr+1;      % kr is used to reproduce
    end
end

%***** 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*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=0.20-[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      %End of kg
kph(k)=BestS(1);
kdh(k)=BestS(2);
BestS
end      %End of k
figure(1);
plot(time,yd,'r',time,y,'b','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('Ideal position signal','Position signal tracking');
```



```
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('Time(s)');ylabel('u');
figure(3);
subplot(211);
plot(time,kph,'r','linewidth',2);
xlabel('Time(s)');ylabel('kp change');
subplot(212);
plot(time,kdh,'r','linewidth',2);
xlabel('Time(s)');ylabel('kd change');
被控对象子程序: chap10_5plant.m
function dx=PlantModel(t,x,flag,para)
dx=zeros(2,1);
u=para;
dx(1)=x(2);
dx(2)=-50*x(2)+400*u;
```



10.5 基于摩擦模型补偿的 PD 控制

10.5.1 摩擦模型辨识

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

$$F_f(t) = 0.8 \operatorname{sgn}(\dot{\theta}(t)) \quad (10.10)$$

当 $F=2$ 时为库仑摩擦加黏性摩擦，摩擦模型为

$$F_f(t) = \operatorname{sgn}(\dot{\theta}(t)) (kx_1 |\dot{\theta}(t)| + kx_2) = \operatorname{sgn}(\dot{\theta}(t)) (0.30 |\dot{\theta}(t)| + 1.50) \quad (10.11)$$

式中， kx_1 和 kx_2 为待辨识参数。

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

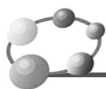
$$J = \int_0^\infty (w_1 |e(t)| + w_2 u^2(t)) dt \quad (10.12)$$

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

$$\text{if } e(t) < 0 \quad J = \int_0^\infty (w_1 |e(t)| + w_2 u^2(t) + w_3 |e(t)|) dt \quad (10.13)$$

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。

10.5.2 仿真实例

仿真中, $S=1$ 为正弦指令, $S=2$ 为阶跃信号, $F=1$ 为库仑摩擦, 取 $F=2$ 为库仑摩擦加黏性摩擦, $M=1$ 为不加摩擦补偿, $M=2$ 为加入摩擦补偿, $u_c(k)$ 为摩擦补偿项。

采样时间为 1ms, 取 $S=2$, 使输入指令为阶跃信号。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 $P_c=0.9$, $P_m=0.10-[1:1:\text{Size}]\times 0.01/\text{Size}$, 最优指标权值取 $w_1=0.999$, $w_2=0.001$, $w_3=10$ 。

被控对象程序 chap10_6plant.m 中, 可取 $u_c(k)=0$, 得到无摩擦补偿情况下阶跃响应, 如图 10-13 所示。

取 $F=2$, 运行程序 chap10_6.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 $\text{BestS}=[0.3523, 1.2257]$, $\text{BestJ}=25.5439$ 。摩擦参数辨识结果为 $kx_1=0.3523$, $kx_2=1.2257$, 控制器采用 PD 控制加摩擦补偿项, 即 $u(k)=50\text{error}(k)+0.5\text{derror}(k)+u_c(k)$, 采用摩擦补偿后的 PD 控制阶跃响应如图 10-14 所示, 代价函数值 J 的优化过程如图 10-15 所示。

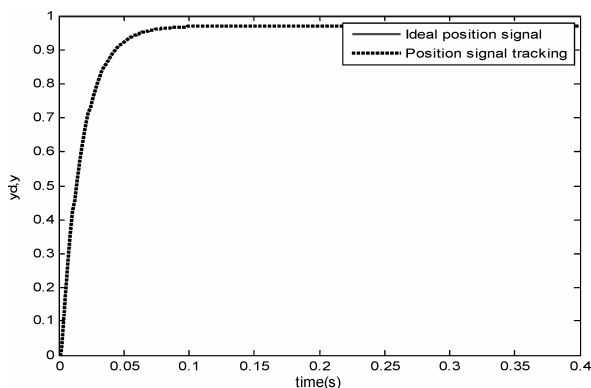


图 10-13 无摩擦补偿的阶跃响应 ($M=1$)

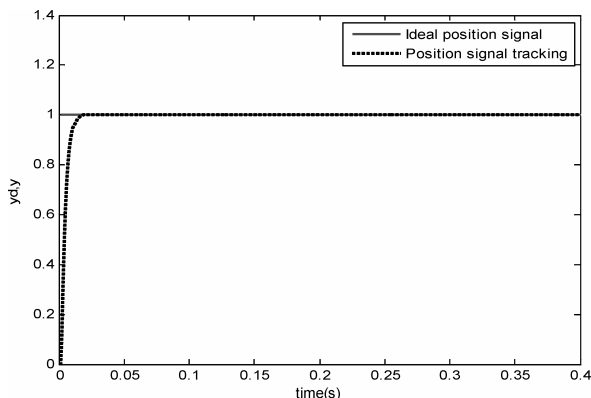
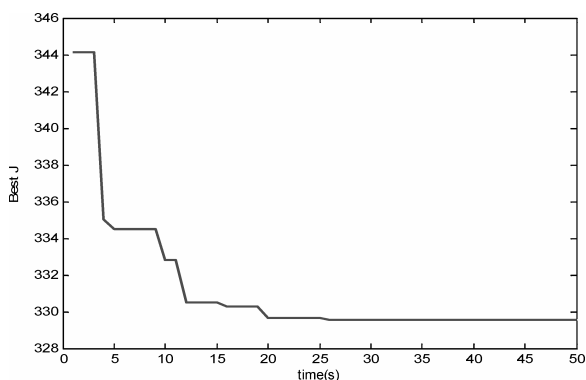
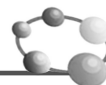


图 10-14 采用摩擦补偿的阶跃响应 ($M=2$)

图 10-15 代价函数值 J 的优化过程

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 PD 控制程序

主程序: chap10_6.m

```
%GA(Generic Algorithm) Program to predict Parameters of Friction
```

```
clear all;
```

```
close all;
```

```
global yd y 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]=chap10_6plant(kx,BsJ);
```



```
BsJi(i)=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)
%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;

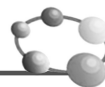
fi_S=floor(fi_Size); %Selecting Bigger fi value
r=Size-sum(fi_S);

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(i,:);
            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,yd,'r',timef,y,'k','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('Ideal position signal','Position signal tracking');
figure(2);
plot(time,BestJ,'r','linewidth',2);
xlabel('Times');ylabel('Best J');
子程序: chap10_6plant.m
function [kx,BsJ]=pid_fm_gaf(kx,BsJ)
global yd y 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;
        yd(k)=AA*sin(2*pi*fre*k*ts);
    end
    if S==2
        yd(k)=1;
    end

    y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
    error(k)=yd(k)-y(k);
    derror(k)=(error(k)-e_1)/ts;
    speed(k)=(y(k)-y_1)/ts;

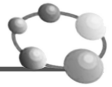
    if F==1 % Disturbance Signal: Coulomb Friction
        Ff(k)=0.8*sign(speed(k));
    end
    if F==2 % Disturbance Signal: Coulomb & Viscous Friction
        Ff(k)=sign(speed(k))*(0.30*abs(speed(k))+1.50);
    end

    %kx=[0.3,1.5]; %Idea Identification

    u(k)=50*error(k)+0.50*derror(k); %PD control
    u(k)=u(k)-Ff(k); %Practical control input

    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

    M=2;
    if M==1 %PD without Friction compensation
        uc(k)=0;
    elseif M==2 %PD with Friction compensation
```



```
uc(k)=Ffc(k);
end
u(k)=u(k)+uc(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=y(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 11 章 伺服系统 PID 控制



11.1 基于 LuGre 摩擦模型的 PID 控制

11.1.1 伺服系统的摩擦现象

在高精度、超低速伺服系统中，由于非线性摩擦环节的存在，使系统的动态及静态性能受到很大程度的影响，主要表现为低速时出现爬行现象，稳态时有较大的静差或出现极限环振荡。

摩擦现象是一种复杂的、非线性的、具有不确定性的自然现象，摩擦学的研究表明，人类目前对于摩擦的物理过程的了解还只停留在定性认识阶段，无法通过数学方法对摩擦过程给出精确描述。在现实生活中，摩擦现象几乎无处不在，在有些情况下，摩擦环节是人们所期望的，如汽车的刹车系统，但对于机械伺服系统而言，摩擦环节却成为提高系统性能的障碍，使系统出现爬行、振荡或稳态误差。为了减轻机械伺服系统中摩擦环节带来的负面影响，人们在大量的实践中总结出很多有效的方法，可概括为三类：

- (1) 改变机械伺服系统的结构设计，减少传动环节。
- (2) 选择更好的润滑剂，减小动静摩擦的差值。
- (3) 采用适当的控制补偿方法，对摩擦力（矩）进行补偿。

有关摩擦建模及动态补偿控制技术方面的研究具有近百年的历史，但由于当时控制理论和摩擦学发展水平的限制，使得这方面的研究一直进展不大，进入 20 世纪 80 年代以后，这一领域的研究渐渐活跃，许多先进的摩擦模型和补偿方法被相继提出，其中许多补偿技术已经在机械伺服系统的控制设计中得到了成功的应用。

在伺服系统辨识中，选择一个合适的摩擦模型是非常重要的，实践表明，采用简单的库仑摩擦加黏性摩擦作为摩擦模型，其效果并不理想。目前，已提出的摩擦模型很多，主要有 Karnopp 模型、LuGre 模型及综合模型。其中，LuGre 模型是 Canudas 等在 1995 年提出的典型伺服系统的摩擦模型^[46]，该模型能够准确地描述摩擦过程的复杂的动态、静态特性，如爬行 (stick slip)、极限环振荡 (hunting)、滑前变形 (presliding displacement)、摩擦记忆 (friction memory)、变静摩擦 (rising static friction) 及静态 Stribeck 曲线。

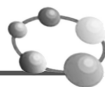
11.1.2 伺服系统的 LuGre 摩擦模型

LuGre 摩擦模型可描述如下。

对于伺服系统，用下面的微分方程表示

$$J\ddot{\theta} = u - F \quad (11.1)$$

式中， J 为转动惯量， θ 为转角， u 为控制力矩， F 为摩擦力矩。设状态变量 z 代表接触面



鬃毛的平均变形，则 F 可由下面的 LuGre 模型来描述^[46]

$$F = \sigma_0 z + \sigma_1 \dot{z} + \alpha \dot{\theta} \quad (11.2)$$

$$\dot{z} = \dot{\theta} - \frac{\sigma_0 |\dot{\theta}|}{g(\dot{\theta})} z \quad (11.3)$$

$$g(\dot{\theta}) = F_c + (F_s - F_c) e^{-\left(\frac{\dot{\theta}}{V_s}\right)^2} + \alpha \dot{\theta} \quad (11.4)$$

在式 (11.2) 至式 (11.4) 中， σ_0 、 σ_1 为动态摩擦参数， F_c 、 F_s 、 α 、 V_s 为静态摩擦参数，其中 F_c 为库仑摩擦， F_s 为静摩擦， α 为黏性摩擦系数， V_s 为切换速度。

11.1.3 仿真实例

在伺服系统 (11.1) 及摩擦模型 (11.2)~(11.4) 中，取 $J=1.0$ ， $\sigma_0=260$ ， $\sigma_1=2.5$ ， $\alpha=0.02$ ， $F_c=0.28$ ， $F_s=0.34$ ， $V_s=0.01$ 。取指令信号为正弦信号 $y_d=0.1\sin(2\pi t)$ 。

采用 Simulink 实现控制算法及带有摩擦模型的被控对象的描述。采用 PD 控制，取 $k_p=20$ ； $k_d=5$ 。如图 11-1 及图 11-2 所示为位置和速度跟踪的仿真结果。在速度过零点时，波形发生畸变，出现位置跟踪“平顶”现象和速度跟踪“死区”现象。

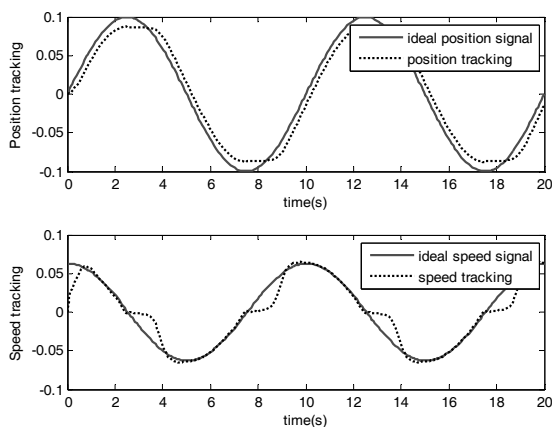


图 11-1 PD 控制的位置和速度跟踪

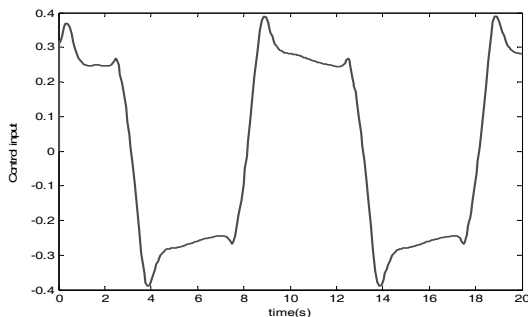


图 11-2 PD 的控制输入

仿真程序

Simulink 主程序: chap11_1sim.mdl (见图 11-3)

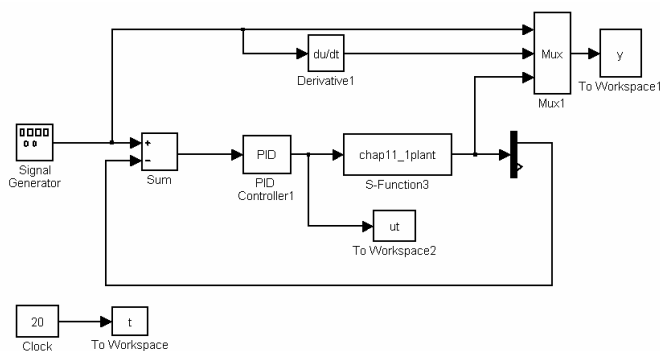
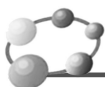
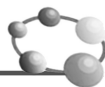


图 11-3 PD 控制 Simulink 主程序

被控对象子程序: chap11_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9}
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0;0;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

sigma0=260;sigma1=2.5;sigma2=0.02;
Fc=0.28;Fs=0.34;
Vs=0.01;
J=1.0;
g=Fc+(Fs-Fc)*exp(-(x(2)/Vs)^2)+sigma2*x(2);
```



```

F=sigma0*x(3)+sigma1*x(3)+sigma2*x(2);

sys(1)=x(2);
sys(2)=1/J*(ut-F);
sys(3)=x(2)-(sigma0*abs(x(2))/g)*x(3);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
作图子程序: chap11_1plot.m
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed signal','speed tracking');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```



11.2 基于 Stribeck 摩擦模型的 PID 控制

11.2.1 Stribeck 摩擦模型描述

Stribeck 曲线是比较著名的摩擦模型^[47]。如图 11-4 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。

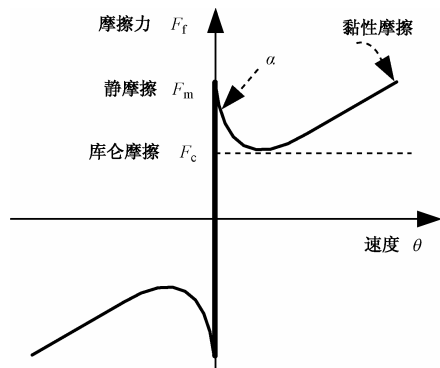


图 11-4 摩擦—速度稳态关系曲线 (Stribeck 曲线)



Stribeck 摩擦模型可表示为^[47]

当 $|\dot{\theta}(t)| < \alpha$ 时, 静摩擦为

$$F_f(t) = \begin{cases} F_m & F(t) > F_m \\ F(t) & -F_m < F < F_m \\ -F_m & F(t) < -F_m \end{cases} \quad (11.5)$$

当 $|\dot{\theta}(t)| > \alpha$ 时, 动摩擦为

$$F_f(t) = \left(F_c + (F_m - F_c) e^{-\alpha_1 |\dot{\theta}(t)|} \right) \text{sgn}(\dot{\theta}(t)) + k_v \dot{\theta} \quad (11.6)$$

式中, $F_f(t)$ 为驱动力, F_m 为最大静摩擦力, F_c 为库仑摩擦力, k_v 为黏性摩擦力矩比例系数, $\dot{\theta}(t)$ 为转动角速度, α_1 为非常小的、正的常数。

11.2.2 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。任意框的伺服结构如图 11-5 所示, 该系统采用直流电动机, 忽略电枢电感, 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 为电动机转矩系数, 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, 即驱动力 $F(t)$ 。

伺服系统的动力学方程为:

$$\ddot{\theta} = -\frac{K_m C_e}{JR} \dot{\theta} + K_u \frac{K_m}{JR} u(t) - \frac{1}{J} F_f(t) \quad (11.7)$$

转换为状态方程可描述如下

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{K_m C_e}{JR} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ K_u \frac{K_m}{JR} \end{bmatrix} u(t) - \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} F_f(t) \quad (11.8)$$

式中, $x_1(t) = \theta(t)$ 为转角, $x_2(t) = \dot{\theta}(t)$ 为转速。

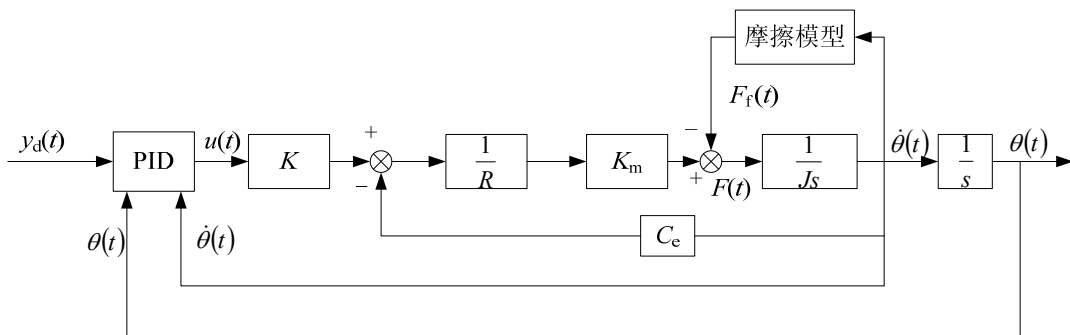
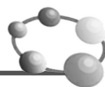


图 11-5 转台伺服系统结构



11.2.3 仿真实例

设某伺服系统参数如下： $R = 7.77\Omega$ ， $K_m = 6\text{N}\cdot\text{m}/\text{A}$ ， $C_e = 1.2\text{V}/(\text{rad}/\text{s})$ ， $J = 0.6\text{Kg}\cdot\text{m}^2$ ， $K_u = 11\text{V}/\text{V}$ 。摩擦模型参数取 $F_c = 15\text{N}\cdot\text{m}$ ， $F_m = 20\text{N}\cdot\text{m}$ ， $a_1 = 1.0$ ， $k_v = 2.0\text{Nms}/\text{rad}$ ， $\alpha = 0.01$ 。

采用连续系统仿真，运行 Simulink 主程序 chap11_3sim.mdl，正弦跟踪信号指令为 $y_d(t) = 0.10\sin(2\pi t)$ ，采用 PD 控制 $u(t) = 200e(t) + 40\dot{e}(t)$ ，带有摩擦环节的 PID 控制仿真结果如图 11-6 至图 11-7 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。

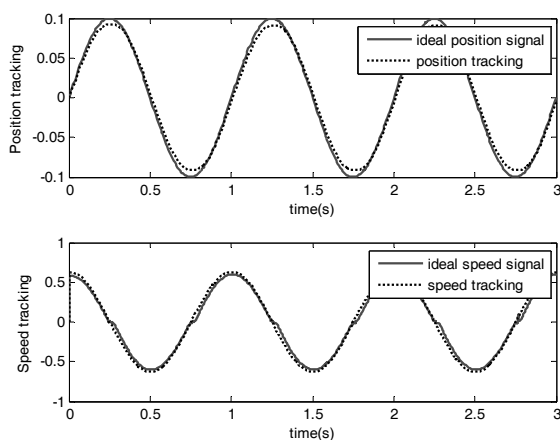


图 11-6 带摩擦时的位置和速度跟踪

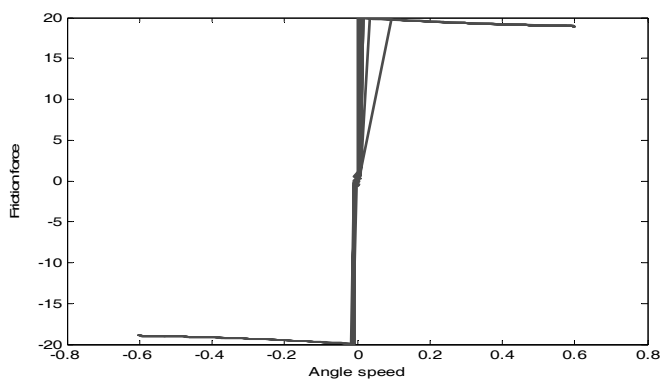


图 11-7 摩擦力 $F_f(t)$ 的变化

仿真程序

(1) M 语言仿真

主程序: chap11_2.m

```
%PID Control with Stribeck Friction Model
clear all;
close all;
global w A alfa J Ce R Km Ku S a1 Fm Fc M kv

%Servo system Parameters
```



```
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;

A=0.10;
w=1*2*pi;

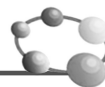
alfa=0.01;

T=1.0;
ts=0.001; %Sampling time
TimeSet=[0:ts:T];

M=1; %If M=0, No Friction works
S=1;
[t,x]=ode45('chap11_2plant',TimeSet,[0,0,0]);
x1=x(:,1);
x2=x(:,2);
x3=x(:,3);

if S==1
    yd=A*sin(w*t);
    dyd=A*w*cos(w*t);
    ddyd=-A*w*w*sin(w*t);
    error=yd-x(:,1);
    derror=dyd-x(:,2);
end
if S==2
    for kk=1:1:T/ts+1
        yd(kk)=1;
        dyd(kk)=0;
        ddyd(kk)=0;
        error(kk)=yd(kk)-x1(kk);
        derror(kk)=dyd(kk)-x2(kk);
    end
end
if S==3
    for kk=1:1:T/ts+1
        yd=A*sign(sin(0.4*2*pi*t));
        dyd(kk)=0;
        ddyd(kk)=0;
        error(kk)=yd(kk)-x1(kk);
        derror(kk)=dyd(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))+kv*x2(kk);
elseif x2(kk)<-alfa
    Ff(kk)=-Fc-(Fm-Fc)*exp(a1*x2(kk))+kv*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
end
figure(1);
plot(t,yd,'r',t,x(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('ideal position signal','position tracking');
figure(2);
plot(t,dyd,'r',t,x(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('speed tracking');
legend('ideal speed signal','speed tracking');
figure(3);
plot(t,error,'r','linewidth',2);
xlabel('time(s)');ylabel('error');
figure(4);
plot(x(:,2),Ff,'r','linewidth',2);
xlabel('speed');ylabel('Friction');
figure(5);
plot(t,Ff,'r','linewidth',2);
xlabel('time(s)');ylabel('Friction');
figure(6);
plot(time,u,'r','linewidth',2);

```



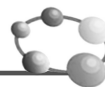

```
xlabel('time(s)');ylabel('Control input');
被控对象程序: chap11_2plant.m
function dx=Model(t,x)
global w A alfa J Ce R Km Ku S a1 Fm Fc M kv
persistent aa
dx=zeros(3,1);

a1=1.0; %Effect on the shape of friction curve
Fm=20;
Fc=15;
kv=2.0;

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(-a1*x(2))+kv*x(2);
    dFf=(Fm-Fc)*exp(-a1*x(2))*(-a1)*x(3)+kv*x(3);
elseif x(2)<=-alfa
    Ff=-Fc-(Fm-Fc)*exp(a1*x(2))+kv*x(2);
    dFf=-(Fm-Fc)*exp(a1*x(2))*a1*x(3)+kv*x(3);
end

if S==1
    yd=A*sin(w*t);
    dyd=A*w*cos(w*t);
    ddyd=-A*w*w*sin(w*t);
    dddy=-A*w*w*w*cos(w*t);
end
if S==2
    yd=1;
    dyd=0;
    ddyd=0;
    dddy=0;
```



```

end
if S==3
    yd=A*sign(sin(0.4*2*pi*t));
    dyd=0;
    ddyd=0;
    dddyd=0;
end
error=yd-x(1);
derror=dyd-x(2);
dderror=ddyd-x(3);

u=200*error+40*derror;    %PID
du=200*derror+40*dderror;

if u>=110
    u=110;
end
if u<=-110
    u=-110;
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);

```

(2) Simulink 仿真

Simulink 主程序: chap11_3sim.mdl (见图 11-8)

