

GOTOP

北京科海培训中心

# MATLAB 5.X 应用与技巧

蒙以正 编著

柳承茂 审校

科学出版社

GOTOP

北京科海培训中心

# MATLAB 5.X 应用与技巧

蒙以正 编著

柳承茂 审校

科学出版社

## 内 容 简 介

本书全面深入地阐述了 MATLAB 5.x 的系统功能与应用技巧。

全书共分 6 章,内容包括:基本入门知识、数学指令的应用、LMI 应用、SIMULINK 的设计、S-function 的设计、模糊理论应用与设计技巧、图形用户界面设计及图形处理技巧等。书中的重点在于范例的应用与设计技巧,以简明的图形大量的程序说明使读者达到活学活用的目的。

本书既适合于广大初学者理解 MATLAB 的强大功能,也可以满足具有一定水平的中、高级用户学习编程的迫切要求。

## 版 权 声 明

本书为著者资讯股份有限公司独家授权的中文简体字版本。本书专有出版权属北京科学培训中心与科学出版社所有。在没有得到本书原版出版者和本书出版者的书面许可时,任何单位和个人不得擅自摘抄、复制本书的一部分或全部内容,不得以任何形式(包括资料和出版物)进行传播。

本书原版权属于著者资讯股份有限公司。

**版权所有,侵权必究**

## 图书在版编目(CIP)数据

MATLAB 5.X 应用与技巧/蒙以正编著. —北京:

科学出版社,1999.11

ISBN 7-03-008004-1

I. M… I. 蒙… III. 计算机辅助计算-软件包,MATLAB  
5.x N. TP391.75

中国版本图书馆 CIP 数据核字(1999)第 65158 号

**图字:01-1999-2094**

科 学 出 版 社 出 版

北京东黄城根北街 16 号

邮政编码:100717

北京市朝阳区科普印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

\*

1999 年 11 月 第 一 版

开本:787×1092 1/16

1999 年 11 月第一次印刷

印张:16

印数:1—5 000

字数:243 200

**定价:25.00 元**

## 内 容 简 介

本书全面深入地阐述了 MATLAB 5.x 的系统功能与应用技巧。

全书共分 6 章,内容包括:基本入门知识、数学指令的应用、LMI 应用、SIMULINK 的设计、S-function 的设计、模糊理论应用与设计技巧、图形用户界面设计及图形处理技巧等。书中的重点在于范例的应用与设计技巧,以简明的图形大量的程序说明使读者达到活学活用的目的。

本书既适合于广大初学者理解 MATLAB 的强大功能,也可以满足具有一定水平的中、高级用户学习编程的迫切要求。

## 版 权 声 明

本书为著者资讯股份有限公司独家授权的中文简体字版本。本书专有出版权属北京科学培训中心与科学出版社所有。在没有得到本书原版出版者和本书出版者的书面许可时,任何单位和个人不得擅自摘抄、复制本书的一部分或全部内容,不得以任何形式(包括资料和出版物)进行传播。

本书原版权属于著者资讯股份有限公司。

**版权所有,侵权必究**

## 图书在版编目(CIP)数据

MATLAB 5.X 应用与技巧/蒙以正编著. —北京:

科学出版社,1999.11

ISBN 7-03-008004-1

I. M… I. 蒙… III. 计算机辅助计算-软件包,MATLAB  
5.x N. TP391.75

中国版本图书馆 CIP 数据核字(1999)第 65158 号

**图字:01-1999-2094**

科 学 出 版 社 出 版

北京东黄城根北街 16 号

邮政编码:100717

北京市朝阳区科普印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

\*

1999 年 11 月 第 一 版

开本:787×1092 1/16

1999 年 11 月 第一次印刷

印张:16

印数:1—5 000

字数:243 200

**定价:25.00 元**

# 序

编写这本书的动机,源于目前市面上找不到有关MATLAB 5.x的图形用户界面(GUI),图形处理(Handle Graphics)方面的书籍,由于工作需要只能靠自己研读手册或查找英文书籍,因而萌生出想把学习MATLAB的心得与大家分享的愿望。所以本书的重点在于范例的应用与设计的技巧,并介绍具有代表性的指令格式,至于理论的论述则重在介绍控制系统、非线性系统(Nonlinear System)与模糊控制(Fuzzy Control)的体会,经过整理去除繁杂的部分,以简单明了的方式展现出来给读者。

本书内容主要针对5.1版,而目前市面上的书籍大多数是4.x版,尽管命令没有很大的差异,但是在功能与画图方面其能力都增强了许多,并且增加联机查询Help Desk的功能,可以对不懂的命令即时在电脑上得到说明,或者通过网络向Math Works公司咨询。本书内容涵盖了MATLAB 5.x版的数学指令的应用、LMI应用、SIMULINK的设计、S-function的设计、Fuzzy设计、ANFIS设计、GUI的设计方法、图形处理技巧(Handle Graphics)等。全书着重范例的推导,以简明的图形、程序说明与循序渐进的范例学习,使读者达到活学活用的目的。下面为各章内容的重点:

第1章是入门部分,学习如何使用MATLAB及正常操作,并介绍Help的用法。

第2章的主要目的是希望就设计上所需要的理论基础知识和常用指令加以说明,以期读者在学习与工作中能够节省额外摸索的时间,并具备独立思考及程序设计的能力。本章从最基本的命令开始,渐进到各种命令的应用。相信读完本章,您已经知道如何灵活地去使用及查寻命令,以设计出完美的程序。

第3章介绍SIMULINK,它是模拟建构模型的重要工具,可以使您设计的程序如虎添翼。学习SIMULINK最需要掌握的是S-function的设计,本章不仅详细地作了介绍,同时还单独地介绍了4.x版的S-function的设计方法,以供读者学习比较。

第4章为模糊理论应用及设计技巧,这是进一步提高的内容,读者可从中了解到理论用于实践的意义。

第5章为图形用户界面(GUI)设计,学习本章可使您的设计能力大增,用GUI的控制面板,可以设计出生动、使用方便且富有表现力的图形,使您的研究成果更具说服力。

第6章为图形处理技巧,我们知道MATLAB最强的就是绘图功能,本章教您如何使用坐标、如何在图形上显示文字,以及如何美化图形的方法。

笔者有幸受教于台湾元智大学韩光渭教授、林志民教授,书中部分范例取自其授课教材,并获两位教授同意出版,在此谨以本书献给恩师,以表达诚挚的谢意。最后谢谢读者的支持,若有疏漏之处,尚祈见谅并请赐指正意见。谢谢!

(E mail:typhon@tpts6.seed.net.tw)

# 序

编写这本书的动机,源于目前市面上找不到有关MATLAB 5.x的图形用户界面(GUI),图形处理(Handle Graphics)方面的书籍,由于工作需要只能靠自己研读手册或查找英文书籍,因而萌生出想把学习MATLAB的心得与大家分享的愿望。所以本书的重点在于范例的应用与设计的技巧,并介绍具有代表性的指令格式,至于理论的论述则重在介绍控制系统、非线性系统(Nonlinear System)与模糊控制(Fuzzy Control)的体会,经过整理去除繁杂的部分,以简单明了的方式展现出来给读者。

本书内容主要针对5.1版,而目前市面上的书籍大多数是4.x版,尽管命令没有很大的差异,但是在功能与画图方面其能力都增强了许多,并且增加联机查询Help Desk的功能,可以对不懂的命令即时在电脑上得到说明,或者通过网络向Math Works公司咨询。本书内容涵盖了MATLAB 5.x版的数学指令的应用、LMI应用、SIMULINK的设计、S-function的设计、Fuzzy设计、ANFIS设计、GUI的设计方法、图形处理技巧(Handle Graphics)等。全书着重范例的推导,以简明的图形、程序说明与循序渐进的范例学习,使读者达到活学活用的目的。下面为各章内容的重点:

第1章是入门部分,学习如何使用MATLAB及正常操作,并介绍Help的用法。

第2章的主要目的是希望就设计上所需要的理论基础知识和常用指令加以说明,以期读者在学习与工作中能够节省额外摸索的时间,并具备独立思考及程序设计的能力。本章从最基本的命令开始,渐进到各种命令的应用。相信读完本章,您已经知道如何灵活地去使用及查寻命令,以设计出完美的程序。

第3章介绍SIMULINK,它是模拟建构模型的重要工具,可以使您设计的程序如虎添翼。学习SIMULINK最需要掌握的是S-function的设计,本章不仅详细地作了介绍,同时还单独地介绍了4.x版的S-function的设计方法,以供读者学习比较。

第4章为模糊理论应用及设计技巧,这是进一步提高的内容,读者可从中了解到理论用于实践的意义。

第5章为图形用户界面(GUI)设计,学习本章可使您的设计能力大增,用GUI的控制面板,可以设计出生动、使用方便且富有表现力的图形,使您的研究成果更具说服力。

第6章为图形处理技巧,我们知道MATLAB最强的就是绘图功能,本章教您如何使用坐标、如何在图形上显示文字,以及如何美化图形的方法。

笔者有幸受教于台湾元智大学韩光渭教授、林志民教授,书中部分范例取自其授课教材,并获两位教授同意出版,在此谨以本书献给恩师,以表达诚挚的谢意。最后谢谢读者的支持,若有疏漏之处,尚祈见谅并请赐指正意见。谢谢!

(E mail:typhon@tpts6.seed.net.tw)







# 目 录

<b>第1章 MATLAB入门</b>	<b>1</b>
1.1 MATLAB概述	1
1.2 Help的使用方法	3
1.3 范例的演示(摘自MathWorks公司的范例)	6
<b>第2章 MATLAB程序设计</b>	<b>9</b>
2.1 MATLAB的基本设计	9
2.1.1 设计原则	9
2.1.2 设置MATLAB的工作路径	10
2.1.3 声明子程序的变量	11
2.1.4 程序的运算符	13
2.1.5 利用字符串模拟运算式	13
2.1.6 调试器的运用	15
2.2 程序流程控制	16
2.2.1 for 循环	17
2.2.2 if then与while语句	18
2.2.3 switch语句	19
2.3 系统分析技巧	20
2.3.1 文件读取与处理	20
2.3.2 字符数据的处理	25
2.3.3 数据的频谱分析	27
2.3.4 系统稳定性分析	28
2.4 多项式的求解技巧	33
2.4.1 多项式的建立与表示法	33
2.4.2 多项式的运算	34
2.4.3 多项式的拟合	36
2.4.4 多项式的插值(Interpolation)	37
2.5 矩阵运算技巧	40
2.5.1 矩阵的建立与表示法	40
2.5.2 矩阵的四则运算与转置	42
2.5.3 逆矩阵与行列式的求解	43
2.5.4 矩阵的次方运算	45
2.5.5 矩阵的分解	46
2.5.6 矩阵应用	48

# 目 录

<b>第1章 MATLAB入门</b>	<b>1</b>
1.1 MATLAB概述	1
1.2 Help的使用方法	3
1.3 范例的演示(摘自MathWorks公司的范例)	6
<b>第2章 MATLAB程序设计</b>	<b>9</b>
2.1 MATLAB的基本设计	9
2.1.1 设计原则	9
2.1.2 设置MATLAB的工作路径	10
2.1.3 声明子程序的变量	11
2.1.4 程序的运算符	13
2.1.5 利用字符串模拟运算式	13
2.1.6 调试器的运用	15
2.2 程序流程控制	16
2.2.1 for 循环	17
2.2.2 if then与while语句	18
2.2.3 switch语句	19
2.3 系统分析技巧	20
2.3.1 文件读取与处理	20
2.3.2 字符数据的处理	25
2.3.3 数据的频谱分析	27
2.3.4 系统稳定性分析	28
2.4 多项式的求解技巧	33
2.4.1 多项式的建立与表示法	33
2.4.2 多项式的运算	34
2.4.3 多项式的拟合	36
2.4.4 多项式的插值(Interpolation)	37
2.5 矩阵运算技巧	40
2.5.1 矩阵的建立与表示法	40
2.5.2 矩阵的四则运算与转置	42
2.5.3 逆矩阵与行列式的求解	43
2.5.4 矩阵的次方运算	45
2.5.5 矩阵的分解	46
2.5.6 矩阵应用	48

2.5.7	利用LMI工具箱解矩阵不等式 .....	51
2.5.8	矩阵方程式的求解 .....	57
2.5.9	矩阵内容的查找与修改 .....	57
2.6	常微分方程ODE设计技巧 .....	58
2.6.1	ODE解题器的指令格式 .....	58
2.6.2	编写ODE文件 .....	59
2.6.3	查看解题器的性能(performance) .....	61
2.6.4	利用odeset指令改变解题器的设定参数 .....	62
2.6.5	用常微分方程ODE解非线性问题 .....	66
2.7	积分方程的设计技巧 .....	68
2.7.1	积分方程的指令格式 .....	68
2.7.2	设计步骤 .....	68
<b>第3章</b>	<b>SIMULINK设计 .....</b>	<b>70</b>
3.1	SIMULINK概述 .....	70
3.2	功能模块的处理 .....	79
3.3	线的处理 .....	83
3.4	自定义功能模块 .....	84
3.4.1	Icon标签页 .....	88
3.4.2	Initialization 标签页 .....	92
3.4.3	Documentation标签页 .....	100
3.4.4	Unmask标签页 .....	101
3.5	模拟参数的设定 .....	102
3.5.1	解题器(Solver) 参数的设定 .....	102
3.5.2	工作空间(Workspace)参数的设定 .....	104
3.6	在命令行中执行SIMULINK .....	105
3.7	模型的线性化 .....	107
3.8	模型的状态稳定平衡点 .....	111
3.9	建立可以激活与触发的功能模块 .....	112
3.10	S-function的设计 .....	119
3.11	4.x版的S-function设计 .....	135
<b>第4章</b>	<b>Fuzzy Toolbox设计 .....</b>	<b>139</b>
4.1	Fuzzy概述 .....	139
4.2	建立Fuzzy推论系统 .....	140
4.3	Fuzzy推论系统的调试 .....	144
4.4	Fuzzy推论系统的模拟 .....	146
4.5	fuzzy常用指令介绍 .....	157
4.6	ANFIS指令的应用 .....	160

2.5.7	利用LMI工具箱解矩阵不等式 .....	51
2.5.8	矩阵方程式的求解 .....	57
2.5.9	矩阵内容的查找与修改 .....	57
2.6	常微分方程ODE设计技巧 .....	58
2.6.1	ODE解题器的指令格式 .....	58
2.6.2	编写ODE文件 .....	59
2.6.3	查看解题器的性能(performance) .....	61
2.6.4	利用odeset指令改变解题器的设定参数 .....	62
2.6.5	用常微分方程ODE解非线性问题 .....	66
2.7	积分方程的设计技巧 .....	68
2.7.1	积分方程的指令格式 .....	68
2.7.2	设计步骤 .....	68
<b>第3章</b>	<b>SIMULINK设计 .....</b>	<b>70</b>
3.1	SIMULINK概述 .....	70
3.2	功能模块的处理 .....	79
3.3	线的处理 .....	83
3.4	自定义功能模块 .....	84
3.4.1	Icon标签页 .....	88
3.4.2	Initialization 标签页 .....	92
3.4.3	Documentation标签页 .....	100
3.4.4	Unmask标签页 .....	101
3.5	模拟参数的设定 .....	102
3.5.1	解题器(Solver) 参数的设定 .....	102
3.5.2	工作空间(Workspace)参数的设定 .....	104
3.6	在命令行中执行SIMULINK .....	105
3.7	模型的线性化 .....	107
3.8	模型的状态稳定平衡点 .....	111
3.9	建立可以激活与触发的功能模块 .....	112
3.10	S-function的设计 .....	119
3.11	4.x版的S-function设计 .....	135
<b>第4章</b>	<b>Fuzzy Toolbox设计 .....</b>	<b>139</b>
4.1	Fuzzy概述 .....	139
4.2	建立Fuzzy推论系统 .....	140
4.3	Fuzzy推论系统的调试 .....	144
4.4	Fuzzy推论系统的模拟 .....	146
4.5	fuzzy常用指令介绍 .....	157
4.6	ANFIS指令的应用 .....	160

<b>第5章 图形用户界面(GUI)的应用 .....</b>	<b>168</b>
5.1 GUI概述 .....	168
5.2 辅助控制面板 .....	169
5.2.1 Property编辑器 .....	169
5.2.2 Callback编辑器 .....	173
5.2.3 Alignment工具 .....	173
5.2.4 Menu编辑器 .....	174
5.2.5 辅助控制列表 .....	175
5.2.6 新建对象面板 .....	176
5.3 初级设计技巧 .....	177
5.4 对象处理指令介绍 .....	180
5.5 动态图形设计技巧 .....	183
5.6 列表框(Listbox)设计技巧 .....	186
5.7 充分利用ButtonDownFcn指令 .....	188
5.8 鼠标属性指令设计技巧 .....	189
5.9 Menu设计技巧 .....	192
5.10 中文对象设计 .....	194
5.11 Slider及Frame的设计技巧 .....	195
5.12 Checkbox设计技巧 .....	198
<b>第6章 图形处理技巧 .....</b>	<b>201</b>
6.1 基本的平面绘图 .....	201
6.1.1 plot的指令格式 .....	201
6.1.2 axis指令的格式 .....	202
6.1.3 标示坐标刻度 .....	203
6.1.4 文字标示 .....	204
6.1.5 legend指令的格式 .....	208
6.1.6 num2str指令的格式 .....	209
6.1.7 fill指令的格式 .....	211
6.1.8 subplot的用法 .....	211
6.1.9 fplot指令的格式 .....	212
6.1.10 应用绘图指令 .....	213
6.1.11 向量绘图指令 .....	220
6.1.12 其他绘图指令 .....	222
6.2 基本的三维绘图 .....	222
6.2.1 产生三维图的参考点 .....	223
6.2.2 plot3 指令的格式 .....	223
6.2.3 mesh及meshz的指令格式 .....	224

<b>第5章 图形用户界面(GUI)的应用 .....</b>	<b>168</b>
5.1 GUI概述 .....	168
5.2 辅助控制面板 .....	169
5.2.1 Property编辑器 .....	169
5.2.2 Callback编辑器 .....	173
5.2.3 Alignment工具 .....	173
5.2.4 Menu编辑器 .....	174
5.2.5 辅助控制列表 .....	175
5.2.6 新建对象面板 .....	176
5.3 初级设计技巧 .....	177
5.4 对象处理指令介绍 .....	180
5.5 动态图形设计技巧 .....	183
5.6 列表框(Listbox)设计技巧 .....	186
5.7 充分利用ButtonDownFcn指令 .....	188
5.8 鼠标属性指令设计技巧 .....	189
5.9 Menu设计技巧 .....	192
5.10 中文对象设计 .....	194
5.11 Slider及Frame的设计技巧 .....	195
5.12 Checkbox设计技巧 .....	198
<b>第6章 图形处理技巧 .....</b>	<b>201</b>
6.1 基本的平面绘图 .....	201
6.1.1 plot的指令格式 .....	201
6.1.2 axis指令的格式 .....	202
6.1.3 标示坐标刻度 .....	203
6.1.4 文字标示 .....	204
6.1.5 legend指令的格式 .....	208
6.1.6 num2str指令的格式 .....	209
6.1.7 fill指令的格式 .....	211
6.1.8 subplot的用法 .....	211
6.1.9 fplot指令的格式 .....	212
6.1.10 应用绘图指令 .....	213
6.1.11 向量绘图指令 .....	220
6.1.12 其他绘图指令 .....	222
6.2 基本的三维绘图 .....	222
6.2.1 产生三维图的参考点 .....	223
6.2.2 plot3 指令的格式 .....	223
6.2.3 mesh及meshz的指令格式 .....	224

---

6.2.4	surf指令的格式 .....	226
6.2.5	contour, contour3及contourf的指令格式.....	228
6.2.6	quiver及quiver3的用法 .....	231
6.2.7	meshc与surf的格式 .....	233
6.2.8	cylinder与sphere的格式 .....	234
6.2.9	设定视角 .....	237
6.2.10	light的指令格式 .....	237
6.3	设定图形属性 .....	240
6.3.1	设定双坐标轴.....	240
6.3.2	设定坐标轴颜色.....	241
6.3.3	设定坐标轴其他属性.....	242
6.3.4	对象中的x, y, z数据的处理.....	245

---

6.2.4	surf指令的格式 .....	226
6.2.5	contour, contour3及contourf的指令格式.....	228
6.2.6	quiver及quiver3的用法 .....	231
6.2.7	meshc与surfc的格式 .....	233
6.2.8	cylinder与sphere的格式 .....	234
6.2.9	设定视角 .....	237
6.2.10	light的指令格式 .....	237
6.3	设定图形属性 .....	240
6.3.1	设定双坐标轴.....	240
6.3.2	设定坐标轴颜色.....	241
6.3.3	设定坐标轴其他属性.....	242
6.3.4	对象中的x, y, z数据的处理.....	245



# 第1章 MATLAB 入门

## 1.1 MATLAB概述

本书所使用的MATLAB为5.1版。5.1版与4.x版最大的不同就是增强了Debug、Help功能,另外也提供了很多新指令(如绘图的rotate3d指令),并且路径的设置方式也更加方便快捷(参见2.1.2节)。对于用4.x版编写的指令,会发出警告信息请用户进行更改,例如在4.x版中,用rk45编写的指令就会被要求改写为sim,但是如果不改同样也可以顺利执行,所以读者不必担心这方面的问题。

假如你已经安装了MATLAB软件,并将本书配套的磁盘内容复制到硬盘(参考磁盘使用说明),下面就可以开始学习了。用鼠标右击MATLAB的图标,出现快捷菜单后选择“属性”,如图1.1所示,接着就会出现“起始位置”的输入框,如图1.2所示。

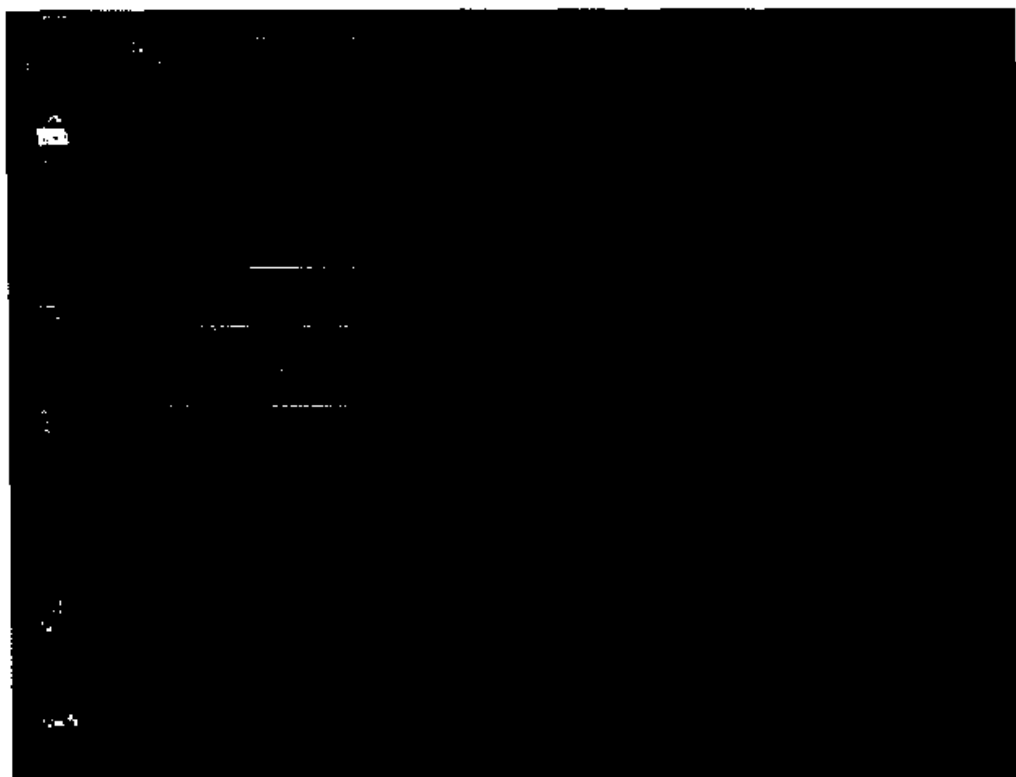


图 1.1 MATLAB 快捷菜单

“起始位置”框内的路径就是MATLAB启动后所在的位置,如果不在set path中进行修改,打开文件的默认目录就是该目录。如果用到的程序都在chp2目录下,就可以将起始位置改为:

c:\chp2

# 第1章 MATLAB 入门

## 1.1 MATLAB概述

本书所使用的MATLAB为5.1版。5.1版与4.x版最大的不同就是增强了Debug、Help功能,另外也提供了很多新指令(如绘图的rotate3d指令),并且路径的设置方式也更加方便快捷(参见2.1.2节)。对于用4.x版编写的指令,会发出警告信息请用户进行更改,例如在4.x版中,用rk45编写的指令就会被要求改写为sim,但是如果不改同样也可以顺利执行,所以读者不必担心这方面的问题。

假如你已经安装了MATLAB软件,并将本书配套的磁盘内容复制到硬盘(参考磁盘使用说明),下面就可以开始学习了。用鼠标右击MATLAB的图标,出现快捷菜单后选择“属性”,如图1.1所示,接着就会出现“起始位置”的输入框,如图1.2所示。

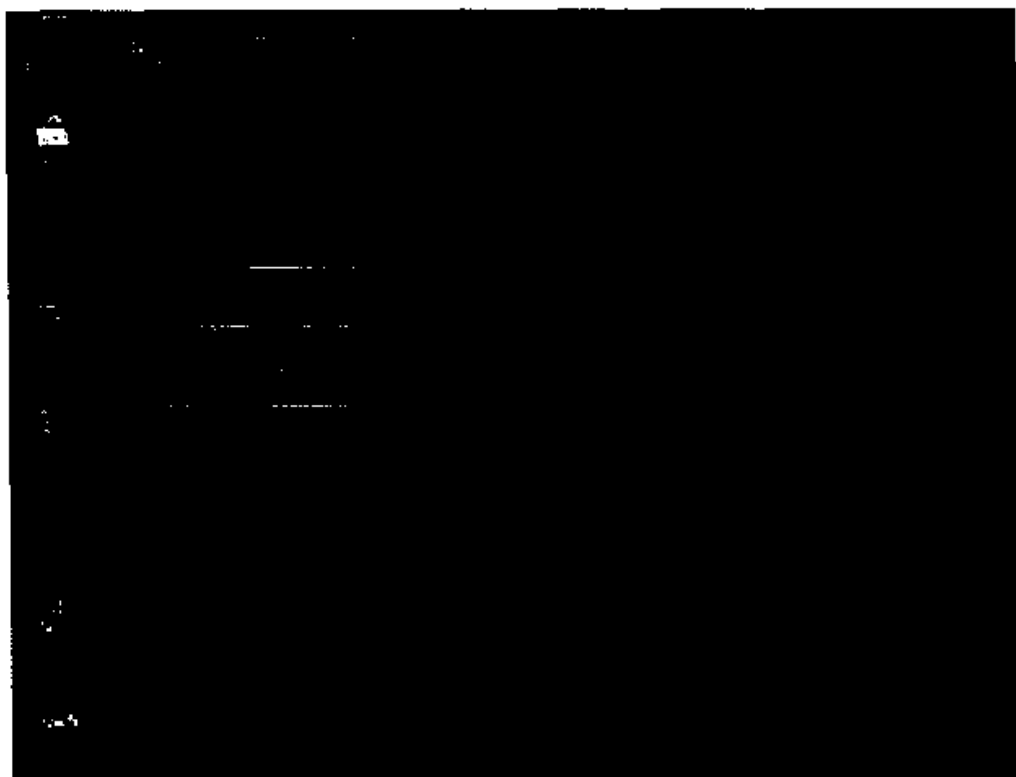


图 1.1 MATLAB 快捷菜单

“起始位置”框内的路径就是MATLAB启动后所在的位置,如果不在set path中进行修改,打开文件的默认目录就是该目录。如果用到的程序都在chp2目录下,就可以将起始位置改为:

c:\chp2

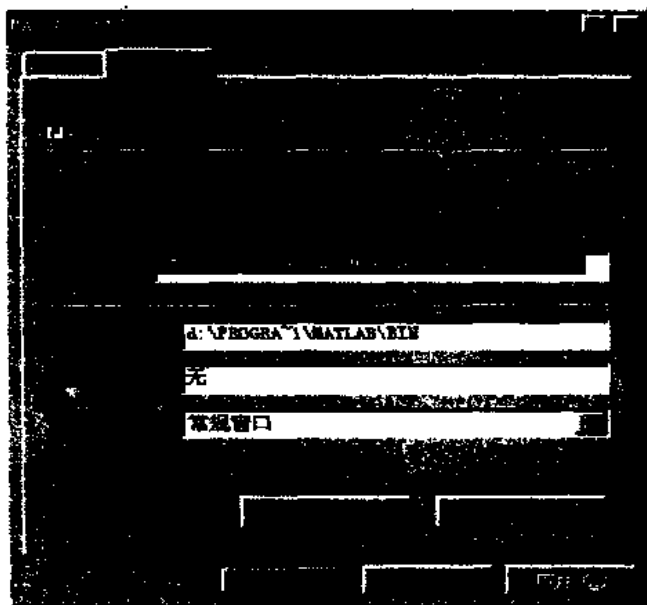


图 1.2 设定起始位置

这样在下次执行时，MATLAB的当前路径就会自动设定为c:\chp2。双击MATLAB图标后，出现的MATLAB窗口称为命令窗口（Command Window），如图1.3所示，是供用户执行Script指令(参见2.1.1节)和显示执行结果信息的区域。窗口上方有进行文件操作的按钮和菜单，包括File, Edit, Window和Help, 可以执行文件的打开、保存、编辑等操作。下面开始尝试执行MATLAB的指令，在命令窗口中执行：

ex235

结果会出现5张图。如果要查看程序的内容，就单击图1.3中的“打开”按钮，会出现如图1.4所示的Open窗口，双击ex235即可。这时会出现Editor/Debugger（编辑调试）窗口编辑调试，显示程序的内容，如图1.5所示。第2章以后所讨论的程序内容均是在该窗口中开发的，文件格式为.m文件格式。

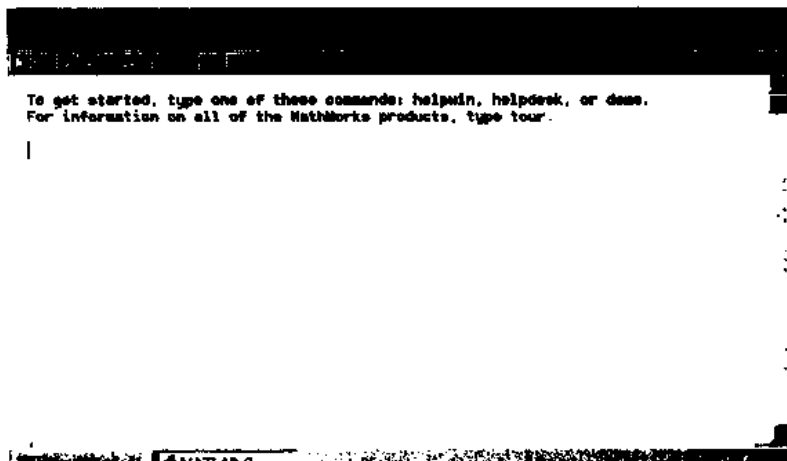


图 1.3 MATLAB 的命令窗口

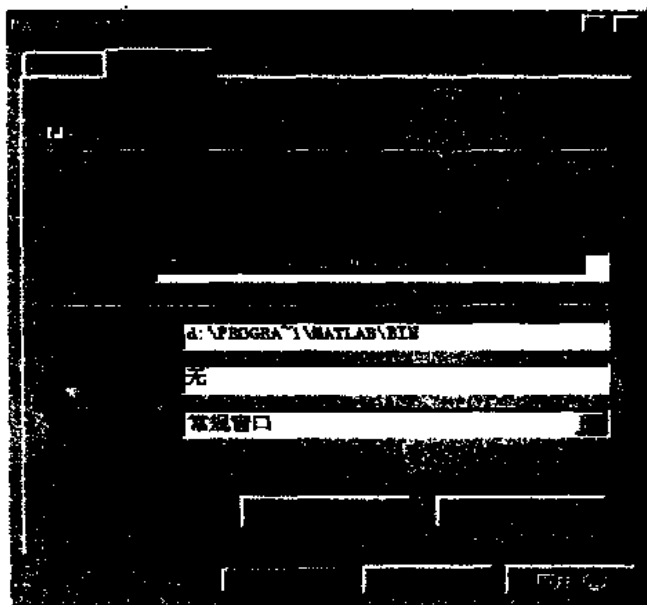


图 1.2 设定起始位置

这样在下次执行时，MATLAB的当前路径就会自动设定为c:\chp2。双击MATLAB图标后，出现的MATLAB窗口称为命令窗口（Command Window），如图1.3所示，是供用户执行Script指令(参见2.1.1节)和显示执行结果信息的区域。窗口上方有进行文件操作的按钮和菜单，包括File, Edit, Window和Help, 可以执行文件的打开、保存、编辑等操作。下面开始尝试执行MATLAB的指令，在命令窗口中执行：

ex235

结果会出现5张图。如果要查看程序的内容，就单击图1.3中的“打开”按钮，会出现如图1.4所示的Open窗口，双击ex235即可。这时会出现Editor/Debugger（编辑调试）窗口编辑调试，显示程序的内容，如图1.5所示。第2章以后所讨论的程序内容均是在该窗口中开发的，文件格式为.m文件格式。

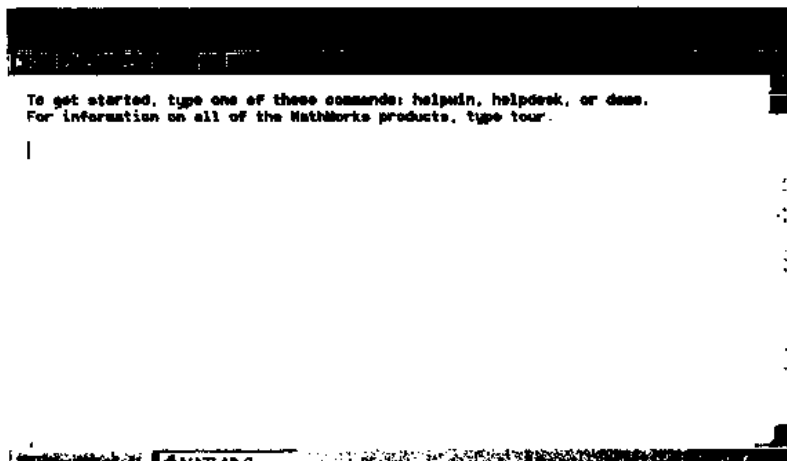


图 1.3 MATLAB 的命令窗口

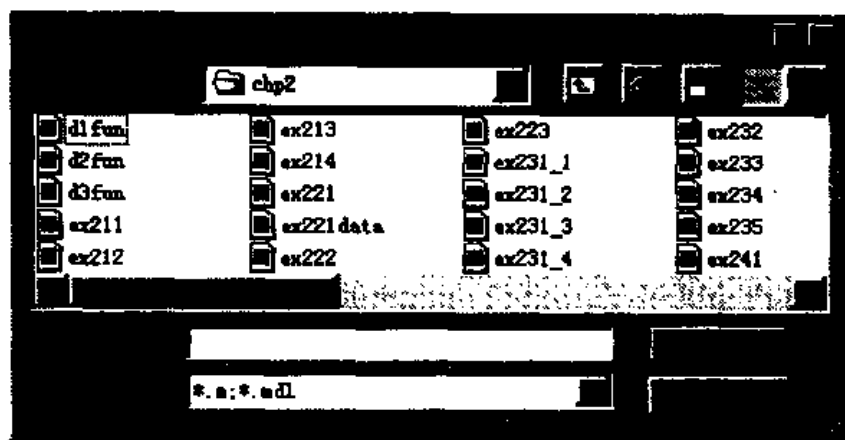


图 1.4 Open 窗口

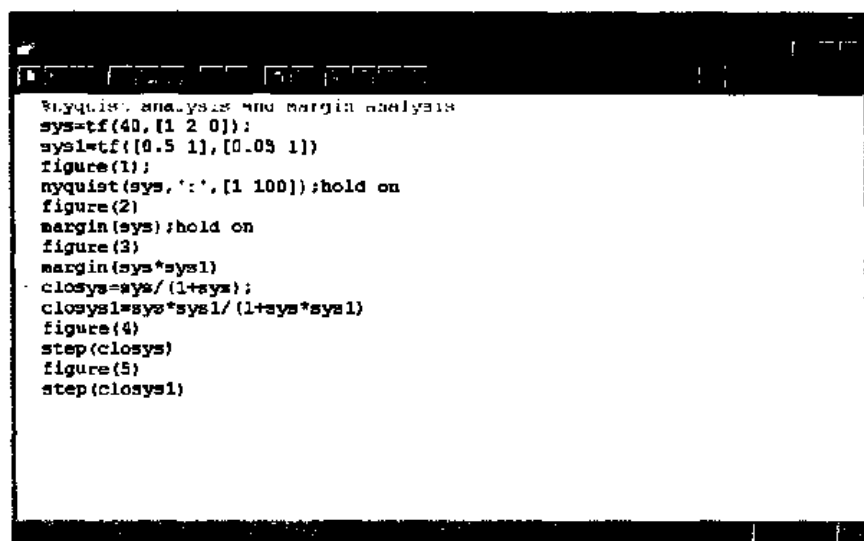


图 1.5 Editor(编辑器)与 Debugger(调试器)

## 1.2 Help的使用方法

在MATLAB 5.1中,最方便的一点就是寻求帮助。需要常常查阅手册的时代已经过去了,现在只要单击鼠标并输入不明白的指令,马上就可以得到所要的信息。最简单的方法就是在命令窗口下执行help指令,此外也可以进入帮助窗口(Help Window),如图1.6所示,用鼠标单击即可查看指令说明,这部分功能保留了4.x版的原有功能。在MATLAB 5.1中加强了双向沟通的功能,让用户使用Help时就如同询问一位专家一样。Help包括了手册中所有的文字和图片内容,并存储为HTML文件格式,只要输入问题的关键字,所有可能的答案就都会显示出来。就像在网络上查询问题一样,并且如果申请了上网账号,还可直接与MathWorks公司联系,可以说方便极了。因此,本书除了对基本而重要的指令进行说明外,并不把重点放在指令的介绍和分析上。因为只要能够充分利用Help功能,所有问题的答案都可以在电脑中找到,所以真正应该下功夫的是技巧训练,也就是读者在遇到问题时,应该知道到哪里找答案,找到答案之后更要知道如何使用。

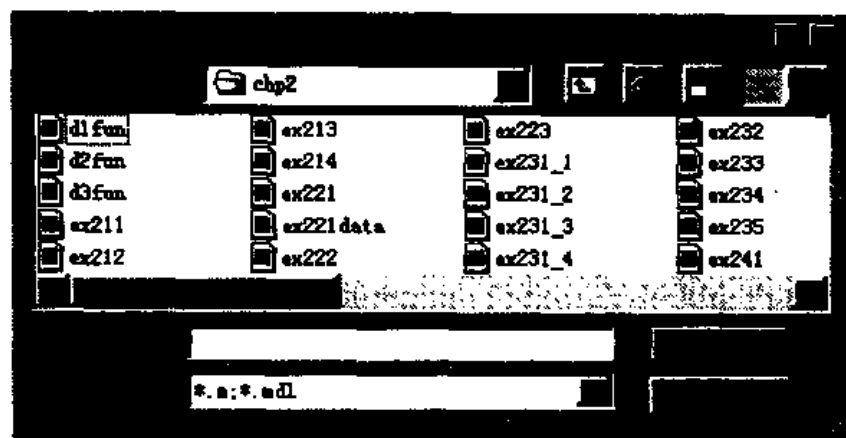


图 1.4 Open 窗口

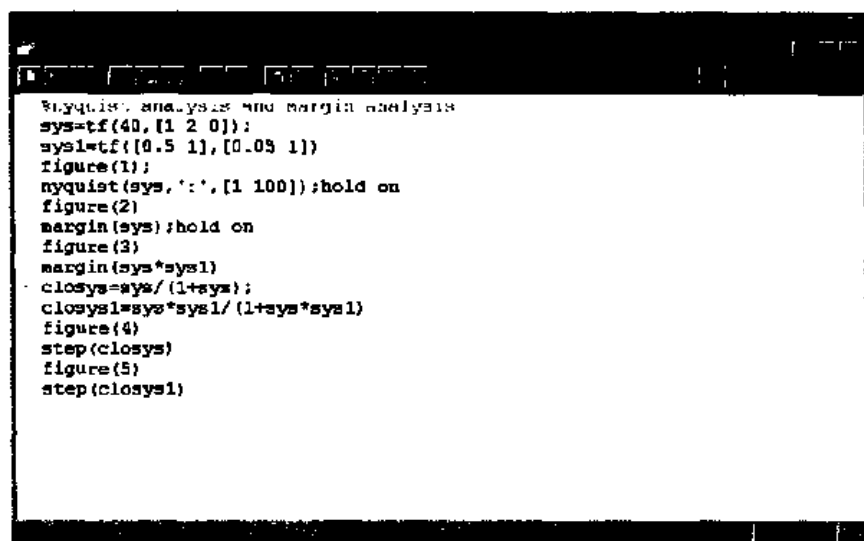


图 1.5 Editor(编辑器)与 Debugger(调试器)

## 1.2 Help的使用方法

在MATLAB 5.1中,最方便的一点就是寻求帮助。需要常常查阅手册的时代已经过去了,现在只要单击鼠标并输入不明白的指令,马上就可以得到所要的信息。最简单的方法就是在命令窗口下执行help指令,此外也可以进入帮助窗口(Help Window),如图1.6所示,用鼠标单击即可查看指令说明,这部分功能保留了4.x版的原有功能。在MATLAB 5.1中加强了双向沟通的功能,让用户使用Help时就如同询问一位专家一样。Help包括了手册中所有的文字和图片内容,并存储为HTML文件格式,只要输入问题的关键字,所有可能的答案就都会显示出来。就像在网络上查询问题一样,并且如果申请了上网账号,还可直接与MathWorks公司联系,可以说方便极了。因此,本书除了对基本而重要的指令进行说明外,并不把重点放在指令的介绍和分析上。因为只要能够充分利用Help功能,所有问题的答案都可以在电脑中找到,所以真正应该下功夫的是技巧训练,也就是读者在遇到问题时,应该知道到哪里找答案,找到答案之后更要知道如何使用。

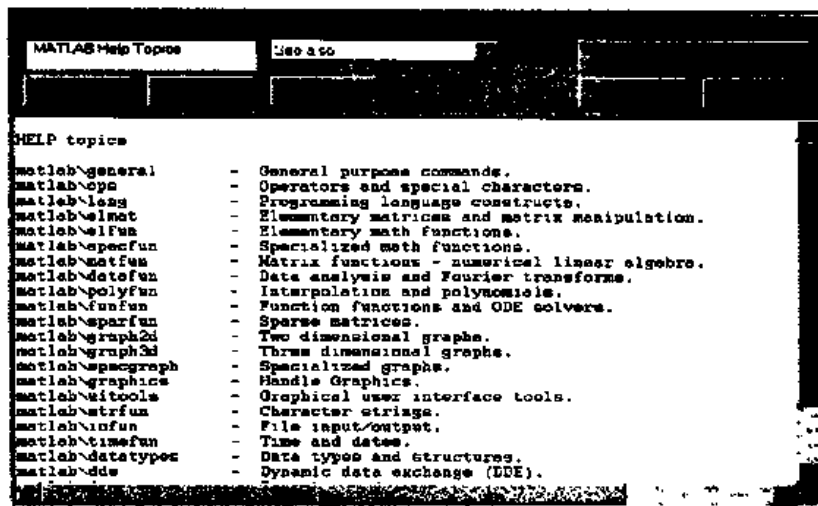


图 1.6 帮助窗口

如果读者已经安装了浏览器，比如Netscape，IE等，就可以进入Help Desk读取HTML文件，如图1.7所示。如果电脑上安装的浏览器是IE，就会出现如图1.8所示的画面。在这里可以查询任何不清楚的问题，如果实在查不到，也可以访问同时提供的网址直接进行查询。

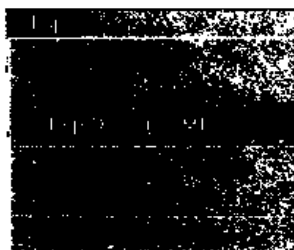


图 1.7 Help 菜单

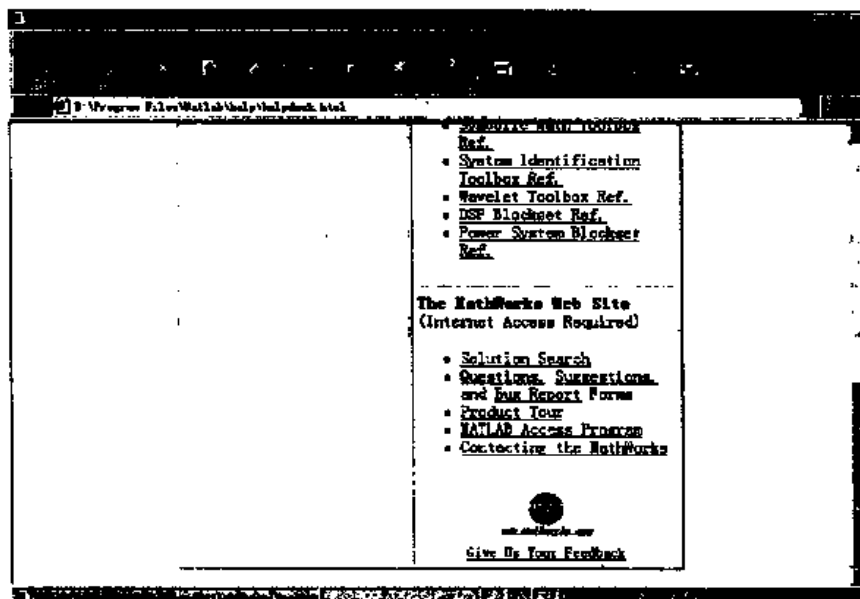


图 1.8 Help Desk

举例来说，如果不明白SIMULINK的Algebraic Constraint指令，就可以单击SIMULINK

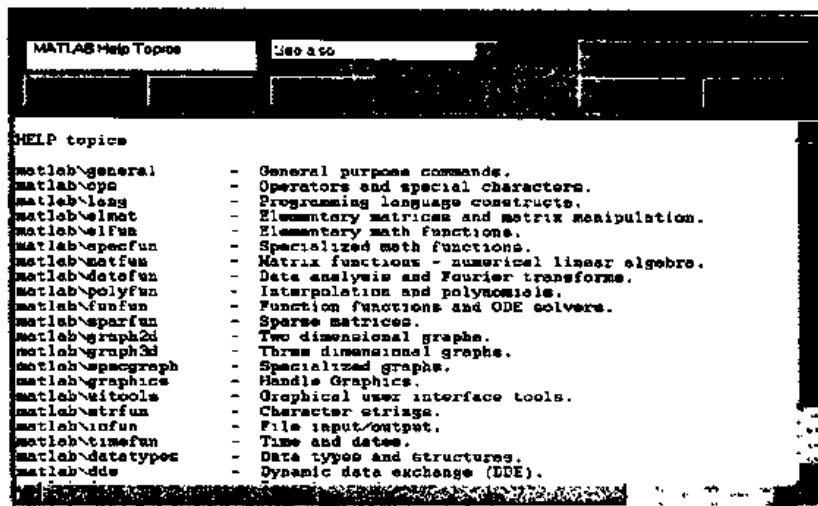


图 1.6 帮助窗口

如果读者已经安装了浏览器，比如Netscape，IE等，就可以进入Help Desk读取HTML文件，如图1.7所示。如果电脑上安装的浏览器是IE，就会出现如图1.8所示的画面。在这里可以查询任何不清楚的问题，如果实在查不到，也可以访问同时提供的网址直接进行查询。

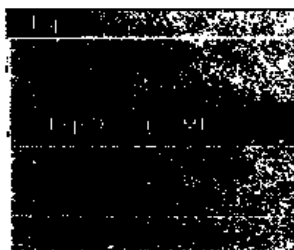


图 1.7 Help 菜单

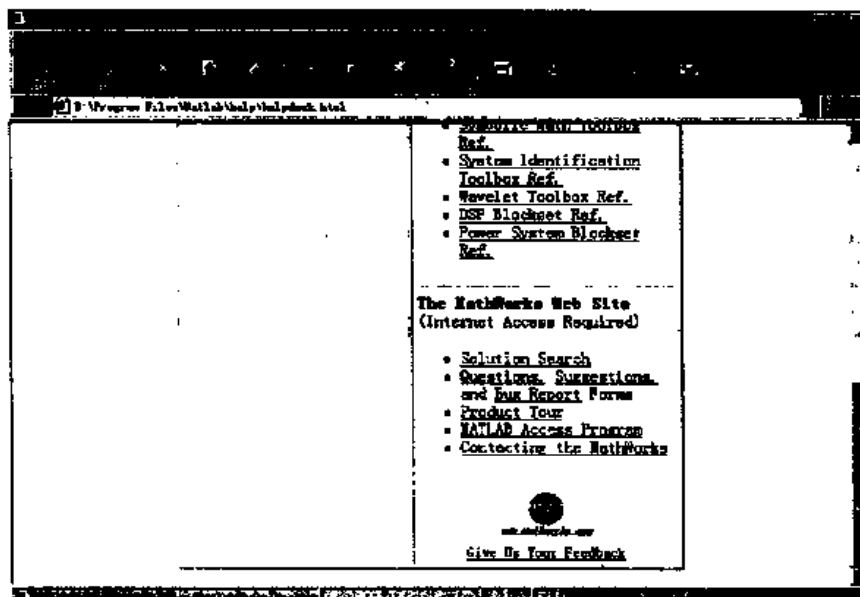


图 1.8 Help Desk

举例来说，如果不明白SIMULINK的Algebraic Constraint指令，就可以单击SIMULINK



Block, 然后会出现Simulink Block的画面, 如图1.9所示。在Search输入框内输入“algebraic”, 然后按Enter键, 即出现所有与“algebraic”相关的项目, 如图1.10所示。在“Algebraic Constraint block”上单击一下, 就出现与使用手册相同的说明内容, 如图1.11所示。

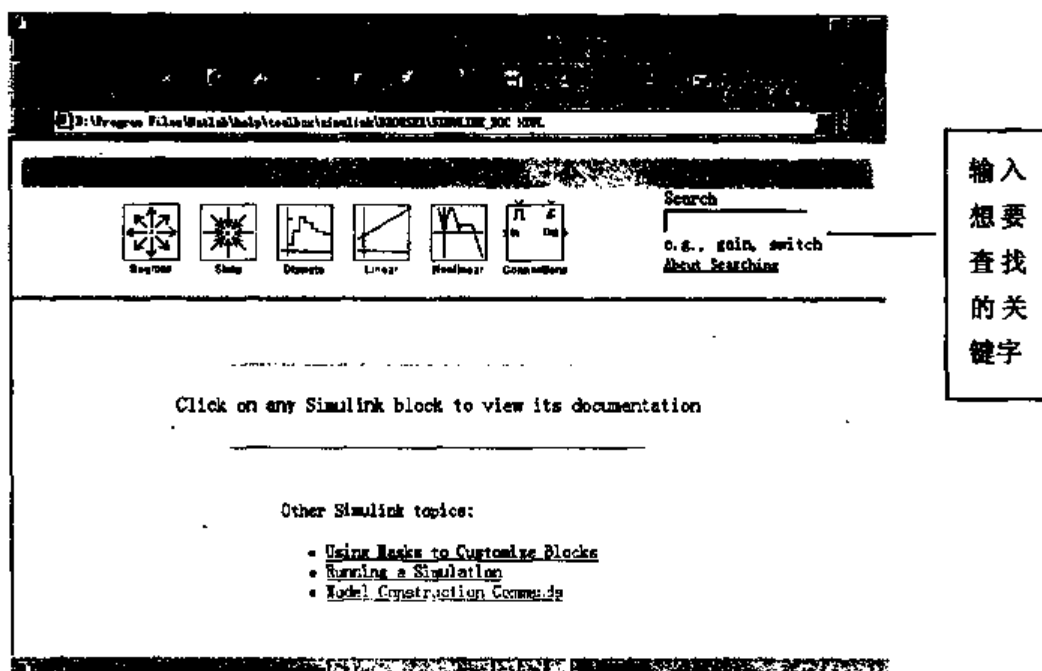


图 1.9 SIMULINK 的 Help Desk

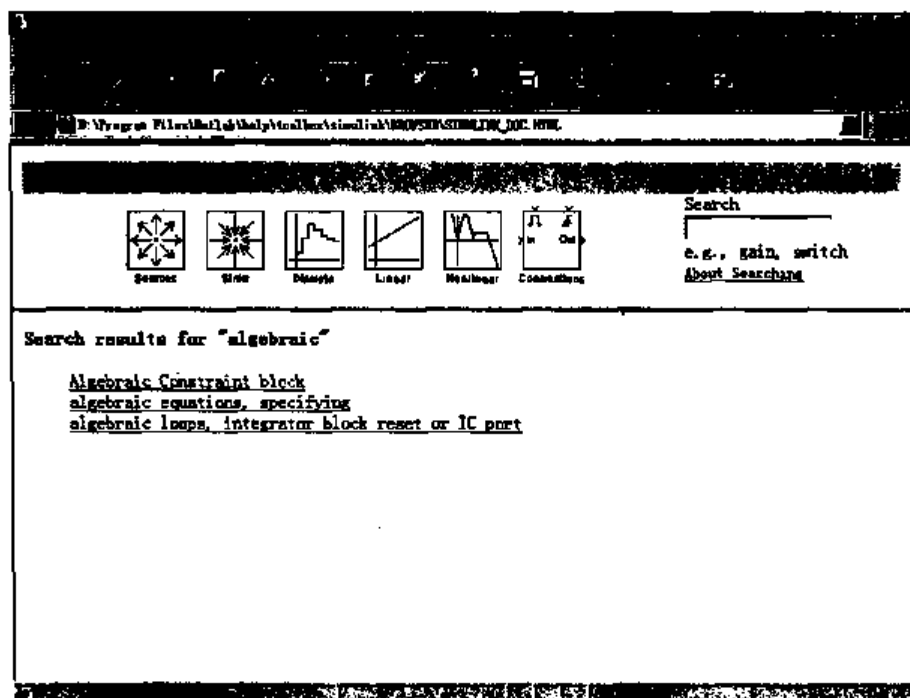


图 1.10 查找后的 Help Desk

Block, 然后会出现Simulink Block的画面, 如图1.9所示。在Search输入框内输入“algebraic”, 然后按Enter键, 即出现所有与“algebraic”相关的项目, 如图1.10所示。在“Algebraic Constraint block”上单击一下, 就出现与使用手册相同的说明内容, 如图1.11所示。

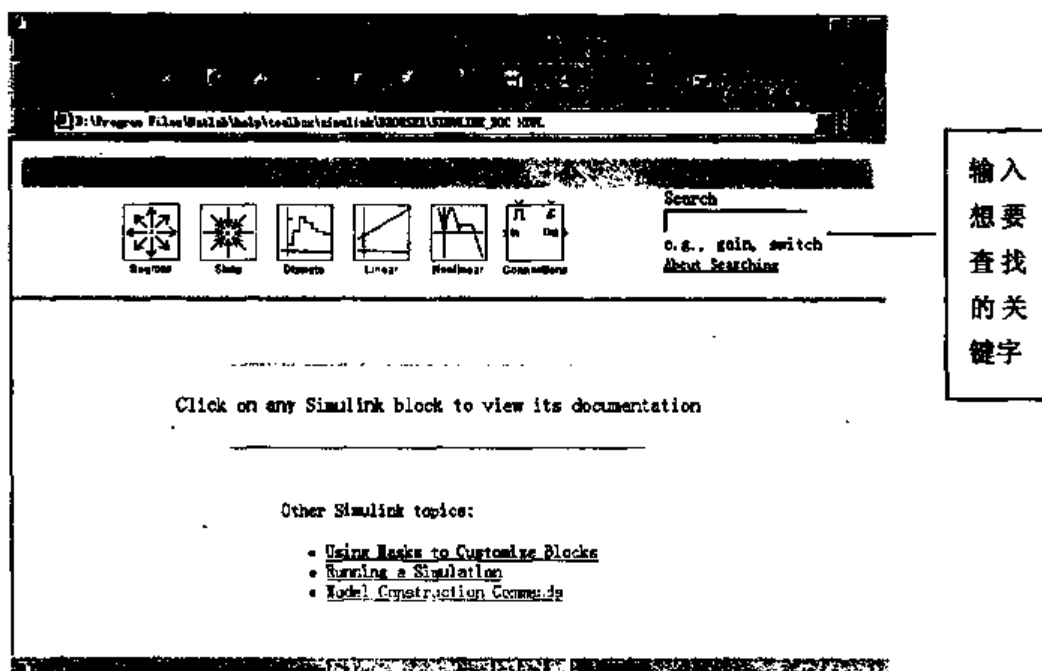


图 1.9 SIMULINK 的 Help Desk

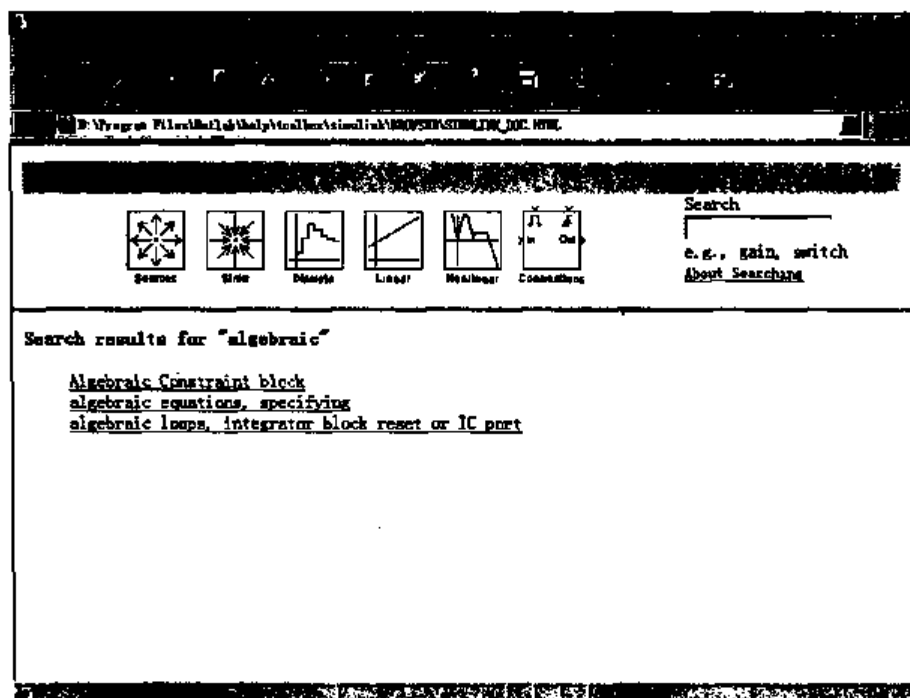


图 1.10 查找后的 Help Desk

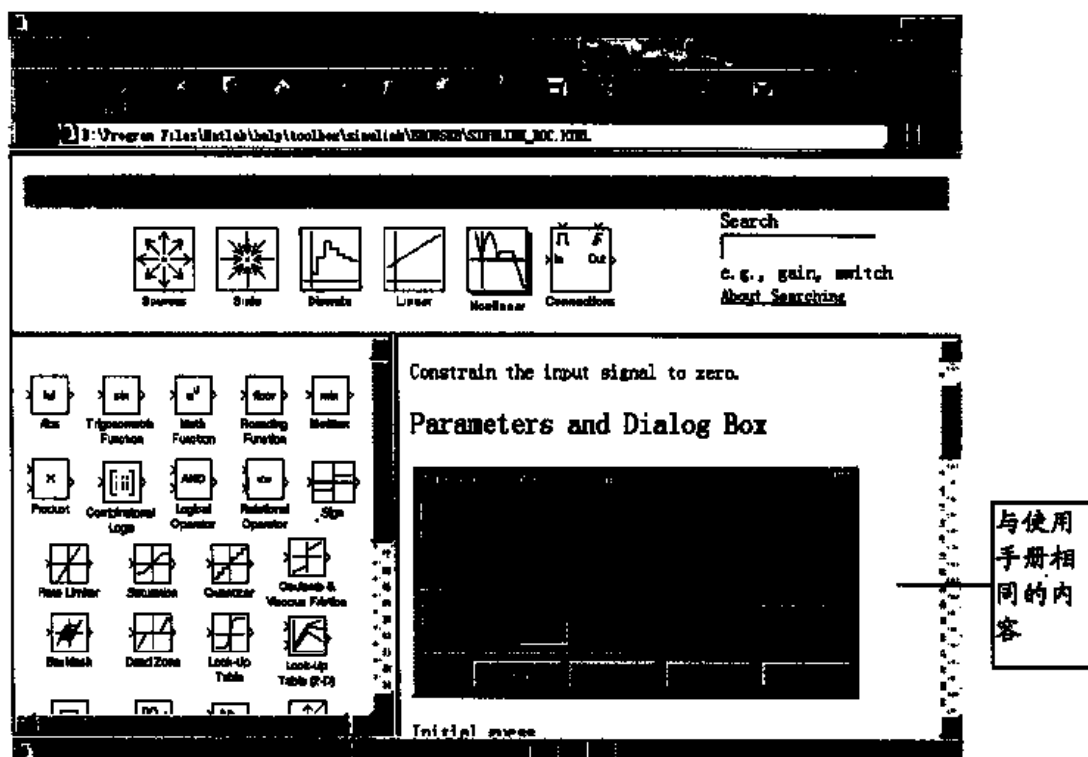


图 1.11 Help Desk 的内容

### 1.3 范例的演示(摘自MathWorks公司的范例)

MATLAB提供了大量的范例演示(Demo)。只要执行“Examples and Demos”指令(见图1.12),即可进入演示画面,如图1.13所示。如果想进行SIMULINK的演示,就可以在SIMULINK选择Inverted Pendulum Animation,再单击Run Inverted Pendulum按钮,如图1.14所示。接着就出现了SIMULINK的设计图形(见图1.15),在“Simulation”菜单中选择“Start”,执行的结果可以看到动态的倒单摆,如图1.16所示。



图 1.12 Help 菜单



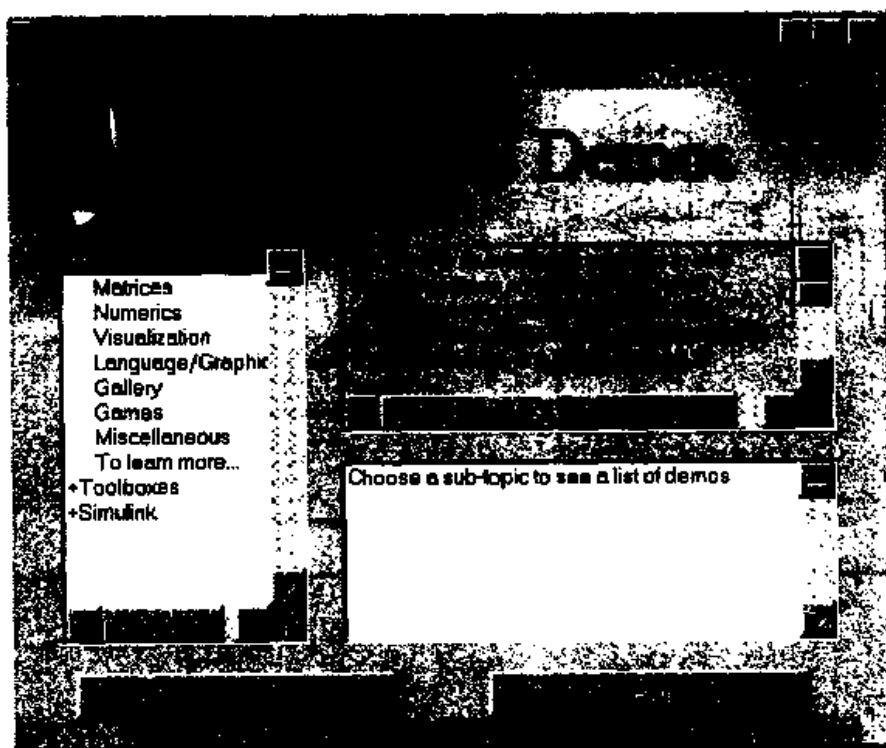


图 1.13 Demo Window 画面

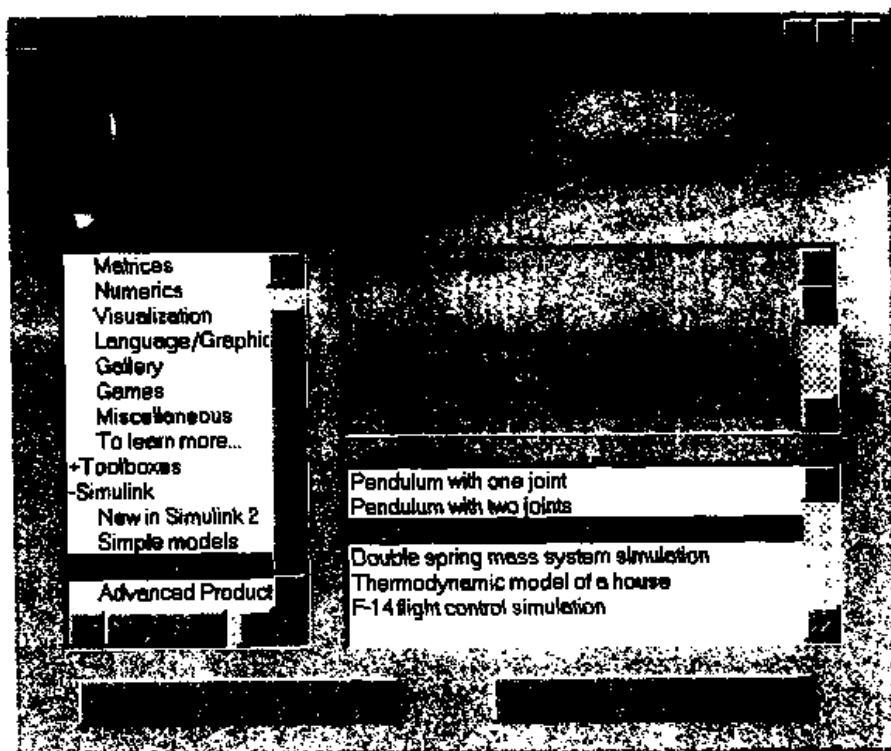


图 1.14 选择 Demo 的窗口

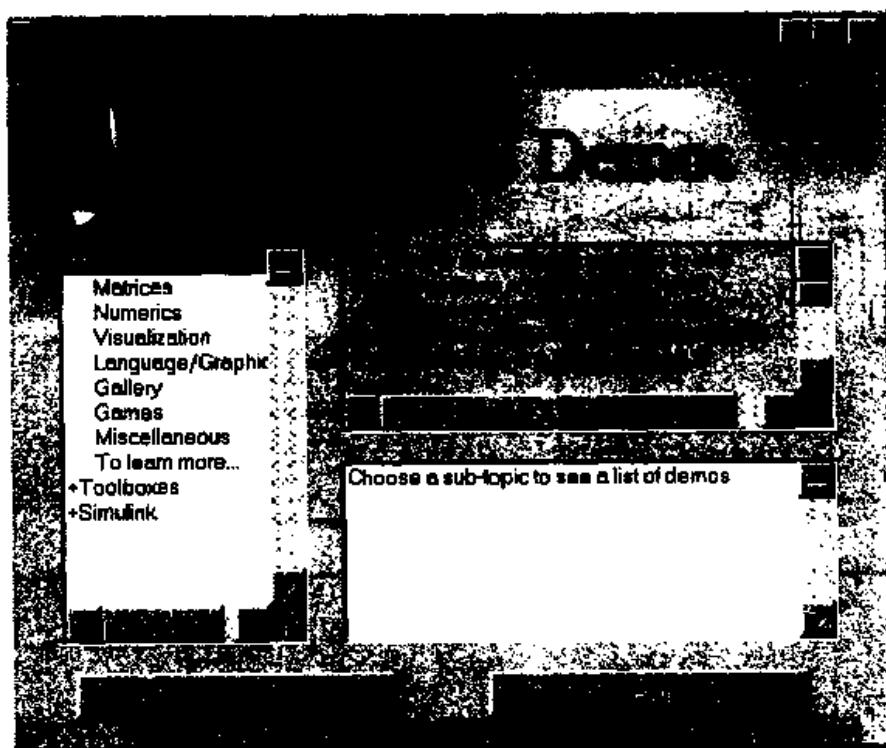


图 1.13 Demo Window 画面

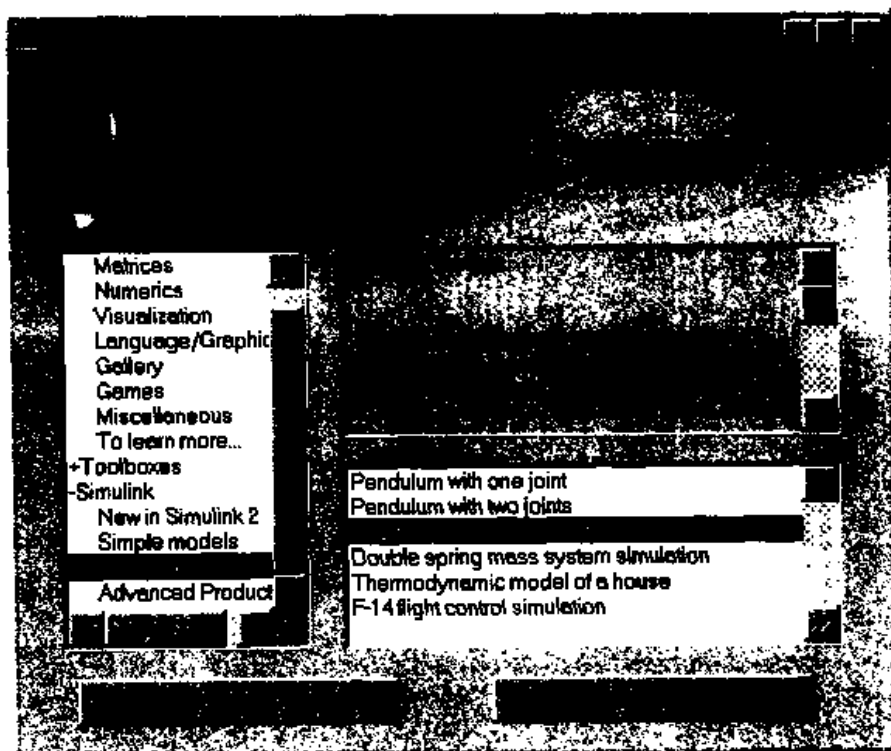


图 1.14 选择 Demo 的窗口

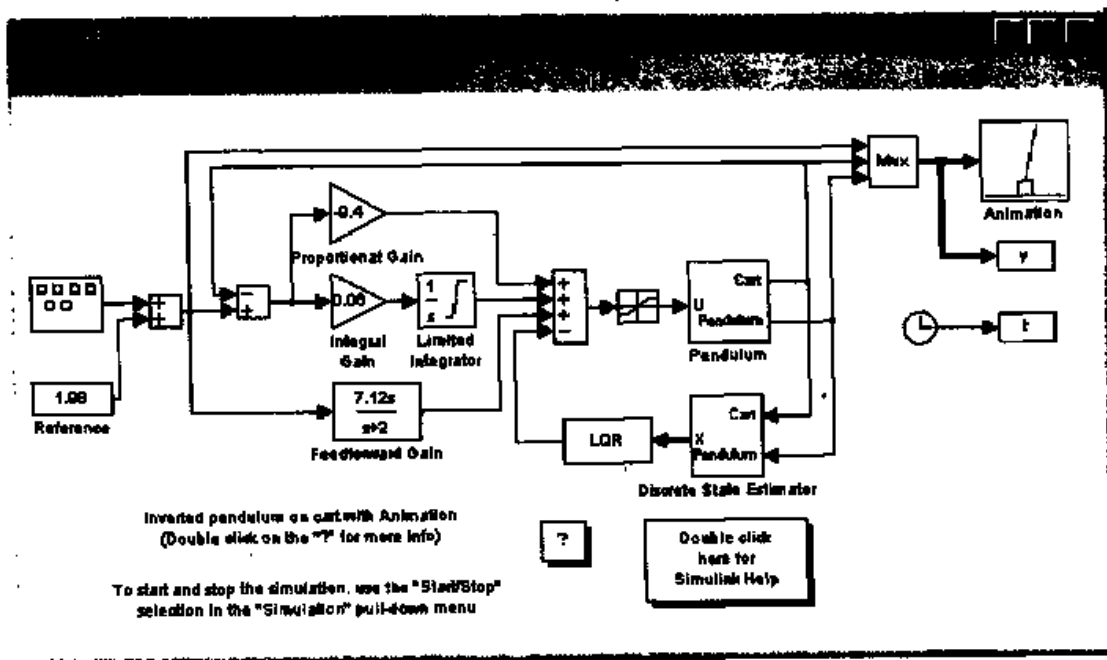


图 1.15 倒单摆的模拟程序

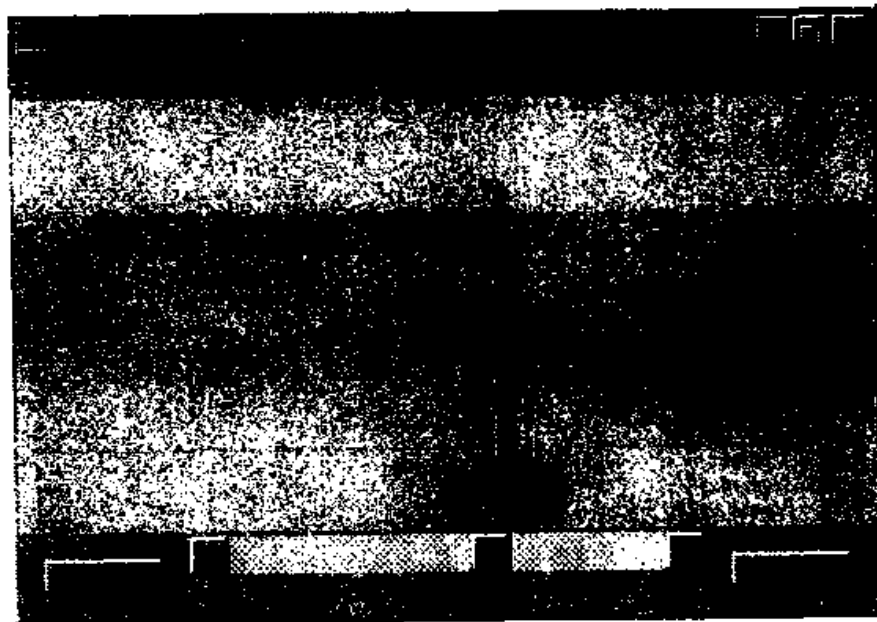


图 1.16 倒单摆的模拟结果

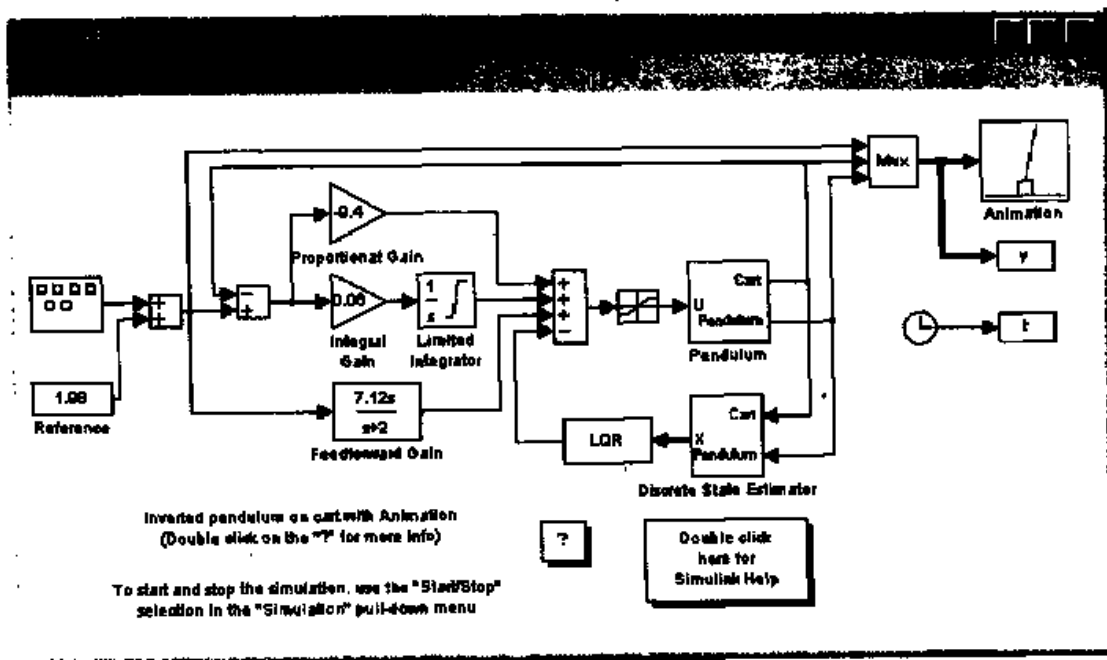


图 1.15 倒单摆的模拟程序

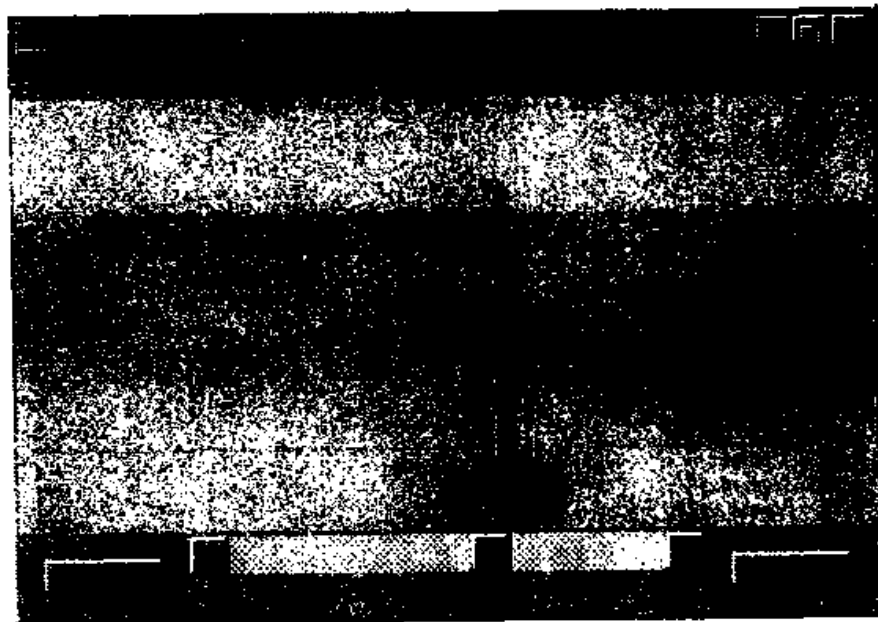


图 1.16 倒单摆的模拟结果



## 第2章 MATLAB 程序设计

本章重点介绍MATLAB 5的应用设计技巧。如果读者能够熟练掌握，就已经具备了基本的设计能力。

### 2.1 MATLAB的基本设计

#### 2.1.1 设计原则

MATLAB的文件格式有两种，一种是在命令窗口下执行的脚本M-file。它所用的变量都要在workspace中建立并获得，不需要输入输出的调用参数，退出MATLAB后就释放了。例如求1到100的和，指令如下：

```
sum(1:100)
ans =
    5050
```