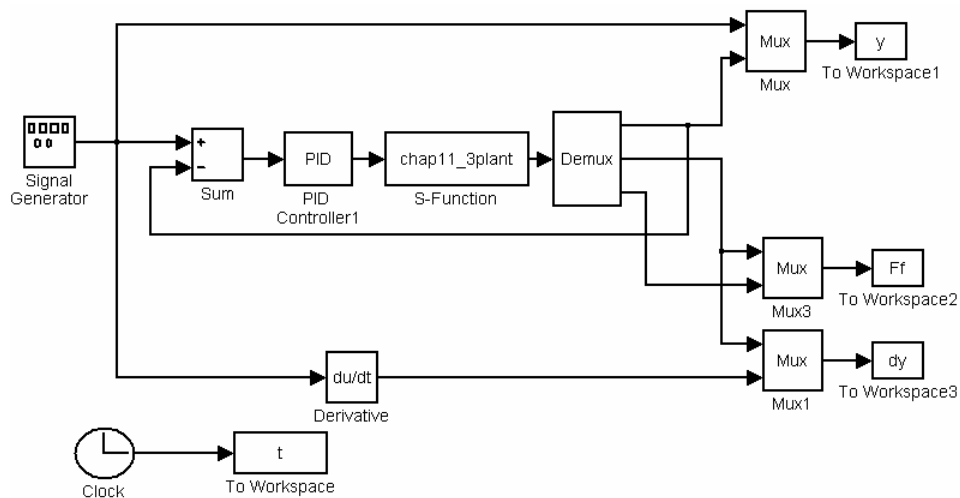


图 11-8 PID 控制主程序



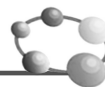
被控对象 S 函数子程序: chap11_3plant.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0;0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
%Servo system Parameters
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;
kv=2.0;

alfa=0.01;
a1=1.0; %Effect on the shape of friction curve
Fm=20;
Fc=15;
kv=2.0;
ut=u(1);
F=ut;
if abs(x(2))<=alfa
    if F>Fm
        Ff=Fm;
    elseif F<-Fm
```



```

        Ff=-Fm;
    else
        Ff=F;
    end
end
if x(2)>alfa
    Ff=Fc+(Fm-Fc)*exp(-a1*x(2))+kv*x(2);
elseif x(2)<=-alfa
    Ff=-Fc-(Fm-Fc)*exp(a1*x(2))+kv*x(2);
end

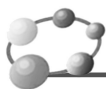
sys(1)=x(2);
sys(2)=-Km*Ce/(J*R)*x(2)+Ku*Km*ut/(J*R)-Ff/J;
function sys=mdlOutputs(t,x,u)

%Servo system Parameters
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;
kv=2.0;

alfa=0.01;
a1=1.0; %Effect on the shape of friction curve
Fm=20;
Fc=15;
kv=2.0;
ut=u(1);
F=ut;
if abs(x(2))<=alfa
    if F>Fm
        Ff=Fm;
    elseif F<-Fm
        Ff=-Fm;
    else
        Ff=F;
    end
end
if x(2)>alfa
    Ff=Fc+(Fm-Fc)*exp(-a1*x(2))+kv*x(2);
elseif x(2)<=-alfa
    Ff=-Fc-(Fm-Fc)*exp(a1*x(2))+kv*x(2);
end

sys(1)=x(1); %Angle
sys(2)=x(2); %Angle speed
sys(3)=Ff; %Friction force
作图程序: chap11_3plot.mdl
close all;
figure(1);

```



```
subplot(211);  
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('Position tracking');  
legend('ideal position signal','position tracking');  
subplot(212);  
plot(t,dy(:,1),'r',t,dy(:,2),'k','linewidth',2);  
xlabel('time(s)');ylabel('Speed tracking');  
legend('ideal speed signal','speed tracking');  
  
figure(2);  
plot(Ff(:,1),Ff(:,2),'r','linewidth',2);  
xlabel('Angle speed');ylabel('Friction force');
```

11.3 伺服系统三环的 PID 控制

11.3.1 伺服系统三环的 PID 控制原理

现代数控机床伺服系统常采用全闭环或半闭环控制系统，而且是三环控制，由里向外分别是电流环、速度环、位置环^[48,49]。以转台伺服系统为例，其控制结构如图 11-9 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，电动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。

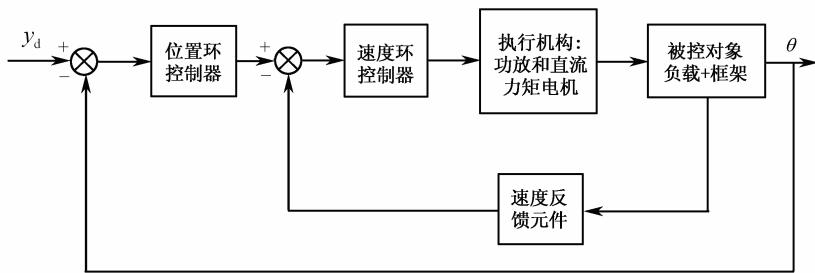


图 11-9 转台伺服系统框图

转台伺服系统单框的位置环、速度环和电流环框图如图 11-10、图 11-11 和图 11-12 所示。

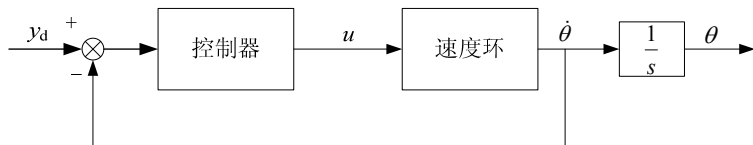


图 11-10 伺服系统位置环框图

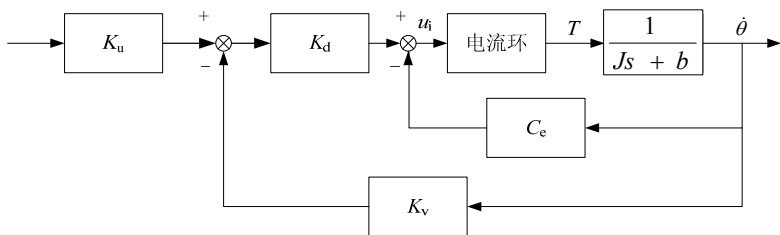
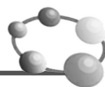


图 11-11 伺服系统速度环框图

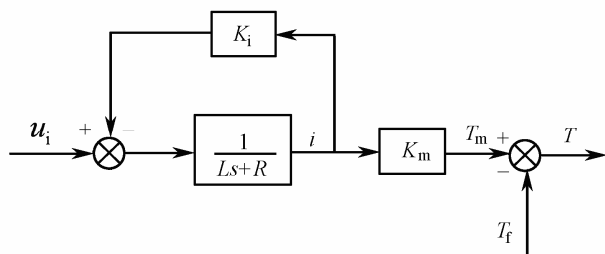


图 11-12 伺服系统电流环框图

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 为电动机转矩系数； C_e 为电动机反电动势系数； J 为等效到转轴上的转动惯量； b 为黏性阻尼系数，其中 $J = J_m + J_L$ ， $b = b_m + b_L$ ， J_m 和 J_L 分别为电动机和负载的转动惯量， b_m 和 b_L 分别为电动机和负载的黏性阻尼系数； 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黏性摩擦模型

$$F_f(t) = F_c \cdot \text{sgn}(\dot{\theta}) + b_c \cdot \dot{\theta} \quad (11.9)$$

控制器采用 PID 控制加前馈控制的形式，加入前馈摩擦补偿控制表示为

$$u_f(t) = F_{c1} \cdot \text{sgn}(\dot{\theta}) + b_{c1} \cdot \dot{\theta} \quad (11.10)$$

式中， F_{c1} 和 b_{c1} 为黏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。

11.3.2 仿真实例

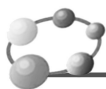
被控对象为一个具有三环结构的伺服系统，如图 11-10、图 11-11 和图 11-12 所示。伺服系统参数和控制参数在程序中给予了描述，系统采样时间为 1ms。取 $M = 2$ ，此时输入指令为正弦迭加信号： $r(t) = A \sin(2\pi Ft) + 0.5A \sin(1.0\pi Ft) + 0.25A \sin(0.5\pi Ft)$ ，其中 $A = 0.50$ ， $F = 0.50$ 。

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 $k_{pp}=15$, $k_{ii}=0.10$, $k_{dd}=1.5$ 。

根据是否加入前馈补偿分别进行仿真。无前馈补偿时正弦跟踪如图 11-13 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。有前馈补偿时的正弦跟踪如图 11-14 所示, 采用 PID 控制加前馈控制可以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。

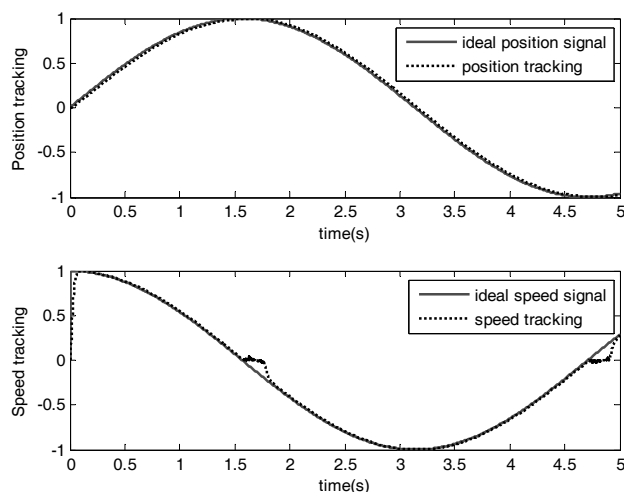


图 11-13 无前馈补偿时位置和速度跟踪

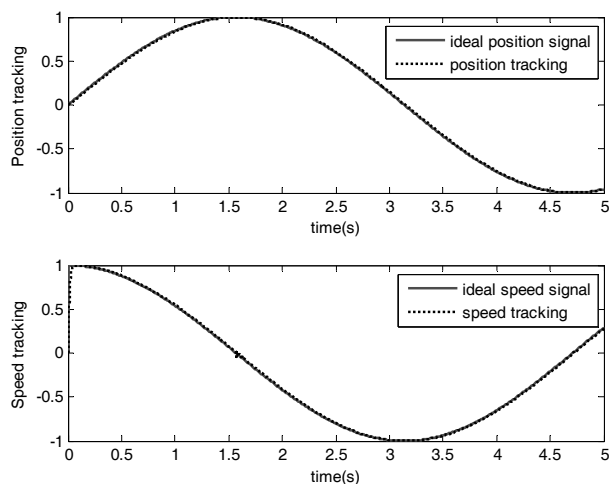
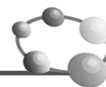


图 11-14 有前馈补偿时位置和速度跟踪

仿真程序

初始化程序: chap11_4int.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
%Equivalent gain from feedforward to practical friction
Gain=ku*kd*1/R*km*1.0;
Fc1=Fc/Gain;    bc1=bc/Gain; %Feedforward compensat

```

控制系统的 Simulink 程序: chap11_4sim.mdl (见图 11-15 和图 11-16)

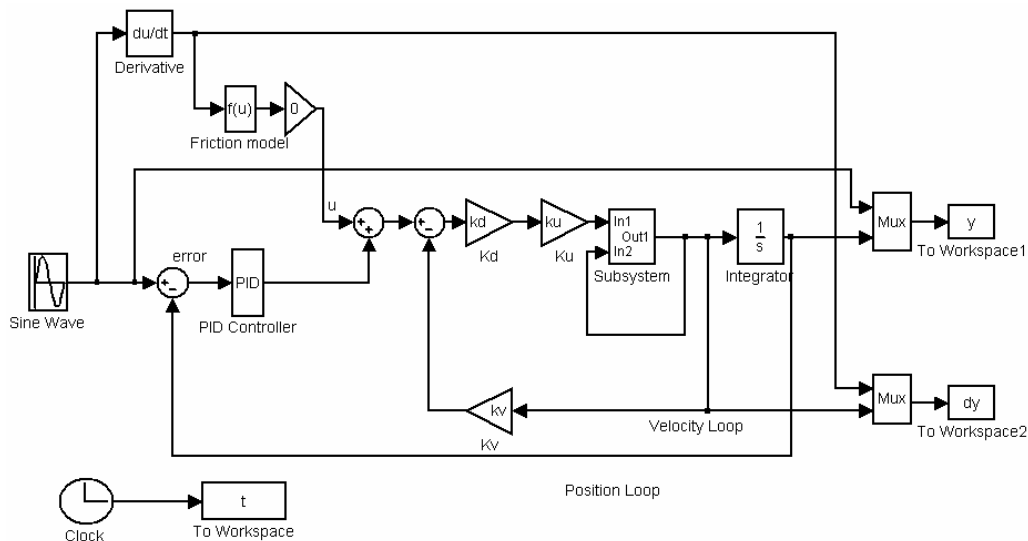


图 11-15 伺服系统位置环模块

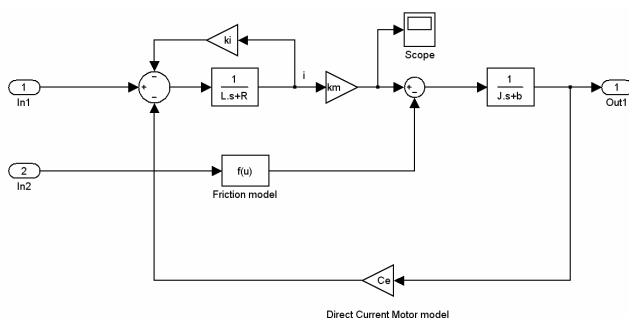
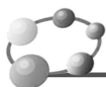


图 11-16 伺服系统速度环和电流环模块

作图程序: chap11_4plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,dy(:,1),'r',t,dy(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed signal','speed tracking');
```



11.4 二质量伺服系统的 PID 控制

11.4.1 二质量伺服系统的 PID 控制原理

如果伺服系统把电动机与负载作为一个刚体来考虑,则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,尽管电动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。在电动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如图 11-17 所示由电动机、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 11-17 可得传动轴动力学方程,根据伺服系统电动机框图 11-18 可得电动机电力学方程。根据伺服系统负载框图 11-19 (不考虑干扰时)可得负载动力学方程。

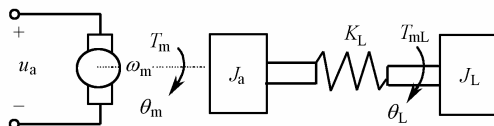


图 11-17 电动机—传动轴—负载模型

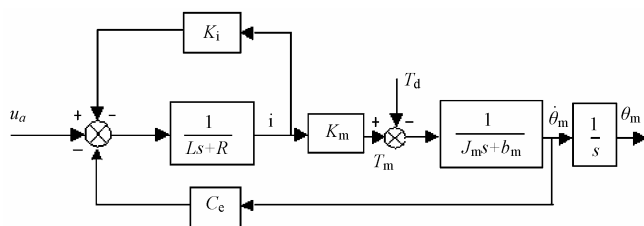
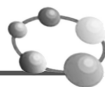


图 11-18 伺服系统电动机框图

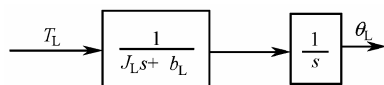


图 11-19 负载框图

三质量伺服系统的电学方程和动力学方程^[48,49]

$$\text{电动机} \quad iR + Li = u_a - C_e \dot{\theta}_m - K_i i \quad (11.11)$$

$$T_m = iK_m \quad (11.12)$$

$$J_m \ddot{\theta}_m = T_m - b_m \dot{\theta}_m - K_L (\theta_m - \theta_L) \quad (11.13)$$

$$\text{传动轴} \quad J_a (\ddot{\theta}_m - \ddot{\theta}_L) = K_L (\theta_m - \theta_L) - T_{mL} \quad (11.14)$$

$$\text{负载} \quad J_L \ddot{\theta}_L = T_{mL} - b_L \dot{\theta}_L \quad (11.15)$$

式中, J_a 为传动轴的转动惯量, θ_m 和 θ_L 分别为电动机和负载的转角, J_m 和 J_L 分别为电动机和负载的转动惯量, b_m 和 b_L 分别为电动机和负载的黏性阻尼系数; K_L 为电动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_L 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

$$\text{电动机} \quad iR + Li = u_a - C_e \dot{\theta}_m - K_i i \quad (11.16)$$

$$T_m = iK_m \quad (11.17)$$

$$J_m \ddot{\theta}_m = T_m - b_m \dot{\theta}_m - K_L (\theta_m - \theta_L) \quad (11.18)$$

$$\text{负载} \quad J_L \ddot{\theta}_L = T_{mL} - b_L \dot{\theta}_L \quad (11.19)$$

$$K_L (\theta_m - \theta_L) - T_{mL} = 0 \quad (11.20)$$

根据上述描述, 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 11-20 所示。

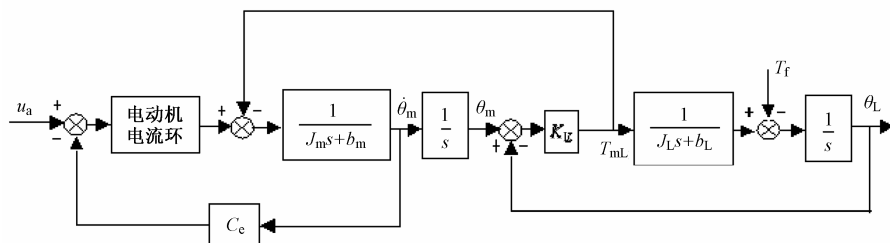


图 11-20 二质量伺服系统部分结构框图



11.4.2 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。输入指令为正弦信号： $y_d = \sin 2\pi t$ ，假设在速度环中的外加干扰 T_f 为黏性摩擦模型： $F_f(t) = F_c \cdot \text{sgn}(\dot{\theta}) + b_c \cdot \dot{\theta}$ ，控制器采用 PID 控制，参数选为 $k_{pp} = 8.0$ ， $k_{ii} = 1.0$ ， $k_{dd} = 5.0$ 。根据是否加入摩擦干扰分别进行仿真。无摩擦时，正弦位置跟踪和速度跟踪的结果如图 11-21 所示。带摩擦时，正弦位置跟踪和速度跟踪的结果如图 11-22 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。

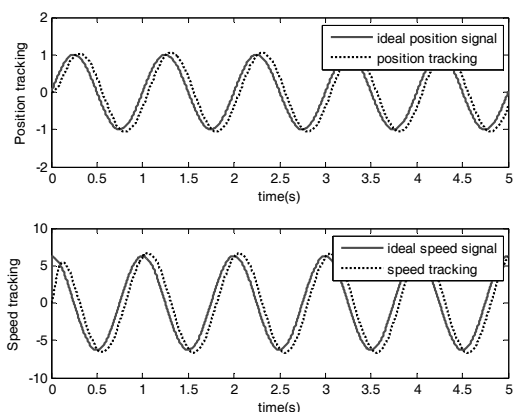


图 11-21 无摩擦时正弦位置和速度跟踪

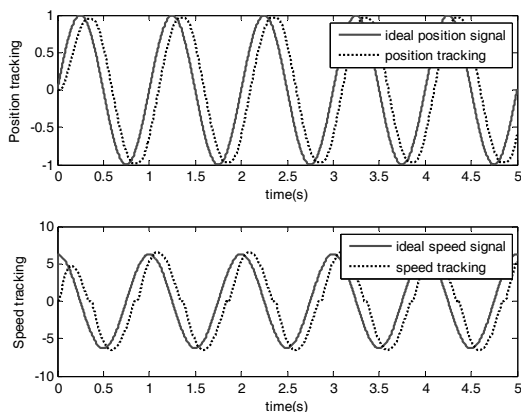


图 11-22 带摩擦时正弦位置跟踪

仿真程序

初始化程序：chap11_5int.m

```
%Three Loop of Flight Simulator Servo System with two-mass of Direct Current Motor
clear all;
close all;

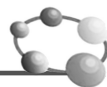
%(1)Current loop
L=0.001;      %L<<1,Inductance of motor armature
R=1.0;        %Resistance of motor armature
ki=0.001;     %Current feedback coefficient

%(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=100;
kii=1.0;
kdd=50;

```

控制系统的 Simulink 程序: chap11_5sim.mdl (见图 11-23、图 11-24 和图 11-25)

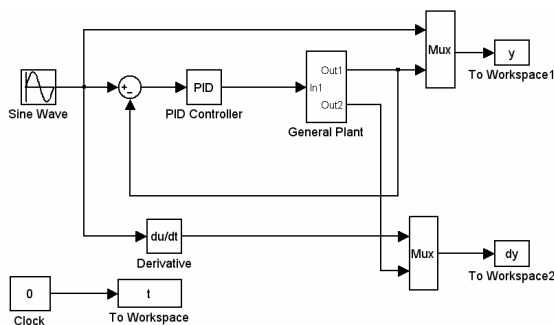


图 11-23 闭环 PID 控制 Simulink 主模型

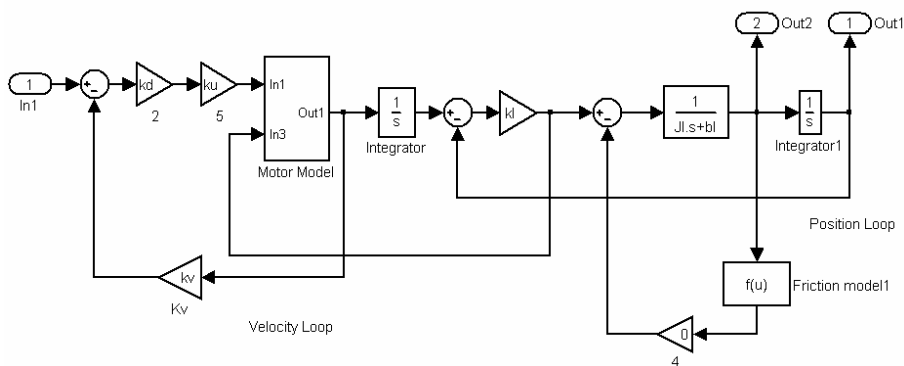


图 11-24 二质量伺服系统 Simulink 模型

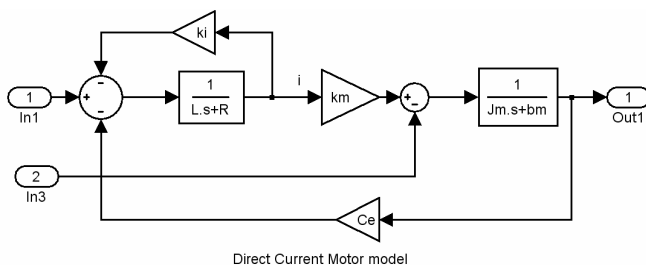


图 11-25 电动机 Simulink 模型



作图程序: chap11_5plot.m

```
close all;

figure(1);
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,dy(:,1),'r',t,dy(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed signal','speed tracking');
```



11.5 伺服系统的模拟 PD+数字前馈控制

11.5.1 伺服系统的模拟 PD+数字前馈控制原理

针对如图 11-9 所示的三环伺服系统, 设电流环为开环, 忽略电动机反电动系数和电枢电感, 将电阻 R 和 k_m 等效到速度环放大系数 K_d 上。即近似取 $K_i \approx 0, z \approx 0, C_e \approx 0$ 。简化后的三环伺服系统结构框图如图 11-26 所示, 其中 u 为控制输入, y_d 为位置指令。

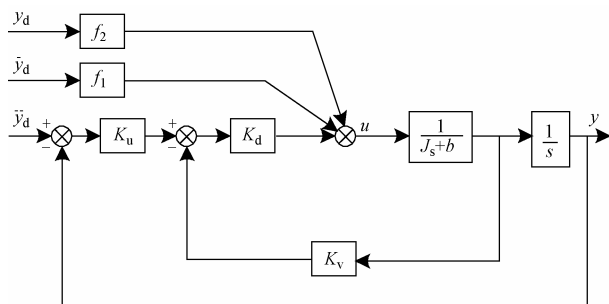


图 11-26 简化后的伺服系统框图

采用 PD+前馈控制方式, 设计的控制律如下:

$$u = k_d [k_p (y_d - \theta) - k_v \dot{\theta}] + f_1 \dot{y}_d + f_2 \ddot{y}_d = k_1 e - k_2 \dot{\theta} + f_1 \dot{y}_d + f_2 \ddot{y}_d \quad (11.21)$$

式中, $k_1 = k_d k_p$, $k_2 = k_d k_v$, $e = y_d - \theta$, f_1 和 f_2 为前馈系数。

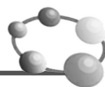
由图 11-26 可知

$$\frac{1}{Js^2 + bs} = \frac{\theta}{u}$$

即

$$J\ddot{\theta} + b\dot{\theta} = u$$

将控制律 u 代入上式, 得



$$f_1 \dot{y}_d + f_2 \ddot{y}_d - J \ddot{\theta} - (k_2 + b) \dot{\theta} + k_1 e = 0$$

取

$$f_1 = k_2 + b, \quad f_2 = J$$

得到系统的误差状态方程如下

$$J \ddot{e} + (k_2 + b) \dot{e} + k_1 e = 0$$

由于 $J > 0, k_2 + b > 0, k_1 > 0$ ，则根据代数稳定性判据，针对二阶系统而言，当系统闭环特征方程式的系数都大于零时，系统稳定，系统的跟踪误差 $e(t)$ 和 $\dot{e}(t)$ 收敛于零。

11.5.2 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，输入指令为正弦迭加信号： $y_d(t) = \sin t$ ， $u(t)$ 为控制器的输出，伺服系统参数为 $J = 2.0$ ， $b = 0.50$ ， $k_v = 2.0$ ， $k_p = 15$ ， $k_d = 6$ ，则 $f_1 = k_2 + b$ ， $f_2 = J$ 。仿真结果如图 11-27 所示。

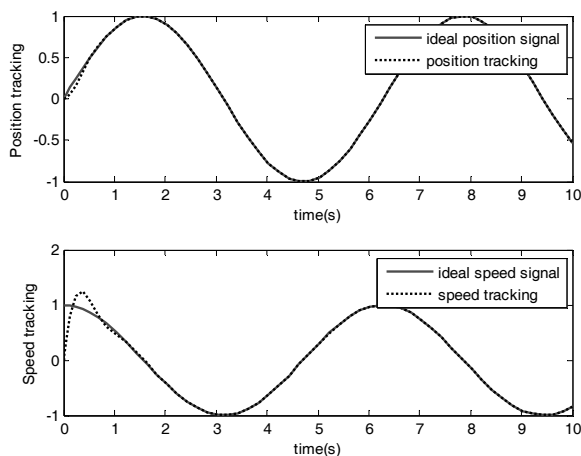


图 11-27 位置跟踪和速度跟踪

仿真程序

初始化程序：chap11_6int.m