另外一种就是可存取的M-file。在命令窗口的菜单中选择File→New→M-file，进入MATLAB的Editor/Debugger窗口编辑程序。编辑器用颜色区分程序内容的类别，分别为：

绿色	注解部分，程序并不执行。
黑色	程序主体部分。
红色	属性值的设定。
蓝色	控制流程，比如for, if...then等语句。

程序的写法非常简单，读者只要掌握以下几个原则就可以开始编写程序了：

- %后面的内容是程序的注解。
- MATLAB的路径设置要完整，并且最好把目前的处理位置设定为current path(参见2.1.2节)。
- 参数值要集中放在程序的开始部分，以便于以后的程序维护。在每行程序的最后输入分号；执行时程序行不会显示在屏幕上，因而提高执行速度。
- 设计时有不明白的指令，多用帮助窗口进行查询或直接输入help command\_name，例如开始时常常忘记画图时线的格式定义，此时不必查书，只要执行help plot即可显示所有需要的信息，非常方便。
- 程序尽量模块化，也就是采用主程序调用子程序的方法，将所有子程序合并在一起执行全部的操作。
- 养成在主程序开头用clear指令清除变量的习惯。注意在子程序中不要用clear，所用到的变量要么是参数，要么定义成全局变量(参见2.1.3节)。
- 了解运算后的数据如何存储才能知道如何去使用，所以在控制流程的语法与矩阵

## 第2章 MATLAB 程序设计

本章重点介绍MATLAB 5的应用设计技巧。如果读者能够熟练掌握，就已经具备了基本的设计能力。

### 2.1 MATLAB的基本设计

#### 2.1.1 设计原则

MATLAB的文件格式有两种，一种是在命令窗口下执行的脚本M-file。它所用的变量都要在workspace中建立并获得，不需要输入输出的调用参数，退出MATLAB后就释放了。例如求1到100的和，指令如下：

```
sum(1:100)
ans =
    5050
```

另外一种就是可存取的M-file。在命令窗口的菜单中选择File→New→M-file，进入MATLAB的Editor/Debugger窗口编辑程序。编辑器用颜色区分程序内容的类别，分别为：

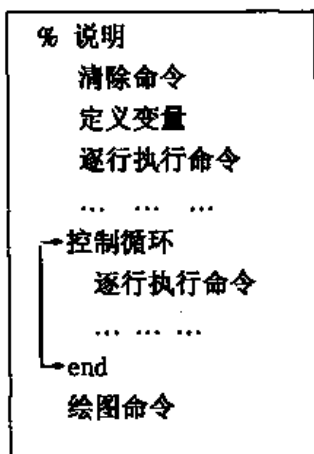
绿色	注解部分，程序并不执行。
黑色	程序主体部分。
红色	属性值的设定。
蓝色	控制流程，比如for, if...then等语句。

程序的写法非常简单，读者只要掌握以下几个原则就可以开始编写程序了：

- %后面的内容是程序的注解。
- MATLAB的路径设置要完整，并且最好把目前的处理位置设定为current path(参见2.1.2节)。
- 参数值要集中放在程序的开始部分，以便于以后的程序维护。在每行程序的最后输入分号；执行时程序行不会显示在屏幕上，因而提高执行速度。
- 设计时有不明白的指令，多用帮助窗口进行查询或直接输入help command\_name，例如开始时常常忘记画图时线的格式定义，此时不必查书，只要执行help plot即可显示所有需要的信息，非常方便。
- 程序尽量模块化，也就是采用主程序调用子程序的方法，将所有子程序合并在一起执行全部的操作。
- 养成在主程序开头用clear指令清除变量的习惯。注意在子程序中不要用clear，所用到的变量要么是参数，要么定义成全局变量(参见2.1.3节)。
- 了解运算后的数据如何存储才能知道如何去使用，所以在控制流程的语法与矩阵

运算中, 注意索引值(index value)的变化对运算产生的影响(参见2.5.6节)。

- 如果要输入大量的数据, 可以用open指令。如果临时由用户输入, 则可以使用input指令。如果是参数, 则建立一个存储参数的子程序, 在主程序中用其子程序的名称(script file name)来调用, 即可将运算所需的数据载入进来(参见2.3.1节)。
- 尽量用simulink来建立复杂的模型, 然后在主程序中用sim指令去调用它, 这样可以简化程序的设计(在第3章详细介绍)。另外要充分利用MATLAB工具箱(Toolbox)提供的指令执行所要进行的运算。多用help指令查看命令的内容, 可以增强编程能力。
- 问题分析、设计、调试、集成和演示是MATLAB的完整设计过程, 充分利用Debugger可以很容易设立breakpoint(断点), 然后选择single step(单步执行)或是continue(连续执行)模式进行调试(参见2.1.6节)。充分利用其他工具箱或图形化用户界面(Graphical User Interface, GUI)的设计技巧; 可以很容易将设计结果集成在一起。最后就是学习绘图的技巧, 将结果展示出来, 这些将在第5章和第6章中详细介绍。
- 程序的基本组成结构如下:



清除命令的目的是先清除workspace(工作空间)中的变量和图形, 包括clear, clf, close等指令(常用close all清除现有的全部图形窗口)。变量声明包括全局变量的声明, 以及程序需要的参数值的设定等。逐行执行命令是MATLAB提供的运算指令或者工具箱提供的专用命令。控制循环则包含for, if then, switch和while等执行语句(在2.2节详细介绍)。绘图命令则用于将上面的运算结果绘制出来。当然, 更复杂的程序还需要调用子程序, 或与SIMULINK以及其他应用程序结合起来, 这将在后面加以说明。

### 2.1.2 设置MATLAB的工作路径

路径的设置很重要, 不然很可能无法读取某些系统文件或数据, 导致程序无法执行。路径设置有两种方法, 一是在命令窗口下使用cd指令, 直接更改工作路径。如果目前程序放在目录chp2中, 执行MATLAB前最好先把路径设到chp2中, 以避免不必要的错误。

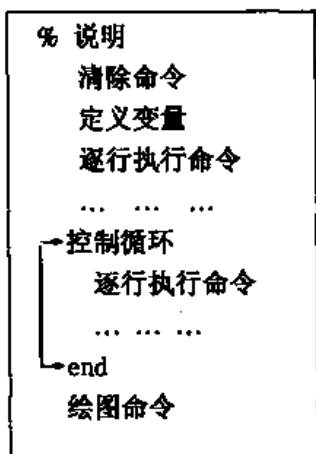
例如要把工作路径设到chp2下, 可以执行:

```
cd chp2
```

另一种方法是在菜单中选择File→Set path, 就会出现如图2.1所示的Path窗口, 用Add to Path

运算中, 注意索引值(index value)的变化对运算产生的影响(参见2.5.6节)。

- 如果要输入大量的数据, 可以用open指令。如果临时由用户输入, 则可以使用input指令。如果是参数, 则建立一个存储参数的子程序, 在主程序中用其子程序的名称(script file name)来调用, 即可将运算所需的数据载入进来(参见2.3.1节)。
- 尽量用simulink来建立复杂的模型, 然后在主程序中用sim指令去调用它, 这样可以简化程序的设计(在第3章详细介绍)。另外要充分利用MATLAB工具箱(Toolbox)提供的指令执行所要进行的运算。多用help指令查看命令的内容, 可以增强编程能力。
- 问题分析、设计、调试、集成和演示是MATLAB的完整设计过程, 充分利用Debugger可以很容易设立breakpoint(断点), 然后选择single step(单步执行)或是continue(连续执行)模式进行调试(参见2.1.6节)。充分利用其他工具箱或图形化用户界面(Graphical User Interface, GUI)的设计技巧; 可以很容易将设计结果集成在一起。最后就是学习绘图的技巧, 将结果展示出来, 这些将在第5章和第6章中详细介绍。
- 程序的基本组成结构如下:



清除命令的目的是先清除workspace(工作空间)中的变量和图形, 包括clear, clf, close等指令(常用close all清除现有的全部图形窗口)。变量声明包括全局变量的声明, 以及程序需要的参数值的设定等。逐行执行命令是MATLAB提供的运算指令或者工具箱提供的专用命令。控制循环则包含for, if then, switch和while等执行语句(在2.2节详细介绍)。绘图命令则用于将上面的运算结果绘制出来。当然, 更复杂的程序还需要调用子程序, 或与SIMULINK以及其他应用程序结合起来, 这将在后面加以说明。

### 2.1.2 设置MATLAB的工作路径

路径的设置很重要, 不然很可能无法读取某些系统文件或数据, 导致程序无法执行。路径设置有两种方法, 一是在命令窗口下使用cd指令, 直接更改工作路径。如果目前程序放在目录chp2中, 执行MATLAB前最好先把路径设到chp2中, 以避免不必要的错误。

例如要把工作路径设到chp2下, 可以执行:

```
cd chp2
```

另一种方法是在菜单中选择File→Set path, 就会出现如图2.1所示的Path窗口, 用Add to Path

添加路径，用Browse更改路径，选择所要的路径后，单击OK即可，如图2.2所示。

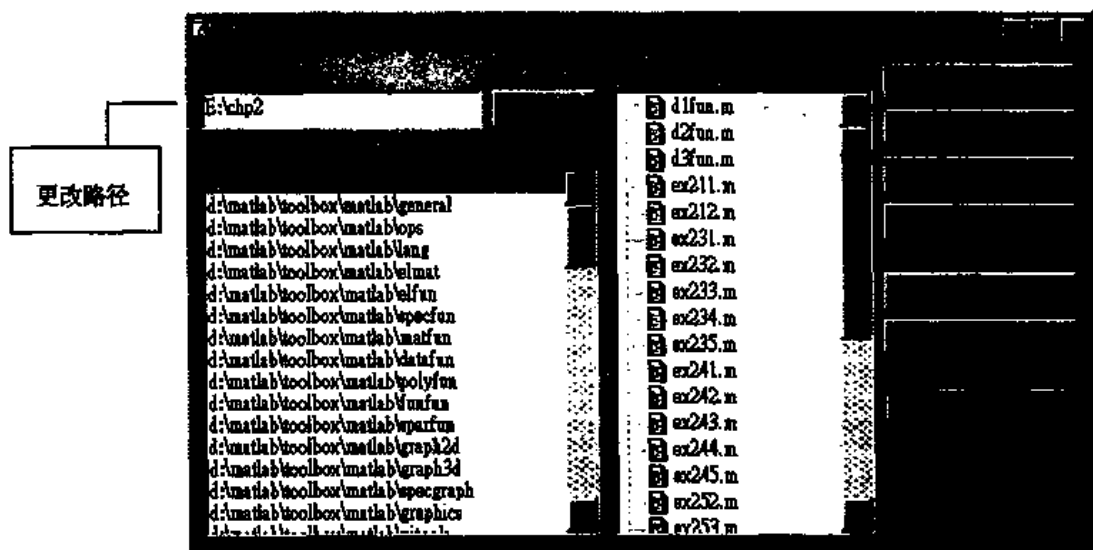


图 2.1 Path 窗口

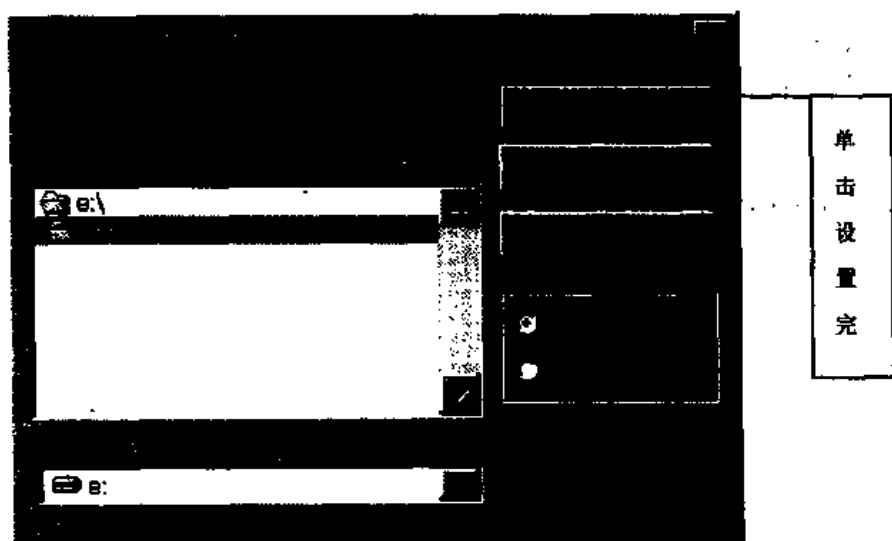


图 2.2 Change Current Directory 窗口

### 2.1.3 声明子程序的变量

子程序与主程序之间的数据靠参数(arguments)值来传递，然后将计算结果返回主程序，例如：

#### 范例2.1

程序：ex211.m

```
a=1;
b=100;
```

添加路径，用Browse更改路径，选择所要的路径后，单击OK即可，如图2.2所示。

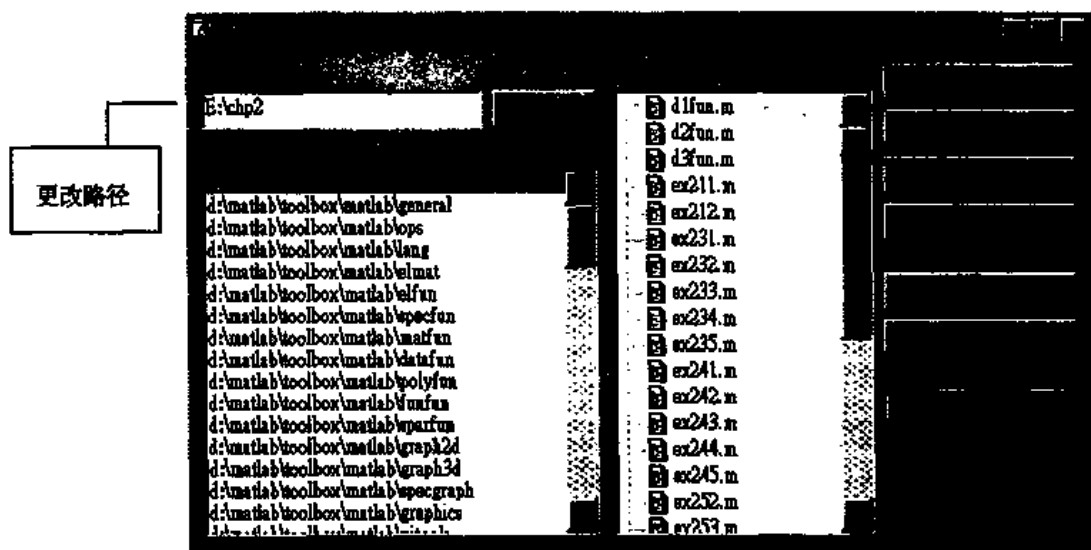


图 2.1 Path 窗口

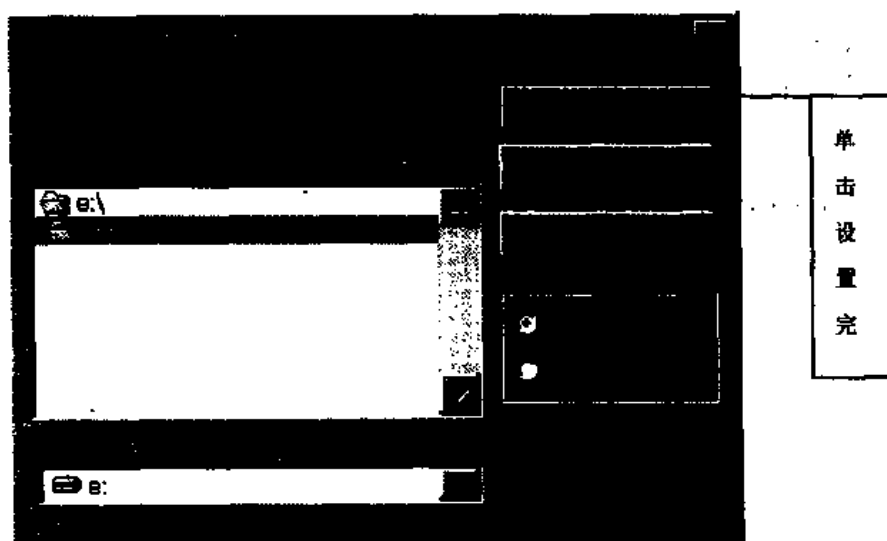


图 2.2 Change Current Directory 窗口

### 2.1.3 声明子程序的变量

子程序与主程序之间的数据靠参数(arguments)值来传递，然后将计算结果返回主程序，例如：

#### 范例2.1

程序：ex211.m

```
a=1;
b=100;
```

```

c=sumation(a,b)
调用子程序sumation.m
%sumation(a,b) sum the serial numbers from a to b.
function result=sumation(a,b)
result=sum(a:b)

```

执行结果:

```

ex211
c=
    5050

```

在此例中,主程序设定两个参数a, b来调用子程序sumation,所以执行子程序sumation时就会引用主程序传递过来的a, b值进行运算,再将结果返回主程序,所以主程序的结果等于子程序sumation的运算结果。其格式为:

```

function 返回值=子程序名称(参数1, 参数2, ...)
    运算式.....

```

如果执行help 子程序名称,就会把子程序的说明部分显示出来。编写程序的说明部分有助于日后了解自己建立的子程序的用途。例如:

```

>> help sumation
sumation(a,b) sum the serial numbers from a to b.

```

但是如果需要用到其他的变量,就要利用在主程序与子程序中分别都声明全局变量的方式,实现变量的传递。例如:

## 范例2.2

程序: ex212

```

%check the function of Global variables
global a _____ 声明程序的全局变量
x=1:100;
a=3;
c=prods(x)

```

调用子程序prods.m

```

%prods(x) get the product sum of serial numbers x.
function result=prods(x)
global a _____ 声明子程序的全局变量
result = a*sum(x);

```

执行结果:

```

ex212

```

```

c=sumation(a,b)
调用子程序sumation.m
%sumation(a,b) sum the serial numbers from a to b.
function result=sumation(a,b)
result=sum(a:b)

```

执行结果:

```

ex211
c=
    5050

```

在此例中,主程序设定两个参数a, b来调用子程序sumation,所以执行子程序sumation时就会引用主程序传递过来的a, b值进行运算,再将结果返回主程序,所以主程序的结果等于子程序sumation的运算结果。其格式为:

```

function 返回值=子程序名称(参数1, 参数2, ...)
    运算式.....

```

如果执行help 子程序名称,就会把子程序的说明部分显示出来。编写程序的说明部分有助于日后了解自己建立的子程序的用途。例如:

```

>> help sumation
sumation(a,b) sum the serial numbers from a to b.

```

但是如果需要用到其他的变量,就要利用在主程序与子程序中分别都声明全局变量的方式,实现变量的传递。例如:

## 范例2.2

程序: ex212

```

%check the function of Global variables
global a _____ 声明程序的全局变量
x=1:100;
a=3;
c=prods(x)

```

调用子程序prods.m

```

%prods(x) get the product sum of serial numbers x.
function result=prods(x)
global a _____ 声明子程序的全局变量
result = a*sum(x);

```

执行结果:

```

ex212

```



```
c=
15150
```

只有在主程序与子程序中将a声明为全局变量，才能在子程序中将其当作求和后的乘数。

#### 2.1.4 程序的运算符

运算符(Operators)分为三大类:

##### 1. 数学运算符(Arithmetic operator)

加	+
减	-
乘	.*
除	/
转置	'
次方	.^
整行(列)	:

##### 2. 关系运算符(Relational operator)

小于	<
大于	>
小于等于	<=
大于等于	>=
等于	==
不等于	~=
a是否为空矩阵	isempty(a)若是则返回1，否则返回0

##### 3. 逻辑运算符(Logical operator)

与	&
或	
非	~

以上所介绍的仅仅是部分重要并且常用的运算符。读者可以使用`help matlab\ops`查看运算符的详细内容。

#### 2.1.5 利用字符串模拟运算式

利用字符串建立运算式后，再用`eval`命令执行它，可以使程序设计更加灵活。但是注意表达式一定要是字符串。其命令格式为：

```
eval('字符串')
```

例如可以先定义字符串t为平方根运算，再用`eval`求出1到10的平方根，以后只要修改t的表

```
c=
15150
```

只有在主程序与子程序中将a声明为全局变量，才能在子程序中将其当作求和后的乘数。

#### 2.1.4 程序的运算符

运算符(Operators)分为三大类:

##### 1. 数学运算符(Arithmetic operator)

加	+
减	-
乘	.*
除	/
转置	'
次方	.^
整行(列)	:

##### 2. 关系运算符(Relational operator)

小于	<
大于	>
小于等于	<=
大于等于	>=
等于	==
不等于	~=
a是否为空矩阵	isempty(a)若是则返回1，否则返回0

##### 3. 逻辑运算符(Logical operator)

与	&
或	
非	~

以上所介绍的仅仅是部分重要并且常用的运算符。读者可以使用`help matlab\ops`查看运算符的详细内容。

#### 2.1.5 利用字符串模拟运算式

利用字符串建立运算式后，再用`eval`命令执行它，可以使程序设计更加灵活。但是注意表达式一定要是字符串。其命令格式为：

```
eval('字符串')
```

例如可以先定义字符串t为平方根运算，再用`eval`求出1到10的平方根，以后只要修改t的表

达式即可。举例如下：

### 范例2.3

程序：ex213.m

```
%check the function of EVAL
clear
t='sqrt(i)';
for i=1:10;
    f(i)={char( ['The square root of',int2str(i), 'is',num2str(eval(t))]) }
end
f(:)
```

{}代表建立数组(Array)

显示数组f的内容

字符串t的运算模拟

执行结果：

```
The square root of 1 is 1'
The square root of 2 is 1.4142'
The square root of 3 is 1.7321'
The square root of 4 is 2'
The square root of 5 is 2.2361'
The square root of 6 is 2.4495'
The square root of 7 is 2.6458'
The square root of 8 is 2.8284'
The square root of 9 is 3'
The square root of 10 is 3.1623'
```

另外一个指令就是feval指令，其格式为：

feval('字符串'，数组)

与eval不同之处在于feval用于模拟功能函数如cos、sin、sqrt等，而不是像eval那样模拟运算式，所以cos(pi)的值同样可以用feval(cos,pi)求出。下面将范例2.3中的t='sqrt(i)'修改为t=sqrt，eval(t)修改为feval(t,i)，可以得到同样的结果。举例如下：

### 范例2.4

程序：ex214.m

```
%check the function of EVAL
clear
t='sqrt';
for i=1:10;
    f(i)={char( ['The square root of',int2str(i), 'is',num2str(feval(t,i))]) }
end
f(:)
```

改为功能函数

改为feval

执行结果与范例2.3相同。

达式即可。举例如下：

### 范例2.3

程序：ex213.m

```
%check the function of EVAL
clear
t='sqrt(i)';
for i=1:10;
    f(i)={char( ['The square root of',int2str(i), 'is',num2str(eval(t))]) }
end
f(:)
```

{}代表建立数组(Array)

显示数组f的内容

字符串t的运算模拟

执行结果：

```
The square root of 1 is 1'
The square root of 2 is 1.4142'
The square root of 3 is 1.7321'
The square root of 4 is 2'
The square root of 5 is 2.2361'
The square root of 6 is 2.4495'
The square root of 7 is 2.6458'
The square root of 8 is 2.8284'
The square root of 9 is 3'
The square root of 10 is 3.1623'
```

另外一个指令就是feval指令，其格式为：

feval('字符串'，数组)

与eval不同之处在于feval用于模拟功能函数如cos、sin、sqrt等，而不是像eval那样模拟运算式，所以cos(pi)的值同样可以用feval(cos,pi)求出。下面将范例2.3中的t='sqrt(i)'修改为t=sqrt，eval(t)修改为feval(t,i)，可以得到同样的结果。举例如下：

### 范例2.4

程序：ex214.m

```
%check the function of EVAL
clear
t='sqrt';
for i=1:10;
    f(i)={char( ['The square root of',int2str(i), 'is',num2str(feval(t,i))]) }
end
f(:)
```

改为功能函数

改为feval

执行结果与范例2.3相同。

### 2.1.6 调试器的运用

MATLAB的文字编辑器(Editor)具有调试功能,因此也称为调试器(Debugger),它使用起来非常简单,在程序很大时尤其有用。在此就以范例2.4为例来说明如何使用Debugger。

#### 第一步: 设定断点(breakpoint)

用鼠标在程序中想要建立断点的地方单击,然后选择菜单中的Debug→Set/Clear Breakpoint,就会出现圆点表示的断点,如图2.3所示。

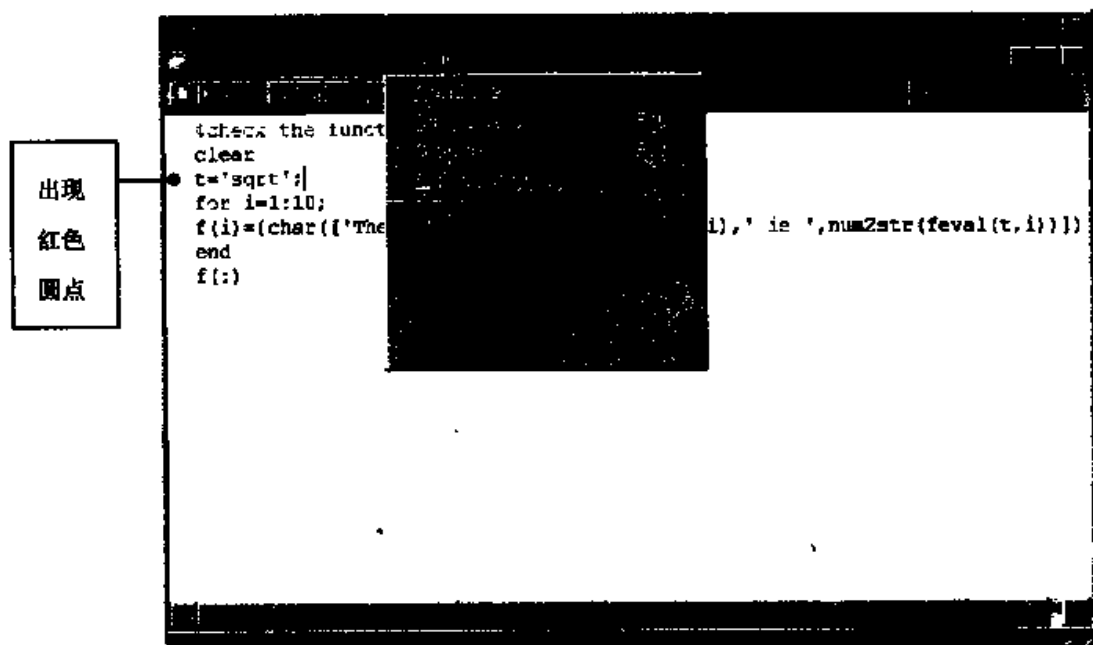


图 2.3 设定调试菜单

#### 第二步: 到命令窗口下执行程序

执行ex214之后,程序就会停在断点处,并出现箭头,如图2.4所示。接着调试选项就会由灰色变为表示可执行的黑色,如图2.5所示。包括:

- Continue:继续执行到结束或下一个断点为止。
- Single Step:在主程序中一步步执行(也可以用F10键)。
- Step In:会跳入所有调用到的子程序中(也可以用F11键)。
- Quit Debugging:退出调试状态。
- Set/Clear Breakpoint:设置/清除断点(也可以用F12键)。
- Clear All Breakpoint:清除所有断点。
- Stop if Error:发生错误时停止。
- Stop if Warning:有警告时停止。
- Stop if NaN or Inf:运算有非数值或无穷大值时停止。

选择需要的方式执行即可。

### 2.1.6 调试器的运用

MATLAB的文字编辑器(Editor)具有调试功能,因此也称为调试器(Debugger),它使用起来非常简单,在程序很大时尤其有用。在此就以范例2.4为例来说明如何使用Debugger。

#### 第一步: 设定断点(breakpoint)

用鼠标在程序中想要建立断点的地方单击,然后选择菜单中的Debug→Set/Clear Breakpoint,就会出现圆点表示的断点,如图2.3所示。

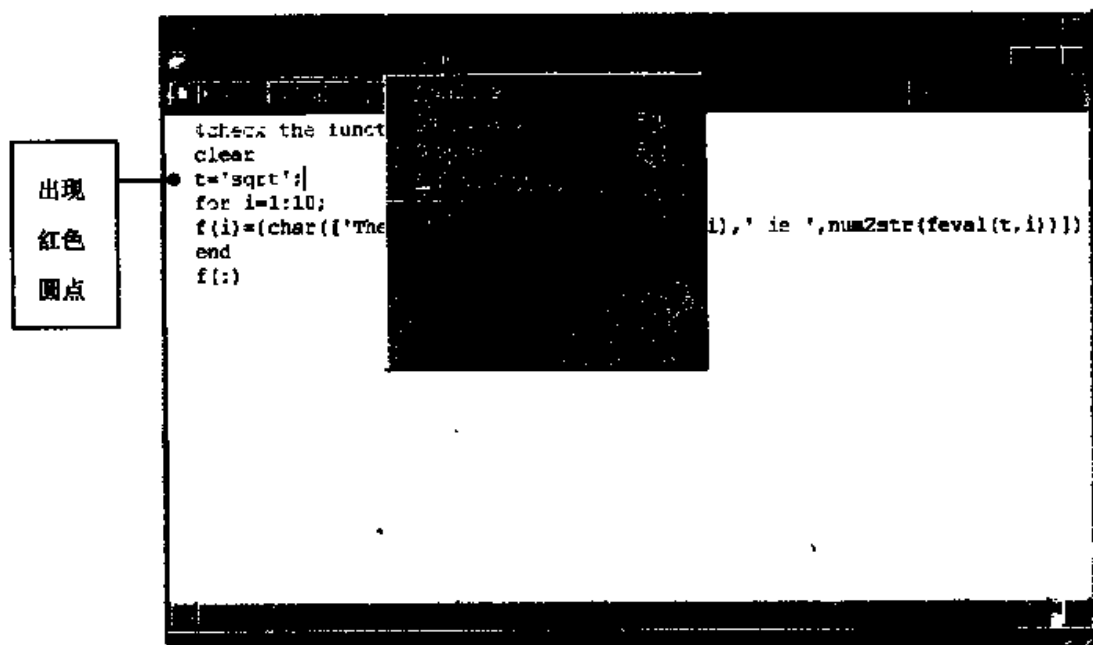


图 2.3 设定调试菜单

#### 第二步: 到命令窗口下执行程序

执行ex214之后,程序就会停在断点处,并出现箭头,如图2.4所示。接着调试选项就会由灰色变为表示可执行的黑色,如图2.5所示。包括:

- Continue:继续执行到结束或下一个断点为止。
- Single Step:在主程序中一步步执行(也可以用F10键)。
- Step In:会跳入所有调用到的子程序中(也可以用F11键)。
- Quit Debugging:退出调试状态。
- Set/Clear Breakpoint:设置/清除断点(也可以用F12键)。
- Clear All Breakpoint:清除所有断点。
- Stop if Error:发生错误时停止。
- Stop if Warning:有警告时停止。
- Stop if NaN or Inf:运算有非数值或无穷大值时停止。

选择需要的方式执行即可。

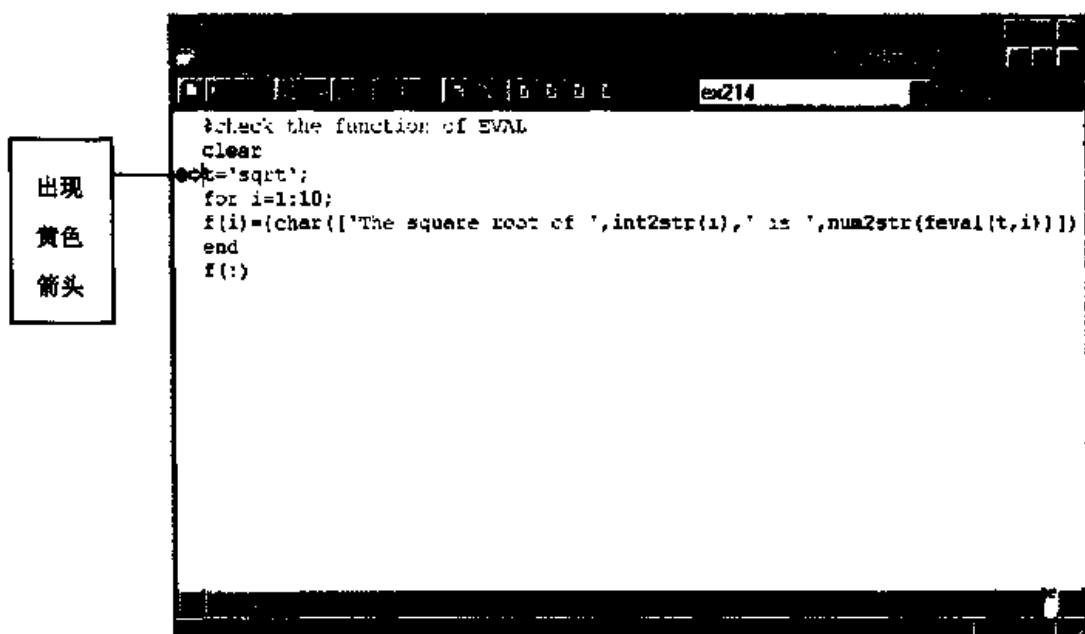


图 2.4 调试与编辑器画面

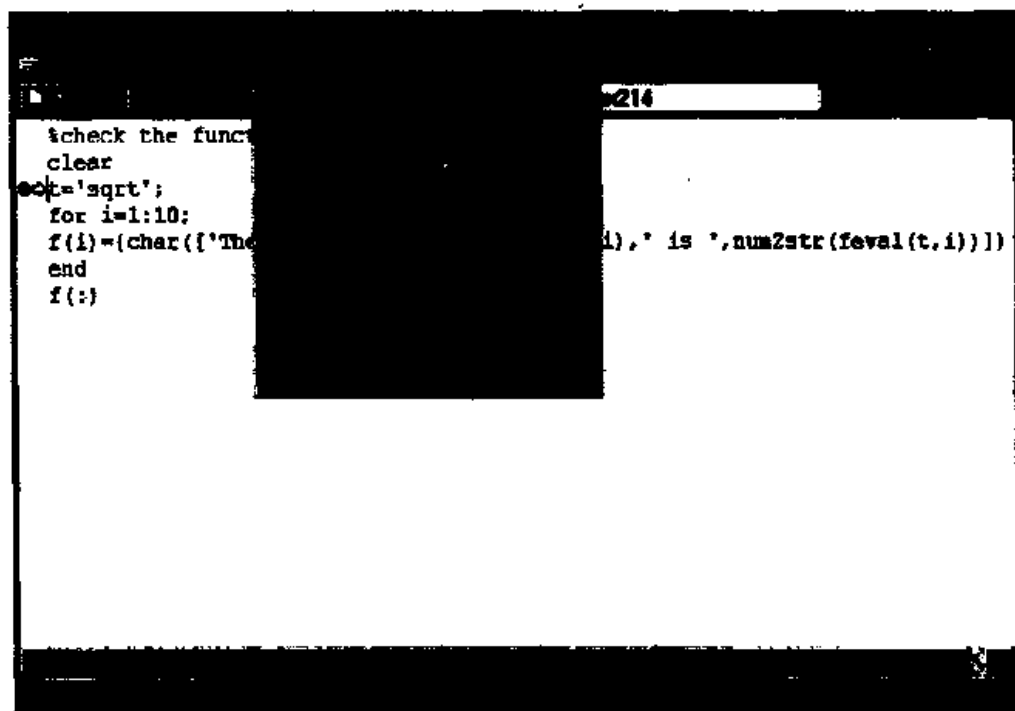


图 2.5 单步执行调试菜单

## 2.2 程序流程控制

程序流程控制是在程序运算中加入判断条件，然后根据不同的条件去执行某一特定的部分。使用最多的指令为for, if then和switch等，至于while则通常用于执行循环运算。下面分别加以介绍。

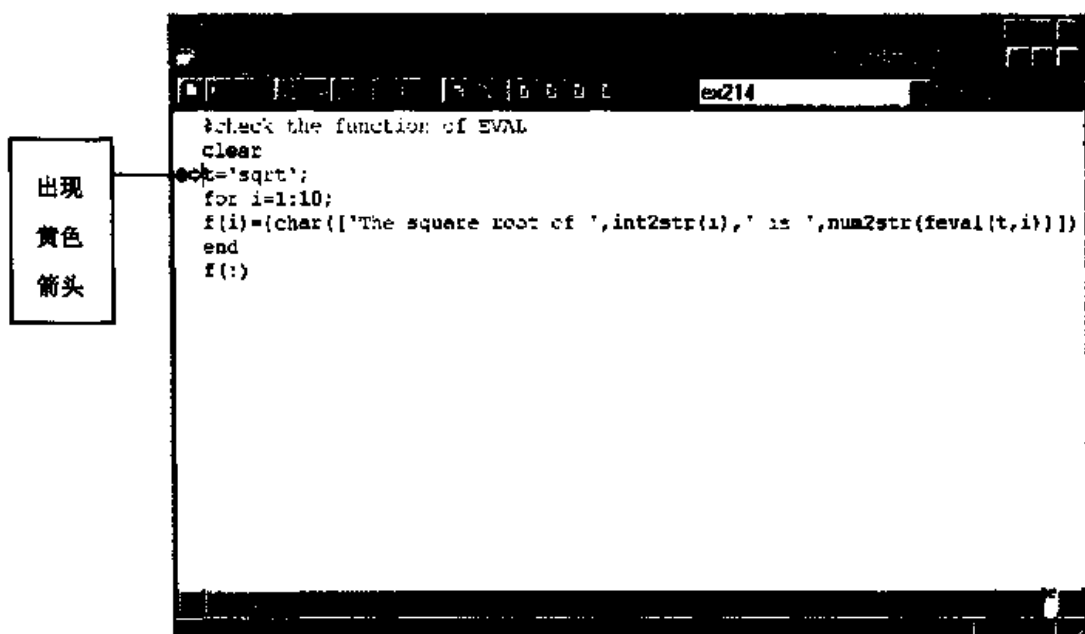


图 2.4 调试与编辑器画面

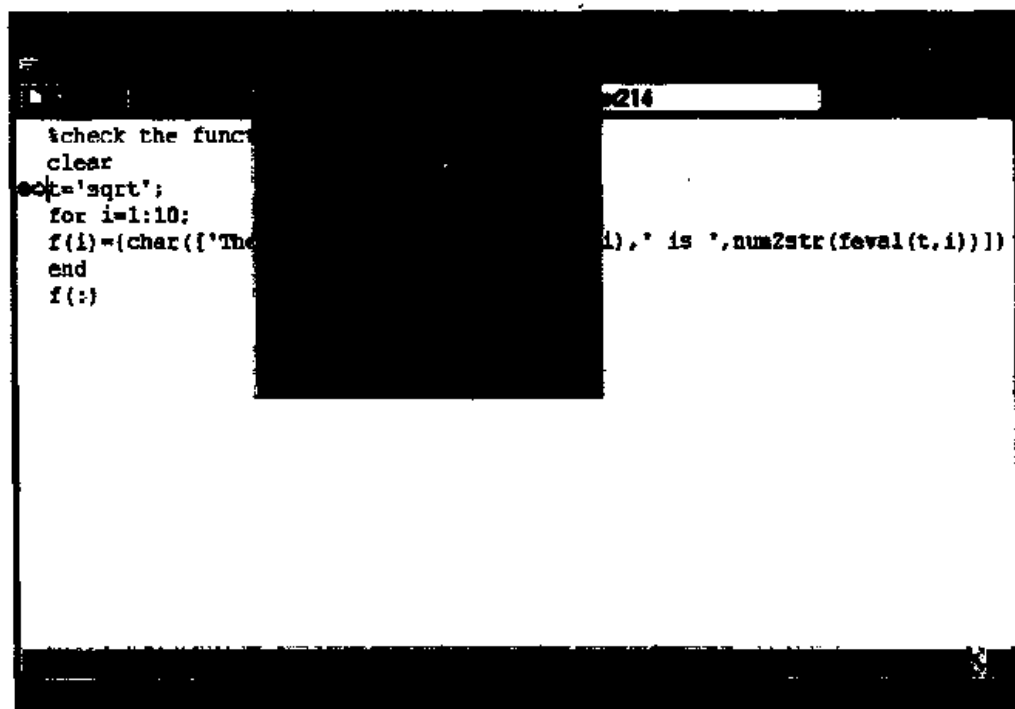


图 2.5 单步执行调试菜单

## 2.2 程序流程控制

程序流程控制是在程序运算中加入判断条件，然后根据不同的条件去执行某一特定的部分。使用最多的指令为for, if then和switch等，至于while则通常用于执行循环运算。下面分别加以介绍。



### 2.2.1 for 循环

for循环的指令格式为:

```
for i=1:n
    运算式
end
```

表示执行运算式n次, 而i的值会由1变到n。在程序中常会结合length指令(参见范例2.5)共同求出n, 然后利用i来求出特定的值。配合break一起使用, 在满足某一条件后可以退出循环。下面的范例说明了如何在波形文件ex221data.m中提取大于3的波形, 如图2.6所示。在该例中大量使用了for循环, 读者可以仔细研究, 必定会有所裨益。

#### 范例2.5 提取大于3的波形。

程序: ex221.m

```
%load the cdata from ex221data.m
ex221data ————— 执行ex221data以便读入波形
%get the start point at the magnitude greater than 3
for k=1:length(cdata); ————— 使用length(cdata)求k的范围, 并找出大于3的点
    if cdata(k)>=3;
        h=k;%got the start point
        break;%if find the point then break
    end;
end;
for u=1:length(cdata)-h;%the total count is the number minus the start point
    ————— u为提取后的data总数
    i=u+h;%the 'h' is used as offset value to catch the actual data
    data(u)=cdata(i);%actual data ————— i为提取后的data位置
    tt(u)=(u-1)*0.1;%the time interval
end
for k=1:length(cdata);
    tc(k)=(k-1)*0.1;%the time interval
end
subplot(211);
plot(tc,cdata');hold on
plot(tc,thr,3, 'r: ');
xlabel('time(sec)')
ylabel('magnitude')
title('The original diagram')
axis( [0 10 -5 15] )
subplot(212)
plot(tt,data)
xlabel('time(sec)')
```

### 2.2.1 for 循环

for循环的指令格式为:

```
for i=1:n
    运算式
end
```

表示执行运算式n次, 而i的值会由1变到n。在程序中常会结合length指令(参见范例2.5)共同求出n, 然后利用i来求出特定的值。配合break一起使用, 在满足某一条件后可以退出循环。下面的范例说明了如何在波形文件ex221data.m中提取大于3的波形, 如图2.6所示。在该例中大量使用了for循环, 读者可以仔细研究, 必定会有所裨益。

#### 范例2.5 提取大于3的波形。

程序: ex221.m

```
%load the cdata from ex221data.m
ex221data ————— 执行ex221data以便读入波形
%get the start point at the magnitude greater than 3
for k=1:length(cdata); ————— 使用length(cdata)求k的范围, 并找出大于3的点
    if cdata(k)>=3;
        h=k;%got the start point
        break;%if find the point then break
    end;
end;
for u=1:length(cdata)-h;%the total count is the number minus the start point
    ————— u为提取后的data总数
    i=u+h;%the 'h' is used as offset value to catch the actual data
    data(u)=cdata(i);%actual data ————— i为提取后的data位置
    tt(u)=(u-1)*0.1;%the time interval
end
for k=1:length(cdata);
    tc(k)=(k-1)*0.1;%the time interval
end
subplot(211);
plot(tc,cdata');hold on
plot(tc,thr,3, 'r: ');
xlabel('time(sec)')
ylabel('magnitude')
title('The original diagram')
axis( [0 10 -5 15] )
subplot(212)
plot(tt,data)
xlabel('time(sec)')
```

```

ylabel('magnitude')
title('The caught diagram')
axis( [0 10 -5 15] )

```

执行结果:

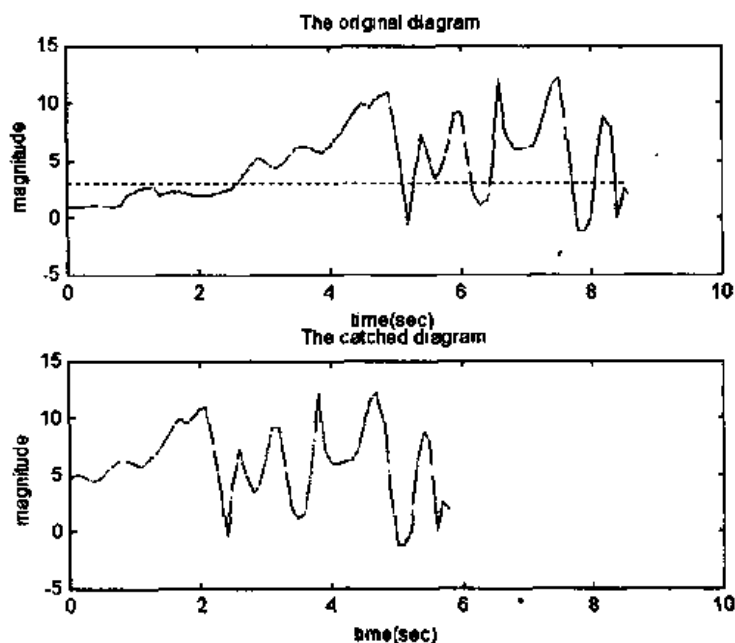


图 2.6 原波形与提取后的波形图

## 2.2.2 if then与while语句

if then的格式为:

```

if 条件1:
    运算式1
elseif 条件2
    运算式2
.....
end

```

while的格式为:

```

while 条件
    运算式
end

```

while很容易出错, 因此建议少用, 不过还是要有所了解。while的关键是只要满足条件, 运算就会一直进行下去, 直到不满足条件时才会中止。而if then则正好相反, 条件只要满足, 就会中止。下面的例子算出累加值小于1000的最大数是多少。同时注意比较while与if的差别。

```
ylabel('magnitude')  
title('The caught diagram')  
axis( [0 10 -5 15] )
```

执行结果:

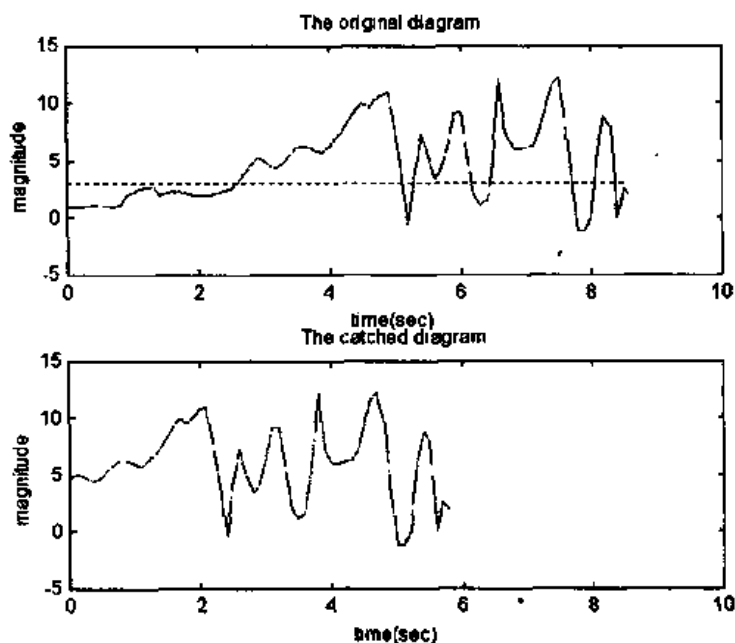


图 2.6 原波形与提取后的波形图

### 2.2.2 if then与while语句

if then的格式为:

```
if 条件1:  
    运算式1  
elseif 条件2  
    运算式2  
.....  
end
```

while的格式为:

```
while 条件  
    运算式  
end
```

while很容易出错, 因此建议少用, 不过还是要有所了解。while的关键是只要满足条件, 运算就会一直进行下去, 直到不满足条件时才会中止。而if then则正好相反, 条件只要满足, 就会中止。下面的例子算出累加值小于1000的最大数是多少。同时注意比较while与if的差别。

### 范例2.6

程序: ex222.m

```
%Utilize while loop to get the element satisfies some condition
a=1;
b=1;
while sumation(a,b)<1000
    b=b+1;
end
n=b-1
```

执行结果:

```
ex222
n=
    44
```

由上述程序可知累加到44时累加值刚好小于1000, 累加到45时就会大于1000了。所以使用while时, 到了不满足条件的45时才会跳出循环。如果将while改为if then, 则第一次就满足循环中断条件, 执行结果为:

```
ex222
n=
    1
```

所以在循环运算时应该使用while还是if, 一定要认真考虑, 因为有时差别会很大。

#### 2.2.3 switch语句

switch语句的指令结构很好, 为条件与条件的排列, 看起来比if then清楚, 可以用于有选择性的程序设计。指令格式为:

```
switch variable
    case 1
        运算式1
    case 2
        运算式2
    .....
    otherwise
        运算式n
end
```

表示当variable为1时, 就执行case 1下面的运算式, 其余依次类推。但要注意的是, 如果variable是字符串, case 1就要把1改为字符串, 变成case '1'。

例如下例, 设计功能菜单, 合并范例2.5与范例2.6的程序, 选1则执行ex221, 选2则执行ex222, 如果选错则会出现错误信息。

### 范例2.6

程序: ex222.m

```
%Utilize while loop to get the element satisfies some condition
a=1;
b=1;
while sumation(a,b)<1000
    b=b+1;
end
n=b-1
```

执行结果:

```
ex222
n=
    44
```

由上述程序可知累加到44时累加值刚好小于1000, 累加到45时就会大于1000了。所以使用while时, 到了不满足条件的45时才会跳出循环。如果将while改为if then, 则第一次就满足循环中断条件, 执行结果为:

```
ex222
n=
    1
```

所以在循环运算时应该使用while还是if, 一定要认真考虑, 因为有时差别会很大。

#### 2.2.3 switch语句

switch语句的指令结构很好, 为条件与条件的排列, 看起来比if then清楚, 可以用于有选择性的程序设计。指令格式为:

```
switch variable
case 1
    运算式1
case 2
    运算式2
.....
otherwise
    运算式n
end
```

表示当variable为1时, 就执行case 1下面的运算式, 其余依次类推。但要注意的是, 如果variable是字符串, case 1就要把1改为字符串, 变成case '1'。

例如下例, 设计功能菜单, 合并范例2.5与范例2.6的程序, 选1则执行ex221, 选2则执行ex222, 如果选错则会出现错误信息。

结果如下:

程序: ex223.m

```
%the switch command evaluation
clear;
['Input the desired number';...
'1.Execute the ex221      ':'...
'2.Execute the ex222      ':']
ip=input('Please input the number: ','s');
switch ip
    case '1'
        ex221
    case '2'
        ex222
    otherwise
        disp('The keyin number is wrong')
end
```

执行结果:

```
Input the desired number
1.Execute the ex221
2.Execute the 2x222
please input the number:1
```

## 2.3 系统分析技巧

### 2.3.1 文件读取与处理

在工程分析中常会需要从硬盘中读取数据,以便供MATLAB使用,用到的指令是fopen与fscanf。指令格式为:

```
fid=fopen('filename','permission')
```

fid是文件标识符(file identifier), fopen指令执行成功后就会返回一个正的fid值,以作为是否完成读取文件的标志。如果fopen指令执行失败, fid就返回-1。

filename是保存在硬盘中的文件名。

permission是文件允许操作的类型,可设为以下几个值:

'r'	只读。
'w'	只写。
'a'	只能追加(append)。

结果如下:

程序: ex223.m

```
%the switch command evaluation
clear;
['Input the desired number';...
'1.Execute the ex221      ':'...
'2.Execute the ex222      ':']
ip=input('Please input the number: ','s');
switch ip
    case '1'
        ex221
    case '2'
        ex222
    otherwise
        disp('The keyin number is wrong')
end
```

执行结果:

```
Input the desired number
1.Execute the ex221
2.Execute the 2x222
please input the number:1
```

## 2.3 系统分析技巧

### 2.3.1 文件读取与处理

在工程分析中常会需要从硬盘中读取数据,以便供MATLAB使用,用到的指令是fopen与fscanf。指令格式为:

```
fid=fopen('filename','permission')
```

fid是文件标识符(file identifier), fopen指令执行成功后就会返回一个正的fid值,以作为是否完成读取文件的标志。如果fopen指令执行失败, fid就返回-1。

filename是保存在硬盘中的文件名。

permission是文件允许操作的类型,可设为以下几个值:

'r'	只读。
'w'	只写。
'a'	只能追加(append)。



'r+' 可读可写。

与fopen对应的指令为fclose，它用于关闭文件，其指令格式为：

```
status=fclose(fid)
```

如果成功关闭文件，status返回的值就是0。

文件打开之后，如何进行读取呢？使用fscanf或fread就可以读取已经打开的文件，不过要先确认所要读的文件是二进制(binary)文件，还是ASCII码文件。其处理的方式并不相同，简而言之，如果文件内容不是二进制文件而是用ASCII码表示的数值，用fread读取之后，再用str2num进行转换即可。格式如下：

```
a=fread(fid);      此时a为一串二进制码。  
b=char(a);        把二进制码转化成横向的字符串。  
bb=str2num(b);     把字符串转化成数字。
```

bb就是一个可以处理的数值矩阵。

如果文件内容已经是数值而不是ASCII码，则使用fscanf就很方便。其指令格式为：

```
Variable=fscanf(fid, '%g')
```

除了%g外还有%d与%s，意义分别如下：

- %g                表示浮点数值。
- %d                表示十进制数值。
- %s                表示字符串。

可以使用fwrite或fprintf指令，将经过处理后的数据保存起来。不过此时的文件是二进制文件，用记事本等文字编辑器看不到数据的内容。如果要存为ASCII文件，就可以使用save指令或dlmwrite指令。save指令可以将workspace中的参数数据存到文件中，以免在执行clear后数据消失，使用load指令就可以把其再调回来。dlmwrite则是将矩阵数据内容用某一符号作分隔符保存为ASCII文件。

这些指令的格式如下：

### 1. fwrite的指令格式

```
先打开(open)一个文件  
fid=fopen('filename','permission')  
接着  
fwrite(fid, [要保存的数据矩阵], '精度格式')
```

执行help fread即可查到精度格式的设定，通常设为int4，float32等，这根据用户的需求而定。

### 2. fprintf的指令格式

```
fprintf(fid, '数据格式', 需要保存的数据矩阵)
```

'r+' 可读可写。

与fopen对应的指令为fclose，它用于关闭文件，其指令格式为：

```
status=fclose(fid)
```

如果成功关闭文件，status返回的值就是0。

文件打开之后，如何进行读取呢？使用fscanf或fread就可以读取已经打开的文件，不过要先确认所要读的文件是二进制(binary)文件，还是ASCII码文件。其处理的方式并不相同，简而言之，如果文件内容不是二进制文件而是用ASCII码表示的数值，用fread读取之后，再用str2num进行转换即可。格式如下：

```
a=fread(fid);      此时a为一串二进制码。  
b=char(a);        把二进制码转化成横向的字符串。  
bb=str2num(b);     把字符串转化成数字。
```

bb就是一个可以处理的数值矩阵。

如果文件内容已经是数值而不是ASCII码，则使用fscanf就很方便。其指令格式为：

```
Variable=fscanf(fid, '%g')
```

除了%g外还有%d与%s，意义分别如下：

- %g                表示浮点数值。
- %d                表示十进制数值。
- %s                表示字符串。

可以使用fwrite或fprintf指令，将经过处理后的数据保存起来。不过此时的文件是二进制文件，用记事本等文字编辑器看不到数据的内容。如果要存为ASCII文件，就可以使用save指令或dlmwrite指令。save指令可以将workspace中的参数数据存到文件中，以免在执行clear后数据消失，使用load指令就可以把其再调回来。dlmwrite则是将矩阵数据内容用某一符号作分隔符保存为ASCII文件。

这些指令的格式如下：

### 1. fwrite的指令格式

```
先打开(open)一个文件  
fid=fopen('filename','permission')  
接着  
fwrite(fid, [要保存的数据矩阵], '精度格式')
```

执行help fread即可查到精度格式的设定，通常设为int4，float32等，这根据用户的需求而定。

### 2. fprintf的指令格式

```
fprintf(fid, '数据格式', 需要保存的数据矩阵)
```

例如:

```
%EX231_1.m fprintf command
clear
x=0:10
y= [x;x.^2]
fid=fopen('001.txt','w')
fprintf(fid,'e e\n\n')
fprintf(fid,'%6.2f %6.2f\n',y)
status=fclose(fid)
```

即可把数据保存到文件001.txt中。

### 3. save的指令格式

```
save filename 变量1 变量2...
```

使用load filename即可把变量1、变量2...调出来。

如果要保存为ASCII码,就要在后面加上-ascii

```
save filename 变量1 变量2...-ascii
```

对于save指令,处理大量数据存取有一个技巧非常有用,即:

```
save('filename','变量1','变量2'...)
```

由于filename是用字符串表示的,所以可以使用程序进行控制,使其每处理完一次就存一个不同的文件名称,例如:

```
%ex231_2.m test the continuous saving function of command
'save;'
clear
m=1:10;
for i=1:length(m)
    n=m.^2
    nf= [m' n'];
    save(['data' int2str(i)], 'nf')
end
```

经过处理后即可连续自动保存到文件data1至data10中。

假设下面范例中的data是从外部通过数据读取系统获得的数据,先保存到磁盘内,再将其复制到硬盘等待处理。文件ex231data.txt是从外部录制的信号文件,将其转换并画出图形,如图2.7所示。最后再将其转换为ASCII码。

**范例2.7** 从硬盘读取文件ex231data.txt并将结果画出,最后将数据保存为ASCII文件。

程序: ex231\_3.m

例如:

```
%EX231_1.m fprintf command
clear
x=0:10
y= [x;x.^2]
fid=fopen('001.txt','w')
fprintf(fid,'e e\n\n')
fprintf(fid,'%6.2f %6.2f\n',y)
status=fclose(fid)
```

即可把数据保存到文件001.txt中。

### 3. save的指令格式

```
save filename 变量1 变量2...
```

使用load filename即可把变量1、变量2...调出来。

如果要保存为ASCII码,就要在后面加上-ascii

```
save filename 变量1 变量2...-ascii
```

对于save指令,处理大量数据存取有一个技巧非常有用,即:

```
save('filename','变量1','变量2'...)
```

由于filename是用字符串表示的,所以可以使用程序进行控制,使其每处理完一次就存一个不同的文件名称,例如:

```
%ex231_2.m test the continuous saving function of command
'save;'
clear
m=1:10;
for i=1:length(m)
    n=m.^2
    nf= [m' n'];
    save(['data' int2str(i)], 'nf')
end
```

经过处理后即可连续自动保存到文件data1至data10中。

假设下面范例中的data是从外部通过数据读取系统获得的数据,先保存到磁盘内,再将其复制到硬盘等待处理。文件ex231data.txt是从外部录制的信号文件,将其转换并画出图形,如图2.7所示。最后再将其转换为ASCII码。

**范例2.7** 从硬盘读取文件ex231data.txt并将结果画出,最后将数据保存为ASCII文件。

程序: ex231\_3.m

```

%data i/o;
clear;
fid=0;%initialize file identifier
while fid<1
    filename='ex231data.txt';
    [fid,message] =fopen(filename, 'r');%open file from disk
    if fid== -1
        disp(message)
    end
end
data=fscanf(fid, '%g');%read data from file
%get the time point reference
for u=1:length(data);
    t(u)=u;
end;
plot(t,data)
status=fclose(fid)

```

打开文件

读取文件

关闭文件

执行的结果如图2.7所示:

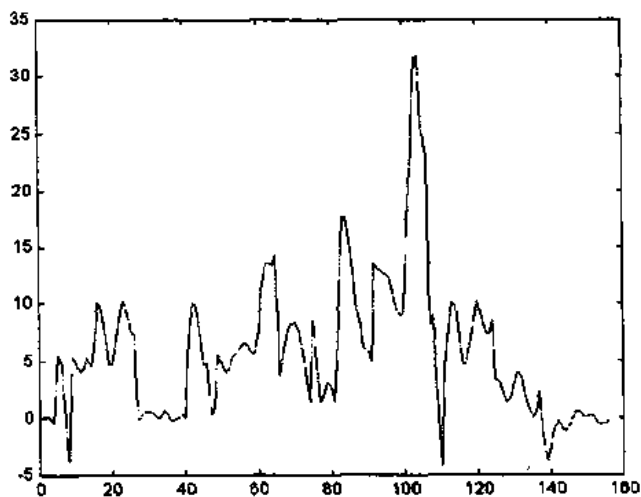


图 2.7 信号波形图

下面的程序在读取文件ex231data.txt后将其保存成名为001.bin的二进制文件, 以及名为001.dat的ASCII文件。

程序: ex231\_4.m

```

%data i/o;
clear;
fid=0;%initialize file identifier
while fid<1

```

```

%data i/o;
clear;
fid=0;%initialize file identifier
while fid<1
    filename='ex231data.txt';
    [fid,message] =fopen(filename, 'r');%open file from disk
    if fid== -1
        disp(message)
    end
end
data=fscanf(fid, '%g');%read data from file
%get the time point reference
for u=1:length(data);
    t(u)=u;
end;
plot(t,data)
status=fclose(fid)

```

打开文件

读取文件

关闭文件

执行的结果如图2.7所示:

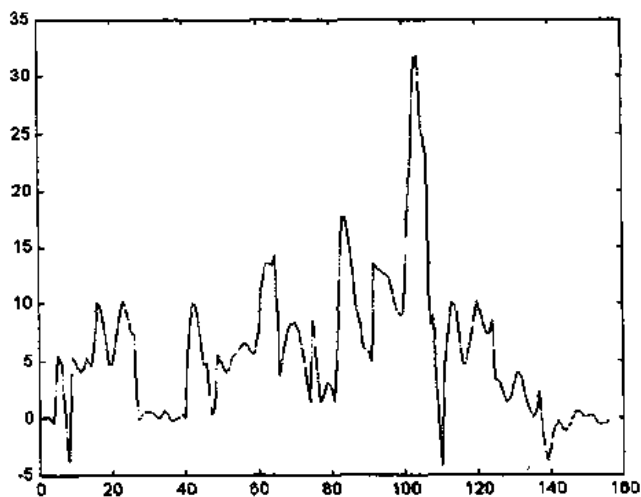


图 2.7 信号波形图

下面的程序在读取文件ex231data.txt后将其保存成名为001.bin的二进制文件, 以及名为001.dat的ASCII文件。

程序: ex231\_4.m

```

%data i/o;
clear;
fid=0;%initialize file identifier
while fid<1

```

```

filename='ex231data.txt';
fid,message]=fopen(filename,'r');    %open file from disk
if fid==-1
    disp(message)
end
end
data=fscanf(fid,'%g');                %read data from file
%get the time point reference
for u=1:length(data);
    tt(u)=u;
end;
fwriteid=fopen('001.bin','w') —— 存为二进制文件
count=fwrite(fwriteid,[tt',data'],'float32')
tf=[tt',data'];
save(['001'.dat'],'tf','-ascii') —— 存为ASCII文件

```

比较文件001.dat的前面部分与文件ex231data.txt的大小，结果相同，只是存储的格式不同而已，所以分辨数据的格式是很重要的。

文件001.dat的输出格式：

```

1.0000000e+000    6.3231000e-002
2.0000000e+000    6.3231000e-002
3.0000000e+000   -1.7901000e-001
4.0000000e+000   -4.2125000e+000
5.0000000e+000    5.3925000e+000
6.0000000e+000    4.7869000e+000
7.0000000e+000   -3.0013000e-001
8.0000000e+000   -4.1760000e+000
9.0000000e+001    5.3925000e+000
1.0000000e+001    4.7869000e+000
1.1000000e+001    4.0602000e+000
1.2000000e+001    4.1813000e+000
1.3000000e+001    5.2714000e+000
1.4000000e+001    4.5447000e+000
1.5000000e+001    6.9670000e+000

```

文件ex231data.txt的输出格式：

```

0.063231
0.063231
-0.179010
-0.421250
5.392500
4.786900
-0.300130
-4.176000

```

```

filename='ex231data.txt';
fid,message]=fopen(filename,'r');    %open file from disk
if fid==-1
    disp(message)
end
end
data=fscanf(fid,'%g');                %read data from file
%get the time point reference
for u=1:length(data);
    tt(u)=u;
end;
fwriteid=fopen('001.bin','w') —— 存为二进制文件
count=fwrite(fwriteid,[tt',data'],'float32')
tf=[tt',data'];
save(['001'.dat'],'tf','-ascii') —— 存为ASCII文件

```

比较文件001.dat的前面部分与文件ex231data.txt的大小，结果相同，只是存储的格式不同而已，所以分辨数据的格式是很重要的。

文件001.dat的输出格式：

```

1.0000000e+000    6.3231000e-002
2.0000000e+000    6.3231000e-002
3.0000000e+000   -1.7901000e-001
4.0000000e+000   -4.2125000e+000
5.0000000e+000    5.3925000e+000
6.0000000e+000    4.7869000e+000
7.0000000e+000   -3.0013000e-001
8.0000000e+000   -4.1760000e+000
9.0000000e+001    5.3925000e+000
1.0000000e+001    4.7869000e+000
1.1000000e+001    4.0602000e+000
1.2000000e+001    4.1813000e+000
1.3000000e+001    5.2714000e+000
1.4000000e+001    4.5447000e+000
1.5000000e+001    6.9670000e+000

```

文件ex231data.txt的输出格式：

```

0.063231
0.063231
-0.179010
-0.421250
5.392500
4.786900
-0.300130
-4.176000

```



```
5.392500
4.786900
4.060200
4.181300
5.271400
4.544700
6.967000
```

### 2.3.2 字符数据的处理

字符(Character)数据的处理主要是创建字符串(String),最常用的指令就是char,其指令格式为:

```
String=char('c1','c2',...)
```

执行后的字符串是左右排列的行向量(row vector),如下例:

```
>> s=char('Hello!','My name is John')
s=
Hello!
My name is John
```

如果想要把字符串变量接到固定的字符串之后,可以使用个别转置后再全部一起转置的技巧来实现,这在绘图指令title中常常用到。

范例2.8 用char指令创建一行文字。

程序: ex232.m

```
%evaluate the function of char making a line word
```

```
name=input('Input your name: ','s'); ——— 最后的s代表输入的是字符串
```

```
char(('Your name is ',name')) ——— 个别转置后再全部转置
```

执行结果:

```
Input your name:John
ans =
Your name is John
```

实际使用中可以输入一组文件名称,配合使用eval指令,可以自动执行一组程序。但要注意,被调用的程序中有clear指令时会清除参数,发生错误,因此要暂时取消被调用程序中的clear指令,在主程序的文件头只执行一次clear即可。下面举例说明如何读取一组数据。假设硬盘上有一组数据,不需要一笔一画的执行读入程序,可以将范例2.7的程序进行修改即可完成所有的读取操作。如范例2.9和图2.8、图2.9及图2.10所示。

范例2.9 连续读取ex231data.txt至ex233data.txt的三组数据,并画出图形。

程序: ex233.m

```
%data i/o;
clear;
files=char('ex231data.txt','ex232data.txt','ex233data.txt');%define file array
[m,n] =size(files);%find the number of files
for f=1:m
    filename=files(f,:);
    fid=0;%initialize file identifier
    while fid<1
        [fid,message] =fopen(filename,'r');%open file from disk
        if fid==-1
            disp(message)
        end
    end
    data=fscanf(fid,'%g');%read data from file
    %get the time point reference
    for u=1:length(data);
        t(u)=u;
    end;
    figure(f)
    plot(t,data)
    xlabel('time(ms)')
    ylabel('magnitude')
    title(char('The Diagram of Data: '),filename));
    status=fclose(fid)
end
```

利用范例2.8的技巧

执行结果:

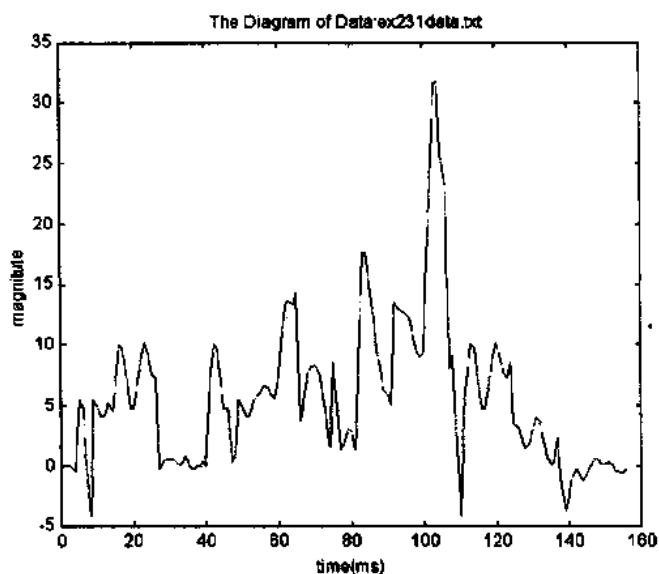


图 2.8

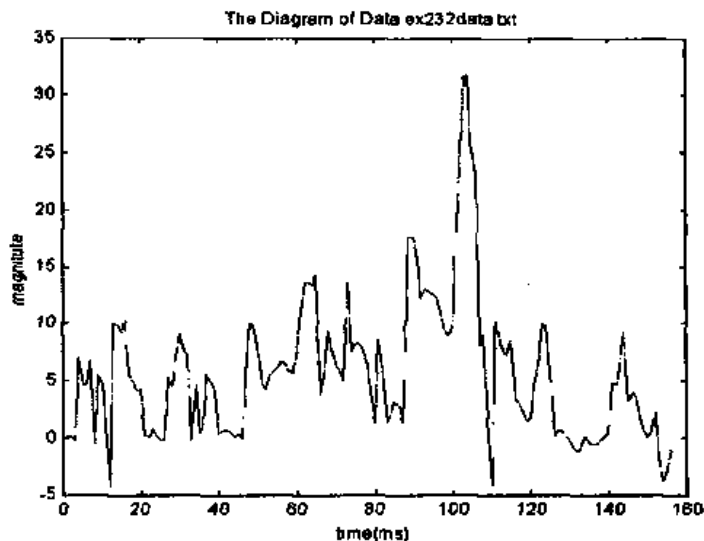


图 2.9

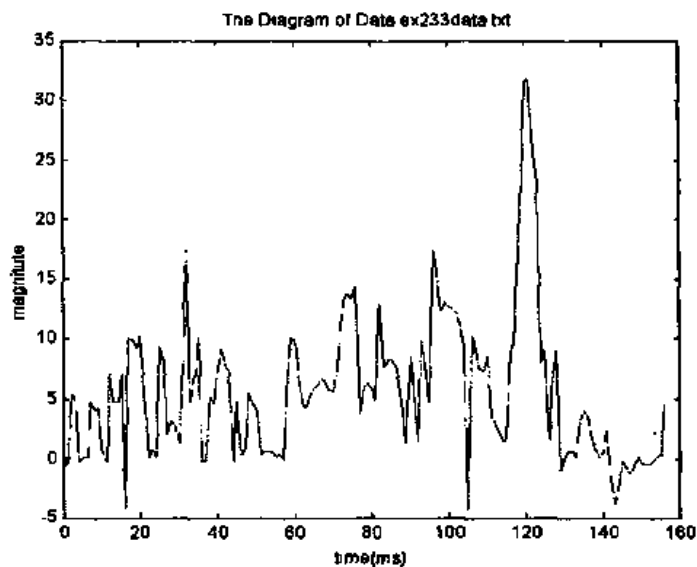


图 2.10

### 2.3.3 数据的频谱分析

在信号处理与图像处理方面,数据频谱分析是很重要的,所需的数学基础是傅立叶变换(Fourier Transform),MATLAB指令则是FFT,指令格式为:

$$y=\text{FFT}(x)$$

$x$ 经过FFT变换后得到复数数据行 $y$ ,因此要靠另外两个指令`abs`与`angle`来求 $y$ 的大小与相位,参照范例2.10的说明。至于其他的相关指令还有`fft2`,`fft`等等,读者可以在命令窗口下执行`help matlab\datafun`,即可查看其指令内容。