```
%Flight Simulator Servo System
clear all;
close all;

J=2;
b=0.5;

kv=2;
kp=15;
kd=6;

f1=(b+kd*kv);
f2=J;
```



Simulink 主程序: chap11_6sim.mdl (见图 11-28)

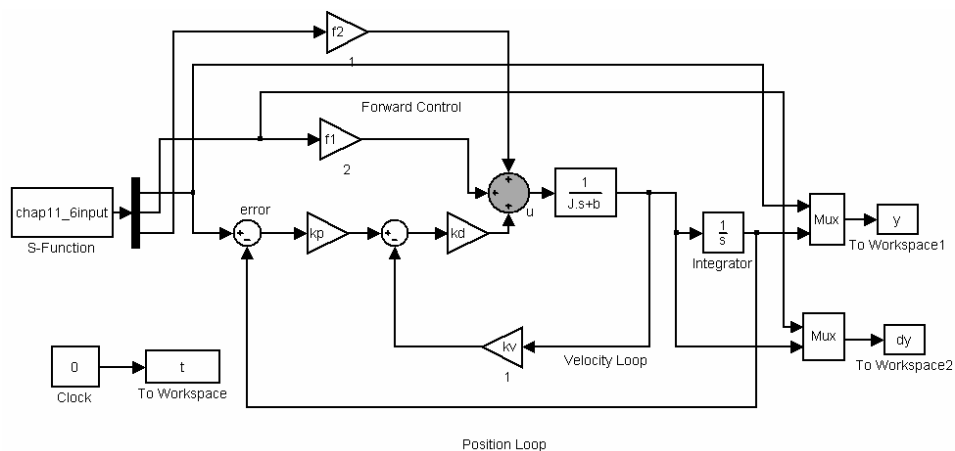


图 11-28 基于反馈前馈的控制主程序

作图程序: chap11_6plot.m

```
close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,dy(:,1),'r',t,dy(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed signal','speed tracking');
```

第 12 章 迭代学习 PID 控制

12.1 迭代学习控制方法介绍

迭代学习控制是通过迭代修正达到某种控制目标的改善，它的算法较为简单，能在给定的时间范围内实现未知对象实际运行轨迹以高精度跟踪给定期望轨迹，且不依赖系统的精确数学模型。因而一经推出，就在机器人控制领域得到了广泛的运用。

迭代学习控制 (Iterative Learning Control, ILC) 是智能控制中具有严格数学描述的一个分支。1984 年, Arimoto^[50]等人提出了迭代学习控制的概念, 该控制方法适合于具有重复运动性质的被控对象, 它不依赖于系统的精确数学模型, 能以非常简单的方式处理不确定度相当高的非线性强耦合动态系统。目前, 迭代学习控制在学习算法、收敛性、鲁棒性、学习速度及工程应用研究上取得了巨大的进展。

近年来, 迭代学习控制理论和应用在国外发展很快, 取得了许多成果。在国内, 迭代学习控制理论也得到广泛的重视, 有许多重要著作出版^[51,52]。

12.2 迭代学习控制基本原理

设被控对象的动态过程为

$$\dot{x}(t) = f(x(t), u(t), t), \quad y(t) = g(x(t), u(t), t) \quad (12.1)$$

式中, $x \in R^n$ 、 $y \in R^m$ 、 $u \in R^r$ 分别为系统的状态、输出和输入变量, $f(\cdot)$ 、 $g(\cdot)$ 为适当维数的向量函数, 其结构与参数均未知。若期望控制 $u_d(t)$ 存在, 则迭代学习控制的目标为: 给定期望输出 $y_d(t)$ 和每次运行的初始状态 $x_k(0)$, 要求在给定的时间 $t \in [0, T]$ 内, 按照一定的学习控制算法通过多次重复的运行, 使控制输入 $u_k(t) \rightarrow u_d(t)$, 而系统输出 $y_k(t) \rightarrow y_d(t)$ 。第 k 次运行时, 式 12.1 表示为

$$\dot{x}_k(t) = f(x_k(t), u_k(t), t) \quad y_k(t) = g(x_k(t), u_k(t), t) \quad (12.2)$$

跟踪误差为

$$e_k(t) = y_d(t) - y_k(t) \quad (12.3)$$

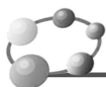
迭代学习控制可分为开环学习和闭环学习。

开环学习控制的方法是^[51]: 第 $k+1$ 次的控制等于第 k 次控制再加上第 k 次输出误差的校正项, 即

$$u_{k+1}(t) = L(u_k(t), e_k(t)) \quad (12.4)$$

闭环学习策略是^[51]: 取第 $k+1$ 次运行的误差作为学习的修正项, 即

$$u_{k+1}(t) = L(u_k(t), e_{k+1}(t)) \quad (12.5)$$



式中, L 为线性或非线性算子。

迭代学习控制的基本结构如图 12-1 所示。

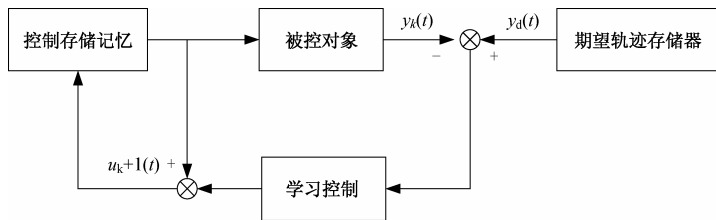


图 12-1 迭代学习控制基本结构



12.3 基本的迭代学习控制算法

Arimoto 等首先给出了线性时变连续系统的 D 型迭代学习控制律^[50]

$$u_{k+1}(t) = u_k(t) + \Gamma \dot{e}_k(t) \quad (12.6)$$

式中, Γ 为常数增益矩阵。在 D 型算法的基础上, 相继出现了 P 型、PI 型、PD 型迭代学习控制律。从一般意义来看它们都是 PID 型迭代学习控制律的特殊形式, PID 型迭代学习控制律表示为

$$u_{k+1}(t) = u_k(t) + \Gamma \dot{e}_k(t) + \Phi e_k(t) + \Psi \int_0^t e_k(\tau) d\tau \quad (12.7)$$

式中, Γ 、 Φ 、 Ψ 为学习增益矩阵。算法中的误差信息使用 $e_k(t)$ 称为开环迭代学习控制, 如果使用 $e_{k+1}(t)$ 则称为闭环迭代学习控制, 如果同时使用 $e_k(t)$ 和 $e_{k+1}(t)$ 则称为开闭环迭代学习控制。

此外, 还有高阶迭代学习控制算法、最优迭代学习控制算法、遗忘因子迭代学习控制算法和反馈—前馈迭代学习控制算法等。

学习算法的收敛性分析是迭代学习控制的核心问题, 基本的收敛性分析方法有压缩映射方法、谱半径条件法、基于 2-D 理论的分析方法和基于 Lyapunov 直接法的设计方法等。



12.4 基于 PID 型的迭代学习控制

12.4.1 系统描述

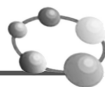
考虑电动机控制系统, 被控对象描述为

$$J\ddot{q} = \tau \quad (12.8)$$

式中, q 为角度, τ 为控制力矩。

设系统所要跟踪的期望轨迹为 $q_d(t)$, $t \in [0, T]$ 。系统第 i 次输出为 $q_i(t)$, 令 $e_i(t) = q_d(t) - q_i(t)$ 。

在学习开始时, 系统的初始状态为 $x_0(0)$ 。学习控制的任务为通过学习控制律设计 $u_{i+1}(t)$, 使第 $i+1$ 次运动误差 $e_{i+1}(t)$ 减少。



12.4.2 控制器设计

采用三种基于反馈的迭代学习控制律^[50~52]

(1) 闭环 D 型

$$u_{k+1}(t) = u_k(t) + K_d(\dot{q}_d(t) - \dot{q}_{k+1}(t)) \quad (12.9)$$

(2) 闭环 PD 型

$$u_{k+1}(t) = u_k(t) + K_p(q_d(t) - q_{k+1}(t)) + K_d(\dot{q}_d(t) - \dot{q}_{k+1}(t)) \quad (12.10)$$

(3) 指数变增益 D 型

$$u_{k+1}(t) = u_k(t) + K_p(q_d(t) - q_{k+1}(t)) + K_d(\dot{q}_d(t) - \dot{q}_{k+1}(t)) \quad (12.11)$$

上述控制算法的收敛性分析可参见相关参考文献^[50~53]。

12.4.3 仿真实例

被控对象取式 (12.8)，被控对象参数为， $J = 10 \text{ kg} \cdot \text{m}^2$ 。采用三种闭环迭代学习控制律，其中 $M=1$ 为 D 型迭代学习控制， $M=2$ 为 PD 型迭代学习控制， $M=3$ 为变增益指数 D 型迭代学习控制。

位置指令为 $\sin t$ ，为了保证被控对象初始输出与指令初值一致，取被控对象的初始状态为 $x(0) = [0, 1]^T$ 。取 PD 型迭代学习控制，即 $M=3$ ， $k_p = 30$ ， $k_d = 50$ ，运行主程序 chap12_1main.m，仿真结果如图 12-2 至图 12-4 所示。

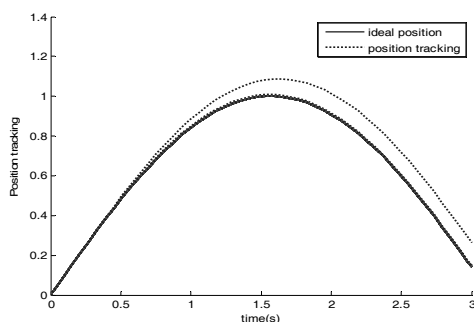


图 12-2 20 次迭代学习的跟踪过程

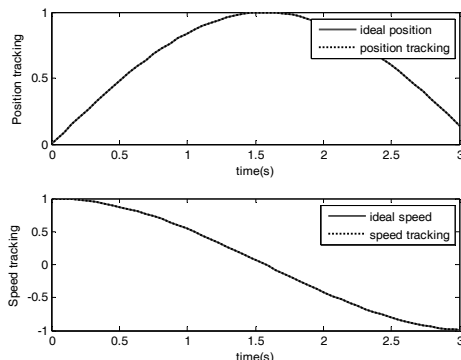


图 12-3 第 20 次迭代学习的位置跟踪

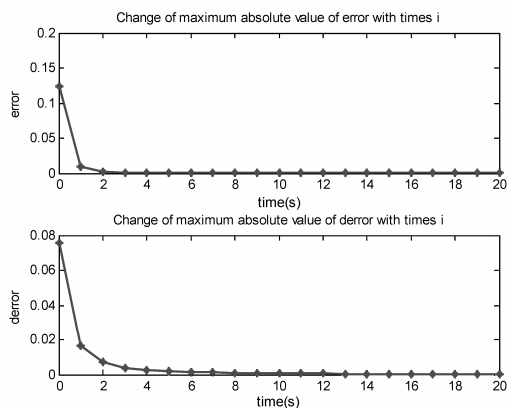


图 12-4 20 次迭代过程中误差范的收敛过程

仿真程序

主程序: chap12_1main.m

```
%PID type Learning Control
clear all;
close all;

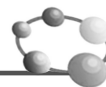
t=[0:0.01:3]';
k(1:301)=0;    %Total initial points
k=k';
T(1:301)=0;
T=T';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=0:1:20    % Start Learning Control
    i
    pause(0.01);

    sim('chap12_1sim',[0,3]);
    q1=q(:,1);
    dq1=q(:,2);
    qd1=qd(:,1);
    dqd1=qd(:,2);

    e=qd1-q1;
    de=dqd1-dq1;

    figure(1);
    hold on;
    plot(t,qd1,'r',t,q1,'b:','linewidth',2);
    xlabel('time(s)');ylabel('Position tracking');
    legend('ideal position','position tracking');

    j=i+1;
```



```

times(j)=i;
ei(j)=max(abs(e));
dei(j)=max(abs(de));
end          %End of i
%%%%%%%%%%%%%%
figure(2);
subplot(211);
plot(t,qd1,'r',t,q1,'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position','position tracking');
subplot(212);
plot(t,dqd1,'r',t,dq1,'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed','speed tracking');
figure(3);
subplot(211);
plot(times,ei,'*-r','linewidth',2);
title('Change of maximum absolute value of error with times i');
xlabel('times');ylabel('error');
subplot(212);
plot(times,dei,'*-r','linewidth',2);
title('Change of maximum absolute value of derror with times i');
xlabel('times');ylabel('derror');

```

Simulink 子程序: chap12_1sim.mdl (见图 12-5)

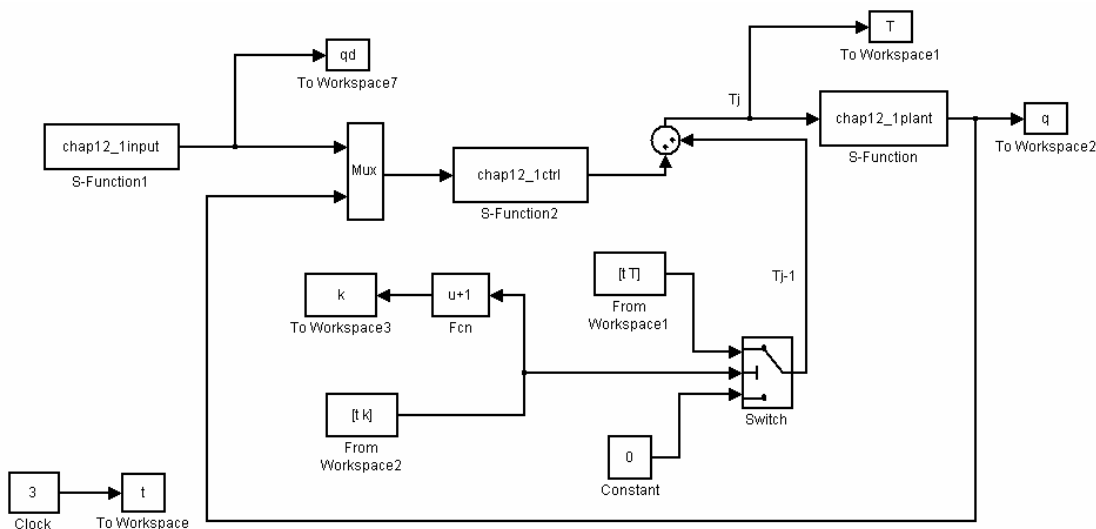


图 12-5 迭代学习控制 Simulink 子程序

被控对象子程序: chap12_1plant.m

```

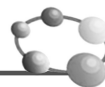
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,

```



```
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0;1];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
Tol=u(1);
J=10;

sys(1)=x(2);
sys(2)=1/J*Tol;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);    %Angle
sys(2)=x(2);    %Angle speed
控制器子程序: chap12_1ctrl.m
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
```



```

sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
qd=u(1);dq=u(2);
q=u(3);dq=u(4);

e=qd-q;
de=dqd-dq;
Kp=30;Kd=50;

M=3;
if M==1
    Tol=Kd*de;          %D Type
elseif M==2
    Tol=Kp*e+Kd*de;     %PD Type
elseif M==3
    Tol=Kd*exp(0.8*t)*de; %Exponential Gain D Type
end
sys(1)=Tol;
指令程序: chap12_1input.m
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;

```



```
sizes.NumSampleTimes = 1;  
sys = simsizes(sizes);  
x0 = [];  
str = [];  
ts = [0 0];  
function sys=mdlOutputs(t,x,u)  
qd=sin(t);  
dqd=cos(t);  
  
sys(1)=qd;  
sys(2)=dqd;
```

第 13 章 其他控制方法的设计与仿真

虽然传统 PID 控制具有算法简单、无须建模的优点，但存在参数难以整定、不能保证稳定性等缺点。在控制理论中，有许多先进的控制方法，如反演控制、滑模控制、自适应控制、鲁棒控制、H 无穷控制等，这些方法都从不同的角度来保证控制系统的稳定性。下面通过实例简要加以介绍。

13.1 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。如图 13-1 所示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 。

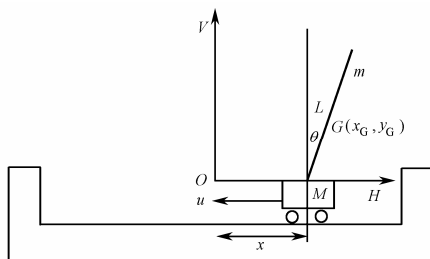


图 13-1 一级倒立摆系统示意图

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

$$x_G = x + l \sin \theta$$

$$y_G = l \cos \theta$$

则根据牛顿力学定律，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， V 为小车对摆杆的垂直分力， H 为小车对摆杆的水平分力。

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, $\sin \theta \approx \theta$, $\cos \theta \approx 1$, 则以上各式变为

$$I \ddot{\theta} = Vl\theta - Hl \quad (13.1)$$

$$m(\ddot{x} + l\ddot{\theta}) = H \quad (13.2)$$

$$0 = V - mg \quad (13.3)$$

$$M\ddot{x} = u - H \quad (13.4)$$

由式 (13-2) 和式 (13-4) 得

$$(M + m)\ddot{x} + ml\ddot{\theta} = u \quad (13.5)$$

由式 (13-1) 和式 (13-3) 得

$$(I + ml^2)\ddot{\theta} + ml\ddot{x} = mgl\theta \quad (13.6)$$

由式 (13-5) 和式 (13-6) 可得单级倒立摆方程

$$\ddot{\theta} = \frac{m(m+M)gl}{(M+m)I + Mml^2} \theta - \frac{ml}{(M+m)I + Mml^2} u \quad (13.7)$$

$$\ddot{x} = -\frac{m^2 gl^2}{(M+m)I + Mml^2} \theta + \frac{I + ml^2}{(M+m)I + Mml^2} u \quad (13.8)$$

式中, $I = \frac{1}{12}mL^2$, $l = \frac{1}{2}L$ 。



13.2 倒立摆 PD 控制

13.2.1 系统描述

假设不考虑小车的控制问题, 被控对象取单级倒立摆, 取 $x_1 = \theta$, $x_2 = \dot{\theta}$, 则动态方程如下

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = f(x) + g(x)u$$

式中, $f(x) = \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, $g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, x_1 和 x_2

分别为摆角和摆速, $g = 9.8\text{m/s}^2$, m_c 为小车质量, $m_c = 1\text{kg}$, m 为摆杆质量, $m = 0.1\text{kg}$, l 为摆长的一半, $l = 0.5\text{m}$, u 为控制输入。

13.2.2 仿真实例

位置指令为 $x_d(t) = 0.1\sin(t)$, 令 $e = x_d - x_1$, 将控制律设计为 $u = k_p e + k_d \dot{e}$, 倒立摆初始状态为 $[\pi/60, 0]$, 仿真结果如图 13-2 所示。

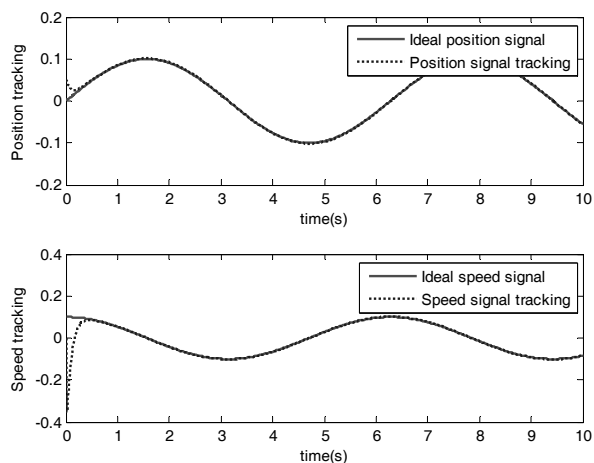
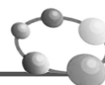


图 13-2 位置和速度跟踪

仿真程序

(1) Simulink 主程序: chap13_1sim.mdl (见图 13-3)

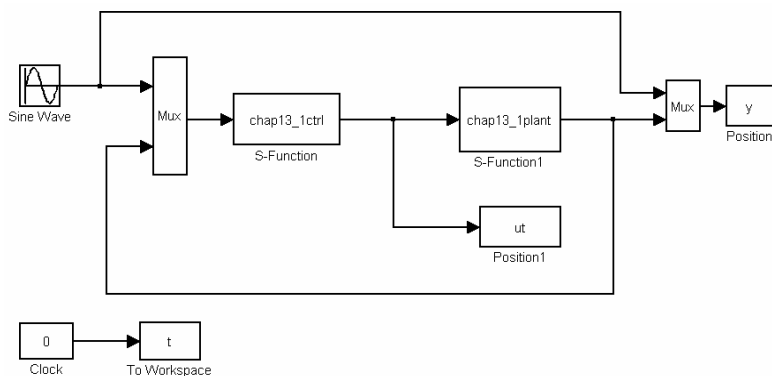


图 13-3 倒立摆 PD 控制主程序

(2) 控制器 S 函数: chap13_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
```



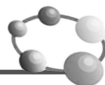
```
sizes.NumOutputs      = 1;
sizes.NumInputs       = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
xd=0.1*sin(t);
dxd=0.1*cos(t);
x1=u(2);
x2=u(3);

e=xd-x1;
de=dxd-x2;

%PD Control
kp=1000;kd=100;
ut=kp*e+kd*de;
sys(1)=ut;
```

(3) 被控对象 S 函数: chap13_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
```



```

str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);

```

(4) 作图程序: chap13_1plot.m

```

close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'l','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position signal','Position signal tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,3),'l','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed signal','Speed signal tracking');
figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```



13.3 单级倒立摆的全状态反馈控制

LQR 控制和极点配置控制是典型的两种全状态反馈控制方法。LQR (Linear Quadratic Regulator) 即线性二次型调节器, 其对象是现代控制理论中以状态空间形式给出的线性系统, 而目标函数为对象状态和控制输入的二次型函数。极点配置 (Pole Assignment) 控制是通过比例环节的反馈把定常线性系统的极点移置到预定位置的一种控制方法。极点配置的实质是用比例反馈去改变原系统的运动模式, 以满足设计规定的性能要求。

13.3.1 系统描述

针对倒立摆模型式 (13-7) 和式 (13-8), 取控制指标为 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 $x(1) = \theta$, $x(2) = \dot{\theta}$, $x(3) = x$, $x(4) = \dot{x}$, 则式 (13-7) 和式 (13-8) 可表示为状态方程

$$\dot{x} = Ax + Bu \quad (13.9)$$

$$\text{式中, } A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ t_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ t_2 & 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ t_3 \\ 0 \\ t_4 \end{pmatrix}, \quad t_1 = \frac{m(m+M)gl}{(M+m)I + Mml^2}, \quad t_2 = -\frac{m^2 gl^2}{(M+m)I + Mml^2},$$

$$t_3 = -\frac{ml}{(M+m)I + Mml^2}, \quad t_4 = \frac{I + ml^2}{(M+m)I + Mml^2}.$$

控制的目的是通过给小车底座施加一个力 u (控制量), 使小车停留在零位置, 并使杆不倒下, 即 $\theta \rightarrow 0$ 、 $\dot{\theta} \rightarrow 0$ 、 $x \rightarrow 0$ 和 $\dot{x} \rightarrow 0$ 。