### 范例2.10 数据的傅立叶分析。

程序: ex234.m

```
%ex234.m
%data FFT;
clear;
t=0:1/99:1;
data=sin(2*pi*5*t)+sin(2*pi*4*t);
power=fft(data)
mag=abs(power);
phs=unwrap(angle(power));
%generate frequency vector
fre=(0:length(t)-1)*99/length(t);
%plot magnitude and phase
subplot(211);
plot(fre,mag);
xlabel('frequency(Hz)')
ylabel('magnitude')
subplot(212)
plot(fre,phs)
xlabel('frequency(Hz)')
ylabel('phase(deg)')
```

傅立叶分析

求大小及相位角

执行结果:

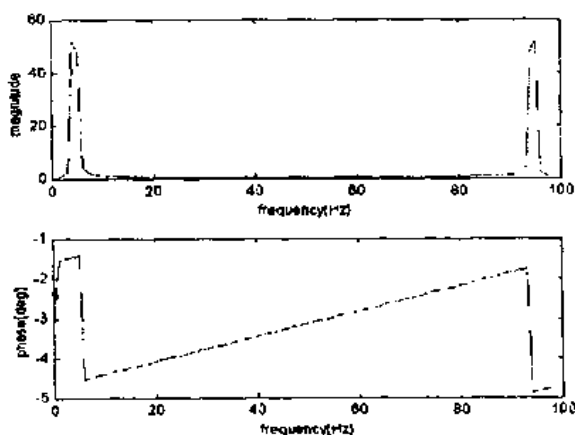


图 2.11 频谱与相位图

### 2.3.4 系统稳定性分析

系统的分析,可以分为时域(Time domain)与频域(Frequency domain)二个区域中的响应(Response)分析。而系统又有单输入单输出(SISO)及多输入多输出(MIMO)的区分,其处理方法与理论也很多,这里不在理论上进行解释,仅就MATLAB的指令应用进行说明。

在时域中,考虑的重点是瞬态响应(Transient Response),也就是系统在接受阶跃输入(Step input)时的响应,分析其稳定时间、超越量(overshoot)以及稳态误差(steady error)。最常用的处理指令是step或是impulse,要注意此时的传递函数是闭环函数(close loop)。在频域中,考虑的重点是相对稳定性(Relative Stability),换句话说,就是考虑增益容限与相位边限的问题,当然改进的办法就是加入校正器。简而言之就是加入:

$$G_c = \frac{1+s}{1+\alpha s}$$

校正器取  $\alpha < 1$  时表示相位优先校正,大于1则表示相位后校正,用MATLAB程序去试验很容易就可以满足实际需求。最常用的处理指令是margin或者nyquist,要注意此时的传递函数是开环函数(open loop)。根据系统的特性方程式为  $1+G(s)H(s)=0$  推算出  $G(s)H(s)=-1$ ,  $-1$  代表什么含义呢?很容易想到的包括  $-180^\circ$ 、0db、 $(-1,0)$  三种含义,这也就是要用开环函数  $G(s)H(s)$  来进行频域分析的原因。在Nyquist定理中,讨论的重点就是用开环函数进行函数映射后,分析其轨迹对  $(-1,0)$  的环绕数,判断闭环的根是否在  $s$  的右半平面,从而决定系统是否稳定。这些对于学过控制理论的读者来说并不陌生,如果想进一步了解指令的内容,可以执行help matlab\toolbox\control目录下或者到帮助窗口中找toolbox\control主题。

范例2.11 求系统开环传递函数为  $\frac{40}{s(s+2)}$  时的时域与频域分析,并校正改进增益容限与相位边限。

由于  $\frac{40}{s(s+2)}$  是系统的开环传递函数,在做频域分析时可以直接使用,因此直接使用指令:

```
nyquist(sys, [1,100])
```

可以发现  $w$  由1变化到100,而路径对  $(-1,0)$  的环绕数  $n$  为0(见图2.12)。由  $\frac{40}{s(s+2)}$  可以知道,在  $s$  右半平面的极点数为0,所以  $p=0$ ,由参数定理(Principle of Argument)可知,根据  $n=z-p$  可知  $z=0$ ,即特性方程式的零点数为0,这就意味着闭环传递函数的极点数为0,所以系统稳定。

为了改善性能,先用margin指令求系统的相位边限与增益容限,再加入校正器

$G_c = \frac{1+s}{1+\alpha s}$  改善其性能。在这里,取  $\alpha < 1$  表示相位优先校正,大于1则表示相位后校正。下面的范例取  $G_c = \frac{1+0.05s}{1+0.05s}$ ,已经大大地改善了性能,与图2.13相比,如图2.14所示,增益容限提高了大约20db,而相位边限也提高了30多度。

最后进行时域分析,注意此时的系统传递函数必须是闭环函数,所以传递函数要变成  $sys/(1+sys)$ 。比较加入校正前的响应图2.15与加入校正后的响应图2.16,可以发现瞬态的性能改善了很多。

程序: ex235.m

```
%nyquist analysis and margin analysis
```

```
sys=tf(40, [1 2 0] );
```

```
sys1=tf( [0.5 1] , [0.05 1] )
```

```
figure(1);
```

```
nyquist(sys, 'r', [1 100] );hold on
```

```
figure(2)
```

```
margin(sys);hold on
```

```
figure(3)
```

```
margin(sys*sys1)
```

```
closys=sys/(1+sys);
```

改为闭环传递函数

```
closys1=sys*sys1/(1+sys*sys1)
```

```
figure(4)
```

```
step(closys)
```

```
figure(5)
```

```
step(closys1)
```

执行结果:

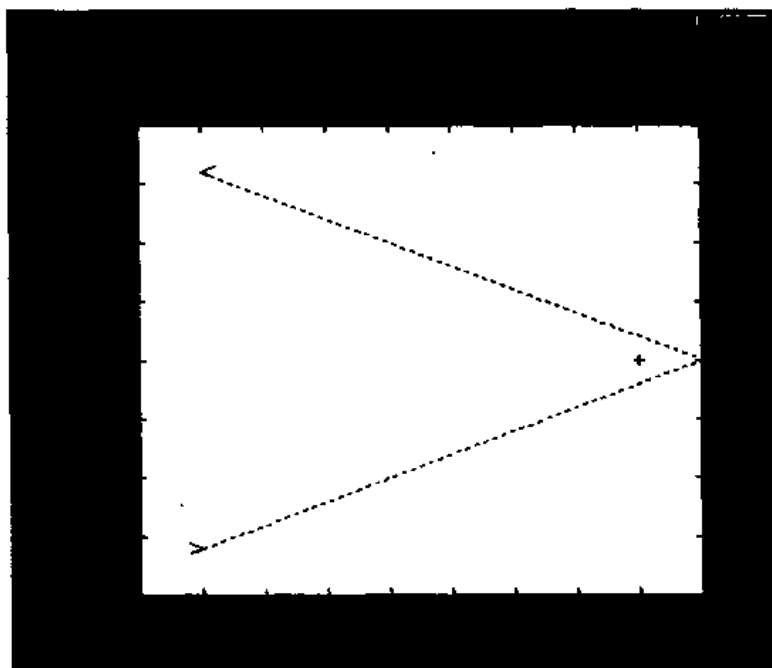


图 2.12 Nyquist 图

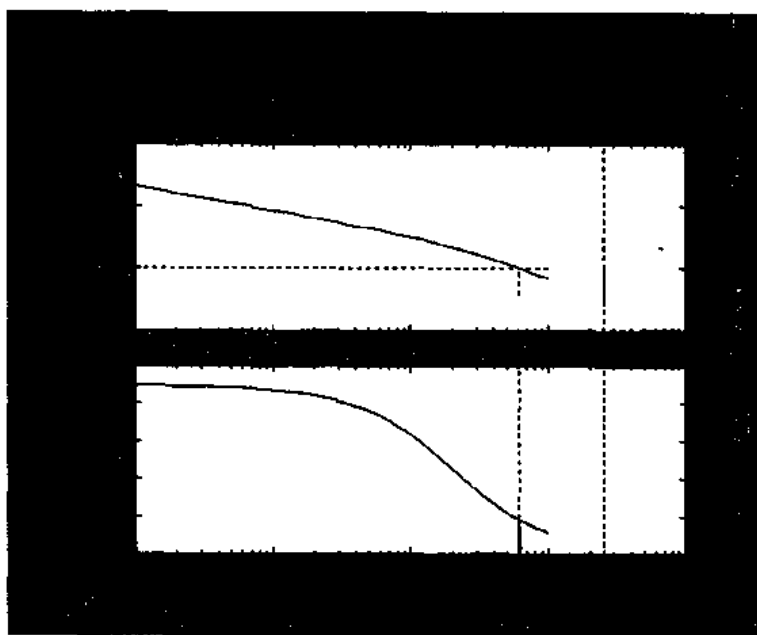


图 2.13 改善前的 Bode 图

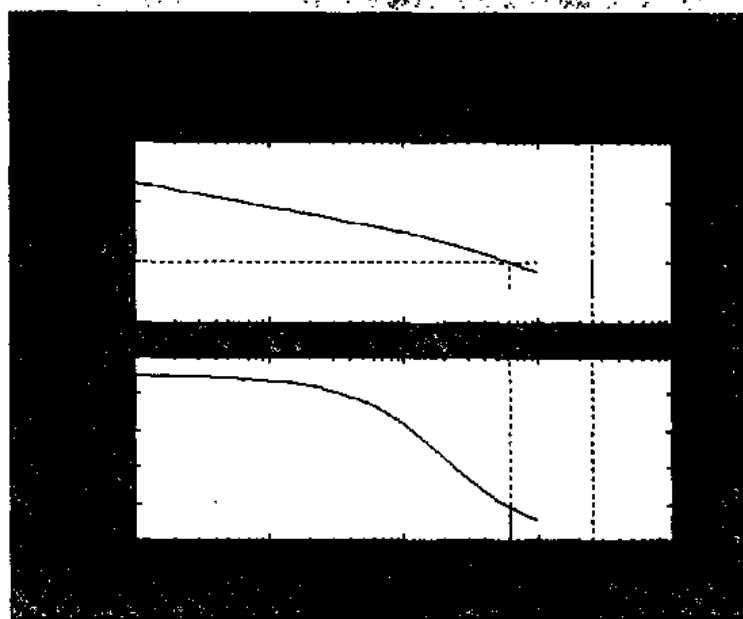


图 2.14 改善后的 Bode 图

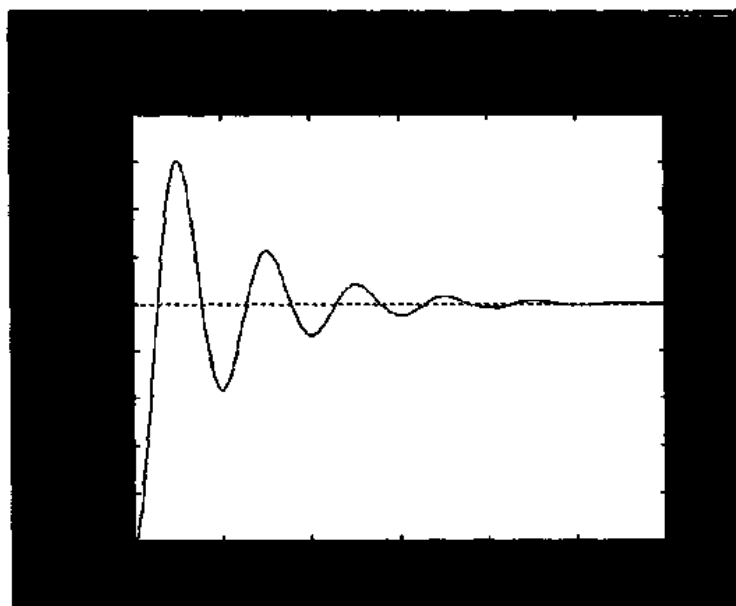


图 2.15 改善前的 Step response

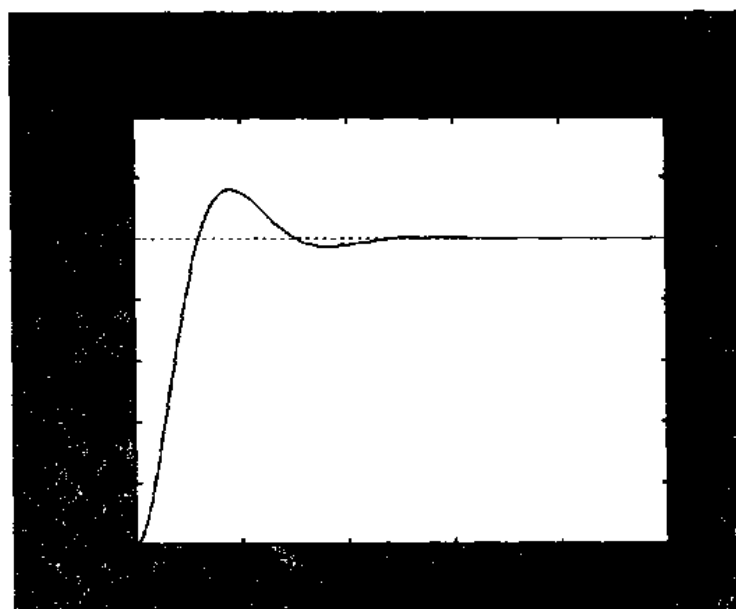


图 2.16 改善后的 Step response

上面的绘图操作可以用一个很好的工具来集成实现，这就是ltiview指令。在命令窗口中执行：

`ltiview`

可以看到如图2.17所示的窗口，它可以节省试验的时间，同时图形间的切换也很容易，使用起来很简单，读者不妨试试看。



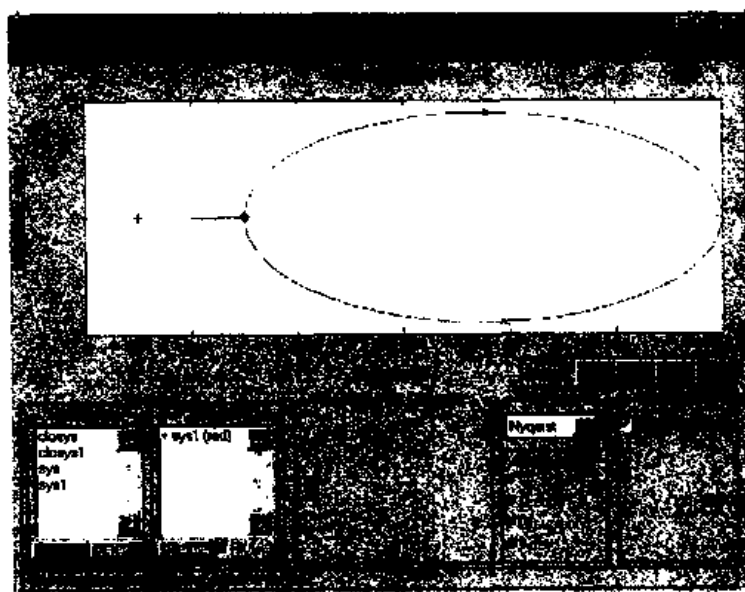


图 2.17 用 ltiview 指令画 Nyquist 图

## 2.4 多项式的求解技巧

### 2.4.1 多项式的建立与表示法

多项式是用向量形式来表示的，从最右边算起，第一个为0阶，第二个为1阶，依次类推。例如一个一元三次多项式

$$4x^3+3x^2+2x+1$$

用  $[4 \ 3 \ 2 \ 1]$  来表示。

conv指令执行多项式的相乘运算，指令格式为：

$$z=\text{conv}(x,y)$$

例如：  $x=[1 \ 3 \ 5]$  ;  $y=[2 \ 4 \ 6]$

$y=$

$$2 \ 4 \ 6$$

$>>z=\text{conv}(x,y)$

$z=$

$$2 \ 10 \ 28 \ 38 \ 30$$

代表  $(x^2+3x+5)*(2x^2+4x+6)$  的结果等于

$$2x^4+10x^3+28x^2+38x+30$$

如果要对两个以上的多项式进行相乘，可以重复使用conv指令，例如：

$$\text{conv}(\text{conv}(x,y),x)$$

```
ans=
```

```
4 28 108 248 380 348 180
```

多项式的分解恰好与上述过程相反，它所用的是dconv指令，其指令格式为：

```
[z,r]=dconv(x,y)
```

表示x除以y，商为z，余数为r。

**范例2.12** 功能求 $2x^4+10x^3+28x^2+38x+30$ 除以 $(x^2+3x+5)$ 的值。

程序：241.m

```
clear;
x=[2 10 28 38 30];
y=[1 3 5];
x=deconv(x,y)
```

执行结果：

```
x=
2 4 6
```

## 2.4.2 多项式的运算

多项式的运算包含求根、求函数值、多项式微分及多项式分解等运算。

下面是相关命令的格式：

a=roots(p)	求出方程 $p=0$ 的根。
p=poly(a)	与roots的指令功能相反，可由根向量a，求出多项式p。 如果a是矩阵，p就是特征方程式(Characteristic polynomials)，可以通过roots(p)求出a的特征值(eigenvalue)。参见范例2.13。
polyval(p,n)	将n代入多项式p求出其函数值。
polyder(p)	求p的微分多项式。
[r,p,k]=residue(x,y)	求x/y的部分分式分解。参见范例2.14。

**范例2.13** 求 $x^2+3x+2=0$ 的根，并验证poly的功能。

程序：ex242.m

```
%calculate the root and recover the polynomials
clear;
x=conv([1 1],[1 2])
rootofx=roots(x)
polyno=poly(rootofx)
```

```
%check the function of poly(x) when x is a matrix
x=[-1 0;0 -2]
cha_x=poly(x);
eig1=roots(cha_x)
eig2=eig(x)
```

说明:

(1) 多项式为:

```
x=
    1    3    2
```

求出根为:

```
rootofx=
    -2
    -1
```

反过来求得多项式为:

```
polyno=
    1    3    2
```

与x相同。

(2) 当x为矩阵时:

```
x=
   -1    0
    0   -2
```

求得特征多项式为:

```
cha_x=
    1    3    2
```

特征值为:

```
eig1 =
    -2
    -1
```

用eig(x)计算的结果为:

```
eig2=
    -1
    -2
```

与poly(x)的计算结果相同。

范例 2.14 用 residue 指令求  $\frac{x}{x^2 + 3x + 2} = 0$  的部分因式分解。

设分子为  $x = [1 \ 0]$  ;分母为  $y = [1 \ 3 \ 2]$

执行

```
>> [r,p,k] = residue(x,y)
```

得到

```
r =
    2
   -1
p =
   -2
   -1
k =
    []
```

表示解为  $\sum_{i=1}^n \frac{r_i}{x-p_i}$  , 亦即  $\frac{x}{x^2 + 3x + 2} = \frac{2}{x+2} + \frac{-1}{x+1} = 0$

### 2.4.3 多项式的拟合

多项式拟合(Polynomial Fitting)又称为曲线拟合(Curve Fitting), 其目的就是在众多的样本点中进行拟合, 找出满足样本点分布的多项式。所用的指令为polyfit, 其指令格式为:

```
p=polyfit(x,y,n)
```

$x$ 与 $y$ 为样本点向量,  $n$ 为所求多项式的阶数,  $p$ 为求出的多项式。

范例2.15 用正弦波验证polyfit的功能。

程序: ex243.m

```
%curve fitting of sin wave
x=0:0.1:2*pi; _____ 产生样本点x,y
y=sin(x)+0.5*rand(size(x))
p=polyfit(x,y,3) _____ 拟合出多项式p
y1=polyval(p,x)
plot(x,y,'+',x,y1,'-')
```

执行结果:

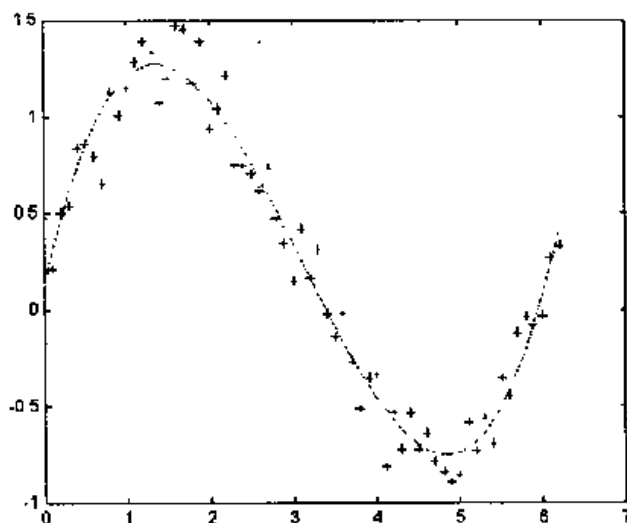


图 2.18 多项式拟合结果图

#### 2.4.4 多项式的插值(Interpolation)

多项式插值是指根据给定的有限个样本点,产生另外的估计点以达到数据更为平滑的效果。该技巧在信号处理与图像处理上应用广泛。所用的指令有一维的`interp1`、二维的`interp2`、三维的`interp3`。这些指令分别有不同的方法(method),在后面会加以介绍,设计者可以按照需求选择适当的方法,以满足系统属性的要求。除了这些指令之外,还有傅立叶方法的`interpft`及n维的`interp`n。如果读者有兴趣,可以在命令窗口下执行`help polyfun`,可以查到更详细的内容。下面介绍主要的插值指令:

`y=interp1(xs,ys,x,'method')`

在向量`xs`与`ys`中,插值产生向量`x`与`y`,所用的方法定义在`method`中,有下面4种选择:

- `nearest`:执行速度最快,输出结果为直角转折。
- `linear`:默认值,在样本点上斜率变化很大。
- `spline`:最花时间,但输出结果也最平滑。
- `cubic`:最占用内存,效果与`spline`差不多。

参见范例2.16。

`z=interp2(xs,ys,zs,x,y,'method')`

在向量`xs`, `ys`与`zs`中,插值产生向量`x`, `y`与`z`,所用的方法定义在`method`中,有下面3种选择:

- `nearest`:产生断片状(piecewise)的平面。
- `linear`:默认值,在样本点上斜率变化很大。
- `cubic`:产生平滑的连续平面,适合于作图像处理。

参见范例2.17。

范例2.16 用interp1验证各种方法的输出结果。

程序: ex244.m

```
%curve interpolation
y= [0 0.9 0.6 1 0 0.1 -0.3 -0.7 -0.9 -0.2] ;
x=0:length(y)-1;
x1=0:0.1:length(y)-1;
y1=interp1(x,y,x1,'nearest');
y2=interp1(x,y,x1,'linear');
y3=interp1(x,y,x1,'spline');
y4=interp1(x,y,x1,'cubic');
figure(1);
plot(x,y,'+k',x1,y1,':k',x1,y2,'-k')
axis( [0 9 -1 1.5] )
legend('sampled point','nearest','linear')
figure(2)
plot(x,y,'+k',x1,y3,':k',x1,y4,'-k')
axis( [0 9 -1 1.5] )
legend('sampled point','spline','cubic')
```

给定x,y后, 由x1插值产生y1(分别使用4种方法)

执行结果:

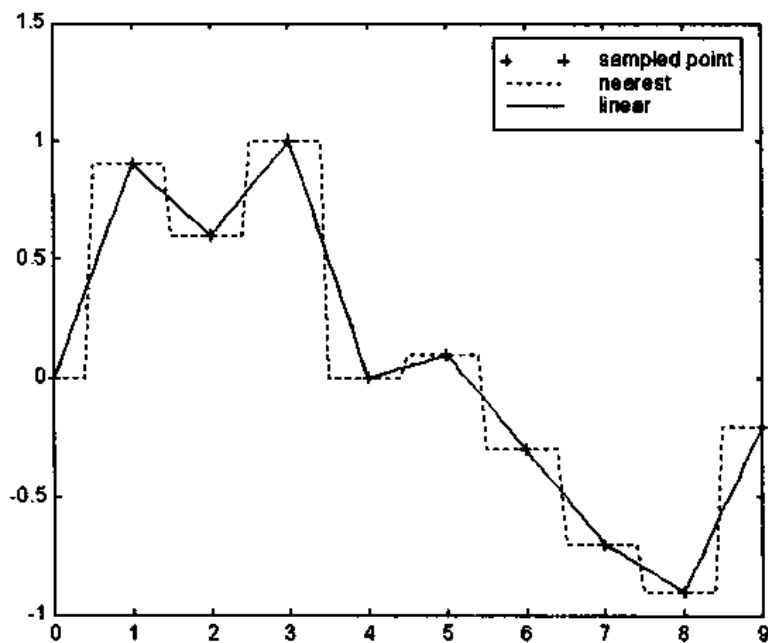


图 2.19 插值结果图

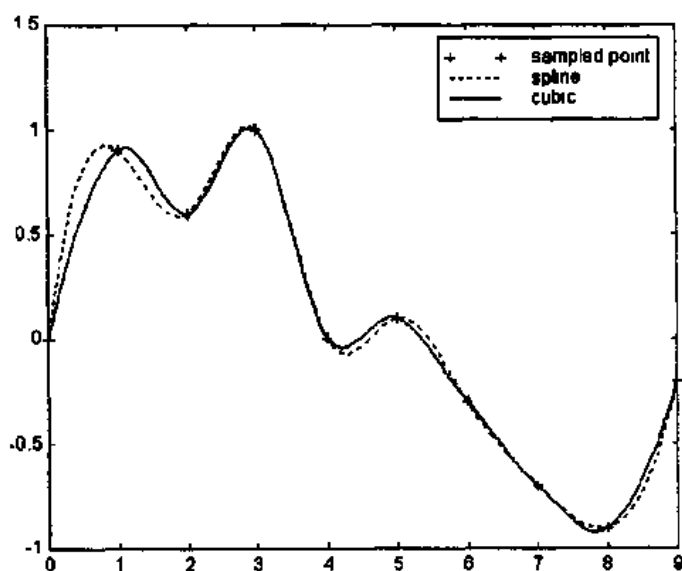


图 2.20 插值结果图

范例2.17 用interp2验证各种方法的输出结果。

程序: ex245.m

```
%surface interpolation
[x,y]=meshgrid(1:10); 参见6.2.1节, 产生立体参考点
[x1,y1]=meshgrid(1:0.4:10);
for i=1:10
    z(i,:)= [0 0.9 0.6 1 0 0.1 -0.3 -0.7 -0.9 -0.2] ;
end
subplot(221)
surf(x,y,z)
legend('sampled data',1) 参见6.1.5节
z1=interp2(x,y,z,x1,y1,'nearest');
└ 给定x, y, z后, 由x1, y1插值产生z1
subplot(222)
surf(x1,y1,z1)
legend('nearest',0)
z1=interp2(x,y,z,x1,y1,'linear');
subplot(223)
surf(x1,y1,z1)
legend('linear',0)
z1=interp2(x,y,z,x1,y1,'cubic');
subplot(224)
surf(x1,y1,z1)
legend('cubic',0)
```

执行结果:

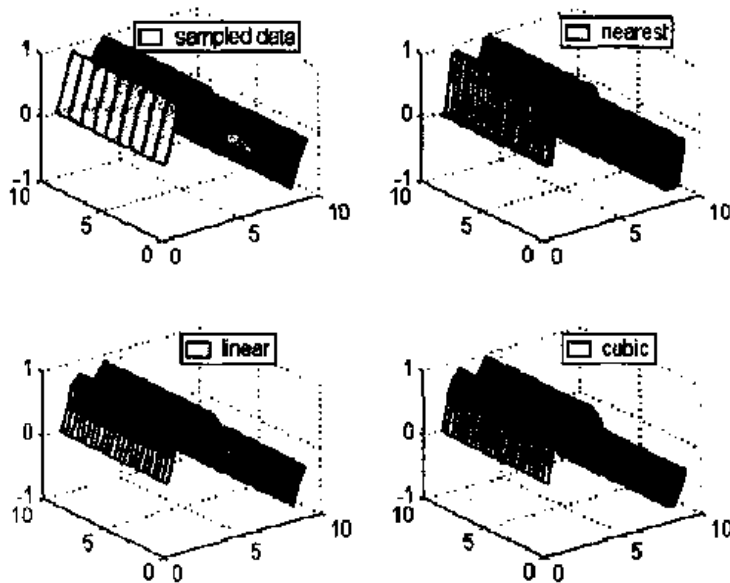


图 2.21 使用 4 种不同方法的插值结果图

## 2.5 矩阵运算技巧

### 2.5.1 矩阵的建立与表示法

在MATLAB中，矩阵的建立非常简单，只要给定一个向量然后用中括号括起来，里面的值就表示矩阵的元素值。例如：

在命令窗口中输入：

```
a=[1 2 3;4 5 6;7 8 9]
```

就可以得到：

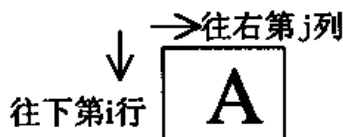
```
a =  
    1    2    3  
    4    5    6  
    7    8    9
```

必须透彻了解MATLAB中表示矩阵行(rows)或列(columns)的方法，这样在以后的程序设计中，尤其是在进行三维空间的数据处理时，才不会出错。下面说明矩阵元素的表示格式：

第*i*行*j*列的矩阵元素=矩阵变量名称(行位置*i*，列位置*j*)

假设有矩阵A，可以想像成下图：





例如输入

```
a(2,2)
```

就会显示出第2行第2列的元素为5。

如果要显示整行或整列，则可以用(:)冒号来表示。冒号(:)代表矩阵中行(rows)或列(columns)的全部。例如：

```
a(:,2)
```

就会显示第2列的全部，结果为：

```
ans =
```

```
2
```

```
5
```

```
8
```

除了可以查看矩阵数据，还可以对其进行替换。例如，可以把刚才看到的矩阵第二行全部换成0，方法如下：

```
a(:,2)=[0;0;0] ——— 注意输入分号(;)才能表示列(column)向量。
```

```
a=
```

```
1  0  3
4  0  6
7  0  9 ——— 第二行全部变成0了。
```

至于其他的矩阵生成方法，可以使用下面的指令：

### 1. 零矩阵与单位矩阵

```
eye(m,n)
```

```
或eye(m)
```

产生 $m \times n$ 或 $m \times m$ 的单位矩阵。

```
zeros(m,n)
```

```
或zeros(m)
```

产生 $m \times n$ 或 $m \times m$ 的零矩阵。

```
ones(m,n)
```

```
或ones(m)
```

产生 $m \times n$ 或 $m \times m$ 的全部元素为1的矩阵。

### 2. 特殊矩阵

```
pascal
```

产生对称矩阵。

```
compan
```

产生伴随矩阵。

```
gallery
```

产生Higham测试矩阵。

```
hadamard
```

产生Hadamard矩阵。

```
hankel
```

产生Hankel矩阵。

```
hilb
```

产生Hilbert矩阵。

invhilb	产生逆Hilbert矩阵。
magic	产生幻方矩阵。
toeplitz	产生Toeplitz矩阵。
vander	产生Vandermonde矩阵。
wilkinson	产生Wilkinson特征值测试矩阵。

### 2.5.2 矩阵的四则运算与转置

矩阵的四则运算符号为：

加	+
减	-
乘	*
除	/

转置运算符号为：

转置(Transpose)

非共轭转置(Unconjugate transpose)

例如：

输入a'

则得到a的转置矩阵：

```
ans=
     1     4     7
     0     0     0
     3     6     9
```

如果输入：

```
b= [1+2i;3+4i]
```

```
b=
```

```
1.0000+2.0000i
3.0000+4.0000i
```

然后得到共轭转置矩阵如下：

```
>> b'
```

```
ans=
```

```
1.0000-2.0000i 3.0000-4.0000i
```

或者得到非共轭转置矩阵如下：

```
>> b.'
```

```
ans =
```

```
1.0000+2.0000i 3.0000+4.0000i
```

要注意的就是只有矩阵维数(Dimension)相同,才能进行运算。  
例如用上面的a和b矩阵做加法,就会产生错误信息。

```
a =  
    1  0  3  
    4  0  6  
    7  0  9  
b =  
    1.0000 + 2.0000i  
    3.0000 + 4.0000i  
a+b  
??? Error using ==>+  
Matrix dimensions must agree.
```

将b改为:

```
b =  
    1  2  3
```

则a与b的转置相乘的结果为:

```
a*b'  
ans =  
    10  
    22  
    34
```

至于其他运算,请读者自行练习。

### 2.5.3 逆矩阵与行列式的求解

求逆矩阵的方法有两种:

方法一:

求解矩阵方程 $AX=B$ , 则 $X=A^{-1}B$ , 指令格式为:

$X=A\backslash B$

求解矩阵方程 $XA=B$ , 则 $X=BA^{-1}$ , 指令格式为:

$X=B/A$

注意上面两者所用的斜线符号不同, 为避免混淆, 建议用方法二。

方法二:

使用求逆矩阵 $X=A^{-1}$ 的指令:

$\text{inv}(A)$

求解矩阵方程 $AX=B$ , 则 $X=A^{-1}B$ , 指令格式为:

$$X = \text{inv}(A) * B$$

求解矩阵方程式  $XA=B$ , 则  $X=BA^{-1}$ , 指令格式为:

$$X = B * \text{inv}(A)$$

求行列式  $|A|$  的指令格式为:

$$\text{det}(A)$$

范例2.18 已知A与B的值, 求出  $AX=B$  和  $BX=A$  的解及  $|X|$  的值。

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 2 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} -2 & 9 & -15 \\ 6 & 15 & -21 \\ -6 & 11 & -2 \end{bmatrix}$$

1. 求出  $AX=B$  的解

方法一的执行结果为:

```
>>a\b
ans=
    2.0000    4.0000   -6.0000
    1.0000    1.0000    0.0000
   -2.0000    1.0000   -3.0000

det(a\b)
ans=
   -12.0000
```

方法二的执行结果为:

```
>> inv(a)*b
ans=
    2.0000    4.0000   -6.0000
    1.0000    1.0000    0.0000
   -2.0000    1.0000   -3.0000

det(inv(a)*b)
ans=
   -12.0000
```

2. 求出  $XA=B$  的解

方法一的执行结果为:

```

b/a
ans=
    25.2500   -3.2500  -17.5000
    30.7500   -0.7500  -22.5000
    35.7500   -5.7500  -24.5000
det(b/a)
ans=
   -12.0000

```

方法二的执行结果为:

```

>> b*inv(a)
ans =
    25.2500   -3.2500  -17.5000
    30.7500   -0.7500  -22.5000
    35.7500   -5.7500  -24.5000
det(b*inv(a))
ans=
   -12.0000

```

#### 2.5.4 矩阵的次方运算

矩阵A的次方运算指令如下:

<code>sqrtm(A)</code>	开根号。
<code>A^2</code>	矩阵平方, 同理3次方则用 <code>A^3</code> 。
<code>expm(A)</code>	A的指数解, 参见范例2.19。
<code>logm(A)</code>	A的对数解。
<code>funm(A,'fun')</code>	求出 <code>fun(A)</code> 的解。例如 <code>log(A)</code> , 可以通过指令 <code>funm(A,'log')</code> 求出, 也可以通过上面的 <code>logm(A)</code> 求出。

范例2.19 如果A值如下, 求出 $\frac{dx}{dt}=Ax$   $x(0)=[1;1;1]$  的解。

```

A=[ -1   0   0
     0  -2   0
     0   0  -3]

```

由 $\frac{dx}{dt}=Ax$ 可知

$$x(t)=e^{At}x(0)$$

程序: ex252.m

```

clear;
a=[-1   0   0
    0  -2   0
    0   0  -3]
x0=[1;1;1];

```

```

t=0:0.1:10;
for i=1:length(t)
    x(:,i)=expm(a*t(i))*x0;
end
plot3(x(1,:),x(2,:),x(3,:));
grid on

```

采用列向量存储数据

采用行向量画图

执行结果:

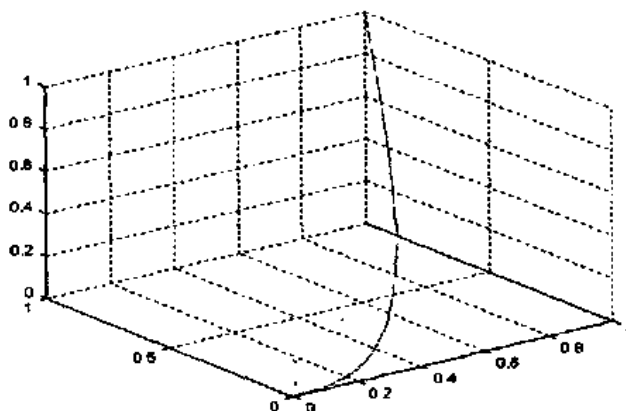


图 2.22 运算结果图

### 2.5.5 矩阵的分解

与矩阵分解相关的指令如下:

- |                           |   |
|---------------------------|---|
| $[U,S,V] = \text{svd}(A)$ | 求矩阵A的奇异值及分解矩阵, 满足 $U \cdot S \cdot V' = A$ (如范例2.20所示)。   |
| $[V,D] = \text{eig}(A)$   | 求矩阵A的特征向量V及特征值D, 满足 $A \cdot V = V \cdot D$ (如范例2.21所示)。  |
| $[Q,R] = \text{qr}(A)$    | 将矩阵A做正交化分解, 使得 $A = Q \cdot R$ 。Q为单位矩阵(unitary matrix), 其范数(norm)为1。R为对角化的上三角矩阵, 如范例2.22所示。                   |
| $[L,U] = \text{lu}(A)$    | 将矩阵A做对角线分解, 使得 $A = L \cdot U$ L为下三角矩阵(lower triangular matrix), U为上三角矩阵(upper triangular matrix), 如范例2.22所示。 |

范例2.20 如果A值如下, 求A的奇异值及分解矩阵。

```

a=[ 9 8
    6 8]
[u,s,v] = svd(a)

```

奇异值为:

```
s=
    15.5765      0
         0     1.5408
```

分解矩阵为:

```
u=
    0.7705   -0.6375
    0.6375    0.7705

v=
    0.6907   -0.7231
    0.7231    0.6907
```

验算结果与矩阵A相同。

```
>> u*s*v'
ans=
    9.0000    8.0000
    6.0000    8.0000
```

范例2.21 同上例, 求A的特征值和特征向量。

```
a= 9  8
    6  8
```

指令:

```
>> [v,d] = eig(a)
```

特征向量为:

```
v= [ 0.7787   -0.7320
     0.6274    0.6813 ]
```

特征值为:

```
d= [ 15.4462      0
      0     1.5538 ]
```

验算:

```
a*v
ans=
    12.0275   -1.1373
     9.6915    1.0586

>> v*d
ans=
    12.0275   -1.1373
     9.6915    1.0586
```

两者运算结果相同。

范例2.22 同上例，求A的正交化分解及LU分解。

```
a= [ 9    8  
     6    8 ]
```

正交化分解：

```
>> [q,r]=qr(a)  
q= [ -0.8321   -0.5547  
     -0.5547    0.8321 ]  
r= [ -10.8167  -11.0940  
      0         2.2188 ]  
  
norm(q)  
ans =  
  
1
```

LU分解：

```
[l,u]=lu(a)  
l= [ 1.0000      0  
     0.6667    1.0000 ]  
u= [ 9.0000    8.0000  
      0     2.6667 ]
```

## 2.5.6 矩阵应用

最常使用矩阵的情况就是先用矩阵保存需要运算的数值，并在运算完毕之后画出图形。上述的一般性指令都是用于运算，当运算完毕之后，最重要的是要将运算结果保存到矩阵中的正确位置上，也就是正确的相对行列位置。也正因为这样，在for loop的程序控制里，要把对应的指标(index)弄清楚才不会出错。下面用一个简单的模拟程序来说明这一设计技巧。

### 范例2.23

本例是求三维空间的导弹制导(Guidance Law)追击模型，导弹追击目标的控制法则称作制导。下面的程序，请读者注意for循环中的指标为k，因此在矩阵ptr与pmr中填入的数值也是以k为指标，通过ptr(:,k)及pmr(:,k)填入数据，最后再通过ptr(1,:), ptr(2,:), ptr(3,:)与pmr(1,:), pmr(2,:), pmr(3,:)画出图形。

程序的算法为比例制导(Proportional Navigation)。对其理论部分有兴趣的读者可以阅读相关资料。

程序：ex253.m

```
%ex253.m..plot guidance law design  
clear;
```



```

clf;
x(1)=0;y(1)=0;z(1)=0;
r(1)=20;o(1)=0;q(1)=0;
kQ=20;kO=100;
axis( [ 0 100 0 200 0 200] );
for k=1:200;
    ptr(:,k)= [ 60+35.*sin(0.1.*k);60;3.*k] ;
    pmr(:,k)= [ x(k);y(k);z(k)] ;
    r(k)=sqrt((ptr(1,k)-pmr(1,k)).^2+(ptr(2,k)-pmr(2,k)).^2+(ptr(3,k)-pmr(3,k)).^2);
    o(k)=asin((ptr(3,k)-pmr(3,k))/r(k));
    q(k)=atan((ptr(2,k)-pmr(2,k))/(ptr(1,k)-pmr(1,k)));
    if q(k)<0;
        q(k)=pi+atan((ptr(2,k)-pmr(2,k))/(ptr(1,k)-pmr(1,k)));
    end;
    t1=0.1;
    %dx=kQ.*cos(q(t)).*tt+kO.*cos(q(t)).*cos(o(t)).*tt;
    %dy=kQ.*sin(q(t)).*tt+kO.*sin(q(t)).*cos(o(t)).*tt;
    %dz=kO.*sin(o(t)).*tt;
    dx=kQ.*cos(q(k)).*t1;
    dy=kQ.*sin(q(k)).*t1;
    dz=kO.*o(k).*t1;
    x(k+1)=x(k)+dx;
    y(k+1)=y(k)+dy;
    z(k+1)=z(k)+dz;
    if r(k)<3.2;break;end;
end;
figure(1);
plot3(ptr(1,:),ptr(2,:),ptr(3,:));
text(ptr(1,k),ptr(2,k),ptr(3,k),...
    'end\rightarrow','horizontalalignment','right','fontsize',10);
text(ptr(1,1),ptr(2,1),ptr(3,1),...
    'start\rightarrow','horizontalalignment','right','fontsize',10);
title('Target trajectory')
xlabel('x^');
ylabel('y^');
zlabel('z^');
axis( [ 0 100 0 200 0 200] );
grid on
figure(2);
plot3(pmr(1,:),pmr(2,:),pmr(3,:));
text(pmr(1,k),pmr(2,k),pmr(3,k),...
    'end\rightarrow','horizontalalignment','right','fontsize',10);
text(pmr(1,1),pmr(2,1),pmr(3,1),...

```

以k为指标

通过ptr(:,k)填入数据

通过pmr(:,k)填入数据

导弹的制导准则

通过ptr(1,:)至ptr(3,:)画出图形

通过pmr(1,:)至pmr(3,:)画出图形

```
    \leftarrowstart','horizontalalignment','left','fontsize',10);
```

```
title('Missile trajectory')
```

```
xlabel('x');
```

```
ylabel('y');
```

```
zlabel('z');
```

```
axis([0 100 0 200 0 200]);
```

```
grid on
```

```
figure(3);
```

```
plot3(pmr(1,:),pmr(2,:),pmr(3,:),'-r',...
```

```
    ptr(1,:),ptr(2,:),ptr(3,:),'-b');
```

```
legend('missile','target',0);
```

```
title('Pursuit trajectory')
```

```
xlabel('x');
```

```
ylabel('y');
```

```
zlabel('z');
```

```
axis([0 100 0 200 0 200]);
```

```
grid on
```

在第6章对画图技巧进行说明

执行结果如图2.23所示:

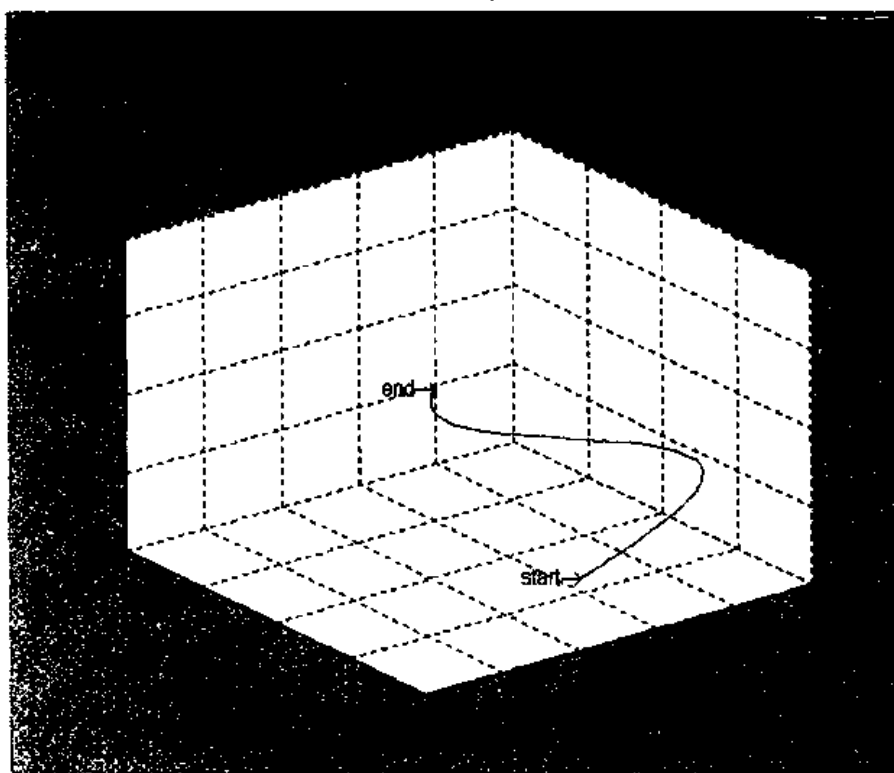


图 2.23 目标的轨迹图

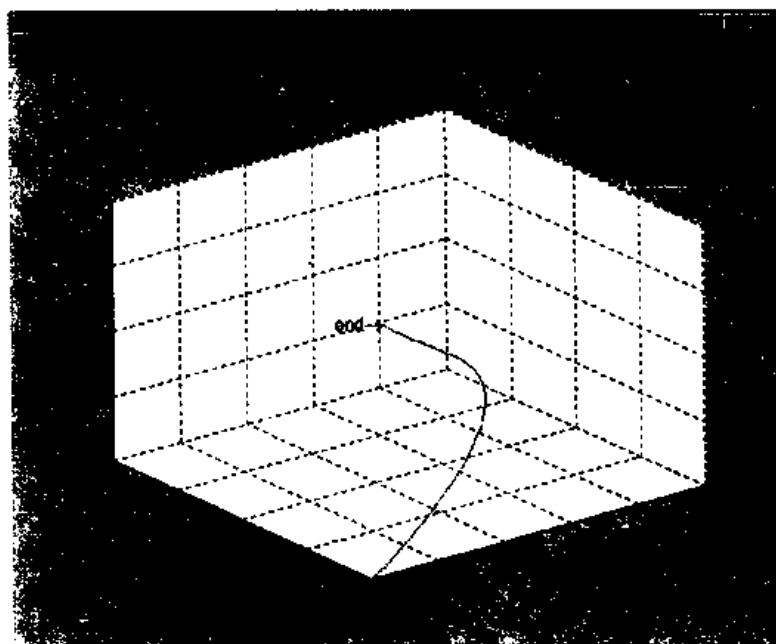


图 2.24 导弹的轨迹图

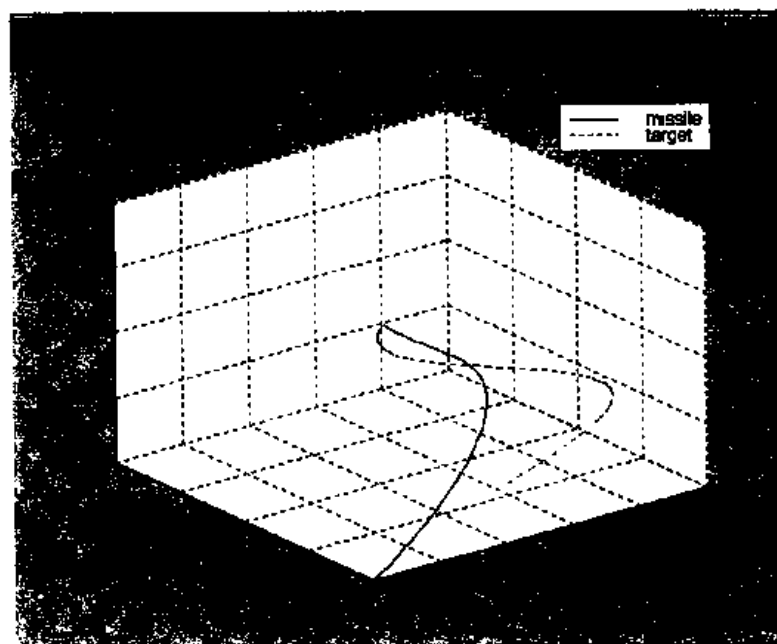


图 2.25 追击的轨迹图

### 2.5.7 利用LMI工具箱解矩阵不等式

在处理多变量控制或强健性(Robust)控制时,常常需要求解矩阵的不等式,以便探讨系统的稳定性问题。例如系统方程式为:

$$\dot{x} = Ax$$

由李亚普诺夫(Lyapunov)定理可知, 必须找到A的对称矩阵, 并满足下列条件:

$$A^T P + P A < 0 \text{ 且 } P > I$$

才能保证系统是稳定的。上式就是矩阵不等式的类型之一, 那么重点就是要求出P, 以满足该不等式。

MATLAB提供了许多解决这一类问题的指令。本节主要介绍用LMI(Linear Matrix Inequality)的方法解矩阵不等式。该方法是用一种特殊算法(Algorithm), 不停地算出其解与条件之间的误差, 当误差落在误差边界(Constraint)以内时, 则表示算出的矩阵与要求的矩阵是接近的, 并且是可接受的(Feasible)。所以LMI的问题不是一种严谨问题, 也就是说解非唯一, 只要结果在接受区间(feasibility)范围内, 就表示所求的解是正确的。

下面先介绍LMI所用到的指令:

1. `setlmis( [ ] )`用于在程序开始时初始化LMI矩阵的值。

2. `lmivar`

指令格式为:

`lmivar(type,struct)`

`type`: 值为1, 2或3。

1代表定义的矩阵为方阵。

2代表定义的矩阵为 $m \times n$ 矩阵。

3代表数值。

`struct`: 配合`type`的值来进行设定, 以定义其维数或值的大小。

用中括号 `[ ]` 输入其值。

在`type`为1时, `[5 1]` 代表 $5 \times 5$ 的方阵, 后面固定输入1。

在`type`为2时, `[5 3]` 代表 $5 \times 3$ 的矩阵。

在`type`为3时, `[5 1]` 代表在其对应位置上分别输入5和1。

3. `lmiedit`: 进入LMI的编辑工具, 主要就是通过这种方式来建立LMI文件。

4. `feasp`: LMI的主要指令, 目的是求出满足边界条件的向量解。最后再用`dec2mat`指令转换成矩阵解。

其指令格式为:

`[tmin,xfeas] = feasp(lmis,options,target)`

`tmin`代表演算结束时`t`的值。

`xfeas`代表求出的可接受的向量解。

`lmis`代表LMI问题的源文件名称。

`options`为含有5个值的列向量, 用中括号表示。

第一个`option`不使用, 默认为0。

第二个`option`定义运算的最大次数。

第三个option定义可接受的边界半径大小。该值会影响求出来的结果。

第四个option定义一个整数值(i)，代表运算的比较次数，当与前面第i个步骤比较，其精度无法小于1%时则停止运算(默认值为10)。

第五个option不使用，默认为0。

target: 设定当 $t < \text{target}$ 时，则停止运算。

下面用一个范例说明LMI的用法。

范例2.24 用lmiedit求P，使其满足 $A^T P + P A < 0$ 且 $P > I$ ，其中 $A = \begin{bmatrix} -1 & 1 \\ 3 & -4 \end{bmatrix}$ 。

第一步：先在命令窗口中输入：

```
a = [ -1    1; 3    -4 ]
a =
    -1     1
     3    -4
```

再执行

```
lmiedit
```

进入编辑窗口，如图2.26所示。

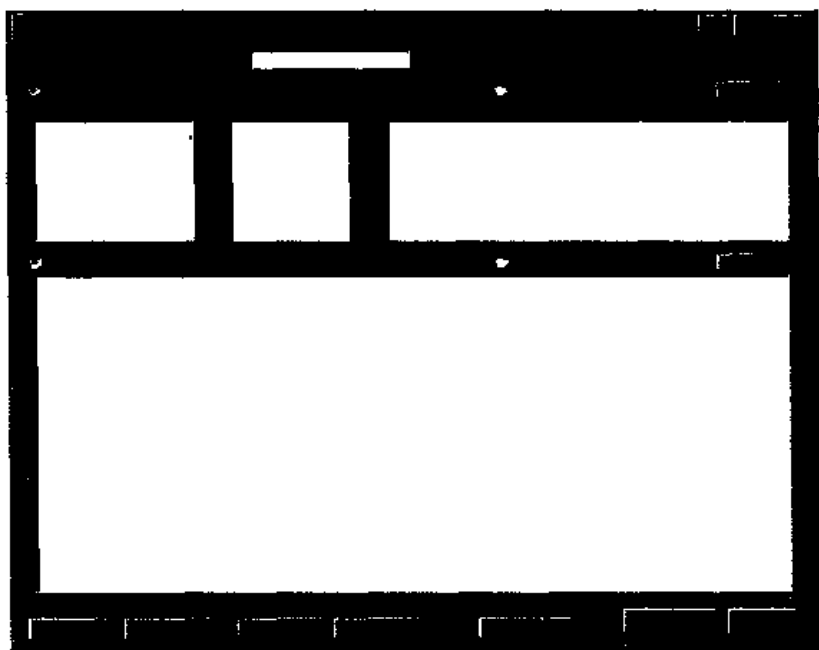


图 2.26 LMI 编辑窗口

第二步：在最上方命名该LMI系统的文件名，例如lmitest。接着定义P矩阵为 $2 \times 2$ 的方阵，定义的方法参见lmivar的说明，在这里s为symmetric的意思，[2 1]为 $2 \times 2$ 方阵的意思；最后再在下方的空白区域内输入矩阵不等式，如图2.27所示。用户可以选择view command选项，查看系统将其转换成指令的内容，如图2.28所示。

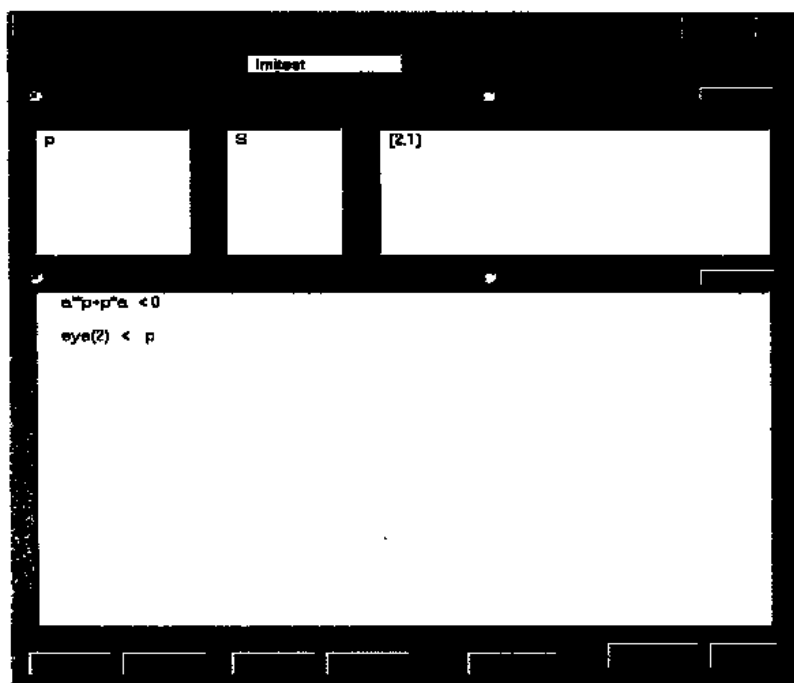


图 2.27 编辑 LMI 不等式

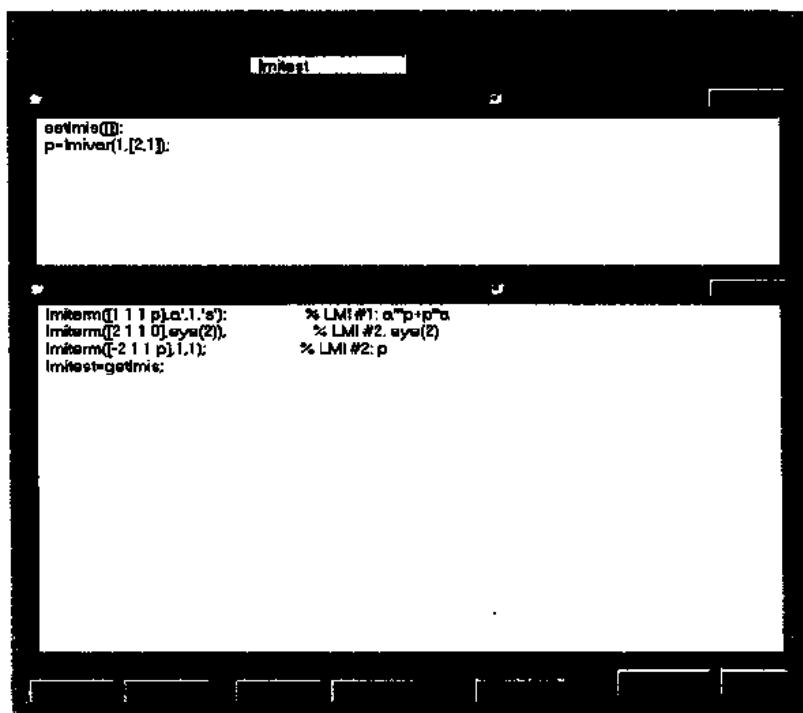


图 2.28 转换后的 LMI 指令内容

**第三步：**在编辑窗口的下方，单击create按钮，产生lmis内部文件。单击write保存文件，例如可将其存为lmitest。在第二次及以后的应用时，先单击read，在出

现输入框后,输入文件名lmitest,确认即可。而第一步及第二步的两个步骤就可以省略了。

第四步:编写执行程序,调用lmi文件,用feasp指令求P。

例如下面程序:feas\_out.m

```
%calculate the LMI feasible matrix P
[tmin,xfeas]=feasp(lmitest,[0;10000;3.7129;5;0],-3);
p=dec2mat(lmitest,xfeas,p)
```

第五步:在命令窗口中给矩阵a赋值

$$a = \begin{bmatrix} -1 & 1 \\ 3 & -4 \end{bmatrix}$$

并执行第四步的程序feas\_out。

执行结果:求出矩阵P为

$$\begin{bmatrix} 3.0727 & -0.1540 \\ -0.1540 & 1.8713 \end{bmatrix}$$

过程如下:

```
Solver for LMI feasibility problems L(x) < R(x)
This solver minimizes t subject to L(x) < R(x) + t*I
The best value of t should be negative for feasibility
Iteration : Best value of t so far
1          0.987595
2          0.273201
3          -0.269137
4          -0.476433
*** new lower bound : -1.562451
5          -0.626669
*** new lower bound : -1.223530
6          -0.791197
*** new lower bound : -1.017599
7          -0.850745
*** new lower bound : -0.927888
Result: best value of t: -0.850745
guaranteed absolute accuracy: 7.71e-002—精度
f-radius saturation: 99.655% of R=3.71e+000
P=
3.0727 -0.1540
-0.1540 1.8713
```

读者可以尝试修改矩阵不等式及feas\_out.m程序中feasp指令的设定值,求出不同的正

确P值。应该注意的是,先要在执行clear指令后,从第三步的read开始调用lmitest,重新执行或者再单击create按钮后,再执行feas\_out,否则会出现下面的错误信息,这或许是LMI工具箱的缺陷。

```
??? Undefined function or variable 'lmitest'.
Error in=> E:\Ousermon\chp2\feas_out.m
On line 2 => [tmin,xfeas] =feasp(lmitest, [0;10000;2;5;0] , -3);
```

如果将feasp指令中的3.7129改为2之后,执行结果为:

```
Solver for LMI feasibility problems L(x) < R(x)
This solver minimizes t subject to L(x) < R(x) + t*I
The best value of t should be negative for feasibility
Iteration :   Best value of t so far
    1         1.013525
    2         0.890978
    3         0.247203
***      new lower bound :   -1.473742
    4         0.071218
***      new lower bound :   -0.505786
    5        -0.191886
***      new lower bound :   -0.364128
    6        -0.223400
***      new lower bound :   -0.300073
    7        -0.240874
***      new lower bound :   -0.272308
    8        -0.247343
***      new lower bound :   -0.260482
Result: best value of t:   -0.247343
      guaranteed absolute accuracy: 1.31e-002———精度
      f-radius saturation: 99.991% of R=2.00e+000
P=
    1.5205    -0.0761
   -0.0761    1.2729
```

可以发现将3.7129改为2之后缩小了误差边界,所得到的误差精度由0.077提高到0.013。验算结果如下,这已经满足设计要求了。

```
eig(p)
ans=
    3.0921
    1.8519
```

均大于1。

```
>> eig(a'*p+p*a)
```



```
ans=
    -0.8649
   -21.4833
```

均小于0。

### 2.5.8 矩阵方程式的求解

矩阵方程式的求解与多项式的求解方式相同，只不过代入的值是矩阵而不是数值，所用的指令则是polyvalm。指令格式为：

$Y = \text{polyvalm}(p, x)$

p为向量表示的多项式，x为要代入的矩阵值。参见范例2.25。

范例2.25 用polyvalm求矩阵方程式 $y(x) = x^3 + x^2 - 3x - 2$ ，其中 $x = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$ 。

程序：ex254.m

```
clear
%y(x)=x^3+x^2-3x-2;
p=[1 1 -3 -2];
x=[2 3; 4 5];
y=polyvalm(p,x)
```

执行结果：

```
y=
    124    165
    220    289
```

### 2.5.9 矩阵内容的查找与修改

矩阵的内容存放着运算的结果，因此有时候需要对其进行变形、查找或更改其值等操作。变形的方法就是用reshape指令，指令格式为：

$\text{reshape}(a, m, n)$

表示将a矩阵变形为m行n列。

要删除矩阵中的整行或整列，可以使用代表空矩阵的中括号[]。例如要删除第二行，就可以执行：

$a(:, 2) = []$

至于查找则使用find指令，去找出满足条件的指标值(indices)，其指令格式为：

指标值=find(查找条件)

下面的范例说明变形、删除、查找和更改其值的过程。

#### 范例2.26

```

a = magic(4) 建立四维的矩阵a
a =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> b=reshape(a,2,8) 将a变形为2*8的矩阵
b =
    16     9     2     7     3     6    13    12
     5     4    11    14    10    15     8     1
>> a(:,2)= [] 删除矩阵a的第二列
a =
    16     3    13
     5    10     8
     9     6    12
     4    15     1
k=find(a>10) 查找矩阵a中值大于10的指标
k =
     1
     8
     9
    11
>> a(k)=99 将对应指标的值变为99
a =
    99     3    99
     5    10     8
     9     6    99
     4    99     1

```

## 2.6 常微分方程ODE设计技巧

常微分方程(ODE)在工程、数学等领域应用非常广泛，是非常重要的解题工具。MATLAB在这方面提供了很好的解题指令，使求解微分方程变得很容易，并能将问题及解答表现在图形上。

### 2.6.1 ODE解题器的指令格式

ODE的解题器(Solver)可以在给定的初始时间及条件下，通过数值方法计算每个程序步骤的解(Solutions)，并验证该解是否满足给定的容许误差(Tolerances)，如果满足，则该解就是一个正确的解；否则就再试一次，直到求出解为止。常用的指令有5个，分别用于解决非难问题(Nonstiff Problems)以及艰难问题(Stiff Problems)，下面是指令的格式及说明。

指令格式：

```
[t,y] = solver('F',tspan,y0)
```

说明:

1. solver有5种, 各不相同的功能可供选择应用, 它们分别是:

- ode45 采用Runge-Kutta解法的中阶解法, 用于解决非难问题, 应用最广。
- ode23 采用Runge-Kutta解法的低阶解法, 用于解决非难问题。在容许误差大的情况下, 比ode45效率高。
- ode113 采用Adams-Bashforth-Moulton解法的变阶解法(variable order), 用于解决非难问题。在容许误差小的情况下, 比ode45有效率。
- ode15s 采用数值差分的方法, 也称为Gear's方法, 用于解决艰难问题, 当ode45、ode113无法解决问题时, 可以尝试采用ode15s去求解。
- ode23s 采用修正的Rosenbrock二阶解法, 解决部分ode15s无法解决的艰难问题。在容许误差大的情况下, 使用ode23s比ode15s有效率。

2. F是描述常微分方程的ODE文件名, 这便于解题器调用它。其建立的指令格式如下:

```
function ydot = F(t,y)
    ydot = [表达式1; 表达式2; 表达式3]
```

表达式的写法在后面章节中详细介绍。

3. tspan为一个行向量, 描述运算的起止时间。例如:

[0 20] 代表起始时间为0, 终止时间为20。

4. y0为初始状态值, 用列向量来表示。例如:

[2;0;0] 代表 $y_1(0)$ 为2,  $y_2(0)$ 为0,  $y_3(0)$ 为0。

### 2.6.2 编写ODE文件

在介绍ODE文件的编写之前, 先对处理问题的步骤加以说明。当我们要解决一个常微分方程例如 $y^{(n)}=f(t,y,y',y'',\dots)$ 的时候, 处理的步骤如下:

**第一步:** 先把问题转化为一阶问题。

利用代换法, 令

```
y1=y
y2=y'
y3=y''
:
yn=y(n-1)
```

所以 $y^{(n)}=f(t,y,y',y'',\dots)$ 变成一阶的微分方程, 如下:

```
y1' = y2
y2' = y3
y3' = y4
```

$$\begin{aligned} &: \\ &y'_{n+1} = y_n \end{aligned}$$

第二步：应用ODE文件描述一阶问题。

$$\text{令 } Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \text{ 则 } y^{(n)} = f(t, y, y', y'' \dots) \text{ 的问题变成}$$

$$Y' = F(t, y)$$

所以就用上一节介绍的建立一个常微分方程文件的方法来描述它，如下：

```
function ydot=F(t,y)
    ydot= [表达式1; 表达式2; 表达式3; ...; 表达式n-1]
    表达式1 → y'_1 = y_2
    表达式2 → y'_2 = y_3
    表达式3 → y'_3 = y_4
    :
    表达式n-1 → y'_{n-1} = y_n
```

范例2.27 求出 $y''' + y'' + y' = 0$ 的ODE文件。

令

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \end{aligned}$$

所以 $y''' + y'' + y' = 0$ 等同于下式：

$$\begin{aligned} y'_1 &= y_2 \\ y'_2 &= y_3 \\ y'_3 &= -y_3 - y_2 \end{aligned}$$

表达式对应如下：

$$\begin{aligned} \text{表达式1为 } y'_1 &= y_2 \\ \text{表达式2为 } y'_2 &= y_3 \\ \text{表达式3为 } y'_3 &= -y_3 - y_2 \end{aligned}$$

因此ODE文件为：

```
function ydot=dlfun(t,y);
    ydot= [y(2);y(3);-y(3)-y(2)]
```

第三步：利用解题器指令求解y。

用2.6.1节介绍的解题器命令调用ODE文件，解出 $[t, y]$ 。t为时间向量，对应的y矩阵为对应该时间的 $y_1, y_2, \dots, y_n$ 的解。

范例2.28 用ode45调用范例2.27的ODE文件解y。

程序为ex262.m

```
clear
y0= [10;1;0];
[t,y]=ode45('dlfun',[0 120],y0);
plot(t,y(:,1))
xlabel('t')
ylabel('y')
```

设定y的初始值

用ode45调用ODE文件dlfun, 运算120秒。

执行结果如图2.29所示:

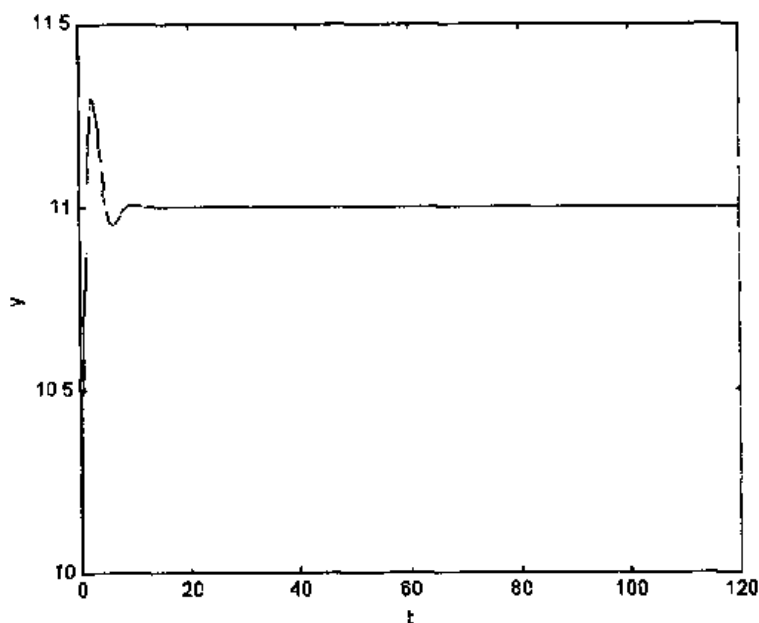


图 2.29 运算结果图

### 2.6.3 查看解题器的性能(performance)

前面介绍了解题器指令

```
[t,y]=solver('F',tspan,y0)
```

这个指令还可以增加一项输出参数,从而获得解题器性能的信息。增加的这个参数为s,其指令格式为:

```
[t,y,s]=solver('F',tspan,y0)
```

s为包含6个元素(s1-s6)的列向量,每个元素的意义说明如下:

- s1: 运算成功时的步骤数。
- s2: 运算失败时的步骤数。

- s3: 调用ODE文件执行F(t,y)的次数。
- s4: 执行偏微分的次数。
- s5: 执行LU分解的次数。
- s6: 线性系统解的个数。

上面的s4至s6只有在解艰难问题时才会用到。

**范例2.29 查看范例2.28的解题器性能。**

利用范例2.28的程序，增加输出参数s后的程序如下：

程序：ex263.m

```
clear
y0= [10;1;0] ;
[t,y,s]=ode45('dfun',[0 120],y0); ——— 增加s作为输出参数
plot(t,y(:,1))
xlabel('t')
ylabel('y')
s
```

执行结果：

```
s=
    62
     5
   403
     0
     0
     0
```

由上述的执行结果可知，解题器总共成功运算62次，失败5次，调用ODE文件403次。

#### 2.6.4 利用odeset指令改变解题器的设定参数

在解题器指令中还可以增加输入参数，以便通过调整这些参数来改进运算某些特殊问题时(比如解Jacobian问题)的解题性能。这些参数的设定是通过指令odeset来实现的，并设到解题器指令的参数options之中。指令格式如下：

```
[t,y]=solver('F',tspan,y0,options)
```

假如没有指定options，则系统会用默认值来解决一般问题。那么有哪些参数可供调整呢？下面来加以说明({}内为默认值)。

##### 1. 容许误差

所用的参数名称为：

RelTol{1e-3}、AbsTol{1e-6}

解题器在每一步的积分运算中, 都会把解出的值与预期值相减得到误差值 $e$ , 且必须满足。

$$e \leq \max(\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i))$$

时, 才算是一个成功的运算步骤。

## 2. 解题器的输出

所用的参数名称为:

OutputFcn、OutputSel、Refine{1}、Stats{off}。

这些参数可用来控制解题器的输出。

- OutputFcn为输出函数的文件名称。

可以自行设定或采用系统的输出函数:

```
odeplot
odephas3
odephas2
odeprint
```

(参见范例2.30的说明)。

- OutputSel是一个行向量, 表示所关心的OutputFcn的元素是什么 (参见范例2.30的说明)。
- Refine是一个整数值, 通过增加输出的点来控制输出的平滑度, 其值愈大则输出愈平滑。ode45的默认值为4 (参见范例2.31的说明)。
- stats用于控制计算结果的性能。当设定为on时, 如同2.6.3节介绍的那样会在s中表示出来 (参见范例2.32的说明)。

## 3. Jacobian矩阵

所用的参数名称为:

```
Jacobian{off}、Jconstant{off}、
Jpattern{off}、Vectorized{off}
```

这些参数用于解决偏微分  $\partial F / \partial y$  的艰难问题或者,

$$\begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_1}{\partial y_2} & \dots \\ \frac{\partial F_2}{\partial y_1} & \frac{\partial F_2}{\partial y_2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

的Jacobian问题, 尤其对于使用ode15s及ode23s的情况, 能够增加可靠性及运算效率。

下面说明参数的应用:

- Jacobian为on时, 解题器执行ODE文件所定义的Jacobian运算, 并返回结果。
- Jconstant为on时, 代表 $\partial F / \partial y$  常数。
- Jpattern为on时, 代表 $\partial F / \partial y$  矩阵。
- Vectorized为on时, 则ODE文件可编写为向量形式 $F(t, [y_1 y_2 \dots])$ 。

#### 4. 步阶大小

所用的参数名称为:

InitialStep、MaxStep

- InitialStep是正数, 设定解题器第一步运算的时间上限。
- MaxStep也是正数, 设定解题器每一步骤运算的时间上限。

通常不必调整这些参数, 因为解题器已经设计好了算法并确定了最佳的上限值。

#### 5. 群矩阵

所用的参数名称为:

Mass{Off}、MassConstant{off}

这些参数用于解决群矩阵(Mass matrix)问题, 问题的形式为 $M(t)y'=F(t,y)$

- Mass为on时, 解题器运算返回 $M(t)$ 值。
- MassConstant为on时,  $M(t)$ 为常数。

#### 6. 事件定位

所用的参数名称为Events{off}

- 当Events为on时, 如果解已经达到某个值或发生了某一情况时, 解题器将返回需要的数据值。例如模拟导弹击中目标的时刻, 即可设定这个参数, 以求得 $t$ 。

范例2.30 画出范例2.28 中 $y$ 的三列元素的三维输出图。

程序: ex264.m

```
%check the odeset function
clear
y0= [10;1;0];
options=odeset('OutputFcn','odephas3','OutputSel', [1 2 3]);
[t,y] =ode45('d1fun', [0 120] ,y0,options);
xlabel('y(1)')
ylabel('y(2)')
zlabel('y(3)')
```

执行结果:

调用 odephas3 来画出  $y(1), y(2), y(3)$ 。除了 odephas3 外, 还有 odephas2 odeplot 可以使用。请读者自行练习。



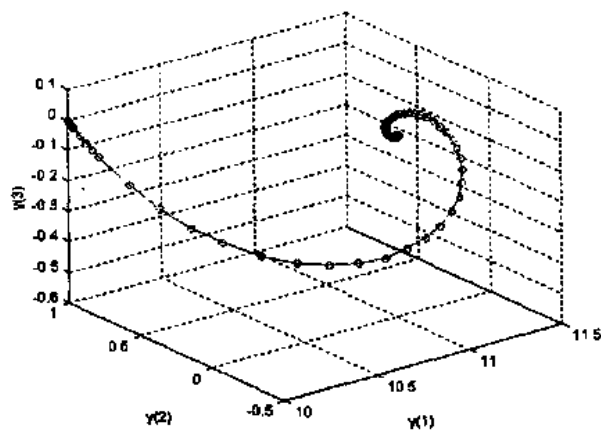


图 2.30

范例2.31 使范例2.30的三维输出图更平滑。

程序: ex265.m

```
%check the odeset function
clear
y0= [10;1;0] ;
options=odeset('OutputFcn
','odephas3','OutputSel', [1 2 3] , 'refine',20);
[t,y] =ode45('dlfun', [0 120] ,y0,options);
xlabel('y(1)')
ylabel('y(2)')
zlabel('y(3)')
```

Refine增加为20, 使得输出点更多, 输出图更为平滑。

执行结果(与图2.30相比较, 输出点更多了):

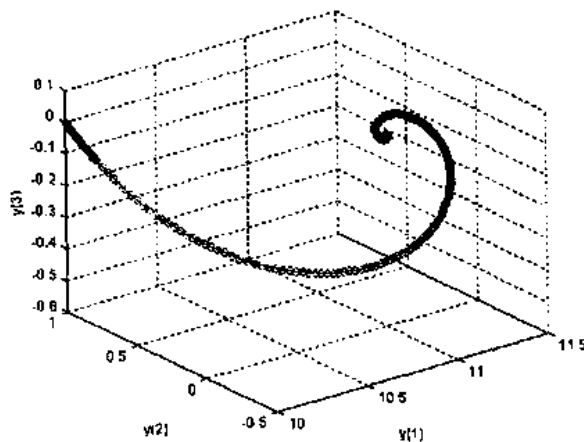


图 2.31

范例2.32 运用stats参数列出 $y''+y''+y'=0$ 的运算性能。

程序: ex266.m

```
%check the odeset(stats)function
clear
y0= [10;1;0] ;
options=odeset('stats','on');
[t,y]=ode45('dlfun',[0,120],y0,options);
```

将stats设定为on, 可显示出解  
题器的运算性能。

执行结果:

```
62 successful steps
5 failed attempts
403 function evaluations
0 partial derivatives
0 LU decompositions
0 solutions of linear systems
```

结果与范例2.6.3相同, 但所用的方  
法不同。

## 2.6.5 用常微分方程ODE解非线性问题

所谓非线性问题就是无法以叠加原理描述系统问题, 在现实生活中, 无论是机械、电子, 还是社会现象中都充满了非线性的问题, 只不过为了分析问题, 我们会假设很多前提, 使系统在某一微小的范围或平衡点上满足线性关系, 以便进行分析、设计工作。但是必须首先了解非线性的特点, 而研究其属性的方法, 就是非线性领域的研究范围, 通过找出系统的位置误差, 对其做时间的微分 $\dot{E}$ 在平面上的移动轨迹, 可以得到所谓的相平面, 再配合Lyapunov定理, 系统的稳定条件:

$$E \geq 0 \text{ 且 } \dot{E} < 0$$

就可以得出非线性系统稳定的范围。这个E就叫做Lyapunov函数, 有很多种方法可以求出它, 例如积分法、等值线法等等。详细的介绍请参见参考文献[6], [8]。ode45指令在解非线性方程时最常用, 下面的范例将说明如何应用该指令进行非线性问题的求解, 并将此解表现在椭圆形的球面上。

范例2.33 运用ode45解非线性问题。

对于非线性系统:

$$\begin{aligned}\dot{x}_1 &= -x_2(x_3-1) \\ \dot{x}_2 &= x_1(x_3-3) \\ \dot{x}_3 &= x_1x_2\end{aligned}$$

经过理论推导后, 得到一组E函数。

$$E = \frac{1}{2}(3x_1^2 + x_2^2 + 2x_3^2)$$

$$\dot{E} = 0$$

取初始条件为:

$$\begin{cases} x_1(0) = 1 \\ x_2(0) = \sqrt{6} \\ x_3(0) = 0 \end{cases}$$

此时,  $E=4.5$

根据非线性系统E函数理论, 由于  $\dot{E}=0$ , 所以系统轨迹会在  $E=4.5$  的橄榄球体上滑动。

ODE文件为: 程序d3fun.m

```
function bd = d3fun(t,b);
bd = zeros(3,1);
bd(1) = b(2) - b(2).*b(3);
bd(2) = b(1).*b(3) - 3.*b(1);
bd(3) = b(1).*b(2);
```

主程序为: ex267.m

```
clear;clf;
t0=0;
tf=200;
%b0= [1 2 1];
b0= [1 sqrt(6) 0];
x=-3:0.05:3;
y=-3:0.05:3;
[x1,x2] =meshgrid(x,y);
c=4.5.*2;
B=1;
b=3;
A=-1;
a=1;
%q=1/2*(B.*b.*x1.^2-A.*a.*x2.^2)-4.5;
p=(c-B.*b.*x1.^2+A.*a.*x2.^2)/(B.*A.*a-A.*B.*b);
q=sqrt(p);
figure(1);
%if p>=0
[t,b] =ode45('d3fun',[t0,tf],b0);
plot3(b(:,1),b(:,2),b(:,3),'*');hold on;
contour3(x1,x2,q,15);hold on;
contour3(x1,x2,-q,15);hold on;
xlabel('X1');
ylabel('X2');
zlabel('X3');
axis([-2 2 -4 4 -6 6]);
view(60,10);
```

设定初始条件

定义的点范围

求出橄榄球体的值

用ode45解属性轨迹

画出轨迹

画出橄榄球

```
grid;
title('THE 3-D CONTOUR FOR E2=0.5(3X1.^2+X2.^2+2X3.^2)');
```

执行结果:

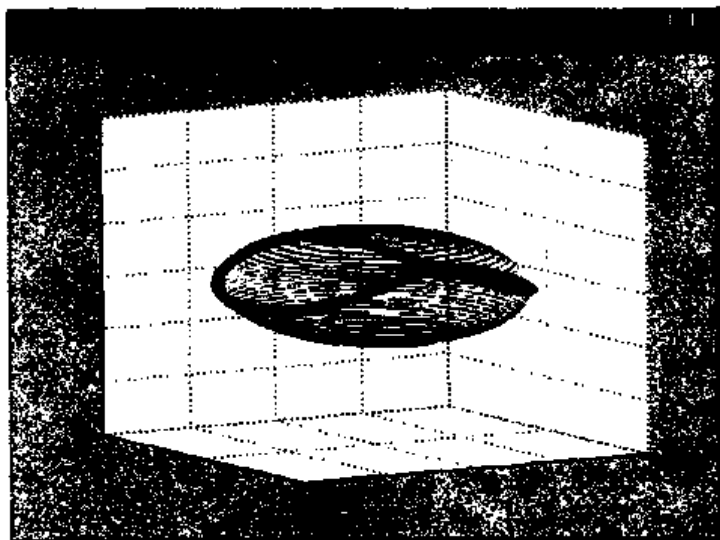


图 2.32 表现在橄榄球体上的非线性解的根

## 2.7 积分方程的设计技巧

### 2.7.1 积分方程的指令格式

当要求出 $F(x)$ 在某一区间的积分值时, 可以使用下面的指令:

```
quad('F',a,b,tol)
```

表示求 $x$ 在区间 $a$ 到 $b$ 上的积分值 $F(x)$ 。 $tol$ 为容许误差, 也可以省略不输入。可以用更高级的算法`quad8`来代替`quad`, 其格式完全相同。

### 2.7.2 设计步骤

与ODE的设计相同, 必须有一个子程序供解题器调用。这里的解题器包括`quad`和`quad8`两种。下面说明设计步骤。

**第一步: 编写积分方程的子程序。**

这个子程序以`function`开头, 其文件名可以由解题器来调用。

**第二步: 编写解题器的主程序去调用子程序, 并将结果画出来。**

下面用一个范例来加以说明。

**范例2.34** 积分方程  $y = \int_0^{10} \frac{1}{x^3 - 2x + 4}$  的积分程序。

先编写积分方程的子程序如下:

程序: d2fun.m

```
%d2fun.m:the equation to be integrated
function y=d2fun(x);
y=1/(x.^3-2.*x+4);
```

接着写主程序去调用子程序。

程序: ex271.m

```
%calculate integration
clear
x_a=0:0.1:10;
for I=1:length(x_a);
    x=x_a(I);
    int(I)=quad('d2fun',0,x);
    y(I)=d2fun(x);
end
ha=line(x_a,y,'color','b','linestyle',':');hold on
legend('y equation')
ap=gca
xlabel('x')
ylabel('y')
bp=axes('position',get(ap,'position'),'YAxisLocation','right',...
        'XAxisLocation','top','color','none')
hb=line(x_a,int,'color','red','parent',bp);hold on
ylabel('integration')
legend('integration result')
```

注意: 积分范围为变量x

本段  
为图  
形处  
理技  
巧,  
请参  
阅第  
6章

执行结果如图2.33所示。

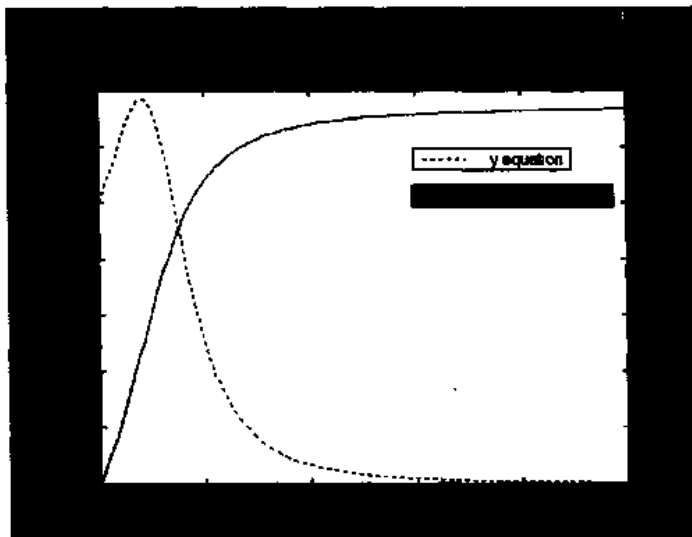


图 2.33 积分结果图

## 第3章 SIMULINK 设计

本章先介绍SIMULINK的基本概念，然后说明功能模块，线的设定、简单的设计、特殊要求的设计，以及如何与其他程序结合使用，包括最重要的s函数的设计技巧。

### 3.1 SIMULINK概述

SIMULINK是MATLAB环境下的模拟工具，其文件类型为.mdl，这与MATLAB 4不同，只要把原先SIMULINK设计好的.m文件改为.mdl即可在当前环境下运行。SIMULINK为用户提供了很方便的图形化功能模块，以便连接成一个模拟系统，简化设计的流程，减轻设计负担。更重要的是，SIMULINK能够用MATLAB自身的语言或C，FORTRAN语言，根据S函数的标准格式，写成用户自定义的功能模块。因此，其扩充性很强，同时也能够调用.dll文件类型的应用程序，实现与其集成应用的目的。所以，有些应用软件会提供.dll文件的s函数，以便能够通过DDE(Dynamic data exchanger)与SIMULINK传递数据。

在命令窗口中执行

`simulink`→Enter

或单击命令窗口中的“New Simulink Model”按钮(■)(见图3.1)，即可出现SIMULINK的功能模块函数库(见图3.2)与空白设计区域(见图3.3)。该空白设计区域用于SIMULINK的设计，也可以在模块函数库中通过File→New→Model来打开。

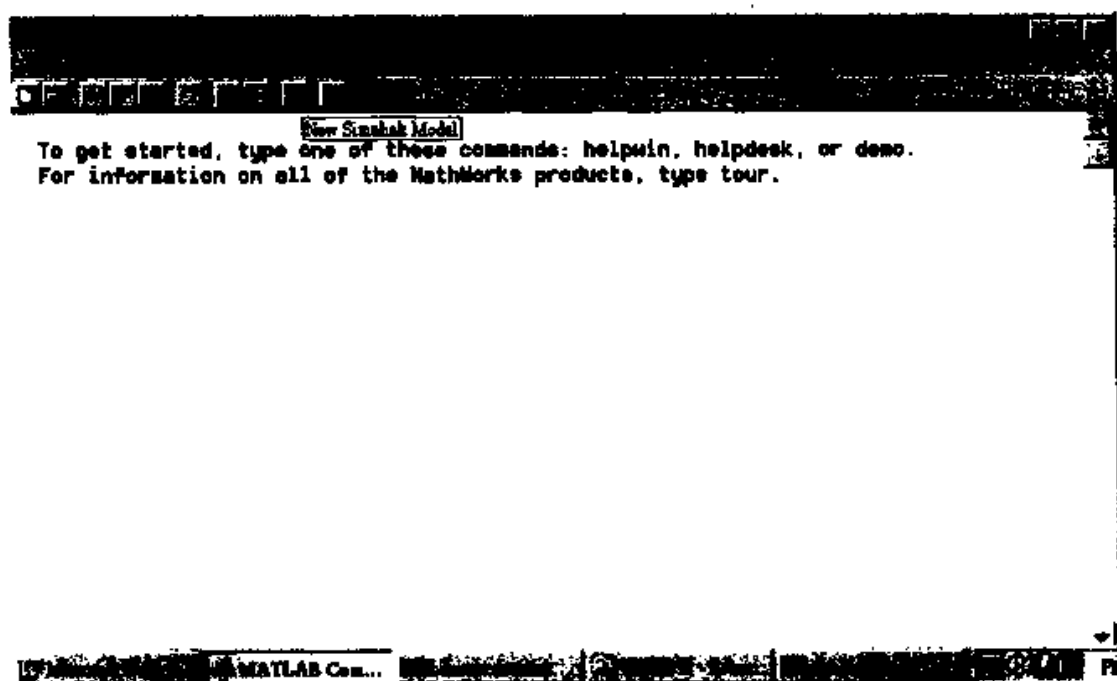


图 3.1 进入 SIMULINK 的按钮

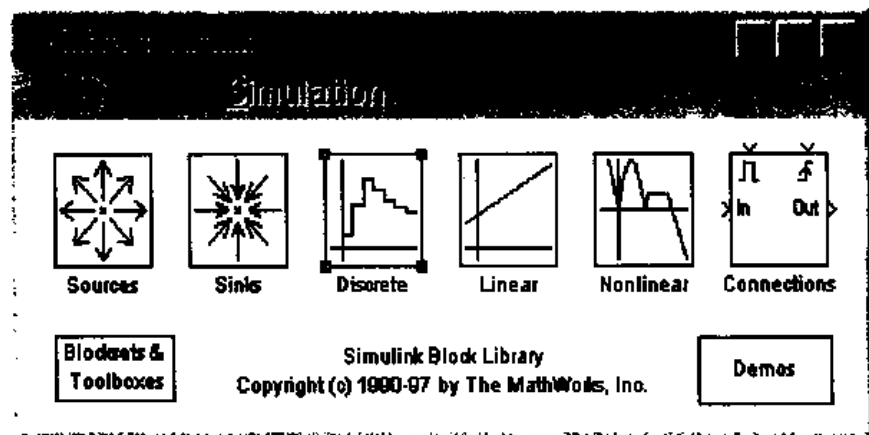


图 3.2 SIMULINK 功能模块函数库



图 3.3 空白设计区域

模块函数库中有很多子模块库，如sources、sinks等，每个子模块库中又有很多功能模块供用户设计时使用。将功能模块放入空白设计区域内组合完成后即可执行程序。

在SIMULINK中，可以设定很多参数来改变其执行的环境与条件，即可以单独执行，也可以由主程序用sim指令调用执行，执行时所需要的参数或变量，同样来自MATLAB的workspace或从“from workspace”模块获得，而执行后的结果则保存到“to workspace”模块中。主程序只要调用“to workspace”的名称，即可获得运算结果。除“from workspace”与“to workspace”之外，与“from workspace”性质相同的功能模块有很多，目的都是产生信号来源，所以这些都被放到“sources”模块库中；而与“to workspace”性质相同的功能模块也有很多，目的都是存放或显示处理结果，所以这些都被放到“sinks”模块库中。图3.4为“sources”模块库的内容(只有输出)，图3.5为“sinks”模块库的内容(只有输入)。

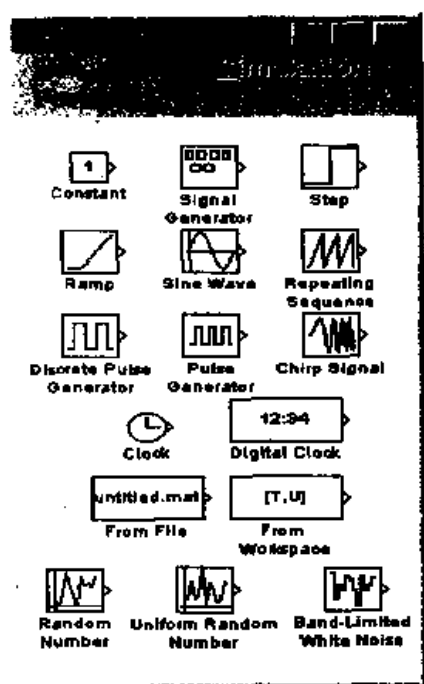


图 3.4 Sources 模块库

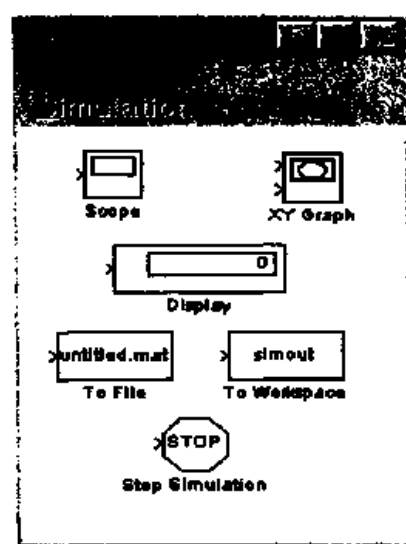


图 3.5 Sinks 模块库

对于处理单元，也就是既有输入也有输出的功能模块，根据其处理的属性分别放到图 3.2 中的“Nonlinear”，“Discrete”和“Linear”模块库中。而只做连接不做处理的功能模块则被放到“connection”模块库中。这些模块库所含的功能模块，其中“Discrete”如图 3.6 所示，“Linear”如图 3.7 所示，“Nonlinear”如图 3.8 所示，“Connections”如图 3.9 所示。

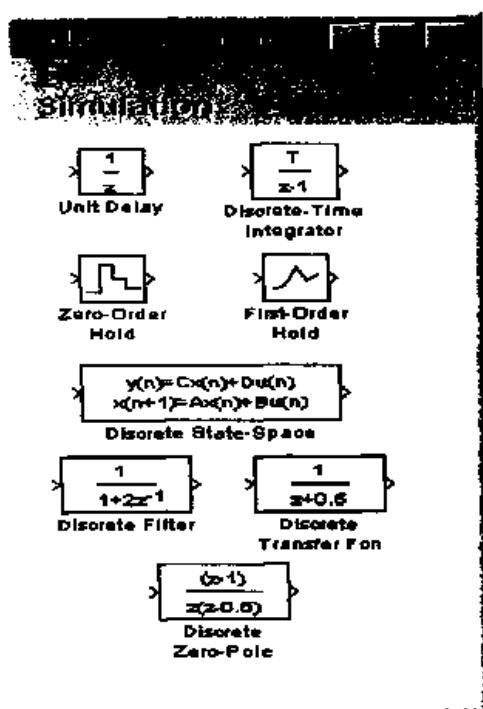


图 3.6 Discrete 模块库

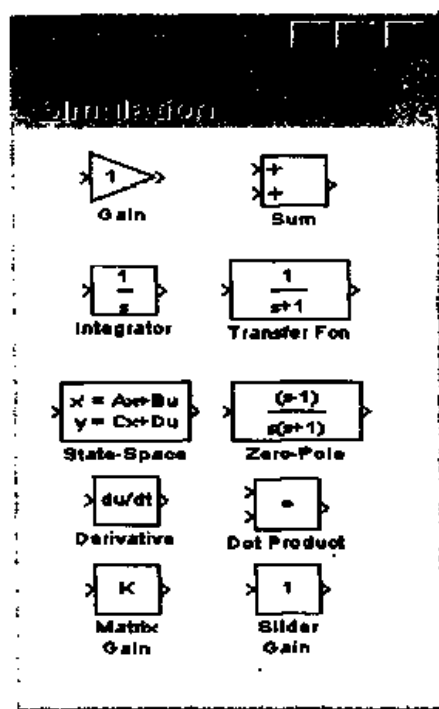


图 3.7 Linear 模块库



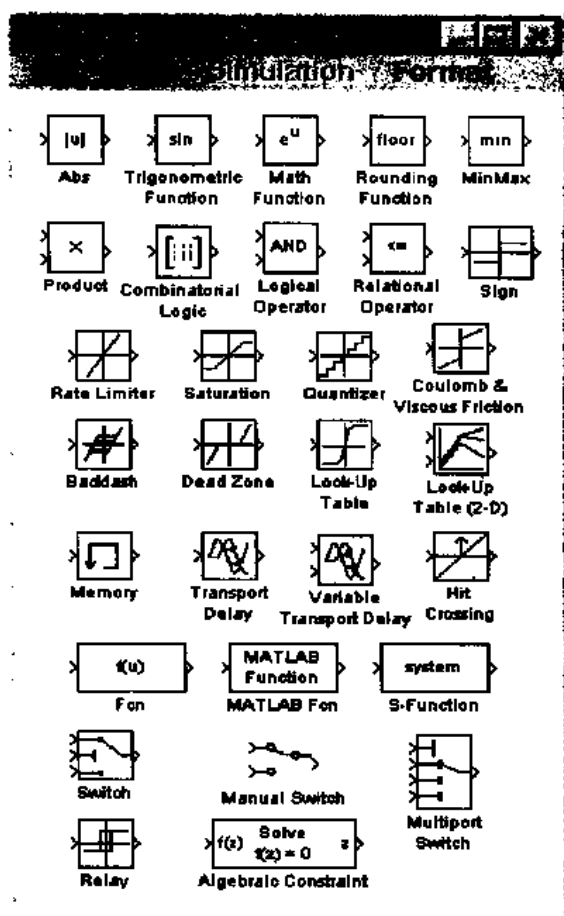


图 3.8 Nonlinear 模块库

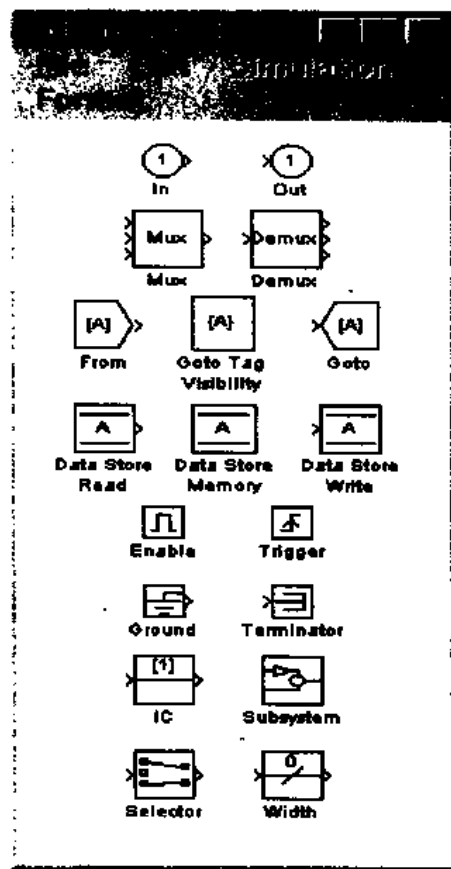


图 3.9 Connections 模块库

除了功能模块以外，另一个重要的部分就是连接功能模块的线(Line)，信号可以沿着线(Lines)前进，并进入功能模块中进行处理，可以设定线的粗细，设定方法在后续章节中进行说明。也可以在设计的模型中加入说明文字，以便让读者了解该设计模型的用途。在空白区域中，双击鼠标左键即可输入文字，并可改变文字的大小与字体。图3.15就是在空白区域内双击后出现的方框中直接输入文字完成的。用鼠标可以直接移动该文字模块。为了能够了解各功能模块的意义，下面说明各功能模块的用途(以下摘自SIMULINK User's Guide)。

表3.1 信号源(Sources)模块库

功能模块	用途
Band- Limited White Noise	产生正态分布的白噪音
Chirp Signal	产生频率与时间成正比的信号
Clock	时钟
Constant	常数
Digital Clock	在固定的取样间隔中产生模拟的时钟
From File	来自文件
From Workspace	来自工作空间

(续表)

功能模块	用 途
Pulse Generator	脉冲发生器
Ramp	斜坡信号
Random Number	随机数
Uniform Random Number	正态分布随机数
Repeating Sequence	重复信号
Signal Generator	信号发生器
Sine Wave	正弦波
Step	步阶波

表3.2 信号槽(Sinks)模块库

功能模块名称	用 途
Display	显示输入值
Scope	示波器
Stop Simulation	输入值非零时停止模拟
To File	写入文件
To Workspace	写入工作空间
XY Graph	显示二维图形

表3.3 离散(Discrete)模块库

功能模块名称	用 途
Discrete Filter	IIR与FIR滤波器
Discrete State-Space	离散状态空间系统
Discrete-Time Integrator	离散时间积分器
Discrete Transfer Fcn	离散传递函数
Discrete Zero-pole	以极零点表示的离散传递函数
First-Order Hold	一阶取样与维持
Unit Delay	一个取样周期的延迟
Zero-Order Hold	零阶取样与维持

表3.4 线性(Linear)模块库

功能模块名称	用 途
Derivative	输入信号微分
Dot Product	内积
Gain	输入信号增益
Integrator	输入信号积分
Matrix Gain	输入信号乘上矩阵
Slider Gain	用slider改变增益

(续表)

功能模块名称	用 途
State-Space	线性状态空间系统
Sum	将输入信号相加
Transfer Fcn	转换函数
Zero-Pole	以极零点表示的转换函数

表3.5 非线性(Nonlinear)模块库

功能模块名称	用 途
Abs	绝对值
Algebraic Constraint	求限制输入信号为零时的状态值
Backlash	由deadband(静带)设定,输出依据输入的改变而改变
Combinational Logic	组合逻辑
Coulomb&Viscous Friction	当输出值达到一定值时才会根据输入而改变
Dead Zone	当输入值达到一定值时输出才会根据输入而改变
Elementary Math	基本数学函数计算
Fcn	自定义函数运算
Hit Crossing	侦测跨越点
Logical Operator	对输入做逻辑判断
Look-Up Table	建立输入信号的查询表
Look-Up Table(2D)	建立两个输入信号的查询表
MATLAB Fcn	用MATLAB的现有函数做运算
Memory	存储上一时刻的状态值
MinMax	求极大与极小值
Multiport Switch	在多输入中选一个输出
Product	输入信号间相乘
Quantizer	把输入转化成阶梯状的量化输出
Rate Limiter	限制输入变化率的变化大小
Relational Operator	对输入信号做比较运算
Relay	限制输出值在某一范围内变化
Saturation	让输出超过某一值时能够饱和
S-Function	调用s函数的程序
Sign	输入的正负号
Switch	当第二个输入端大于临界值时,输出由第一个输入端而来,否则输出由第三个输入端而来
Transport Delay	输入信号延迟一个固定时间再输出
Variable Transport Delay	输入信号延迟一个可变时间再输出

表3.6 连接(Connections)模块库

功能模块名称	用 途
Data Store Memory	定义一个可读写的数据库并赋初始值
Data Store Read	定义一个可读的数据库并赋初始值

(续表)

功能模块名称	用途
Data Store Write	定义一个可写的空间并赋初始值
Demux	将一个向量值拆成多个单一输出
Enable	激活分系统的功能模块
From	从Goto功能模块中接收信号
Goto	连接到From的功能模块
Goto Tag VisibilityGoto	功能模块的标签
Ground	连接到没有连接到的输入端
IC	为信号赋初始值
Import	输入端
Mux	将多个单一输入转化成u向量
Output	输出端
Selector	对多个输入信号执行选择性的输出
Subsystem	建立新的封装 (Mask)功能模块
Terminator	连接未连信号的输出端
Trigger	触发功能模块
Width	产生与输入信号相同宽度的输出

对于上述各表功能模块的详细说明,读者除了可以参阅SIMULINK的使用手册之外,还可以使用Help Desk直接查阅,能够立即找到所需要的解答(参见第1章的说明)。

设计好系统之后,在Simulation中执行start, SIMULINK就会开始执行所设计的程序。

为了说明方便起见,先回到图3.3中去设计程序。在2.3.4节中采用  $\frac{40}{s(s+2)}$  为转换函数

的设计例子,如果用SIMULINK来设计的话会怎么样呢?下面举例进行说明。

**范例3.1** 用SIMULINK模拟  $\frac{Y(S)}{X(S)} = \frac{G(S)}{1+G(S)}$ , 其中  $G(S) = \frac{40}{S(S+2)}$ 。

程序: ex311.mdl(可在图3.2中通过菜单File→Open→ex311.mdl来打开)。

在Sources模块库中找到step功能模块,放到图3.3的空白设计区域中,其他也类似,把功能模块组合成如图3.10的系统。再执行Simulation菜单中的start(见图3.11),在Scope上单击鼠标即可看到执行的结果,如图3.12所示。

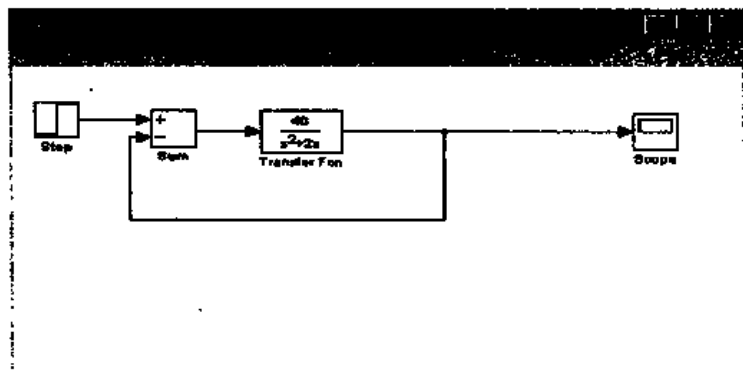


图 3.10 功能模块组合图

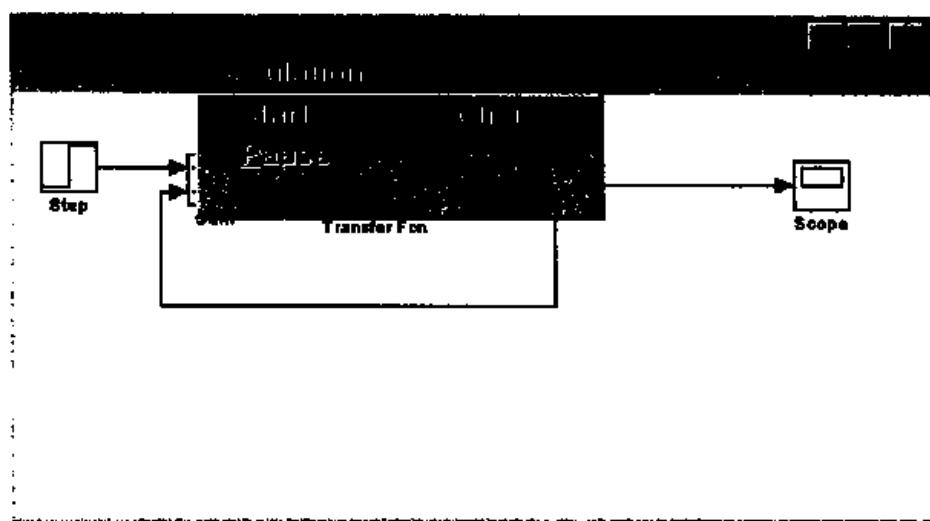


图 3.11 单击 Start 执行模拟

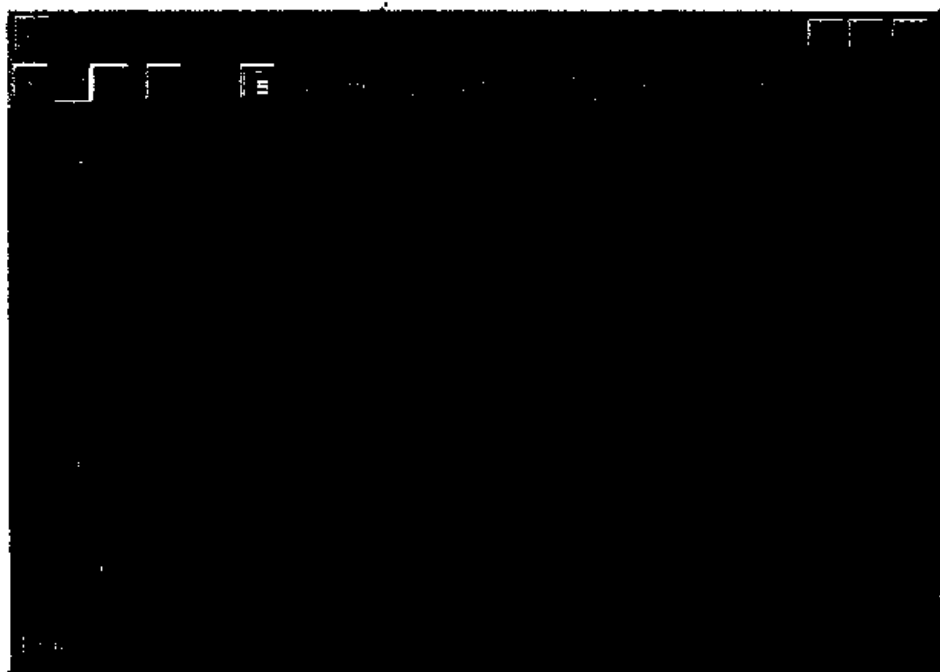


图 3.12 模拟结果

图3.12中的瞬态响应并不理想，如果按照2.3.4节采用  $G_c = \frac{1+0.5S}{1+0.05S}$  来作为校正器，用SIMULINK设计是很简单的，举例如下。

**范例3.2** 在范例3.1的基础上加入校正器后重新模拟系统响应。

程序：ex312.mdl

将图3.10略做补充，加入  $G_c = \frac{1+0.5S}{1+0.02S}$  校正器。设计结果如图3.13所示，模拟结果如图3.14所示，与2.3.4节相比较可发现结果是相同的。

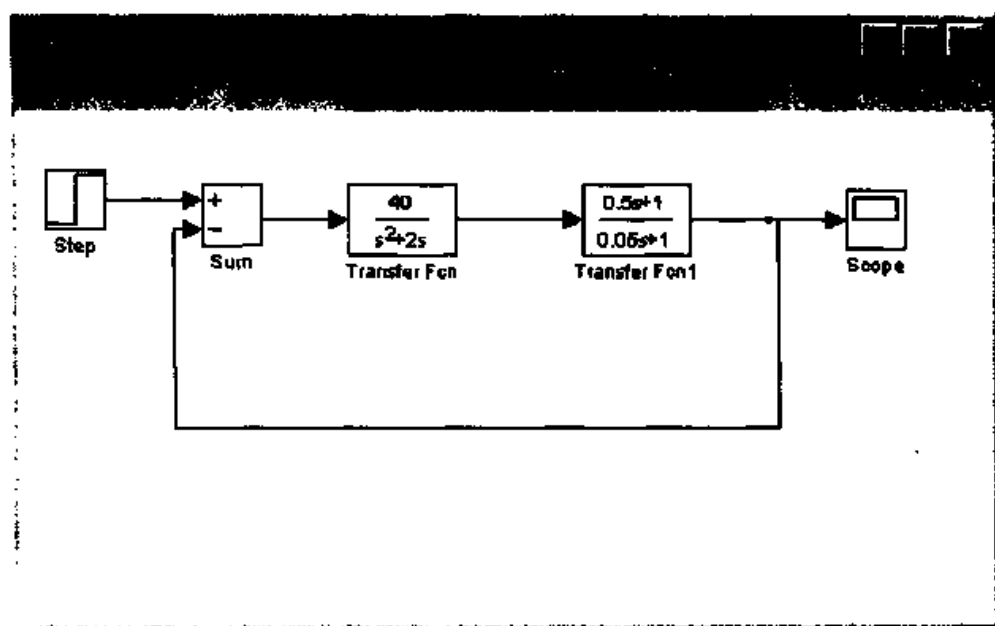


图 3.13 加入校正设计的功能模块图

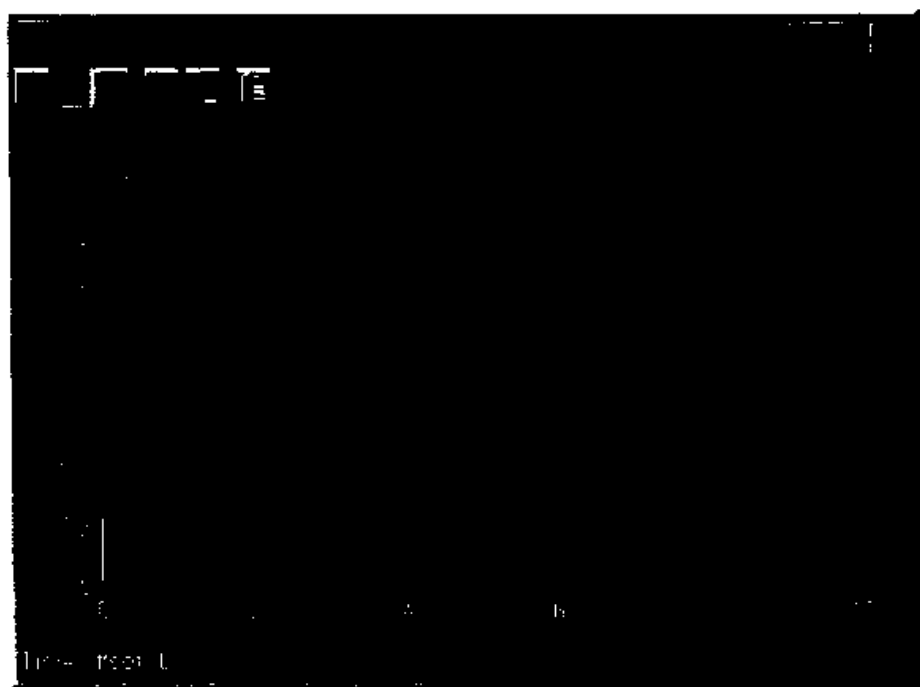


图 3.14 执行结果

范例3.3 在范例3.2的基础上, 重新设计校正器, 改善overshoot.