13.3.2 全状态反馈控制

(1) LQR 控制

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 分别表示各个状态跟踪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。基于 LQR 的增益及控制律为

$$u(k) = -Kx \quad (13.10)$$

$$K = LQR(A, B, Q, R) \quad (13.11)$$

式中, LQR 为 MATLAB 下的线性二次型调节器。

(2) 极点配置

状态方程 $\dot{x} = Ax + Bu$ 的全状态反馈控制系统闭环特征多项式为 $(sI - (A - BK))$, 基于极点配置的增益及控制律为

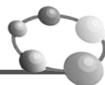
$$u(k) = -Kx \quad (13.12)$$

$$K = \text{place}(A, B, P) \quad (13.13)$$

式中, place 为 MATLAB 下的极点配置命令。

13.3.3 仿真实例

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。

仿真中，取\$M=1\$，采用LQR控制。取\$\mathbf{Q}=\begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\$，\$R=0.10\$，则由式(13.10)

可得LQR控制器增益\$\mathbf{K}=[-64.0799, -14.2285, -3.1623, -6.6632]\$。采用LQR时倒立摆响应结果及控制器输出如图13-4至图13-6所示，可见，采用LQR可实现单级倒立摆的最优控制。

仿真中，取\$M=2\$，采用极点配置控制。在本例中，\$\mathbf{A}\$和\$\mathbf{BK}\$都是\$4\times 4\$矩阵，因此闭环系统应该有4个极点。根据控制系统的要求，取希望的主导极点为\$-10-10i\$和\$-10+10i\$，其他的两个极点应远离主导极点，分别取\$-10\$和\$-20\$。根据极点配置命令place，可得控制系统的增益矩阵\$\mathbf{K}\$，\$\mathbf{K}=[-817.3, -133.3, -1394.6, -348.6]\$，采用极点配置控制时倒立摆响应结果及控制器输出如图13-7至图13-9所示。

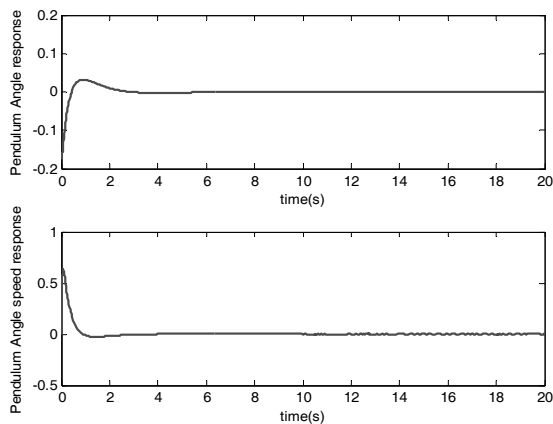


图 13-4 摆的角度和角速度响应 (LQR)

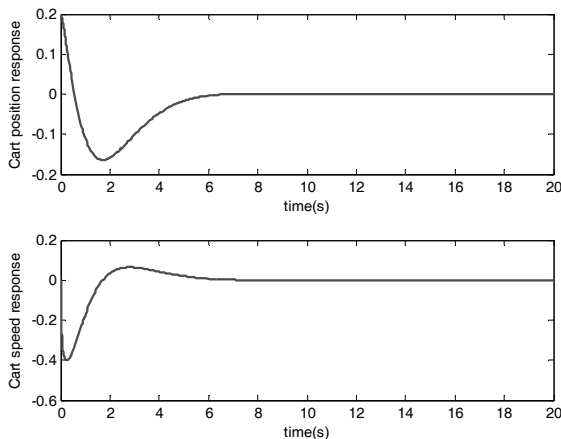


图 13-5 小车的角度和角速度响应 (LQR)

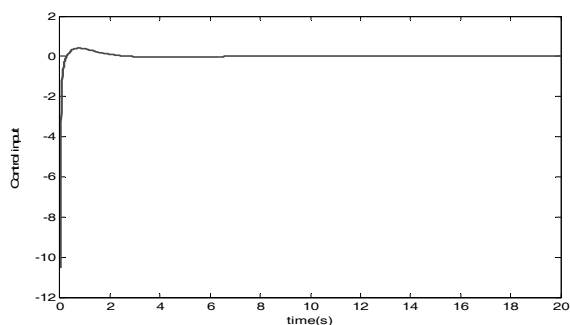


图 13-6 LQR 控制输入

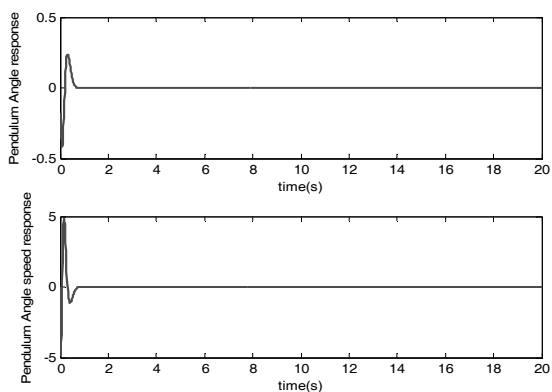


图 13-7 摆的角度和角速度响应（极点配置）

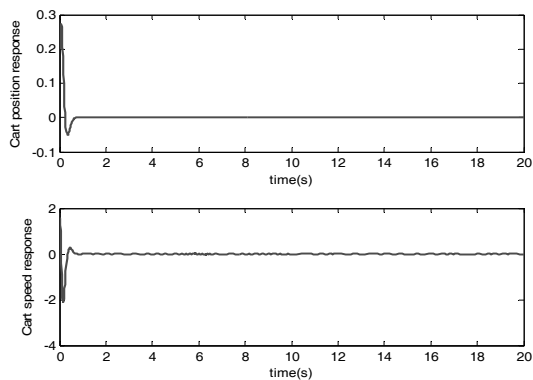


图 13-8 小车的角度和角速度响应（极点配置）

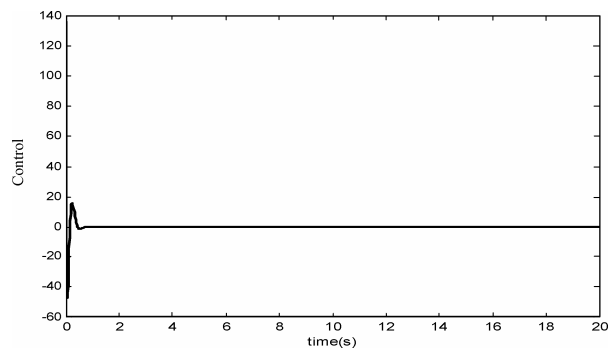
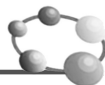


图 13-9 极点配置控制输入



仿真程序

(1) 连续系统控制仿真

Simulink 主程序: chap13_2sim.mdl (见图 13-10)

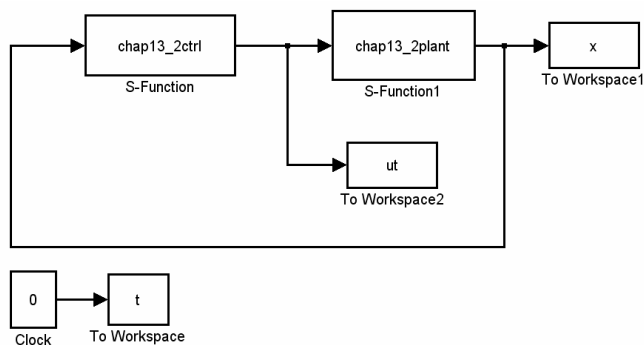


图 13-10 全状态反馈控制主程序

被控对象子程序: chap13_2plant.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0 = [-10/57.3,0,0.20,0]; %Initial state
str = [];
ts = [0 0];

function sys=mdlDerivatives(t,x,u) %Time-varying model
```




```
%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];

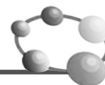
%State equation for one link inverted pendulum
D=A*x+B*u;
sys(1)=x(2);
sys(2)=D(2);
sys(3)=x(4);
sys(4)=D(4);

function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
```

控制器子程序: chap13_2ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
```



```

        sys=mdlOutputs(t,x,u);
    case {2,4,9}
        sys=[];
    otherwise
        error(['Unhandled flag = ',num2str(flag)]);
    end

function [sys,x0,str,ts]=mdlInitializeSizes
    sizes = simsizes;
    sizes.NumContStates = 0;
    sizes.NumDiscStates = 0;
    sizes.NumOutputs = 1;
    sizes.NumInputs = 4;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 0; % At least one sample time is needed
    sys = simsizes(sizes);
    x0 = []; %Initial state
    str = [];
    ts = [];

function sys=mdlOutputs(t,x,u)

%Single Link Inverted Pendulum Parameters
g=9.8;
M=1.0;
m=0.1;
L=0.5;

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];

M=1;
if M==1 % LQR

```



```
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
elseif M==2    % Pole point placement
    P=[-10-10i -10+10i -10 -20];    %Stable pole point
    K=place(A,B,P);
end

X=[u(1) u(2) u(3) u(4)]';
ut=-K*X;

sys(1)=ut;
```

作图子程序: chap13_2plot.m

```
close all;
figure(1);
subplot(211);
plot(t,x(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Pendulum Angle response');
subplot(212);
plot(t,x(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Pendulum Angle speed response');

figure(2);
subplot(211);
plot(t,x(:,3),'k','linewidth',2);
xlabel('time(s)');ylabel('Cart position response');
subplot(212);
plot(t,x(:,4),'k','linewidth',2);
xlabel('time(s)');ylabel('Cart speed response');

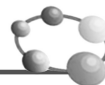
figure(3);
plot(t,ut,'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

(2) 离散系统控制仿真

主程序: chap13_3.m

```
%State Feedback Control for Single Link Inverted Pendulum
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];

M=1;
if M==1
    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
elseif M==2    % Pole point placement
    P=[-10-10i -10+10i -10 -20];    %Stable pole point
    K=place(A,B,P);
end

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('chap13_3plant',Tspan,xk,[],para);
    xk=x(length(x),:);

    x1(k)=xk(1);
    x2(k)=xk(2);
    x3(k)=xk(3);
    x4(k)=xk(4);

```



```
x=[x1(k) x2(k) x3(k) x4(k)];

u(k)=-K*x; %LQR
u_1=u(k);
end
figure(1);
subplot(211);
plot(time,x1,'k','linewidth',2);
xlabel('time(s)');ylabel('Pendulum Angle response');
subplot(212);
plot(time,x2,'k','linewidth',2);
xlabel('time(s)');ylabel('Pendulum Angle speed response');

figure(2);
subplot(211);
plot(time,x3,'k','linewidth',2);
xlabel('time(s)');ylabel('Cart position response');
subplot(212);
plot(time,x4,'k','linewidth',2);
xlabel('time(s)');ylabel('Cart speed response');

figure(3);
plot(time,u,'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

子程序: chap13_3plant.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;
```

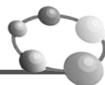


13.4 输入/输出反馈线性化

反馈线性化的基本思想就是利用全状态反馈抵消原系统中的非线性特性, 得到输入/输出之间具有线性行为的新系统(称为伪线性系统), 从而可以应用线性方法对新系统进行控制。^[63] 反馈线性化方法与其他方法相比, 其主要特点是不依赖于非线性系统的求解或稳定性分析, 而只需讨论系统的反馈变换, 因而它具有一定的普遍性。

13.4.1 系统描述

考虑三阶系统



$$\begin{aligned}
 \dot{x}_1 &= \sin x_2 + (x_2 + 1)x_3 \\
 \dot{x}_2 &= x_1^5 + x_3 \\
 \dot{x}_3 &= x_1^2 + u \\
 y &= x_1
 \end{aligned} \tag{13.14}$$

控制问题为：使输出 y 跟踪期望的轨迹 y_d 。由式 (13.14) 可见，输出 y 只是通过状态 x 间接地与控制输入 u 联系，很难直接进行控制器的设计。

13.4.2 控制律设计

为了得到输出 y 与输入 u 的直接关系，对输出 y 进行微分

$$\dot{y} = \dot{x}_1 = \sin x_2 + (x_2 + 1)x_3 \tag{13.15}$$

可见， $\dot{y}$ 与 u 没有直接关系，这就需要对 $\dot{y}$ 再进行微分

$$\begin{aligned}
 \ddot{y} &= \ddot{x}_1 = \dot{x}_2 \cos x_2 + \dot{x}_2 x_3 + (x_2 + 1)\dot{x}_3 \\
 &= (x_1^5 + x_3) \cos x_2 + (x_1^5 + x_3)x_3 + (x_2 + 1)(x_1^2 + u) \\
 &= (x_1^5 + x_3)(\cos x_2 + x_3) + (x_2 + 1)x_1^2 + (x_2 + 1)u
 \end{aligned} \tag{13.16}$$

取 $f(x) = (x_1^5 + x_3)(\cos x_2 + x_3) + (x_2 + 1)x_1^2$ ，则有

$$\ddot{y} = (x_2 + 1)u + f(x) \tag{13.17}$$

式 (13.17) 表示了输出 y 与输入 u 的显式关系。选择控制输入

$$u = \frac{1}{x_2 + 1}(v - f) \tag{13.18}$$

式中， v 为待设计的控制输入。

将式 (13.18) 代入式 (13.17) 中，得

$$\ddot{y} = v \tag{13.19}$$

定义跟踪误差为 $e = y_d - y$ ，设计具有 PD+前馈形式的控制输入 v 为

$$v = \ddot{y}_d + k_1 e + k_2 \dot{e} \tag{13.20}$$

式中， k_1 和 k_2 为正常数。

将式 (13.20) 代入式 (13.19)，则得到如下闭环系统

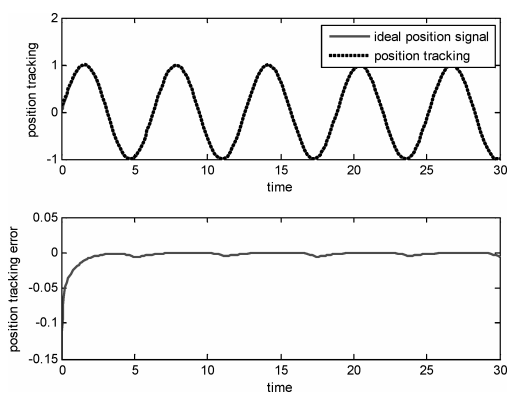
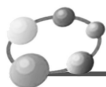
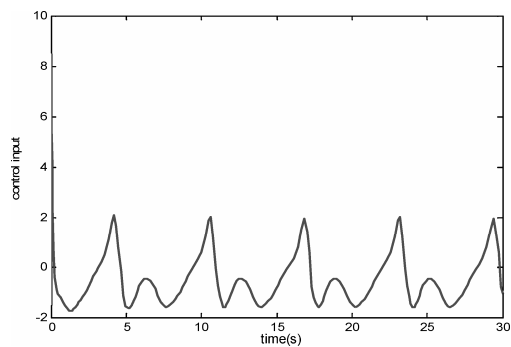
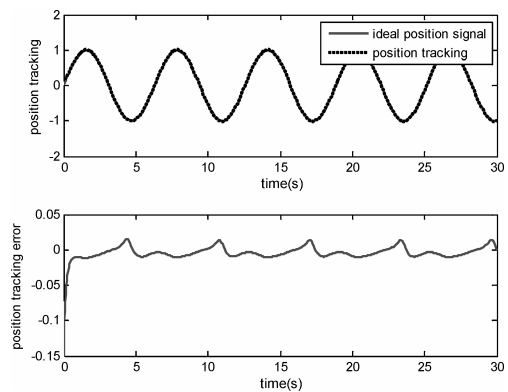
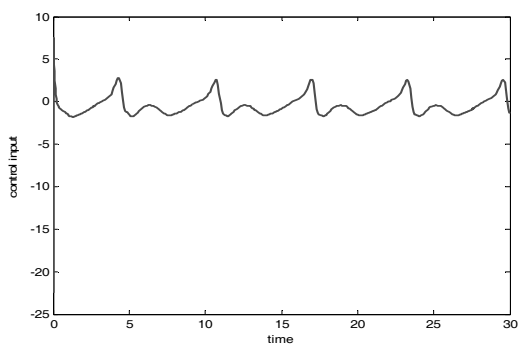
$$\ddot{e} + k_2 \dot{e} + k_1 e = 0 \tag{13.21}$$

式 (13.21) 说明闭环系统指数收敛。

本方法的缺陷是：控制律式 (13.18) 需要 f 项已知。解决的方法是在控制律中加入鲁棒项，或者采用模糊系统或神经网络逼近 f 的方法。

13.4.3 仿真实例

理想指令为 $y_d = \sin t$ ，取 $k_1 = k_2 = 10$ ，取 $M=1$ ，控制律为式 (13.18)，仿真结果如图 13-11 和图 13-12 所示。取 $M=2$ ，取 PD 控制，取 $k_p = 150$ ， $k_d = 30$ ，仿真结果如图 13-13 和图 13-14 所示。可见，采用线性化反馈控制方法，可实现高精度的位置跟踪，并可获得较小的控制输入信号。

图 13-11 采用线性化反馈的位置跟踪 ($M=1$)图 13-12 采用线性化反馈的控制输入 ($M=1$)图 13-13 采用 PD 的位置跟踪 ($M=2$)图 13-14 采用 PD 的控制输入 ($M=2$)



```
x1=u(4);
x2=u(5);
x3=u(6);

f=(x1^5+x3)*(x3+cos(x2))+(x2+1)*x1^2;

M=2;
if M==1
    k1=10;k2=10;
    v=ddyd+k1*e+k2*de;
    ut=1.0/(x2+1)*(v-f);
elseif M==2
    ut=150*e+30*de; %PD
end
sys(1)=ut;
```

被控对象子程序: chap13_4plant.m

```
function [sys,x0,str,ts]=obser(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.15 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
sys(1)=sin(x(2))+(x(2)+1)*x(3);
sys(2)=x(1)^5+x(3);
```



```
sys(3)=x(1)^2+ut;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
```

作图程序: chap13_4plot.m

```
close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time');ylabel('position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,y(:,1)-y(:,2),'r','linewidth',2);
xlabel('time');ylabel('position tracking error');
figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time');ylabel('control input');
```

9 13.5 倒立摆反演控制

反演 (Backstepping) 设计方法的基本思想是将复杂的非线性系统分解成不超过系统阶数的子系统, 然后为每个子系统分别设计李亚普诺夫函数和中间虚拟控制量, 一直“后退”到整个系统, 直到完成整个控制律的设计。反演设计方法又称反步法、回推法或后推法, 使整个闭环系统满足期望的动静态性能指标。

13.5.1 系统描述

假设被控对象为非线性系统

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x,t) + b(x,t)u + d(x,t)\end{aligned}\quad (13.22)$$

式中, $f(x,t)$ 和 $b(x,t)$ 为非线性函数, $d(x,t)$ 为建模不确定项与外加干扰的总和, $|d(x,t)| \leq D$, $b(x,t) \neq 0$ 。

13.5.2 控制律设计

基本反演控制方法设计步骤为

Step1:

定义位置误差



$$e_1 = x_1 - y_d \quad (13.23)$$

式中, y_d 为位置指令信号。

则

$$\dot{e}_1 = \dot{x}_1 - \dot{y}_d = x_2 - \dot{y}_d$$

定义 Lyapunov 函数

$$V_1 = \frac{1}{2} e_1^2 \quad (13.24)$$

则

$$\dot{V}_1 = e_1 \dot{e}_1 = e_1 (x_2 - \dot{y}_d)$$

为了实现 $\dot{V}_1 \leq 0$, 取虚拟控制 $x_2 = s - c_1 e_1 + \dot{y}_d$, 即 $s = x_2 + c_1 e_1 - \dot{y}_d = c_1 e_1 + \dot{e}_1$, $c_1 > 0$, 则

$$\dot{V}_1 = e_1 s - c_1 e_1^2$$

如果 $s = 0$, 则 $\dot{V}_1 \leq 0$ 。为此, 需要进行下一步设计。

Step2:

定义 Lyapunov 函数

$$V_2 = V_1 + \frac{1}{2} s^2 \quad (13.25)$$

由于 $\dot{s} = \dot{x}_2 + c_1 \dot{e}_1 - \ddot{y}_d = f(x, t) + b(x, t)u + d(x, t) + c_1 \dot{e}_1 - \ddot{y}_d$, 则

$$\dot{V}_2 = \dot{V}_1 + s\dot{s} = e_1 s - c_1 e_1^2 + s(f(x, t) + b(x, t)u + d(x, t) + c_1 \dot{e}_1 - \ddot{y}_d)$$

为使 $\dot{V}_2 \leq 0$, 设计控制器为

$$u = \frac{1}{b(x, t)} (-f(x, t) - c_2 s - e_1 - c_1 \dot{e}_1 + \ddot{y}_d - \eta \operatorname{sgn}(s)) \quad (13.26)$$

式中, $c_2 > 0$, $\eta \geq D$ 。

则

$$\dot{V}_2 = -c_1 e_1^2 - c_2 s^2 + s d(x, t) - \eta |s| \leq 0$$

通过控制律的设计, 使得系统满足了李亚普诺夫稳定性理论条件, e_1 和 e_2 以指数形式渐进稳定。

13.5.3 仿真实例

假设不考虑小车的控制问题, 被控对象取单级倒立摆, 则动态方程如下

$$\dot{x}_1 = x_2$$

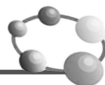
$$\dot{x}_2 = f(x) + g(x)u$$

式中,

$$f(x) = \frac{g \sin x_1 - m l x_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}, \quad g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}, \quad x_1 \text{ 和 } x_2$$

分别为摆角和摆速, $g = 9.8 \text{ m/s}^2$, m_c 为小车质量, $m_c = 1 \text{ kg}$, m 为摆杆质量, $m = 0.1 \text{ kg}$, l 为摆长的一半, $l = 0.5 \text{ m}$, u 为控制输入。

位置指令为 $y_d(t) = 0.1 \sin(\pi t)$, 采用控制律 (13.26), 倒立摆初始状态为 $[-\pi/60, 0]$, 仿



真结果如图 13-16 和图 13-17 所示。

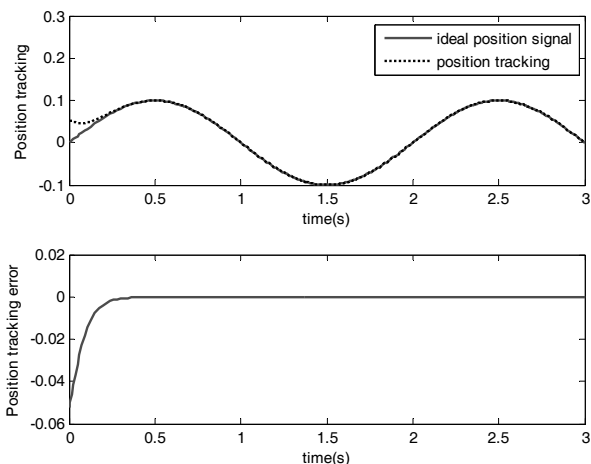


图 13-16 正弦位置跟踪

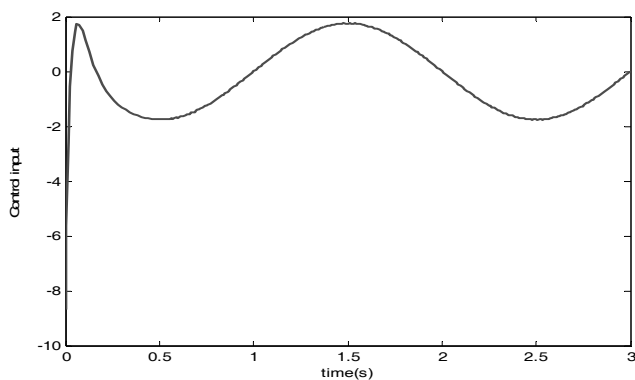


图 13-17 控制输入

仿真程序

Simulink 主程序: chap13_5sim.mdl (见图 13-18)

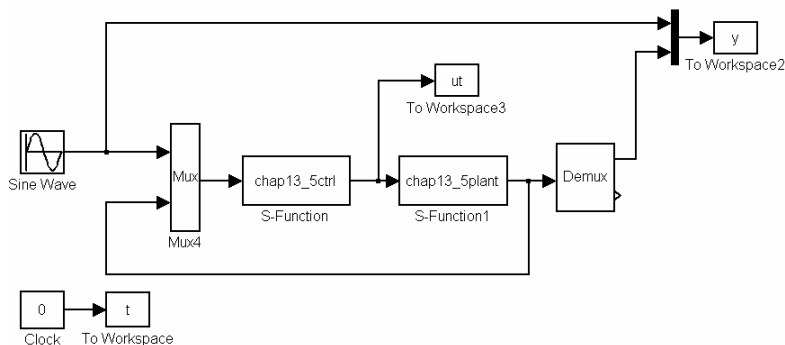


图 13-18 反演控制主程序

控制器 S 函数子程序: chap13_5ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
```



```
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
global M V x0 fai

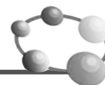
sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0=[];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
c1=35;
c2=15;

yd=u(1);
dyd=0.1*pi*cos(pi*t);
ddy=-0.1*pi^2*sin(pi*t);
x1=u(2);
x2=u(3);

g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x1))^2/(mc+m));
fx=g*sin(x1)-m*l*x2^2*cos(x1)*sin(x1)/(mc+m);
fx=fx/S;
gx=cos(x1)/(mc+m);
gx=gx/S;

e1=x1-yd;
de1=x2-dyd;

s=x2+c1*e1-dyd;
```



```
xite=0.010;
ut=(1/gx)*(-fx-c2*s-e1-c1*de1+ddy-d-xite*sign(s));

sys(1)=ut;
```

被控对象子程序: chap13_5plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;

S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```



作图程序: chap13_5plot.m

```
close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,y(:,1)-y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('Position tracking error');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```



13.6 倒立摆滑模控制

滑模控制 (Sliding Mode Control, SMC) 本质上是一类特殊的非线性控制, 其非线性表现为控制的不连续性, 这种控制策略与其他控制的不同之处在于系统的“结构”并不固定, 而是可以在动态过程中, 根据系统当前的状态 (如偏差及其各阶导数等), 有目的地不断变化, 迫使系统按照预定“滑动模态”的状态轨迹运动。由于滑动模态可以进行设计且与对象参数及扰动无关, 这就使得变结构控制具有快速响应、对参数变化及扰动不灵敏、无须系统在线辨识, 物理实现简单等优点。该方法的缺点在于当状态轨迹到达滑模面后, 难以严格地沿着滑面向着平衡点滑动, 而是在滑模面两侧来回穿越, 从而产生颤动。

13.6.1 问题描述

以倒立摆的摆杆控制为例, 考虑如下二阶非线性倒立摆系统

$$\ddot{\theta} = f(\theta, \dot{\theta}) + \Delta f(\theta, \dot{\theta}) + g(\theta, \dot{\theta})u + \Delta g(\theta, \dot{\theta})u + d_0(t) \quad (13.27)$$

式中, f 和 g 为已知非线性函数, $u \in R$ 和 $y = \theta \in R$ 分别为系统的输入和输出, $d_0(t)$ 为外加干扰, $|d(\theta, \dot{\theta}, t)| \leq D$ 。

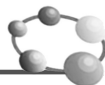
取 $d(\theta, \dot{\theta}, t) = \Delta f(\theta, \dot{\theta}) + \Delta g(\theta, \dot{\theta})u + d_0(t)$, 则式 (13.27) 可写为

$$\ddot{\theta} = f(\theta, \dot{\theta}) + g(\theta, \dot{\theta})u + d(\theta, \dot{\theta}, t) \quad (13.28)$$

13.6.2 控制律设计

设位置指令为 θ_d , 令 $e = \theta_d - \theta$, 设计滑模切换函数为

$$s = \dot{e} + ce \quad (13.29)$$



式中, $c > 0$, 则

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_d - \ddot{\theta} + c\dot{e} = \ddot{\theta}_d - f - gu - d + c\dot{e}$$

将控制律设计为

$$u = \frac{1}{g} \left[-f + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s) \right] \quad (13.30)$$

定义 Lyapunov 函数

$$L = \frac{1}{2} s^2$$

对 L 求导, 并将式 (13.30) 代入, 得

$$\begin{aligned} \dot{L} &= s\dot{s} = s(\ddot{\theta}_d - f - gu - d + c\dot{e}) \\ &= s(\ddot{\theta}_d - f - (-f + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)) - d + c\dot{e}) \\ &= s(-d - \eta \operatorname{sgn}(s)) \\ &= -sd - \eta|s| \end{aligned}$$

只要满足 $\eta \geq D$, 则有

$$\dot{L} = -sd - \eta|s| \leq 0$$

为了消除滑模控制中的抖振, 采用饱和函数 $\operatorname{sat}(s)$ 代替控制律式 (13.30) 中的符号函数 $\operatorname{sgn}(s)$, 表达如下

$$\operatorname{sat}(s) = \begin{cases} 1 & s > \Delta \\ ks & |s| \leq \Delta, k = 1/\Delta \\ -1 & s < -\Delta \end{cases} \quad (13.31)$$

其中 Δ 称为“边界层”。

饱和函数消除抖振的本质为: 在边界层外, 采用切换控制, 在边界层内, 采用反馈控制。由于滑模控制中, 切换函数 s 的值大部分时间都在边界层之内, 即控制律式 (13.30) 中采用的是 $\operatorname{sat}(s)$ 而不是切换项 $\operatorname{sgn}(s)$, 因此抖振得到了很好的抑制。

13.6.3 仿真实例

假设不考虑小车的控制问题, 被控对象取单级倒立摆, 取 $x_1 = \theta$, $x_2 = \dot{\theta}$, 则动态方程如下

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + g(x)u \end{aligned}$$

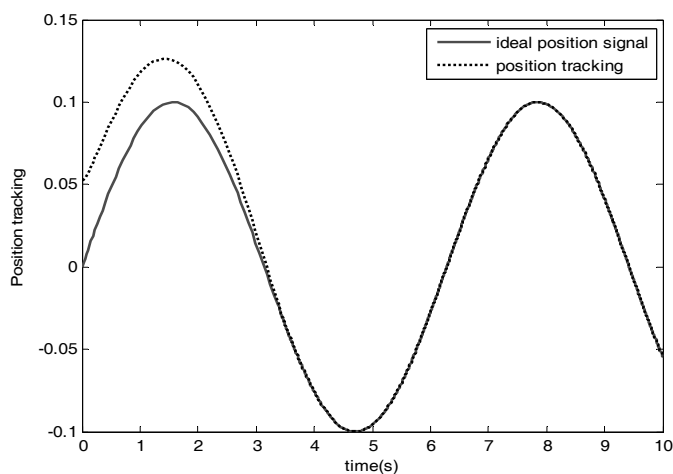
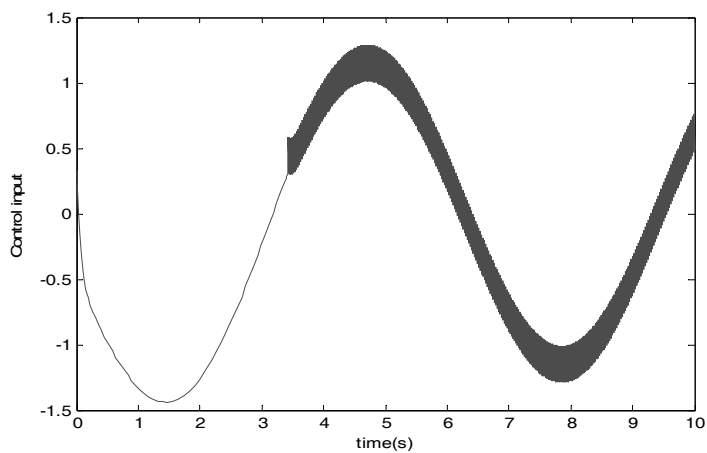
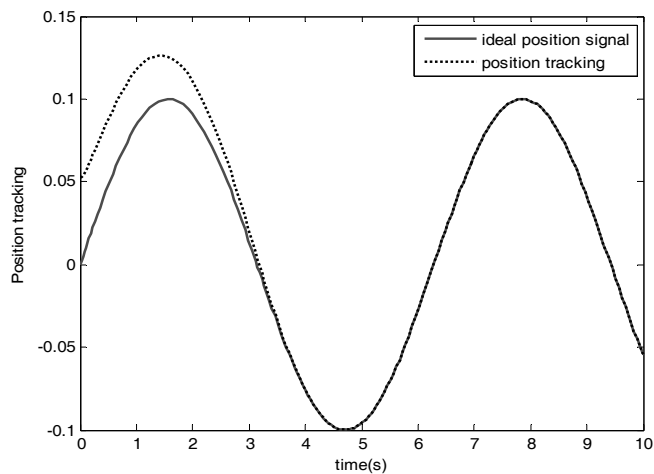
$$\text{式中, } f(x) = \frac{g \sin x_1 - m l x_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}, g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}, x_1 \text{ 和 } x_2$$

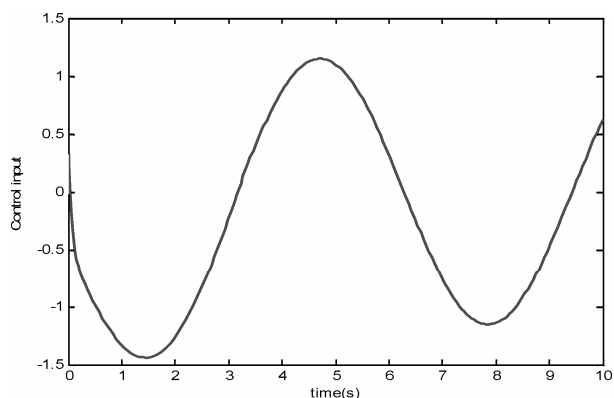
分别为摆角和摆速, $g = 9.8 \text{ m/s}^2$, m_c 为小车质量, $m_c = 1 \text{ kg}$, m 为摆杆质量, $m = 0.1 \text{ kg}$, l 为摆长的一半, $l = 0.5 \text{ m}$, u 为控制输入。

位置指令为 $\theta_d(t) = 0.1 \sin(t)$, 倒立摆初始状态为 $[\pi/60, 0]$ 。为了保证稳定性, 取 $\eta = 0.20$, 采用控制律式 (13.30), 首先采用切换函数, 程序中取 $M=1$, 仿真结果如图 13-19 和图 13-20



所示。为了减小抖振,采用饱和函数代替切换函数,程序中取 $M=2$, $\Delta=0.05$ 。仿真结果如图 13-21 和图 13-22 所示。

图 13-19 基于切换函数的位置跟踪 ($M=1$)图 13-20 基于切换函数的控制输入 ($M=1$)图 13-21 基于饱和函数的位置跟踪 ($M=2$)

图 13-22 基于饱和函数的控制输入 ($M=2$)

倒立摆滑模控制仿真程序有 4 个。

(1) Simulink 主程序: chap13_6sim.mdl (见图 13-23)

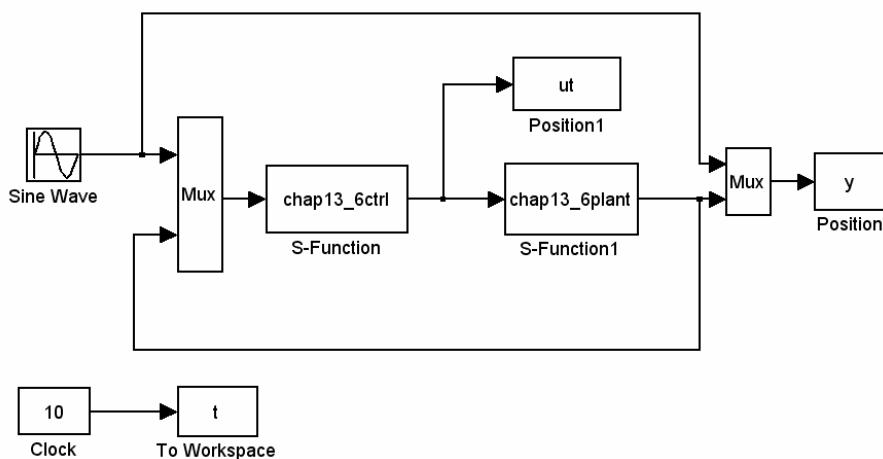


图 13-23 滑模控制主程序

(2) 控制器 S 函数: chap13_6ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
```



```
sizes.NumOutputs      = 1;
sizes.NumInputs       = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
thd=0.1*sin(t);
dthd=0.1*cos(t);
ddthd=-0.1*sin(t);

x1=u(2);
x2=u(3);
e=thd-x1;
de=dthd-x2;

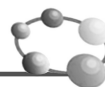
c=15;
s=c*e+de;

g=9.8;mc=1.0;m=0.1;l=0.5;
T=l*(4/3-m*(cos(x1))^2/(mc+m));

fx=g*sin(x1)-m*l*x2^2*cos(x1)*sin(x1)/(mc+m);
fx=fx/T;
gx=cos(x1)/(mc+m);
gx=gx/T;
xite=0.20;

M=2;
if M==1
    ut=1/gx*(-fx+ddthd+c*de+xite*sign(s));
elseif M==2
    %Saturated function
    delta=0.05;
    kk=1/delta;
    if abs(s)>delta
        sats=sign(s);
    else
        sats=kk*s;
    end
    ut=1/gx*(-fx+ddthd+c*de+xite*sats);
end
sys(1)=ut;
```

(3) 被控对象 S 函数: chap13_6plant.m



```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;
%%%%%%%%%%%%
dt=0*10*sin(t);
%%%%%%%%%%%%

sys(1)=x(2);
sys(2)=fx+gx*u+dt;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);

```

(4) 作图程序: chap13_6plot.m

```

close all;

figure(1);

```



```

plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position signal','position tracking');
figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```



13.7 自适应鲁棒控制

自适应控制 (Adaptive Control) 是一种能修正自己特性以适应对象和扰动动态特性变化的一种控制方法。鲁棒控制 (Robust Control) 是指控制系统在一定的参数摄动下, 维持某些性能的特性。通过采用自适应鲁棒控制方法, 可达到很好的控制系统性能。

13.7.1 问题的提出

不确定性机械系统可描述为

$$\frac{dx_1}{dt} = x_2 \quad (13.32)$$

$$J \frac{dx_2}{dt} = u(t) + \Delta \quad (13.33)$$

式中, $x = [x_1 \ x_2]^T$ 表示位置和速度, J 为系统未知转动惯量, J 为大于零的常数, Δ 表示包括干扰和模型不确定部分的总的不确定性。

取 $\theta = J$, 则式 (13.33) 可写为

$$\theta \frac{dx_2}{dt} = u + \Delta \quad (13.34)$$

假设 1 不确定参数 θ 的上下界定义为

$$\theta \in \Omega \triangleq \{\theta : 0 < \theta_{\min} \leq \theta \leq \theta_{\max}\} \quad (13.35)$$

假设 2 不确定项 Δ 有界, 表示为

$$|\Delta| \leq D \quad (13.36)$$

13.7.2 自适应控制律的设计

定义滑模函数为

$$\begin{aligned} s &= \dot{e} + ce = x_2 - \dot{q} \\ q &= \dot{x}_d - ce \end{aligned} \quad (13.37)$$

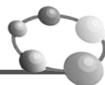
式中, $e = x_1 - x_d$ 为位置跟踪误差, $c > 0$ 。

则

$$\dot{s} = \dot{x}_2 - \dot{q}$$

控制律设计为

$$u = u_a + u_{s1} + u_{s2} \quad (13.38)$$



控制律式 (13.38) 中各项分别表示为

$$u_a = \hat{\theta} \dot{q} \quad (13.39)$$

$$u_{s1} = -k_s s \quad (13.40)$$

$$u_{s2} = -\eta \text{sign}(s) \quad (13.41)$$

式中, u_a 为自适应补偿项, u_{s1} 为反馈项, u_{s2} 为鲁棒项, $k_s > 0$, $\eta > D$ 。

控制律式 (13.38) 可写为

$$u = \hat{\theta} \dot{q} - k_s s - \eta \text{sign}(s)$$

定义 Lyapunov 函数为

$$V = \frac{1}{2} \theta s^2 + \frac{1}{2\gamma} \tilde{\theta}^2 \quad (13.42)$$

式中, $\tilde{\theta} = \hat{\theta} - \theta$, $\gamma > 0$ 。

则

$$\dot{V} = \theta s \dot{s} + \frac{1}{\gamma} \tilde{\theta} \dot{\tilde{\theta}} = s(\theta \dot{x}_2 - \theta \dot{q}) + \frac{1}{\gamma} \tilde{\theta} \dot{\tilde{\theta}}$$

取自适应律为

$$\dot{\tilde{\theta}} = -\gamma \dot{q} s \quad (13.43)$$

则

$$\begin{aligned} \dot{V} &= s(u + \Delta - \theta \dot{q}) + \frac{1}{\gamma} \tilde{\theta} \dot{\tilde{\theta}} \\ &= s(\hat{\theta} \dot{q} - k_s s - \eta \text{sign}(s) + \Delta - \theta \dot{q}) + \frac{1}{\gamma} \tilde{\theta}(-\gamma \dot{q} s) \\ &= s(\tilde{\theta} \dot{q} - k_s s - \eta \text{sign}(s) + \Delta) - \tilde{\theta}(\dot{q} s) \\ &= -k_s s^2 - \eta |s| + \Delta \cdot s < -k_s s^2 \leq 0 \end{aligned}$$

为了防止 $\hat{\theta}$ 过大而造成控制输入信号 $u(t)$ 过大, 对式 (13.43) 进行以下修正^[54]

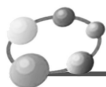
$$\begin{aligned} \dot{\tilde{\theta}} &= \text{Proj}_{\tilde{\theta}}(-\gamma \dot{q} s) \quad (13.44) \\ \text{Proj}_{\tilde{\theta}}(\cdot) &= \begin{cases} 0 & \text{if } \hat{\theta} = \theta_{\max} \text{ and } \cdot > 0 \\ 0 & \text{if } \hat{\theta} = \theta_{\min} \text{ and } \cdot < 0 \\ \cdot & \text{otherwise} \end{cases} \end{aligned}$$

13.7.3 仿真实例

取被控对象为

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ J \frac{dx_2}{dt} &= u(t) + \Delta \end{aligned}$$

式中, $J = 1.0$, Δ 取摩擦模型, 表示为 $\Delta = 0.5\dot{\theta} + 1.5\text{sign}(\dot{\theta})$ 。



位置指令信号取 $\sin t$ ，参数 θ 的变化范围取 $\theta_{\min} = 0.5$ ， $\theta_{\max} = 1.5$ 。控制参数取 $c = 15$ ， $k_s = 15$ ， $\gamma = 500$ ， $\eta = D + 0.01 = 1.01$ 。采用自适应鲁棒控制律式 (13.38)，如果自适应律取式 (13.43)，则仿真结果如图 13-24 至图 13-26 所示；如果自适应律取式 (13.44)，则仿真结果如图 13-27 至图 13-29 所示。可见，通过采用改进的自适应律，可限制参数 θ 的自适应变化范围，防止控制输入信号 $u(t)$ 过大。

仿真中，自适应估计参数 $\hat{\theta}$ 未收敛于 J ，这是由于位置指令信号未能达到“持续激励”条件，即不足够丰富而造成。^[63]如果采用 PD 控制 ($M=2$)，取 $k_p = 100$ ， $k_d = 50$ ，仿真结果如图 13-30 所示，由仿真结果可见，位置跟踪出现了平顶现象，PD 控制无法获得高精度控制效果。

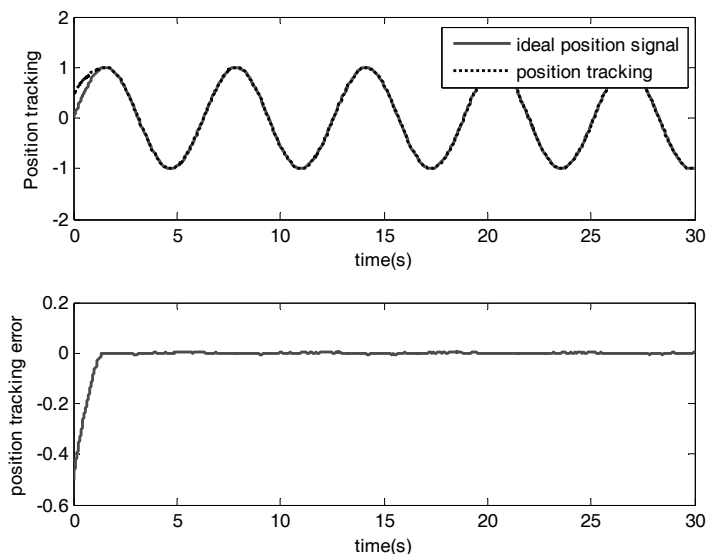


图 13-24 基于自适应律式 (13.43) 的鲁棒自适应控制位置跟踪 ($M=1$, $N=1$)

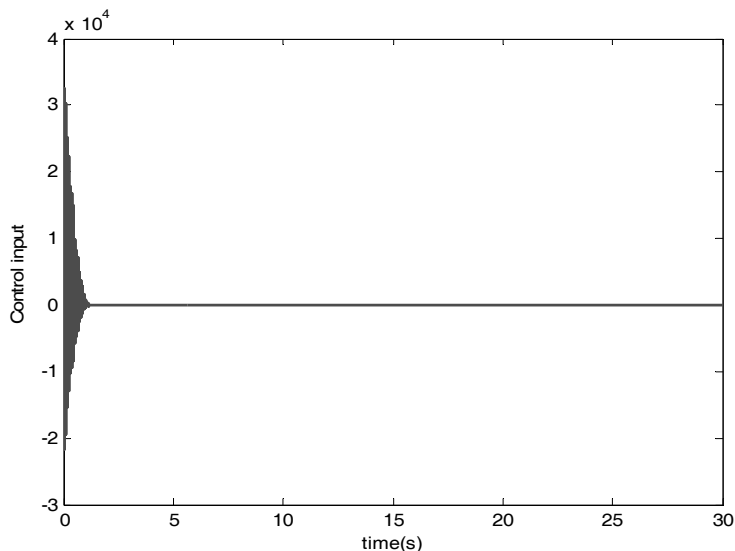
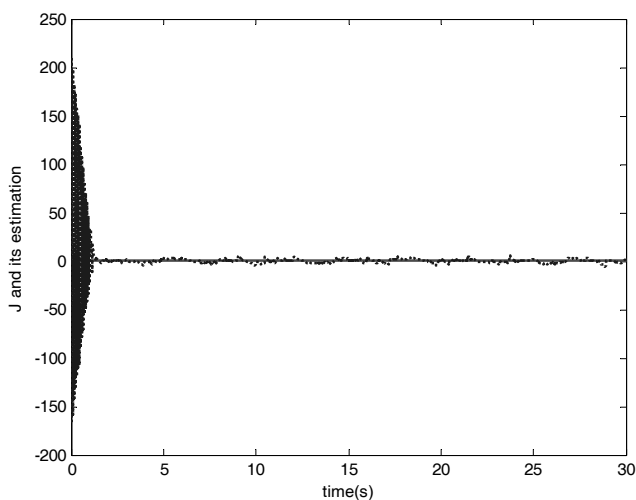
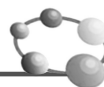
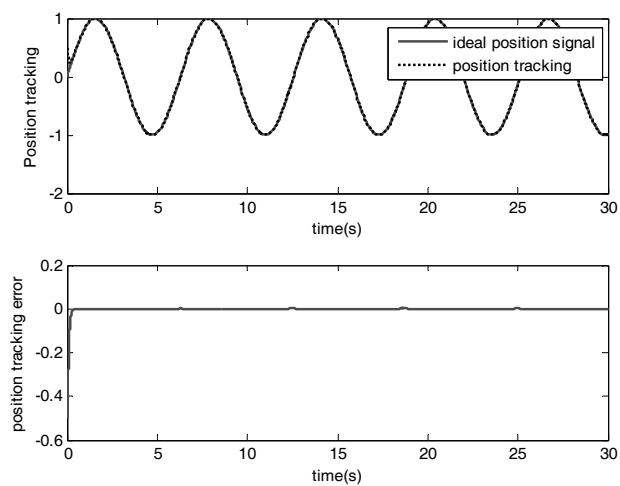
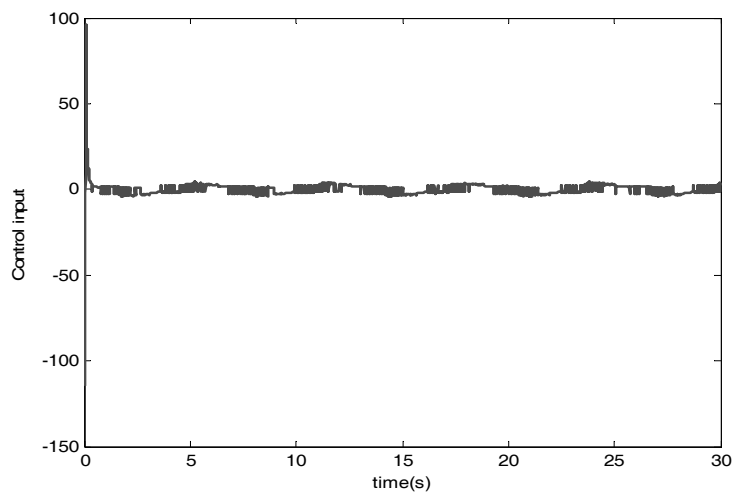
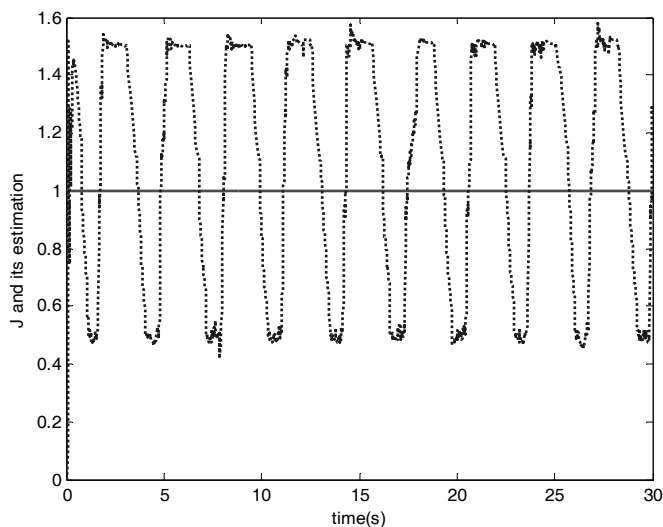
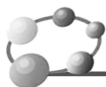
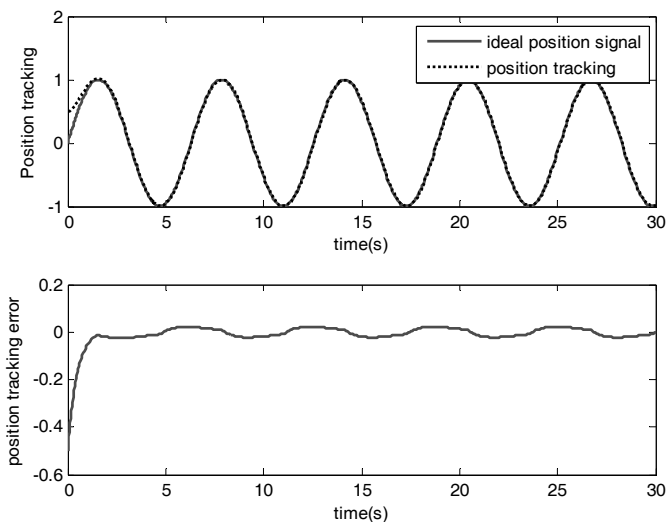


图 13-25 基于自适应律式 (13.43) 的控制输入 ($M=1$, $N=1$)

图 13-26 基于自适应律式 (13.43) 的自适应参数变化过程 ($M=1$, $N=1$)图 13-27 基于修正自适应律式 (13.44) 的鲁棒自适应控制位置跟踪 ($M=1$, $N=2$)图 13-28 基于修正自适应律式 (13.44) 的控制输入 ($M=1$, $N=2$)

图 13-29 基于修正自适应律式 (13.44) 的自适应参数变化过程 ($M=1$, $N=2$)图 13-30 PD 控制位置跟踪 ($M=2$)

仿真程序

Simulink 主程序: chap13_7sim.mdl (见图 13-31)

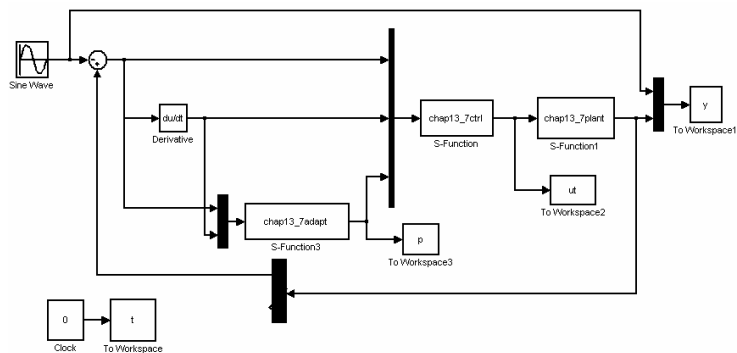
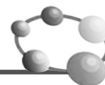


图 13-31 自适应鲁棒控制主程序



控制律程序: chap13_7ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {1,2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[0 0];
function sys=mdlOutputs(t,x,u)
e=u(1);
de=u(2);
dxd=cos(t);
ddxd=-sin(t);
thp=u(3);

c=15;
s=de+c*e;    %Sliding Mode
x2=dxd+de;
dq=ddxd-c*de;

ks=15;
xite=2.01;
ua=thp*dq;
us1=-ks*s;
us2=-xite*sign(s);

M=1;
if M==1    %DRC
    ut=ua+us1+us2;
```



```
elseif M==2 %PD
    kp=100;kd=50;
    ut=-kp*e-kd*de;
end

sys(1)=ut;
```

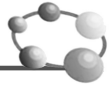
自适应律程序: chap13_7adapt.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
e=u(1);
de=u(2);
dxd=cos(t);
ddxd=-sin(t);

x2=dxd+de;

c=15;
gama=500;

s=de+c*e;
thp=x(1);
```



```

dq=ddxd-c*de;

th_min=0.5;
th_max=1.5;

alaw=-gama*dq*s;    %Adaptive law

N=1;
if N==1
    sys(1)=alaw;
elseif N==2
    if thp>=th_max&alaw>0
        sys(1)=0;
    elseif thp<=th_min&alaw<0
        sys(1)=0;
    else
        sys(1)=alaw;
    end
end
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);        %J estimate

```

被控对象程序: chap13_7plant.m

```

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.5;0];
str=[];

```



```
ts=[];  
function sys=mdlDerivatives(t,x,u)  
J=1.0;  
ut=u(1);  
  
F=0.5*x(2)+1.5*sign(x(2));  
sys(1)=x(2);  
sys(2)=1/J*(ut-F);  
function sys=mdlOutputs(t,x,u)  
J=1.0;  
  
sys(1)=x(1);  
sys(2)=J;
```

作图程序: chap13_7plot.m

```
close all;  
figure(1);  
subplot(211);  
plot(t,y(:,1),'r',t,y(:,2),'k','linewidth',2)  
xlabel('time(s)');ylabel('Position tracking');  
legend('ideal position signal','position tracking');  
subplot(212);  
plot(t,y(:,1)-y(:,2),'r','linewidth',2)  
xlabel('time(s)');ylabel('position tracking error');  
  
figure(2);  
plot(t,ut(:,1),'r','linewidth',2)  
xlabel('time(s)');ylabel('Control input');  
  
figure(3);  
plot(t,y(:,3),'r',t,p(:,1),'l','linewidth',2)  
xlabel('time(s)');ylabel('J and its estimation');
```

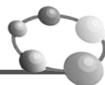


13.8 单级倒立摆的 H_∞ 控制

鲁棒控制 (Robust Control) 是指控制系统在一定 (结构, 大小) 的参数摄动下, 维持某些性能的特性, H_∞ 控制是一种重要的鲁棒控制方法。 H_∞ 优化控制问题可归纳为: 求出一个使系统内部稳定的控制器 $K(s)$, 使闭环传函 $T(s)$ 的无穷范数极小。^[55]

13.8.1 系统描述

为了使倒立摆线性化, 必须满足倒立摆的各级摆杆的转角是小角度, 此时 $\sin \theta \approx \theta$, $\cos \theta \approx 1$ 。线性化后的单级倒立摆方程为



$$\ddot{\theta} = \frac{m(m+M)gl}{(M+m)I + Mml^2} \theta - \frac{ml}{(M+m)I + Mml^2} u \quad (13.45)$$

$$\ddot{x} = -\frac{m^2 gl^2}{(M+m)I + Mml^2} \theta + \frac{I + ml^2}{(M+m)I + Mml^2} u \quad (13.46)$$

式中, $I = \frac{1}{12}mL^2$, $l = \frac{1}{2}L$ 。

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 $x(1) = \theta$, $x(2) = \dot{\theta}$, $x(3) = x$, $x(4) = \dot{x}$, 则式 (13.45) 和 (13.46) 可表示为状态方程

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (13.47)$$

$$\text{式中, } \mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ t_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ t_2 & 0 & 0 & 0 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 0 \\ t_3 \\ 0 \\ t_4 \end{pmatrix}, \text{其中 } t_1 = \frac{m(m+M)gl}{(M+m)I + Mml^2}, t_2 = -\frac{m^2 gl^2}{(M+m)I + Mml^2},$$

$$t_3 = -\frac{ml}{(M+m)I + Mml^2}, t_4 = \frac{I + ml^2}{(M+m)I + Mml^2}。$$

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。

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 $\dot{x}$ 。将倒立摆方程转化为状态方程的形式, 令 $\mathbf{x}(1) = \theta$, $\mathbf{x}(2) = x$, $\mathbf{x}(3) = \dot{\theta}$, $\mathbf{x}(4) = \dot{x}$, 考虑控制输入干扰 ω , 则式 (13.47) 可表示为状态方程

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}_1\omega + \mathbf{B}_2u \quad (13.48)$$

$$\text{式中, } \mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ t_1 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 0 \end{pmatrix}, \mathbf{B}_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \mathbf{B}_2 = \begin{pmatrix} 0 \\ 0 \\ t_3 \\ t_4 \end{pmatrix}, \text{其中 } t_1 = \frac{m(m+M)gl}{(M+m)I + Mml^2},$$

$$t_2 = -\frac{m^2 gl^2}{(M+m)I + Mml^2}, t_3 = -\frac{ml}{(M+m)I + Mml^2}, t_4 = \frac{I + ml^2}{(M+m)I + Mml^2}。$$

13.8.2 H_∞ 控制器要求

针对系统

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}_1\omega + \mathbf{B}_2u \\ \mathbf{z} &= \mathbf{C}_1\mathbf{x} + \mathbf{D}_{11}\omega + \mathbf{D}_{12}u \\ \mathbf{y} &= \mathbf{x} \end{aligned} \quad (13.49)$$

式中, ω 为控制扰动, $\mathbf{z}$ 为控制系统性能评价信号, 取 $D_{11} = 0$ 。

本控制系统的设计要求为:

(1) $\mathbf{x} = 0$ 是闭环系统的局部渐进稳定平衡点, 即对于任意初始状态 $\mathbf{x}(0) \in R^4$, $\mathbf{x}(t) \rightarrow 0$ 。



(2) 对于任意扰动 $\omega \in L_2[0, +\infty)$, 闭环系统具有扰动抑制性能, 即

$$\int_0^\infty (q_1 x^2(t) + q_2 \theta^2(t) + q_3 \dot{x}^2(t) + q_4 \dot{\theta}^2(t) + \rho u^2(t)) dt < \int_0^\infty \omega^2(t) dt \quad (13.50)$$

式中, $q_i \geq 0 (i=1, 2, 3, 4)$ 和 $\rho > 0$ 为加权系数, 令

$$C_1 = \begin{bmatrix} \sqrt{q_1} & 0 & 0 & 0 \\ 0 & \sqrt{q_2} & 0 & 0 \\ 0 & 0 & \sqrt{q_3} & 0 \\ 0 & 0 & 0 & \sqrt{q_4} \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad D_{12} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \sqrt{\rho} \end{bmatrix}$$

则式 (13.50) 等价于

$$\|z\|_2 < \|\omega\|_2 \quad (13.51)$$

定义 $T_\omega(s)$ 为 ω 至 z 的闭环传递函数, 表达式为

$$\|T_\omega(s)\|_\infty = \sup_{\omega \neq 0} \frac{\|z\|_2}{\|\omega\|_2} \quad (13.52)$$

则闭环系统的扰动抑制性能等价于 $\|T_\omega(s)\|_\infty < 1$ 。

13.8.3 基于 Riccati 方程的 H_∞ 控制

针对系统 (13.49), 申铁龙^[55]给出了相应的 Riccati 方程和控制器表达式 (见著作^[55]中定理 3.3.1 和推论 3.3.1), 描述如下:

定理^[55]: 针对式 (13.49), 存在反馈控制器, 使得闭环系统稳定且

$$\|T_\omega(s)\|_\infty < 1 \quad (13.53)$$

成立的充分必要条件是 Riccati 方程

$$A^T X + X A + X (B_1 B_1^T - B_2 B_2^T) X + C_1^T C_1 = 0 \quad (13.54)$$

具有使 $A + (B_1 B_1^T - B_2 B_2^T) X$ 稳定的半正定解 $X \geq 0$ 。如果有解, 则增益为

$$K = -B_2^T X \quad (13.55)$$

满足要求的控制器为

$$u = Kx \quad (13.56)$$

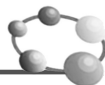
13.8.4 基于 LMI 的 H_∞ 控制

针对系统 (13.49), 俞立^[56]给出了相应的 LMI (见著作^[56]中定理 4.2.1), 该定理描述如下:

定理^[56] 对于式 (13.49), 给定 $\gamma > 0$, 存在 $P_1 = P_1^T > 0$ 和 P_2 , 如果满足不等式

$$\begin{bmatrix} AP_1 + P_1 A^T + B_2 P_2 + P_2^T B_2^T + \gamma^{-2} B_1 B_1^T & (C_1 P_1 + D_{12} P_2)^T \\ C_1 P_1 + D_{12} P_2 & -I \end{bmatrix} < 0 \quad (13.57)$$

则状态反馈控制器为



$$u = Kx = P_2 P_1^{-1} x \quad (13.58)$$

式中, $K = [k_1 \ k_2 \ k_3 \ k_4]$ 。

实现倒立摆控制律的设计, 采用定理求解倒立摆系统的状态反馈控制增益 K 时, 需要两个 LMI, 其中一个 LMI 为式 (13.57), 另一个 LMI 为 $P_1 > 0$, 即

$$-P_1 < 0 \quad (13.59)$$

LMI 工具箱提供了 3 个线性矩阵不等式求解函数^[56], 其中 `feasp` 为用于解决可行性问题的求解函数。用来建立和描述不等式 (13.57) 的 LMI 主要函数如下。

(1) `setlmis` 和 `getlmis`: 一个线性矩阵不等式系统的描述以 `setlmis` 开始, 以 `getlmis` 结束。当需要确定一个新的 LMI 系统时, 输入 “`setlmis[]`”。当线性矩阵不等式确定好后, 输入 “`LMIs=getlmis`”, 该命令将 LMI 系统表示为 “`LMIs`”。

(2) `lmivar`: 用于描述出现在线性矩阵不等式中的矩阵变量, 该函数的一般表达形式为 $X = \text{lmivar}(\text{type}, \text{struct})$, 其中 `type` 确定了矩阵 X 的类型, `struct` 确定了矩阵的结构。例如, `type=1` 时, X 为对称块对角类型。

(3) `lmiterm`: 用于描述每一个线性不等式中各项的内容。第 1 项包含 4 个元素, 第 1 个元素表示了所描述的项属于哪一个线性矩阵不等式, 如该项为正, 则属于不等式的左边, 如该项为负, 则属于不等式的右边; 第 2 个元素和第 3 个元素表示所描述的项在矩阵中的位置; 第 2 项和第 3 项表示变量项的左右系数或常数项的值; 第 4 项为可选, 且只能是 “s”, 该项实现由一条 `lmiterm` 命令描述一个变量与该变量项的转置的和。

(4) `feasp`: 实现对不等式的求解, 求解结果赋给 `feasolution`。

(5) `dec2mat`: 将求解的结果 `feasolution` 按矩阵和向量的形式给出, 赋给 P_1 和 P_2 。

求解 P_1 和 P_2 的线性矩阵不等式 MATLAB 表示如下。

(a) 在
$$\begin{bmatrix} AP_1 + P_1 A^T + B_2 P_2 + P_2^T B_2^T + \gamma^{-2} B_1 B_1^T & (C_1 P_1 + D_{12} P_2)^T \\ C_1 P_1 + D_{12} P_2 & -I \end{bmatrix} < 0$$
 中, 由于 $C_1 P_1 + D_{12} P_2$ 与 $(C_1 P_1 + D_{12} P_2)^T$ 互为转置, 故只需要按 $C_1 P_1 + D_{12} P_2 < 0$ 设计便可, 程序为:

```
%First LMI: X=P1
lmiterm([1 1 1 P1],A,1,'s');
lmiterm([1 1 1 P2],B2,1,'s');
lmiterm([1 1 1 0],gamma^(-2)*B1*B1');

lmiterm([1 2 1 P1],C1,1);
lmiterm([1 2 1 P2],D12,1);

lmiterm([1 2 2 0],-1);    %-I<0
```

(b) $-P_1 < 0$ 的程序为:

```
%Second LMI
lmiterm([2 1 1 P1],-1,1);
```




13.8.5 仿真实例

针对倒立摆式 (13.45) 和式 (13.46), 仿真中取参数为: $g = 9.8\text{m/s}^2$ (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)。初始条件取 $\theta(0) = 30^\circ$, $\dot{\theta}(0) = 0.2^\circ/\text{s}$, $x(0) = 0.20$, $\dot{x}(0) = 0$, 其中摆动角度及角速度值应转变为弧度值。

在式 (13.50) 中, 取 $q_1 = 1.0$, $q_2 = 1.0$, $q_3 = 1.0$, $q_4 = 1.0$, $\rho = 1$ 。采用以下两种方法进行仿真。

仿真之一: 基于 Riccati 方程的 H_∞ 控制

为了求解 Riccati 方程式 (13.54), 按 MATLAB 求解 H_∞ 控制的 Riccati 方程帮助信息, 需要将其转化为

$$A^T X + XA - X \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} -I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B_1^T \\ B_2^T \end{bmatrix} X + C_1^T C_1 = 0$$

为了保证 $A + (B_1 B_1^T - B_2 B_2^T)X$ 具有稳定的半正定解 $X \geq 0$, 取 $B_1 = [0 \ 0 \ 0.1 \ 0.1]^T$ 。

取 $B = [B_1 \ B_2]$, $R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$, $C = C_1$, $X = \text{care}(A, B, C^T C, R)$, 则可得到 $X \geq 0$ 的解, 将 X 代入可验证 $A + (B_1 B_1^T - B_2 B_2^T)X$ 的稳定性, 可知其特征值全在负半面。

由式 (13.55) 可得控制器增益 $K = -B_2^T X = [29.0040 \ 1.0395 \ 5.3031 \ 2.2403]$ 。采用控制律式 (13.56), 倒立摆响应结果及控制器输出如图 13-32 至图 13-34 所示。

仿真之二: 基于 LMI 的 H_∞ 控制

取 $\gamma = 100$, 运行基于 LMI 的控制器增益求解程序 K_LMI_design.m, 求解 LMI 不等式 (13.57) 和式 (13.59), 得 $K = [36.3149 \ 1.8765 \ 6.3851 \ 3.6704]$ 。倒立摆响应结果及控制器输出如图 13-35 至图 13-37 所示。

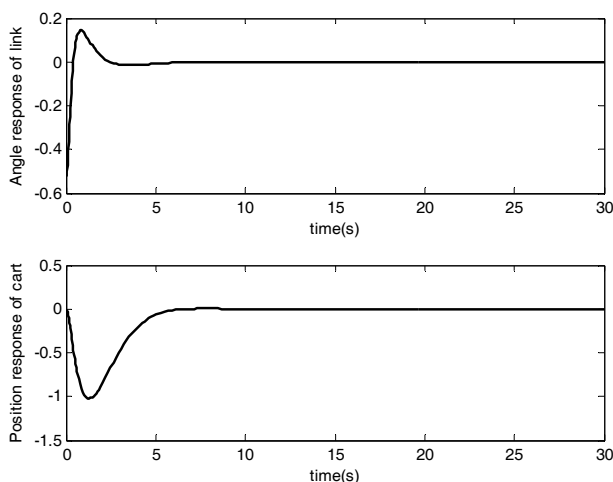


图 13-32 摆的角度和角速度响应 (Riccati 方程)

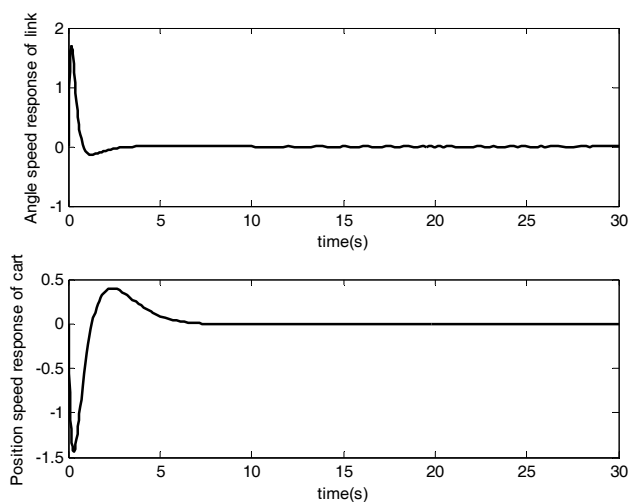
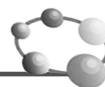


图 13-33 小车的角度和角速度响应 (Riccati 方程)

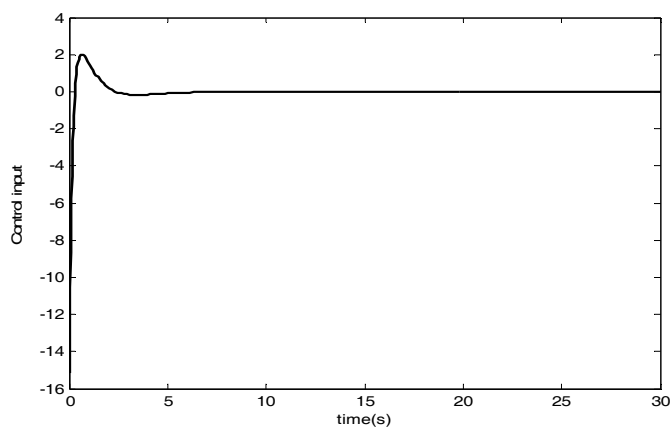


图 13-34 控制输入 (Riccati 方程)

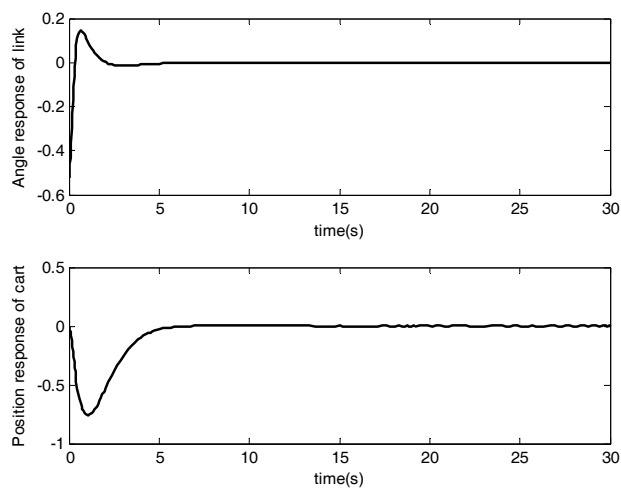


图 13-35 摆的角度和小车位置响应 (LMI 方法)

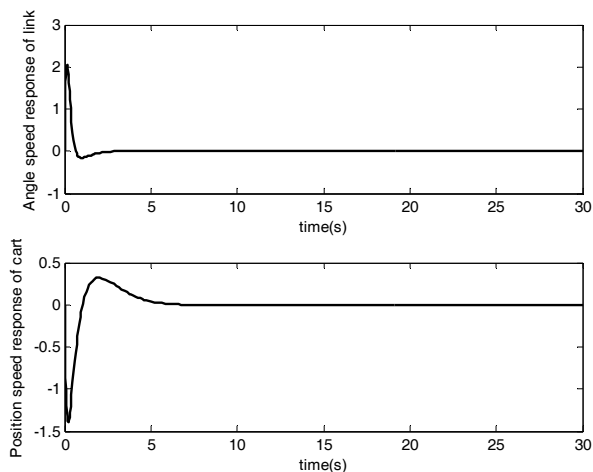


图 13-36 摆的角速度和小车速度响应 (LMI 方法)

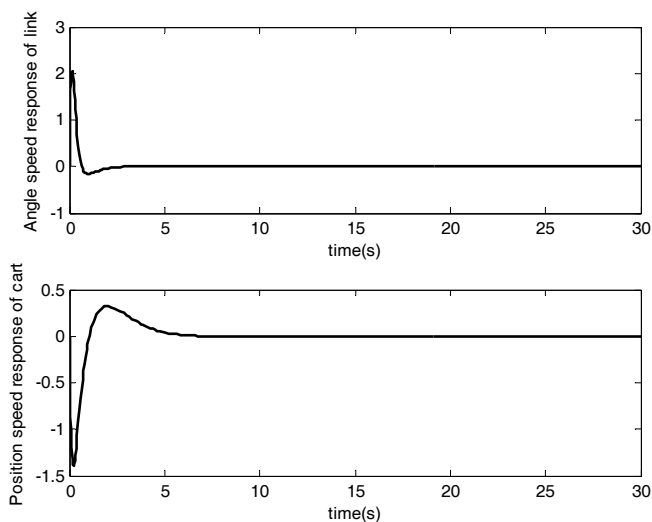
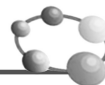


图 13-37 控制输入 (LMI 方法)

连续系统控制仿真

Riccati 控制器增益求解程序: chap13_8riccati.m

```
clear all;
close all;
%Single Link Inverted Pendulum Parameters
g=9.8;M=1.0;
%M=0.1;
m=0.1;L=0.5;
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=[0,0,1,0;
    0,0,0,1;
    t1,0,0,0;
    t2,0,0,0];
B2=[0;0;t3;t4];
B1=[0;0;0.1;0.1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
q1=1;q2=1;
q3=1;q4=1;
rho=1;

C1=[sqrt(q1), 0, 0, 0;
    0, sqrt(q2), 0, 0;
    0, 0, sqrt(q3), 0;
    0, 0, 0, sqrt(q4);
    0, 0, 0, 0];
D12=[0;0;0;0;sqrt(rho)];
%Continuous-time algebraic Riccati equation: Help-->search-->care
B=[B1 B2];
R=[-1 0;0 1];
C=C1;

X=care(A,B,C'*C,R)
%Verify the stability of A+(B1*B1'-B2*B2')*X
eig(A+(B1*B1'-B2*B2')*X)

K=-B2'*X

```

LMI 的控制器增益求解程序: chap13_8LMI.m

```

% H Infinity Controller Design based on LMI for Single Link Inverted Pendulum
clear all;
close all;

%Single Link Inverted Pendulum Parameters
g=9.8;M=1.0;m=0.1;L=0.5;
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*I/[(M+m)*I+M*m*l^2];
t4=(I+m*l^2)/[(m+M)*I+M*m*l^2];

A=[0,0,1,0;
    0,0,0,1;

```



```
t1,0,0,0;
t2,0,0,0];
B2=[0;0;t3;t4];
B1=[0;0;1;1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
q1=1;q2=1;q3=1;q4=1;
q=[q1,q2,q3,q4];
gama=100;

C1=[diag(q);zeros(1,4)];
rho=1;
D12=[0;0;0;0;rho];
D11=zeros(5,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%          LMI          Model          Design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
setlmis([]); %Initializes the description of a new LMI system.

P1=lmivar(1,[4 1]); % define P1
P2=lmivar(2,[1 4]); % define P2

%First LMI: X=P1
lmiterm([1 1 1 P1],A,1,'s');
lmiterm([1 1 1 P2],B2,1,'s');
lmiterm([1 1 1 0],gama^(-2)*B1*B1');

lmiterm([1 2 1 P1],C1,1);
lmiterm([1 2 1 P2],D12,1);

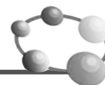
lmiterm([1 2 2 0],-1); %-I<0
%Second LMI
lmiterm([2 1 1 P1],-1,1);

LMIs=getlmis;

[tmin,feasolution]=feasp(LMIs);

if tmin>0
    X=[];Y=[];
else
    % Solving P1,P2
    P1=dec2mat(LMIs,feasolution,1); %P1
    P2=dec2mat(LMIs,feasolution,2); %P2
end

K=P2*inv(P1)
```



```
%K=[36.3149 1.8765 6.3851 3.6704];
```

Simulink 主程序: chap13_9sim.mdl (见图 13-38)

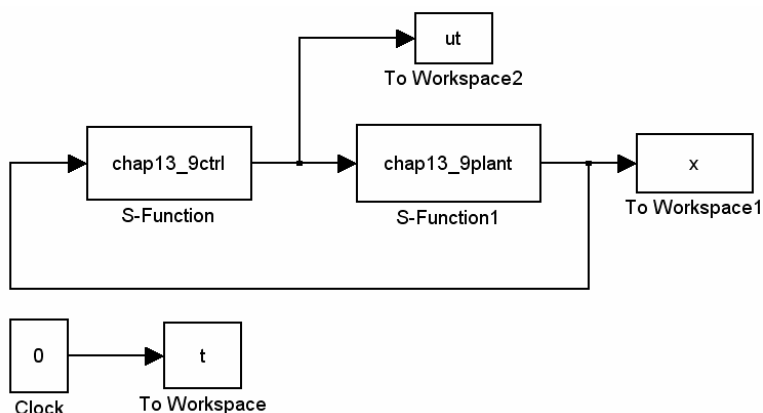


图 13-38 H_{∞} 控制主程序

被控对象子程序: chap13_9plant.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0 = [-30/57.3,0,0.20/57.3,0]; %Initial state
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u) %Time-varying model
%Single Link Inverted Pendulum Parameters
g=9.8;M=1.0;m=0.1;L=0.5;
I=1/12*m*L^2;
l=1/2*L;
```



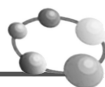
```
t1=m*(M+m)*g*I/[(M+m)*I+M*m*I^2];
t2=-m^2*g*I^2/[(m+M)*I+M*m*I^2];
t3=-m*I/[(M+m)*I+M*m*I^2];
t4=(I+m*I^2)/[(m+M)*I+M*m*I^2];

A=[0,0,1,0;
   0,0,0,1;
   t1,0,0,0;
   t2,0,0,0];
B2=[0;0;t3;t4];
B1=[0;0;1;1];

w=0*sin(t);
%State equation for one link inverted pendulum
D=A*x+B1*w+B2*u;
sys(1)=x(3);
sys(2)=x(4);
sys(3)=D(3);
sys(4)=D(4);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);    %Angle
sys(2)=x(2);    %Cart position
sys(3)=x(3);    %Angle speed
sys(4)=x(4);    %Cart speed
```

控制子程序: chap13_9ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
```



```
function sys=mdlOutputs(t,x,u)
M=2;
if M==1    %Riccati equation
    K=[29.0040 1.0395 5.3031 2.2403];
elseif M==2    %LMI
    K=[36.3149 1.8765 6.3851 3.6704];
end

X=[u(1) u(2) u(3) u(4)]';    % x=[th,x.dth,dx]
ut=K*X;
sys(1)=ut;
```

作图子程序: chap13_9plot.m

```
close all;
figure(1);
subplot(211);
plot(t,x(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Angle response of link');
subplot(212);
plot(t,x(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Position response of cart');

figure(2);
subplot(211);
plot(t,x(:,3),'k','linewidth',2);
xlabel('time(s)');ylabel('Angle speed response of link');
subplot(212);
plot(t,x(:,4),'k','linewidth',2);
xlabel('time(s)');ylabel('Position speed response of cart');

figure(3);
plot(t,ut,'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

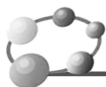


13.9 基于 GUI 的倒立摆控制动画演示

13.9.1 GUI 介绍

图形用户界面 (Graphical User Interface, GUI), 又称图形用户接口, 是指采用图形方式显示的计算机操作用户界面。GUI 是一种结合计算机科学、美学、心理学、行为学及各商业领域需求分析的人机系统工程, 强调人一机—环境三者作为一个系统进行总体设计。与早期计算机使用的命令行界面相比, 图形界面对于用户来说在视觉上更易于接受。MATLAB 提供了很好的 GUI 开发环境^[57]。

本实例是采用 GUI 技术, 参考 13.3 节中倒立摆模型的描述和 LQR 控制算法, 实现倒立



摆控制的动画演示。

13.9.2 演示程序的构成

演示程序由以下几个文件构成。

- (1) 主程序: `dlb.m`。
- (2) 演示界面程序: `dlb.fig`, 采用 GUI 来实现, 通过运行 “open dlb.fig” 可打开演示界面。
- (3) 倒立摆示意图: `model.jpg`, 采用绘图软件设计。

13.9.3 主程序的实现

采用 LQR 控制算法实现倒立摆和小车的控制, 主程序 `dlb.m` 包括以下几个部分。

- (1) 模型参数创建: 采用 `mc_CreateFcn()`、`mc_Callback()` 实现小车质量的创建, 同理可实现摆杆长度和质量的创建。
- (2) LQR 参数创建: 采用 `qi_CreateFcn()` 和 `qi_Callback()` 实现 q_i , $i=1,2,3,4$ 。采用 `r_CreateFcn()` 和 `r_Callback()` 实现 R 。
- (3) 采用 LQR 计算 K : 由 `lqrok_Callback()` 实现。
- (4) K 的创建: 采用 `ki_CreateFcn()` 和 `ki_Callback()` 实现 K_i , $i=1,2,3,4$ 。
- (5) 摆角度和小车位置初始值设定: 实现小车水平位置创建与回调、小车水平拖动条的创建与回调, 摆杆角度创建与回调、摆杆角度拖动条的创建与回调。
- (6) 干扰的输入: 共有冲击、阶跃和正弦三种干扰可以选择。
- (7) 仿真时间和步长的设定: `Tedit_CreateFcn()` 和 `Step_CreateFcn()`。
- (8) 仿真启动的设定。
- (9) 重置按钮: `reset_Callback()`, 实现倒立摆模型参数和 LQR 参数的重置。
- (10) 退出: `exit_Callback()`。

13.9.4 演示界面的 GUI 设计

首先在 MATLAB 环境下输入 “guide” 便可进入 GUI 界面的设计。本系统的 GUI 界面文件为 `dlb.fig`, 创建并保存该文件时, 可自动生成主程序框架 `dlb.m`。

用户在主程序框架 `dlb.m` 下, 在相应的 GUI 组件回调函数中描述模型和编写控制算法。通过在 MATLAB 环境下运行 “guide dlb.fig” 可打开 GUI 界面。

在本系统中, 采用了 GUI 开发环境的触控按钮、静态文本、可编辑文本、滑动条、坐标轴等组件。以小车质量 “M” 的 GUI 界面设计为例, 首先建立小车质量的 Edit text, 将其属性 tag 标签定义为 “mc”。基于 GUI 的倒立摆控制动画演示如图 13-39 所示。

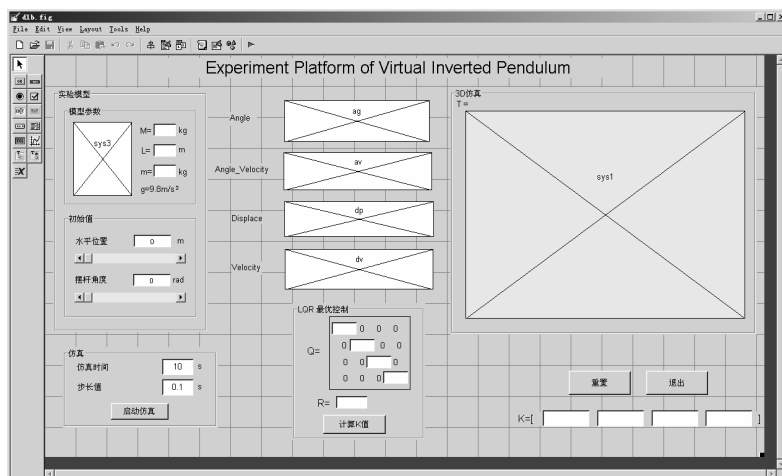


图 13-39 倒立摆控制动画演示 GUI 程序

13.9.5 演示步骤

通过以下 4 个步骤可实现倒立摆的动态演示。

- (1) 输入倒立摆参数：倒立摆摆杆质量 m 和长度 L ，小车质量 M 。
- (2) 通过“初始值”下的“水平位置”和“摆杆角度”可设定倒立摆和小车的初始角度和位置；选择一种干扰输入（冲击、阶跃和正弦），并设定干扰输入的幅值。
- (3) 根据控制系统要求的性能输入控制器设计参数 q_1 、 q_2 、 q_3 、 q_4 和 R ，单击“确定”按钮便可以得到控制器增益 K 。
- (4) 单击“启动仿真”便可以实现倒立摆的动态仿真演示。

仿真实例如下：取 $m=1$ ， $L=1$ ， $M=1$ ，取初始时刻的水平位置为 -0.15 ，摆杆角度为 -0.20 。控制器参数取 $q_1=5$ 、 $q_2=5$ 、 $q_3=5$ 、 $q_4=5$ 和 $R=5$ ，采用 LQR 方法求长，由式 (13.11)，则可得控制器增益 $K=[-51.0162 \quad -12.8235 \quad -1 \quad -2.7224]$ 。取仿真时间为 10，仿真步长为 0.1，控制目标位小车左侧停在零点，摆垂直不动。单击“启动仿真”，仿真结果如图 13-40 和图 13-41 所示。

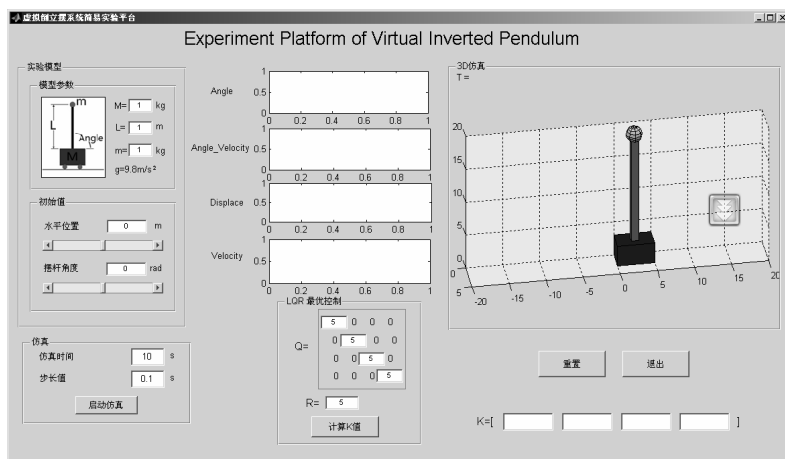


图 13-40 倒立摆控制演示的界面

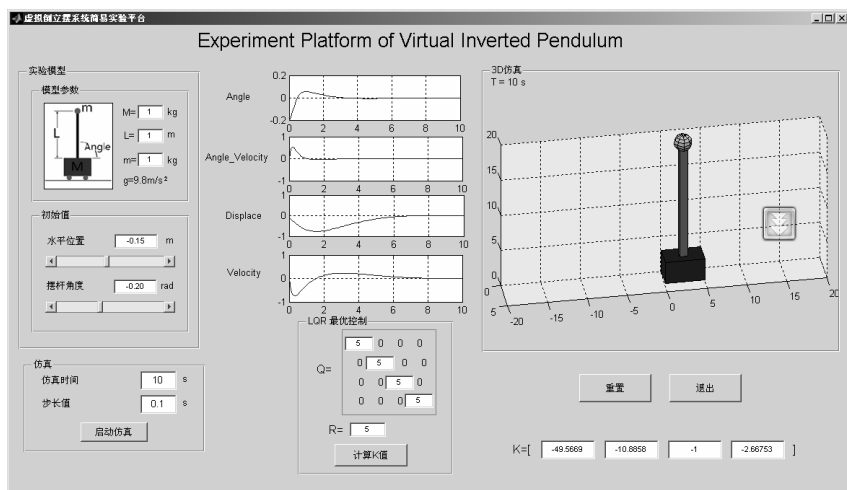


图 13-41 倒立摆控制仿真结果

仿真程序

(1) 主程序: `dlb.m` (程序较长, 可从本书所提供的的网址上下载)。

(2) 演示界面程序: `dlb.fig` (见图 13-42)。

通过 MATLAB 命令“`guide`”可打开 GUI 设计环境。在 MATLAB 环境下输入“`guide dlb.fig`”可打开倒立摆控制的演示界面程序进行设计或修改。

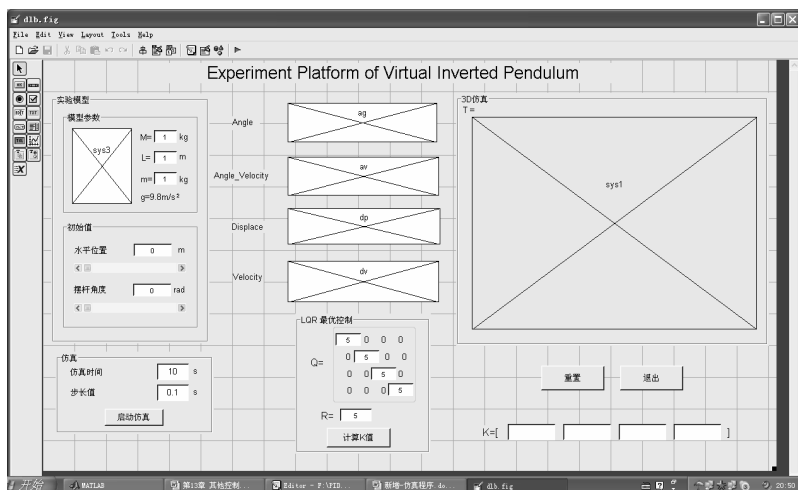


图 13-42 倒立摆控制系统的 GUI 设计界面

(3) 倒立摆 `model.jpg` (见图 13-43)。

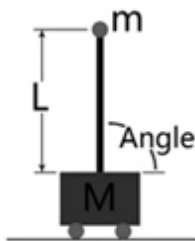


图 13-43 倒立摆示意图

第 14 章 PID 实时控制的 C++语言 设计及应用



14.1 控制系统仿真的 C++实现

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 chap14_1.m，PID 阶跃响应如图 14-1 所示。转化的 C++语言（Borland C++）程序为 chap14_2.cpp。

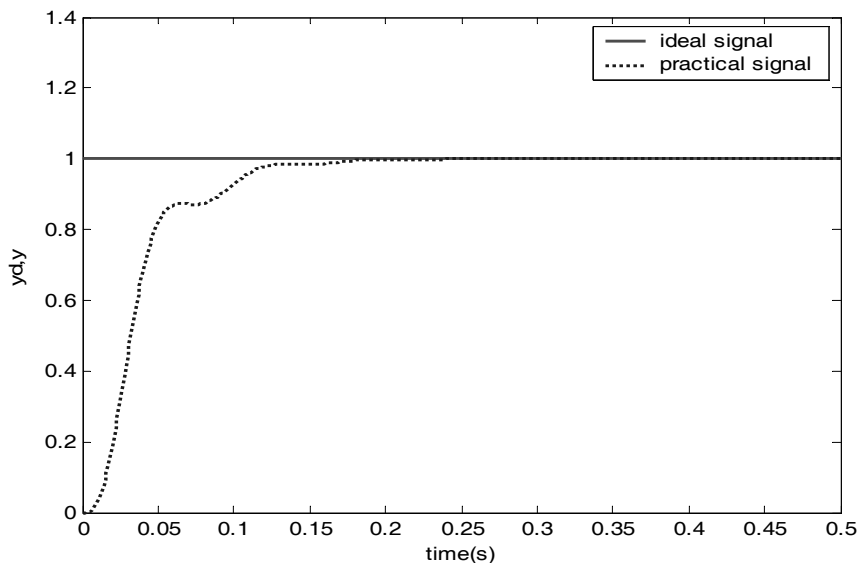


图 14-1 PID 的阶跃响应

(1) M 语言仿真程序：chap14_1.m

```
%PID Controler  
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;

yd(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
y(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)=yd(k)-y(k);

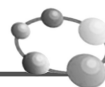
%Return of parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(k);

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,yd,'r',time,y,'b','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal signal','practical signal');
```

(2) C++语言仿真程序: chap14_2.cpp

```
//PID 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;
}
}

```



14.2 基于 C++ 的三轴飞行模拟转台伺服系统 PID 实时控制

由仿真到实时控制，需要在仿真的基础上加入 A/D、D/A 转换，并设计中断服务程序。下面以一个实际工程为例加以介绍。



14.2.1 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,如图 14-2 所示。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。

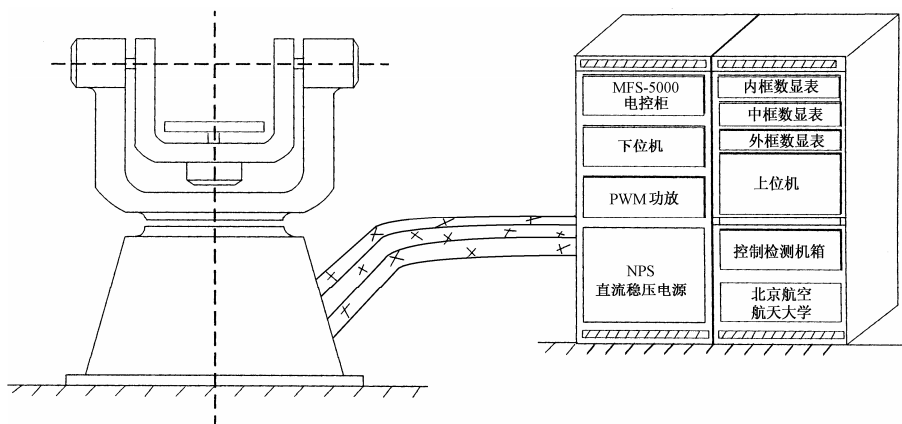


图 14-2 转台系统总体结构图

内、中、外框三通道数字伺服控制系统原理结构,如图 14-3 所示。^[48]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。

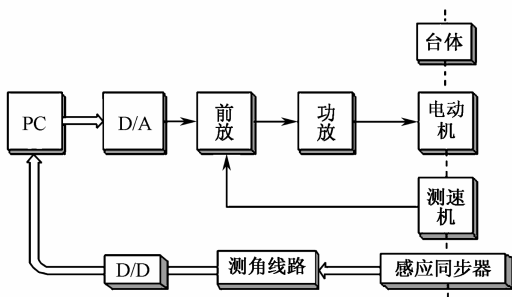


图 14-3 内、中、外三通道数字伺服控制系统原理结构

14.2.2 实时控制程序分析

(1) 程序流程图,如图 14-4 所示。

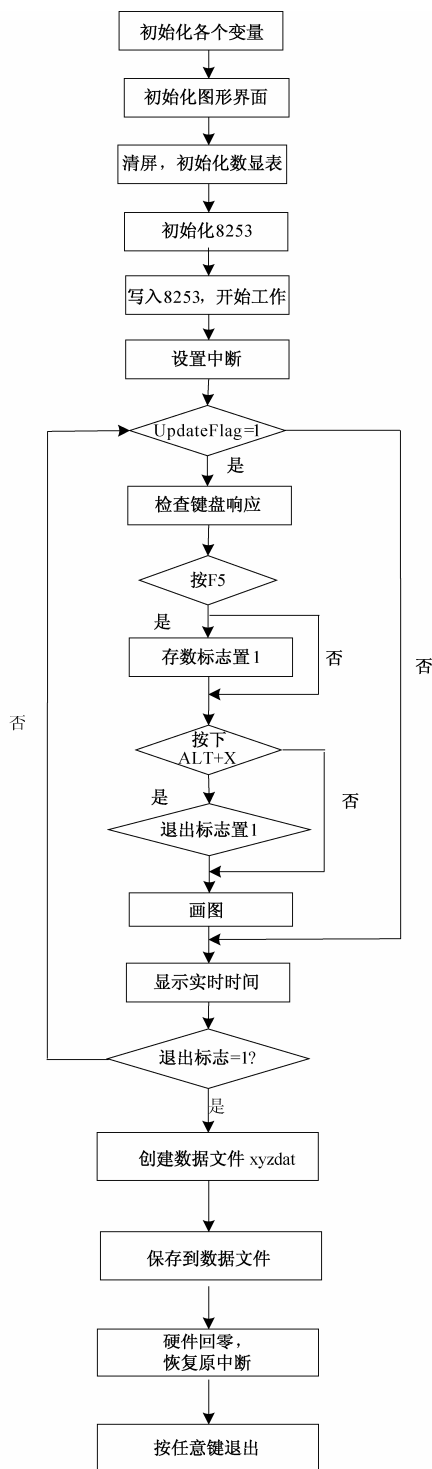
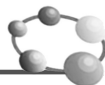


图 14-4 程序流程图

(2) 中断流程图，如图 14-5 所示。

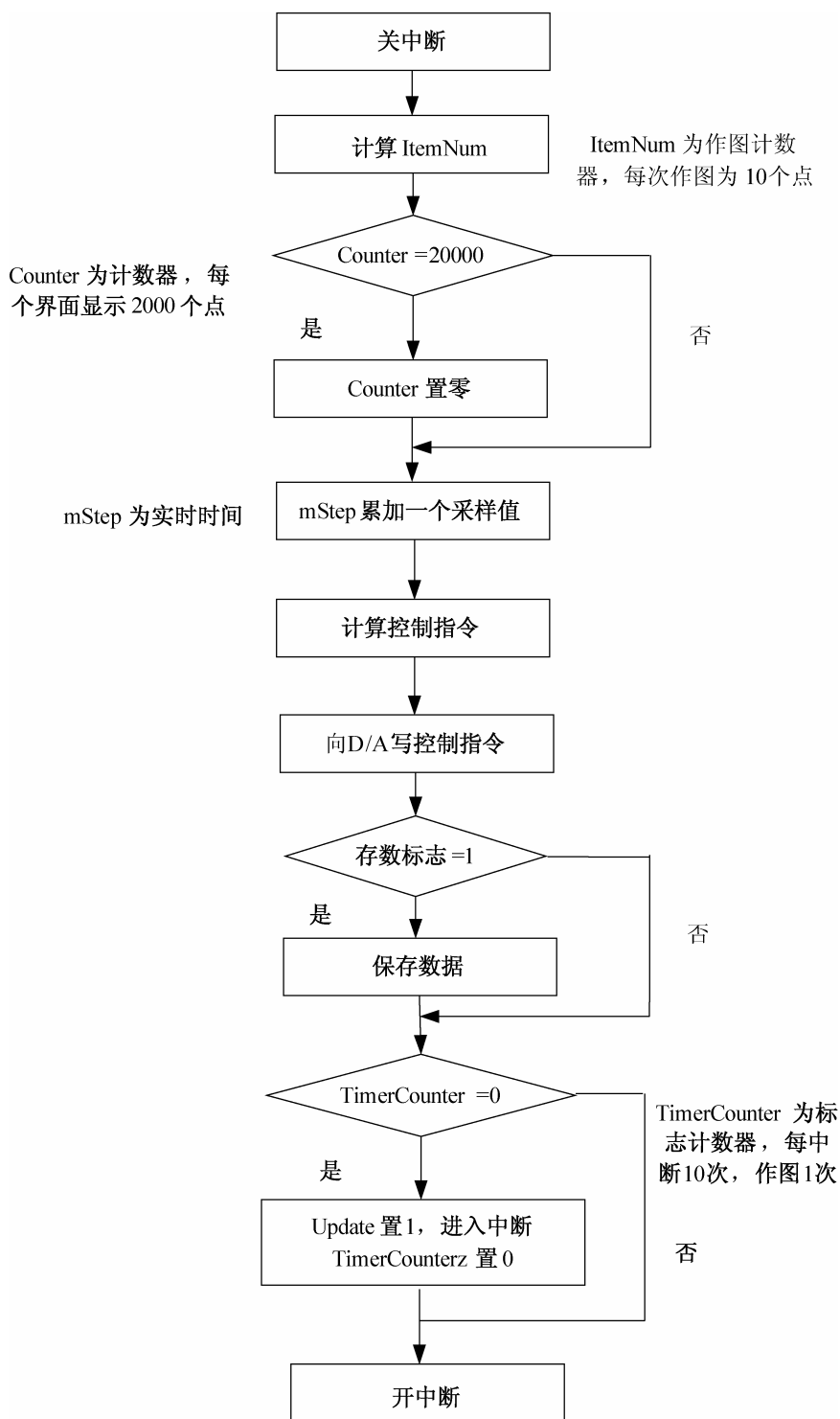


图 14-5 中断流程图

(3) 程序中变量及宏定义说明。

A 和 F 分别表示输入正弦指令的幅值和频率。channel 用来设定控制对象，当 channel=0 时，被控对象为内框；当 channel=1 时，被控对象为中框；当 channel=2 时，被控对象为外框。



channel 变化时改变 D/A 板和 D/D 板的地址,从而达到改变控制对象的目的。

TIMER_BASE 为 8253 定时器的地址, TIMER_RATIO 定义为 10, 表示每次画图画 10 个点, TIMER_CYCLE 与 TIMER_RATIO 的比值为采样周期。

TIMER_VALUE 用来定义 8253 定时器的初始计数值, 当 $N=1$ 时, 8253 定时器每隔 1ms 发一次信号, 也即采样周期为 1ms, 当 $N=10$ 时, 8253 定时器每隔 10ms 发一次信号, 也即采样周期为 10ms。

Irqnumber 为中断服务程序对应的中断向量号。

DATA_DIMENTION 表示用于存数的数组的维数, DATA_LENGTH 表示存数数组的长度, 可调整它们改变存数的数量。

T8253_MODE_0 到 T8253_MODE_5 用来表示 8253 的 5 种计数方式, 控制程序中实际使用的是 T8253_MODE_3, 在该种方式下, 8253 计数器产生周期性的方波信号, 每隔一个采样周期发出一个上升沿信号给 8259 中断计数器, 触发中断服务程序。

T8253_CHANNEL_0 到 T8253_CHANNEL_2 用来表示 8253 计数器中不同的计数寄存器, 分别与内框、中框和外框对应。

T8253_BIN_MODE 表示 8253 计数器的计数方法为使用二进制, T8253_BCD_MODE 表示 8253 计数器的计数方法为使用十进制 BCD 码。

T8253_COUNT_LOCK 表示 8253 计数器锁存, T8253_COUNT_LOW 表示 8253 计数器的写入方法为写入低 8 位, T8253_COUNT_HI 表示 8253 计数器的写入方法为写入高 8 位, T8253_LOW_FIRST 表示 8253 计数器的写入方法为先写低 8 位, 再写高 8 位。

EOI 为 8259 中断管理器的选通信号。

KB_C_N_F4 到 KB_C_A_X 表示键盘上字符对应的数值及其转化结果。

当使用的编译器为 C++ 编译器时, 将 _CPPARGS 定义为 ..., 用于中断服务程序。

(4) 整体设计思路。

每隔 0.001s 执行一次中断, 在中断中完成控制律的计算、采集实时输出信号、发出实时控制信号; 若存数标志位为 1 时, 将每次采样时的输入信号和实时位置信号存入存数数组; 每隔 0.01s 将画图标志位置为 1。

主程序中执行一个循环, 循环的跳出条件是跳出标志位为 1。在循环中, 当画图标志位被置为 1 时, 执行画图函数和键盘响应函数, 也即每隔 0.01s 画一次图, 检查一次键盘响应。当检查到 F5 时, 将存数标志置为 1, 这样在接下来的中断中可以执行存数操作; 当检查到 ALT+X 时, 将跳出标志位置 1, 这时将跳出循环。循环中一直执行显示实时时间的功能。

将存数数组中的数据存入名为 XYZ.dat 的文件中, 其中第一列位输入指令信号, 第二列位输出信号。

(5) 子函数说明。

● ResetShuxianbiao()

实现数显表的初始化, 将内框、中框和外框数显表全部清零, 通过向数显表端口写入相应的信号实现。

● ReadD_D()

采用 D/D 板实现位置信号的采集。将数显表产生的 23 位传感信号读入计算机, 并转化为相应的实际位置值。连续读入两次数显表的输出值, 当它们相等时, 认为信号稳定。读入信号与实际角位置之间的比例系数为 0.9, 乘以 0.9 后的信号单位为角秒, 要再除以 3600 转化为角度。23 位传感信号的第 23 位是符号位, 当其为 1 时, 表示该实际角位置信号为负值。



该函数的返回值为实际角度。

- Write_DA()

将控制信号送出到 D/A 板, 实现控制器的输出。

- KbGetKey()和 SetupKeyReaction()

实现键盘管理, 键盘参数在头文件 chap14_3.h 中设定。判断按键是否是 F5 或 ALT+X, 如果是 F5, 就将存数标志位置 1, 如果是 ALT+X, 就将跳出标志位置 1。数据保存由 F5 键实现, 退出由 ALT+X 键实现。

- DataSavedRoutine()

实现数据存储功能。按 F5 键则保存数据, 保存文件名为 xyz.dat。此时存数标志位是 1, 执行存数操作, 将指令信号和实际输出信号存入二位数组, 由数组长度决定存数多少, 存满为止。

- Control()

调用 ReadD_D()读入实时位置数据, 根据指令信号算出误差, 利用 PID 控制算法算出控制指令。在该函数中可以实现各种控制算法。

- DynamicDisplay()

绘制实时控制曲线, 每隔 10 个采样周期执行一次, 每次绘制 10 个点。界面采取方框的形式, 由 line()函数实现, 每五个点共用一个横坐标, 每画 2500 个点清屏一次。

中断响应是实时控制中最重要的部分。在本程序中, 中断响应采用函数 IniTimer()、interrupt()和 IrqHook()来实现, 中断时间为 1ms, 在中断时间内完成控制算法及绘图等功能, 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 chap14_3.h 中设定。

- IniTimer()

实现中断初始化。向 8253 定时器中写入控制字, 采用工作方式 3, 二进制计数, 选用计数寄存器 0, 先读低 8 位再读高 8 位。CPU 时钟频率为 1.193MHz。

- TimerIrqVect()

中断服务程序, 每隔一个采样周期执行一次。完成的任务为: 计算实时时间; 调用 Control()实现控制律; 调用 Write_DA()将控制信号送出; 调用 DataSaveRoutine()存数; 每隔 10 个采样周期将画图标志位置 1, 发 8259 中断管理器选通信号。

- IrqHook()

实现中断地址勾挂, 即保存旧的中断向量, 设置新的中断向量。

- ReleaseHardware()

恢复旧的中断向量, 调用 Write_DA()输出零控制信号, 实现控制电动机清零。

14.2.3 仿真实例

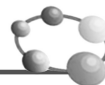
采用 C++编程, 设计一个实时控制的仿真程序, 被控对象取转台中框, 为二阶传递函数

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

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 $k_p=10, k_d=1.0$, 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。位置跟踪实时控制结果如图 14-6 所示。

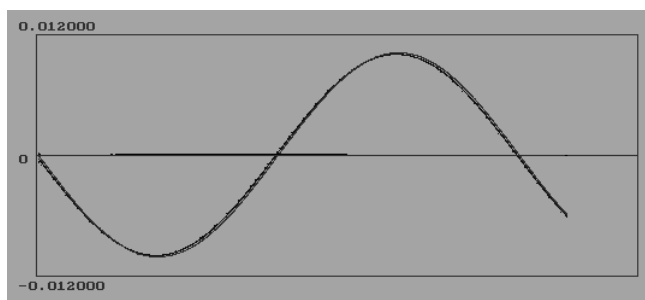


图 14-6 位置跟踪实时控制结果

仿真程序: chap14_3.cpp

```
//Real-time PID Control
//Options->Linker->Libraries->Graphics Library
//Copy c:\Borlandc\bg\*.bgi files to the current directory
//Copy the H file "chap14_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 <chap14_3.h>       //Interrupt 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;

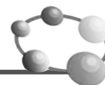
unsigned short    flagF5;
unsigned short    FinishSimulate;
FILE *xyz;

#define DD_PORT    0x380    //Inner frame
void ResetShuXianBiao()
{
    output(DD_PORT,0x0);
    output(DD_PORT,0xf);
    output(DD_PORT+4,0x0);
    output(DD_PORT+4,0xf);
    output(DD_PORT+8,0x0);
    output(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,error1;
    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;
        }
    }
}

```



chap14_3.h

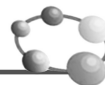
```
        if(Key==KB_C_A_X && Scan==KB_S_A_X)
        {
            FinishSimulate = 1;
        }
    }

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
        {
            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();

    //Setting control word
    OUTPORT(TIMER_BASE+3,timer_mode);
    value = TIMER_VALUE ;           //Low 8 bits
    OUTPORT( TIMER_BASE , value );
    value = TIMER_VALUE / 256 ;    //High 8 bits
    OUTPORT( 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;
    }
    OUTPORT(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,stime[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);

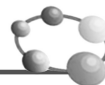
    ReleaseHardware();    //Clear all output u
    getch();    //Any key to exit
    closegraph();
    //Restores the original video mode detected by initgraph
    restorecrtmode();
    return 0;
}    //End of main{}

参数初始化程序: chap14_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 interrupt time:sampling time(ts=0.001s)

#define Irqnumber 0x08    //Realtime interrupt kind:Clock Interrupt

//Save data parameters
```



```
#define DATA_DIMENTION      2
#define DATA_LENGTH          100

//8253 timer
#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
```

附录 A 常用符号说明

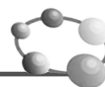
P	比例
I	积分
D	微分
T	采样时间
K	采样
A	正弦信号幅值
F	正弦信号频率
x_d, y_d	输入理想信号
y, y_{out}	输出信号
θ	角度
e, error	误差
$d e, derror$	误差变化率
k_p, k_i, k_d	PID 控制的比例、积分、微分系数
u	控制器输出
w_{ij}	神经网络权值
η	学习速率
α	惯性量
M, S	信号选择变量
n	噪声信号
d	干扰
τ	延迟时间

参 考 文 献

- [1] A.N.Gundes, A.B.Ozguler. PID Stabilization of MIMO Plants, IEEE Transactions on Automatic Control, 2007, 52(8): 1502-1508.
- [2] V.A.Oliveira, L.V.Cossi, M.C.M.Teixeira, A.M.F.Silva. Synthesis of PID controllers for a class of time delay systems, Automatica, 2009,45(7):1778-1782.
- [3] J.A.Ramirez, R.Kelly, I.Cervantes.. Semiglobal stability of saturated linear PID control for robot manipulators, Automatica, 2003, 39(6): 989-995.
- [4] J.Q.Han. From PID to Active Disturbance Rejection Control. IEEE Transactions on Industrial Electronics, 2009, 56, (3): 900-906.
- [5] Y.X.Su, B.Y.Duan, C.H.Zheng. Nonlinear PID control of a six-DOF parallel manipulator. IEE Proceedings - Control Theory and Applications, 2004, 151(1): 95- 102.
- [6] J.Chen, T.C.Huang. Applying neural networks to on-line updated PID controllers for nonlinear process control. Journal of Process Control, 2004, 14(2): 211-230, 2004.
- [7] T.K.Teng, J.S.Shieh, C.S.Chen. Genetic algorithms applied in online autotuning PID parameters of a liquid-level control system. Transactions of the Institute of Measurement and Control, 2003, 25(5): 433-450.
- [8] T.H.Kim, I.Maruta, T.Sugie. Robust PID controller tuning based on the constrained particle swarm optimization. Automatica, 2008, 44(4):1104-1110.
- [9] K.S.Tang, K.F.Man, G.Chen, S.Kwong. An Optimal Fuzzy PID Controller. IEEE Transactions on Industrial Electronics, 2001, 48(4): 757-765.
- [10] F.Zheng, Q.G.Wang, T.H Lee. On the design of multivariable PID controllers via LMI approach. Automatica, 2002, 38(3) 517-526.
- [11] K.H Ang, G.Chong, Y.Li. PID Control System Analysis, Design, and Technology. IEEE Transactions on Control Systems Technology, 2005, 13(4): 559-576.
- [12] Y.Li, K.H.Ang, G.C.Y.Chong. Patents, software and hardware for PID control: an overview and analysis of the current art. IEEE Control Systems Magazine, 2006,26(1): 42-54.
- [13] A.Datta, M.T.Ho, and S.P.Bhattacharyya. Structure and Synthesis of PID Controllers. Springer-Verlag Press, 2000.
- [14] G.J.Silva, A.Datta, S.P.Bhattacharyya. PID controllers for time-delay systems. 2005 , Boston, Birkhauser.
- [15] A.Visioli. Practical PID control. 2006, Springer-Verlag Press.
- [16] A.O.Dwyer. Handbook of PI and PID controller tuning rules. 2006, 2nd Edition, Imperial College Press.
- [17] Astrom K J . Hagglund T. Automatic Tuning of PID Controllers. Research Triangle Park, North Carolina: Instrument Society of America, 1988.



- [18] Astrom K J, Hagglund T. PID Controllers: Theory, Design, and Tuning, 2nd Edition. Research Triangle Park. North Carolina: Instrument Society of America, 1995.
- [19] Ziegler J G, Nichols N B. Optimum settings for automatic controllers. Transaction of ASME, 1942, 64:759-768.
- [20] Ho W K, Lim K W, Xu W. Optimal gain and phase margin tuning for PID controllers. Automatica, 1998, 34 (8):1009-1014.
- [21] Wang Q G, Zou B, Lee T H, Bi Q. Auto tuning of multivariable PID controllers from decentralized relay feedback. Automatica, 1997, 33 (3):319-330.
- [22] 韩京清. 非线性 PID 控制器. 自动化学报, 1994, 20 (4): 487-490.
- [23] 王广雄, 何朕. 控制系统设计. 北京: 清华大学出版社, 2008.
- [24] 肖永利, 张琛. 位置伺服系统的一类非线性 PID 调节器设计. 电气自动化, 2000, 1: 20-22.
- [25] 王新华, 刘金琨 著. 微分器设计与应用-信号滤波与求导. 北京: 电子工业出版社, 2010, 4.
- [26] 王新华, 陈增强, 袁著祉. 全程快速非线性跟踪—微分器. 控制理论与应用, 2003, 20 (6): 875-878.
- [27] A.Levant. Robust exact differentiation via sliding mode technique. Automatica, 1998, 34: 379-384.
- [28] Atsuo K, Hiroshi I, Kiyoshi S. Chattering reduction of disturbance observer based sliding mode control. IEEE Transactions on Industry Applications, 1994, 30(2): 456-461.
- [29] C.J.Kempf, S.Kobayashi. Disturbance Observer and Feedforward Design for a High-Speed Direct-Drive Positioning Table. IEEE Transactions on Control Systems Technology, 1999, 7: 513-526.
- [30] J.C.Doyle, B.Francis. A.R.Tannenbaum, Feedback Control Theory, Macmillan Publishing Co., 1992.
- [31] H. S. Lee. Robust Digital Tracking Controllers for High-Speed/High-Accuracy Positioning Systems. Ph.D. Dissertation, Mech. Eng. Dep, Univ. California, Berkeley, 1994.
- [32] 王新华, 陈增强, 袁著祉. 基于扩张观测器的非线性不确定系统输出跟踪. 控制与决策, 2004, 19(10): 1113-1116.
- [33] H. K. Khalil. Nonlinear Systems, Prentice Hall, Upper Saddle River, New Jersey, 3rd edition, 2002.
- [34] J.H. Ahrens, H.K.Khalil. High-gain observers in the presence of measurement noise: A switched-gain approach, Automatica, 2009, 45: 936-943.
- [35] Xinhua Wang, Jinkun Liu, Kai-Yuan Cai. Tracking control for a velocity-sensorless VTOL aircraft with delayed outputs, Automatica, 2009, 45: 2876-2882.
- [36] 韩京清, 袁露林. 跟踪微分器的离散形式. 系统科学与数学, 1999, 19(3): 268-273.
- [37] 韩京清. 自抗扰控制技术. 北京: 国防工业出版社, 2008.
- [38] 韩京清. 从 PID 技术到自抗扰控制技术. 控制工程, 2002, 9(3): 13-18.
- [39] 胡庆雷. 挠性航天器姿态机动的主动控制[D]. 哈尔滨: 哈尔滨工业大学, 2006.
- [40] S.S.Ge, T.H.Lee, C.J.Harris. Adaptive Neural Network Control of Robotic Manipulators. World Scientific, London, 1998.
- [41] A.S.Hodel, C.E.Hall. Variable-structure PID Control to prevent integrator windup, IEEE



- Transactions on Industrial Electronics. 2001, 48(2): 442-451.
- [42] 刘强, 扈宏杰, 刘金琨, 尔联洁. 高精度飞行仿真转台的鲁棒自适应控制. 系统工程与电子技术, 2001, 23 (10): 35-38.
- [43] P.T.Chan, A.B.Rad, K.M.Tsang. Optimization of fused fuzzy systems via genetic algorithms. IEEE Transactions on Industrial Electronics, 2002, 49(3):685-692.
- [44] J.Park, I.W.Sandberg. Universal approximation using radial basis function networks, Neural Computation. 1990, 3: 246-257.
- [45] 周明, 孙树栋. 遗传算法原理及应用. 北京: 国防工业出版社, 1999.
- [46] C. Canudas de Wit, H. Olsson, K. J.Astrom, and P. Lischinsk. A new model for control of systems with friction. IEEE trans. Automatic Control, 1995, 40(3): 419-425.
- [47] Karnopp D.Comput. Computer Simulation of Stick-slip Friction in Mechanical Dynamic Systems. Journal of Dynamic Systems, Measurement and Control, 1985, 107: 100-103.
- [48] 尔联洁. 自动控制系统. 北京: 航空工业出版社, 1994.
- [49] 冯国楠. 现代伺服系统的分析与设计. 北京: 机械工业出版社, 1990.
- [50] Arimoto S, Kawamura S. Miyazaki F. Bettering Operation of robotics by leaning. Journal of Robotic System. 1984, 1(2): 123~140.
- [51] 孙明轩, 黄宝健. 迭代学习控制. 北京: 国防工业出版社, 1999.
- [52] 谢胜利. 迭代学习控制的理论与应用. 北京: 科学出版社, 2005.
- [53] 刘金琨. 《机器人控制系统的设计与 MATLAB 仿真》, 北京: 清华大学出版社, 2008.
- [54] L.Xu, B. Yao. Adaptive robust control of mechanical systems with non-linear dynamic friction compensation. International Journal of control, February 2008, 81(2): 167-176.
- [55] 申铁龙. H_∞ 控制理论及应用. 北京: 清华大学出版社, 1996.
- [56] 俞立著. 鲁棒控制-线性矩阵不等式处理方法. 北京: 清华大学出版社, 2002.
- [57] 罗华飞. MATLAB GUI 设计学习手记. 北京: 北京航空航天大学出版社, 2009.
- [58] 薛定宇, 控制系统计算机辅助设计. 北京: 清华大学出版社, 1996.
- [59] 王立新 王迎军译. 模糊系统与模糊控制教程. 北京: 清华大学出版社, 2003.
- [60] 王宁, 涂健. 电渣重熔过程的神经元智能控制, 自动化学报, 1993, 19 (5): 634-636.
- [61] 张建明, 王宁, 王树青. PID 自适应调整增益的神经元非模型控制. 机电工程, 1999, 16 (5): 72-73.
- [62] 王耀南. 智能控制系统. 长沙: 湖南大学出版社, 1996.
- [63] Slotine J. E., Li W. 著, 程代展译. 应用非线性控制. 北京: 机械工业出版社, 2006.
- [64] 张建民, 王涛, 王忠礼 编著. 智能控制原理及应用. 北京: 冶金工业出版社, 2003.
- [65] 王永骥, 涂健 编著. 神经网络控制. 北京: 机械工业出版社, 1998.
- [66] 舒怀林 编著. PID 神经网络及其控制系统. 北京: 国防工业出版社, 2006.
- [67] 王顺晃, 舒迪前 编著. 智能控制系统及应用. 北京: 机械工业出版社, 1995.
- [68] 傅信 主编. 过程计算机控制系统. 西安: 西北工业大学出版社, 1995.
- [69] 张绍德. 使用 NCD 模块优化 PID 调节器参数, 自动化与仪器仪表, 2003, 3: 52-53.
- [70] 陈剑桥, 非线性 PID 控制器的计算机辅助设计, 扬州职业大学学报, 2001, 5(4): 12-15.
- [71] Peter H. Windup in control: its effects and their prevention Springer, 2006.

免费下载仿真程序使用说明

1. 为方便读者学习，将本书所有相关程序免费赠送给读者，请读者到网站：<http://yydz.phei.com.cn> 的“资源下载”栏目（具体网址为：http://yydz.phei.com.cn/html/download_List_1.html）或网址 <http://si.buaa.edu.cn> 中的“其他资料下载（Download）”下载相关电子文件。在下载过程中若遇到问题请与本书策划编辑赵丽松联系，E-mail: zls@phei.com.cn，电话 010-88254452。

2. 所有控制算法按章归类，程序名与书中一一对应。

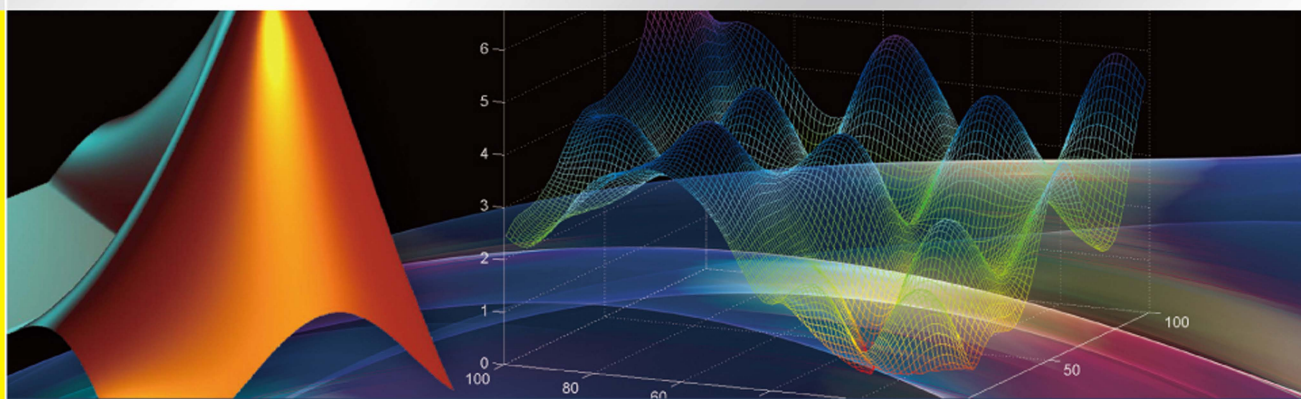
3. 将下载的仿真程序复制到硬盘 MATLAB 运行的路径中，便可仿真运行。

4. 基于 NCD 优化的 PID 控制在 MATLAB 6.5 版下运行成功；其余所有算法均在 MATLAB 7.1 版下运行成功，并适用于其他更高级版本。

5. 假如您对算法和仿真程序有疑问，请通过 E-mail 与作者联系。

北京航空航天大学 刘金琨

E-mail 地址: ljik@buaa.edu.cn



先进PID控制 MATLAB 仿真 (第3版)

免费下载资源!

- 为方便读者学习，特免费提供本书相关仿真程序的下载资源，
下载网址：<http://yydz.phei.com.cn>的“资源下载”栏目，
或：<http://si.buaa.edu.cn>“其他资料下载 (Download)”。



策划编辑：赵丽松
责任编辑：李雪梅
封面设计：徐海燕

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

ISBN 978-7-121-13049-6



9 787121 130496 >