程序: ex313.mdl

由图3.14可以看出存在overshoot. 改进的方法就是将校正器改成 $G_c=1+0.5S+0.02S$ , 如图3.15所示. 模拟结果如图3.16所示, 可以看出效果改进了很多.

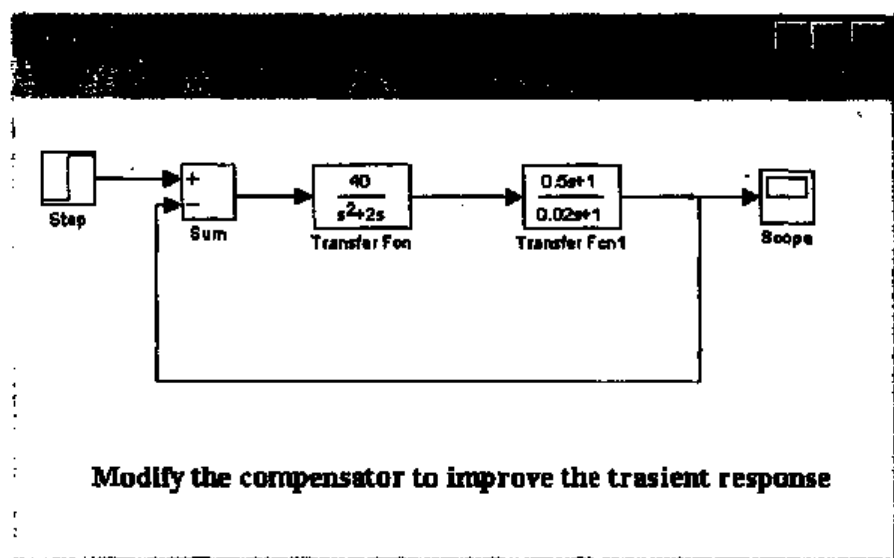


图 3.15 修改校正器的功能模块图

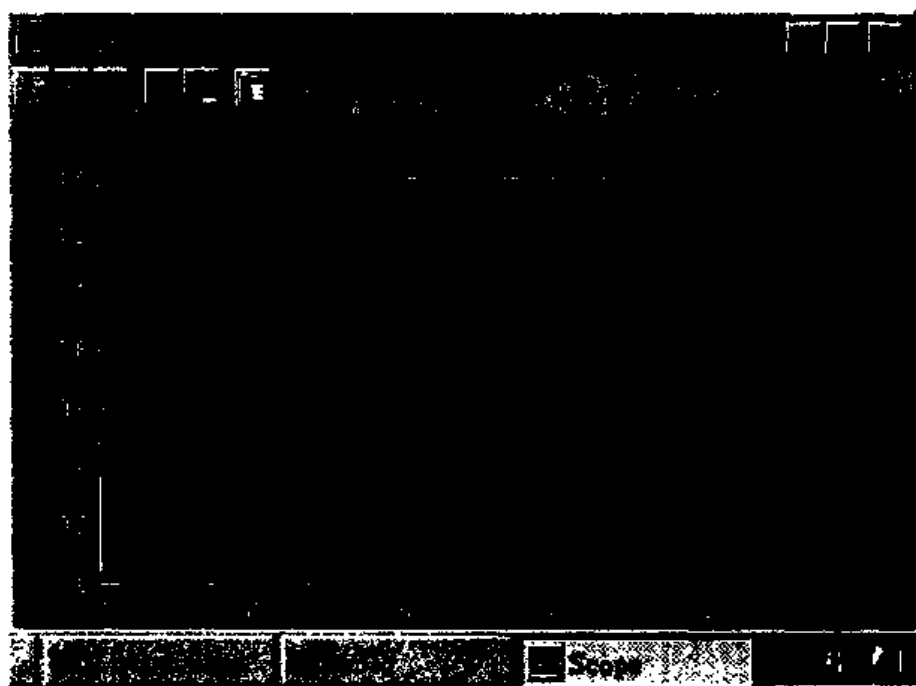


图 3.16 模拟结果图

### 3.2 功能模块的处理

每个模块库中的功能模块，都可以直接用鼠标拖曳到设计区域上，再用线(Lines)将其连接后执行，这在上一节已经介绍过了。本节将对功能模块的基本操作加以说明，也就是模块的移动、复制、删除、转向、改变大小、模块命名、颜色设定、参数设定等等。至于每个功能方块的功能设定，则会因不同的执行功能而有所不同，这部分将在后续章节中加以介绍。

用鼠标选中的功能模块的4个角会出现黑色标记，如图3.17所示。

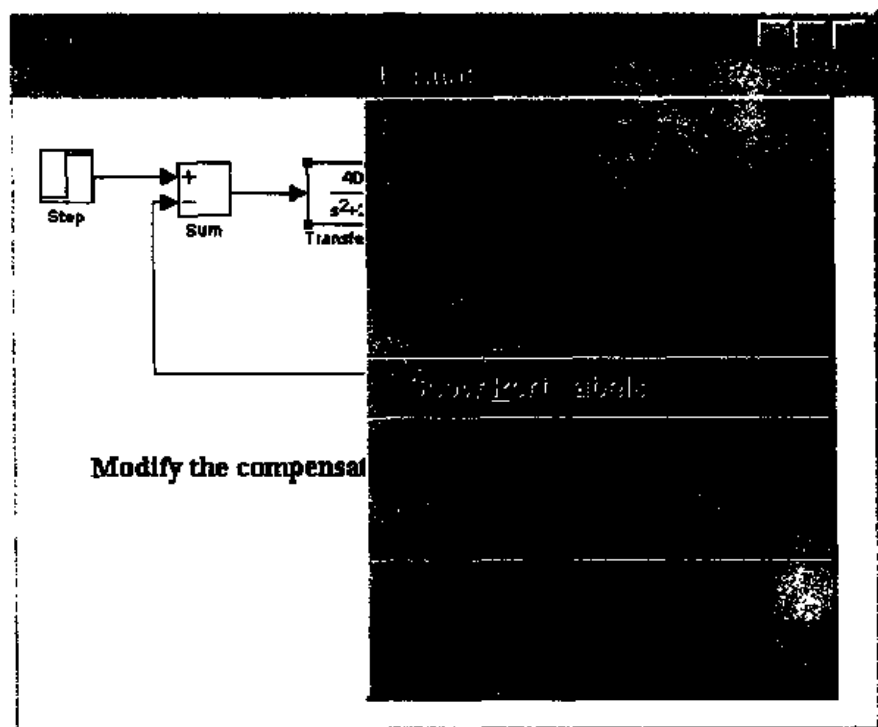


图 3.17 功能模块设定菜单

此时就可以很容易用鼠标进行下面的基本操作。

- **移动：**只要用鼠标将其拖曳到所需位置即可。
- **复制：**先把鼠标指针指到要复制的功能模块上，然后单击鼠标右键即可复制同样的一个功能模块。如果要取出模块库中的功能模块，可以直接从模块库中拖曳出来，不必单击右键。
- **删除：**按Del键即可。若要删除多个功能模块，可以同时按住Shift键，再用鼠标选取多个模块，按Del键即可。也可以直接用鼠标选取某区域，再按Del键就可以把该区域内的所有模块、线等全部删除。
- **转向：**为了能够顺序连接功能模块的输入与输出端，功能模块有时需要转向。在菜单Format中选择Flip Block旋转180°，选择Rotate Block顺时针旋转90°，或者直接按Ctrl+F键执行Flip Block，按Ctrl+R键执行Rotate Block。
- **改变大小：**用鼠标选取功能模块出现的4个黑色控制点，改变功能模块大小。
- **模块命名：**先用鼠标在需要更改的名称上单击一下，然后直接更改即可。名称在功能模块上的位置也可以变换180°，这可以用Format→Flip Name实现。若要隐藏模块名称则用Format→Hide Name。
- **颜色设定：**可以更改功能模块中的前景(Foreground)与背景(Background)颜色，用Format→Foreground Color改变前景颜色，用Format→Background Color改变背景颜色。另外，空白设计区域的颜色可以用Format→Screen Color来设定。
- **参数设定：**用set\_param指令可设定回调程序(Callback Routines)的参数(Parameters)，



以便执行特定的MATLAB程序。其指令格式为：

```
set_param(block名称, '参数', '参数值1', '参数2', '参数值2'...)
```

与之相对应，如果要获得参数值，只要用get\_param取代set\_param即可。范例如下。

#### 范例3.4 设计一个画图按钮。

本范例以图3.15为例说明如何用set\_param指令设定回调程序的参数openfcn，以便能够设计一个功能模块，从而画出模拟的结果。

程序：ex321.mdl

第一步：首先建好simulink模型，如图3.18所示，增加一个名为out的“to workspace”功能模块，目的是存放运算结果，以便供后面画图时使用。接着就是编写要画图简单程序sparam.m如下：

```
figure(1)
plot(t,out)
```

第二步：在“Connections”模块库中将“Subsystem”功能模块复制到设计区域中，如图3.18所示。

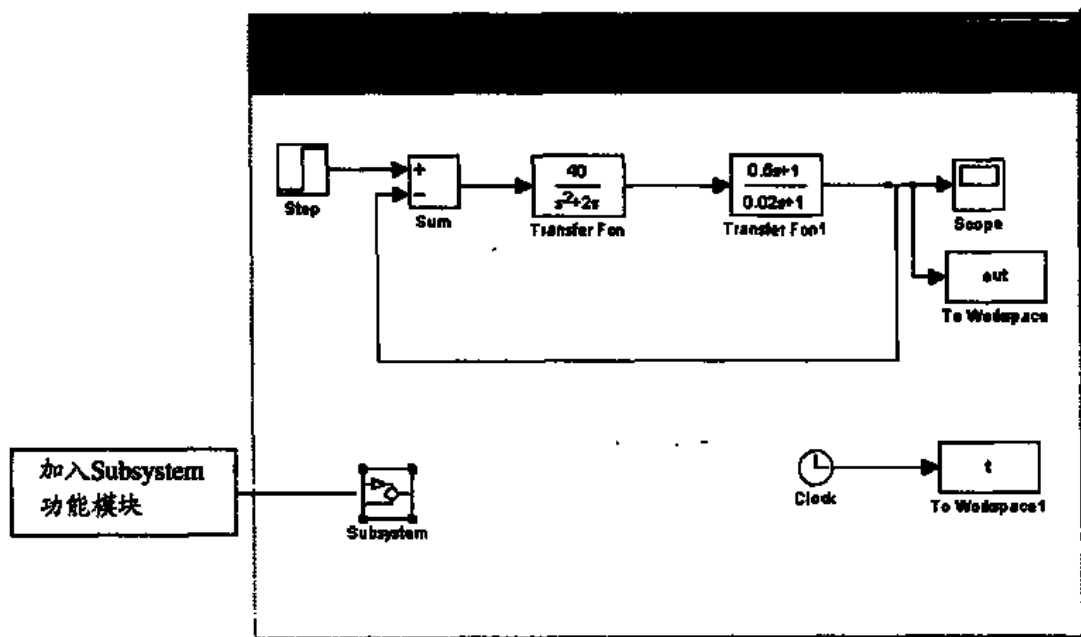


图 3.18 将 Subsystem 功能模块复制进来

第三步：在命令窗口下执行

```
set_param('ex321/Subsystem', 'openfcn', 'spara')
```

注意Subsystem中的S要大写。

第四步：功能模块的命名及外观设定。

选取Subsystem功能模块后, 选择Edit→Mask Subsystem出现如图3.19所示的画面, 输入disp('Double Click here to Plot Result'), 则该功能模块将会出现上述文字内容。另外在Format下选择Hide Name, 并设定好Background颜色, 结果如图3.20所示。当我们执行Simulation→Start后, 再用鼠标双击此功能模块就会画出图形, 如图3.21所示。

还有很多其他回调程序的参数, 在进行不同的应用时可做不同的设定。读者如果要深入探讨, 可参考相关资料。

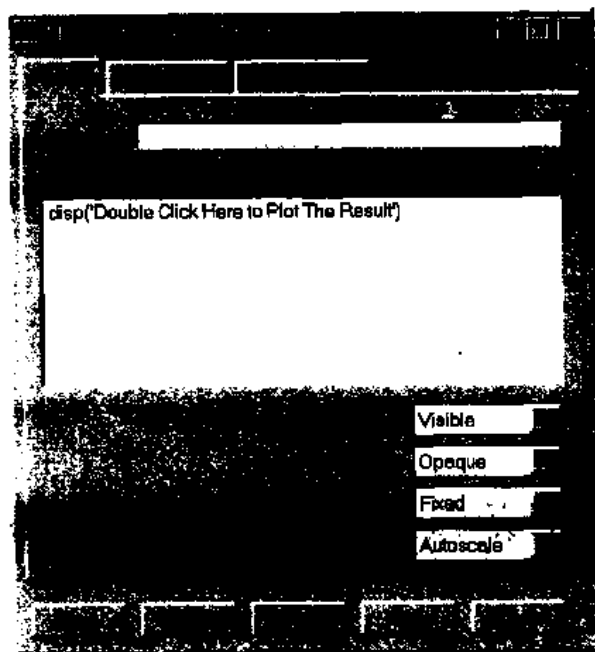


图 3.19 设定模块外观文字

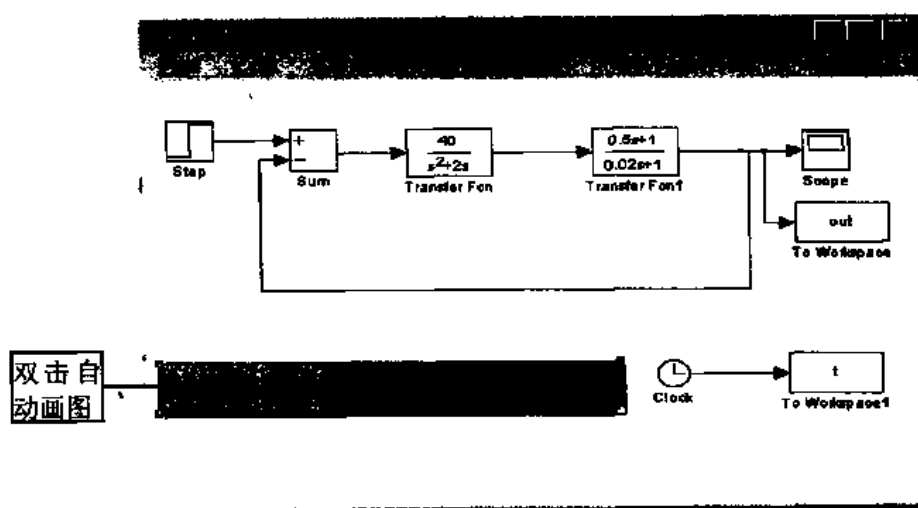


图 3.20 设计结果

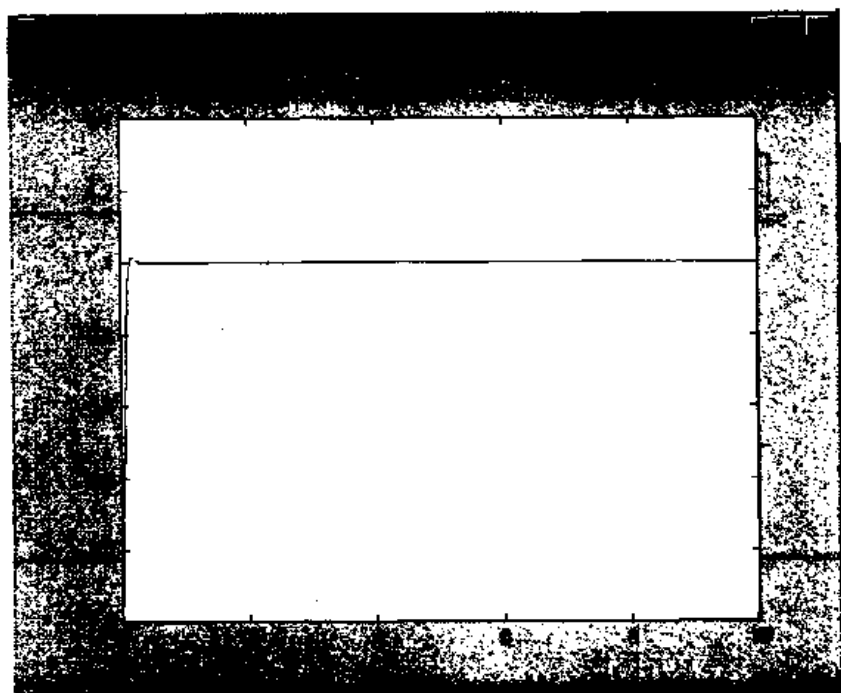


图 3.21 执行结果

### 3.3 线的处理

线具有连接功能模块的功能。用鼠标可以在功能模块的输入与输出端之间直接连线。所画的线可以改变粗细、设定标签,也可以把线折弯、分支。下面分别说明其用法:

1. **改变粗细:** 当选中Format→Wide Vector Lines(见图3.17)时,线的粗细会根据在线上传输的数据是数值(Scale)还是向量(Vector)而改变——如果为数值则是细线,如果是向量则是粗线。举例来说,在设计区域内建立一个含向量的模型并选中WideVector Lines,在执行完Simulation→Start或Edit→Update Diagram之后,传输向量的线就会变粗,如图3.22所示,程序为ex331.mdl。
2. **设定标签:** 只要在线上双击鼠标,即可输入该线的说明标签。如图3.22所示,在粗线上双击后,输入sin vectors即可出现该线的标签。
3. **线的折弯:** 按住Shift键,再用鼠标在要折弯的地方单击一下,就会出现圆圈,表示折点,利用折点就可以改变线的形状。以图3.22为例,对粗线设定三个折点,变形如图3.23所示。
4. **线的分支:** 按住Ctrl键,并在要建立分支的地方用鼠标拉出即可。另一种方法是由输入端拉线到分支点。图3.22中的sin wave分出了三支,它们就是用这种方法画出的。

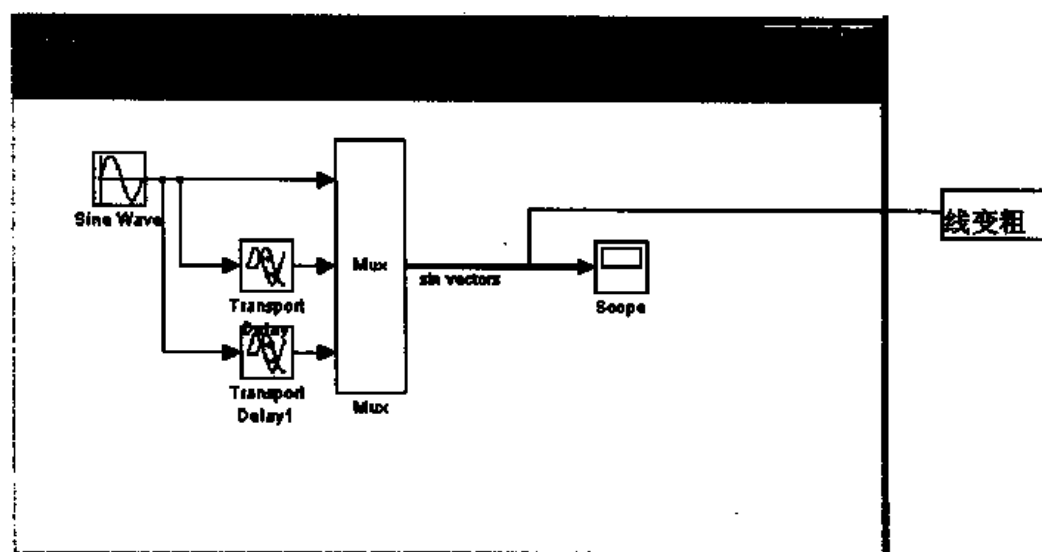


图 3.22 向量线变粗

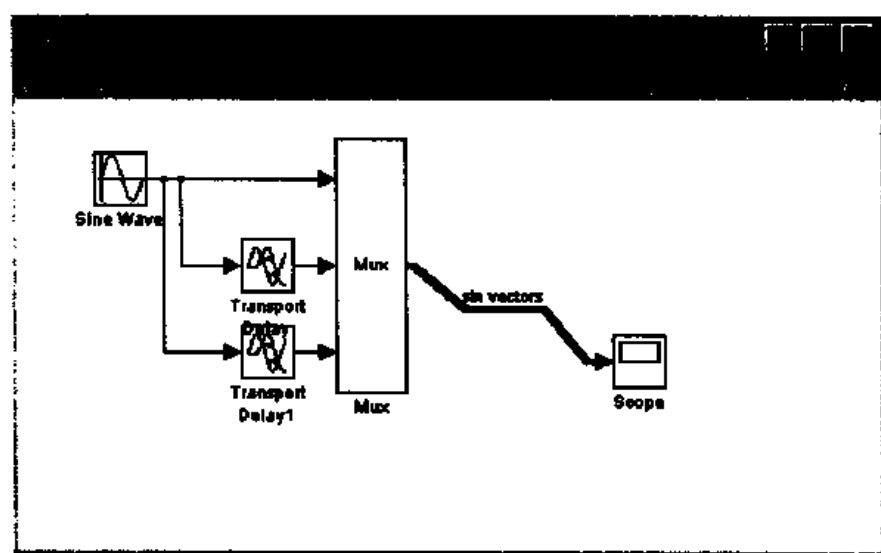


图 3.23 线的转折

### 3.4 自定义功能模块

自定义功能模块有两种方法，一种方法是采用Connections模块库中的Subsystem功能模块，利用其编辑区设计组合新的功能模块；另一种方法是将现有的多个功能模块组合(Group)起来，形成新的功能模块。对于很大的SIMULINK模型，这样的自定义模块可以简化图形，减少功能模块的个数，所以这样很有必要。

对于第一种方法，说明如下(其程序为ex341.mdl)。

先到“Connections”模块库中把功能模块“Subsystem”复制到设计区域内，然后双击，就将会出现Subsystem的设计区域。在此设计区域中进行设计，并在输入与输出端加入“in”和“out”两个功能模块(见图3.24)，之后返回SIMULINK的空白设计区域，将其

组合成一个功能模块，如图3.25所示。模拟结果（见图3.26）可以把前一时刻的输出和当前时刻的输入相加，结果如图3.27所示。

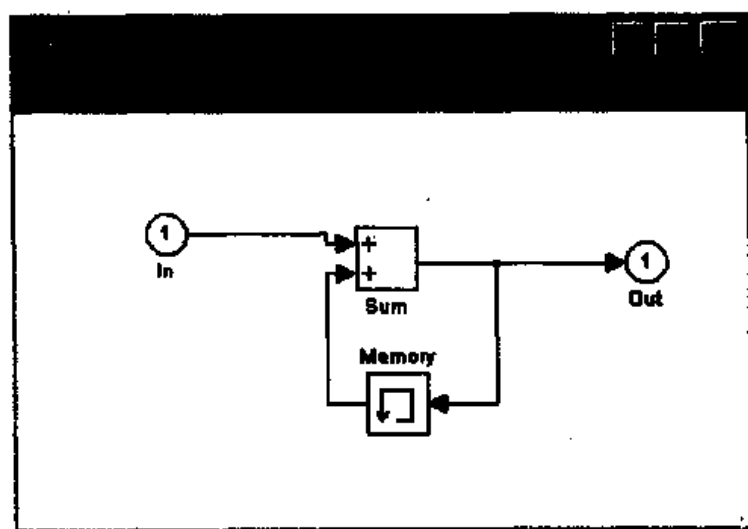


图 3.24 组合前的设计模块

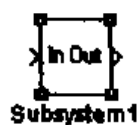


图 3.25 组合后的功能模块

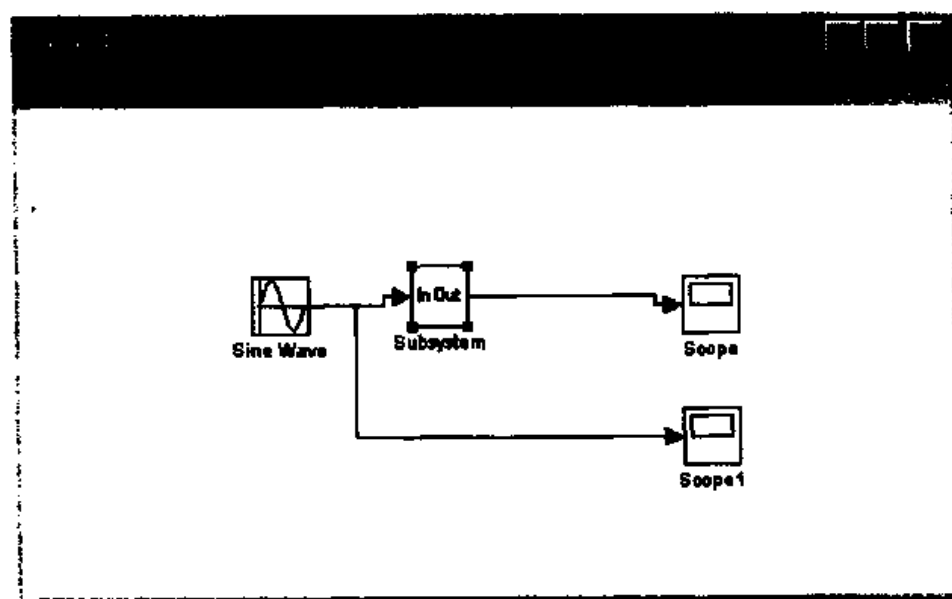


图 3.26 功能模块设计

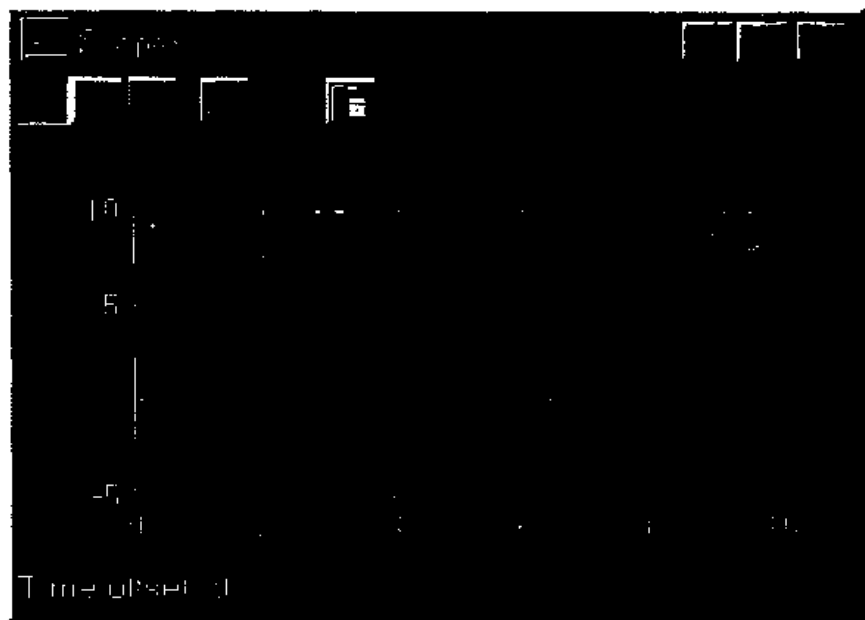


图 3.27 模拟结果

第二种方法更简单，只要用鼠标将要组合的功能模块框住，再选择Edit→Create Subsystem即可。以图3.22中的ex331.mdl为例，将其组合后的结果如图3.28所示。执行后可以发现产生了一个向量输出，如图3.29所示。

程序：ex342.mdl

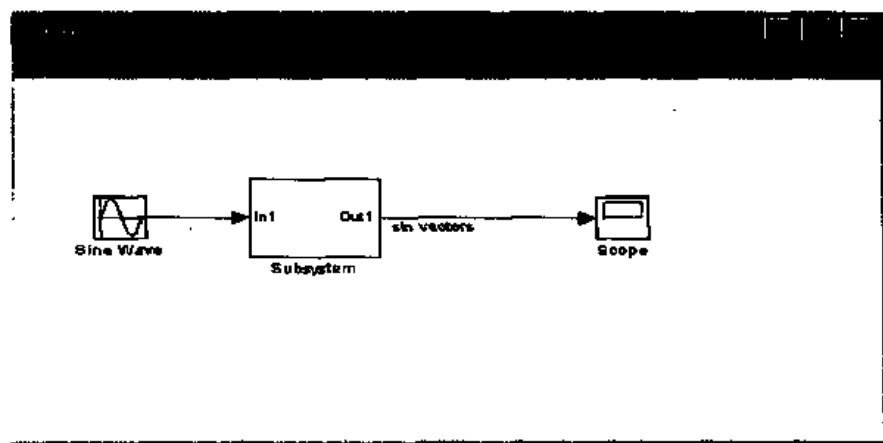


图 3.28 第二种组合功能模块的方法

上面提到的两种方法都只是创建一个功能模块而已，如同3.2节范例3.4提到的一样我们还需要命名功能模块并选定其外观，这方面的技巧称为图罩(mask)。任何一个subsystem功能模块都可以进行图罩。以图3.28的ex342.mdl为例，先选取Subsystem功能模块，再选择Edit→Mask Subsystem进入mask的编辑画面，如图3.30所示，可以看到有Icon, Initialization, Documentation三个标签页。其主要功能分别为：

- Icon：设定功能模块的外观。

- Initialization: 设定输入数据窗口(Prompt List)。
- Documentation: 设计该功能模块的文字说明。

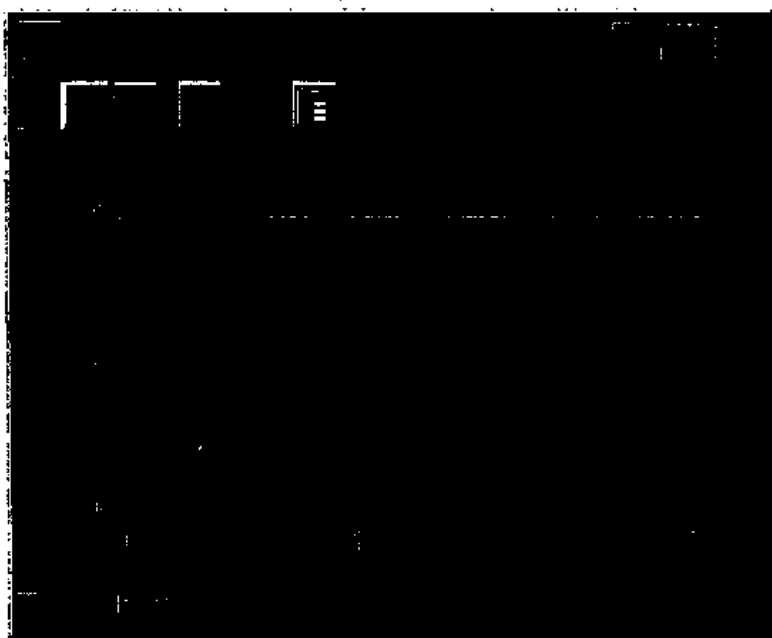


图 3.29 模拟结果

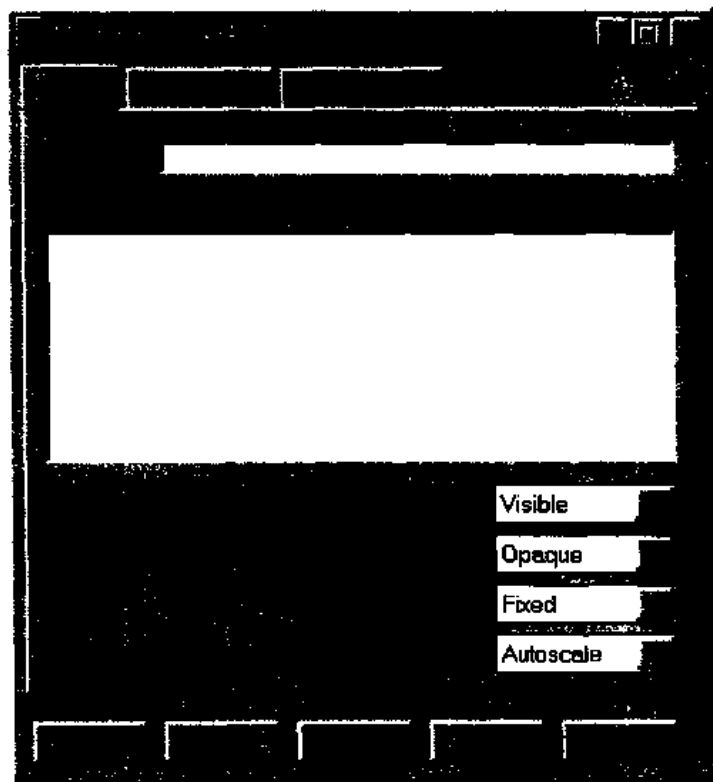


图 3.30 Mask 窗口

### 3.4.1 Icon标签页

此页最重要的部分是Drawing Commands，在该区域内可以用disp指令设定功能模块的文字名称，用plot指令画线，用dpoly指令画转换函数。Drawing Commands区域下方的4个下拉式列表框用于设定功能模块的属性。

#### 1. disp指令的运用

可以在功能模块上显示出设定的文字内容，如果输入图3.31所示的指令，则Subsystem功能模块如图3.32所示。

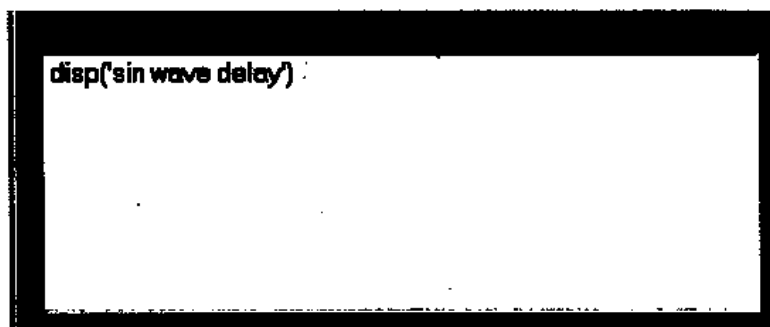


图 3.31



图 3.32

#### 2. plot指令的运用

`plot([x1 x2...xn], [y1 y2...yn])`指令会在功能模块上画出由  $[x1\ y1]$  经  $[x2\ y2]$  经  $[x3\ y3]$  ...直到  $[xn\ yn]$  为止的直线。功能模块的左下角会根据目前的坐标刻度被正规化为  $[0\ 0]$ ，右上角则会依据目前的坐标刻度被正规化为  $[1\ 1]$ 。如果执行图3.33所示的指令，则Subsystem功能模块变为如图3.34所示的对象。

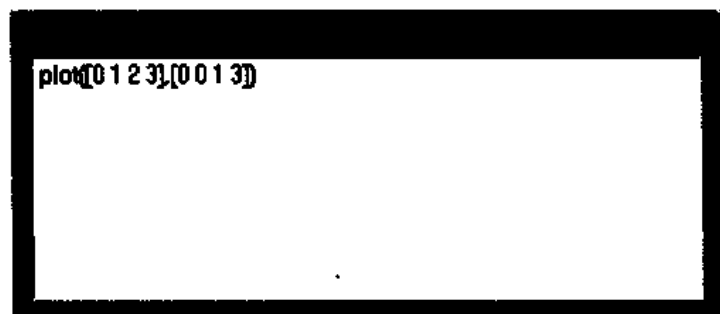


图 3.33



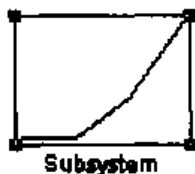


图 3.34

### 3. dpoly指令的运用

执行dpoly([分子],[分母])指令后,功能模块会显示s变量的转换函数式。如果执行图3.35所示的指令,则Subsystem功能模块变为如图3.36所示的对象。

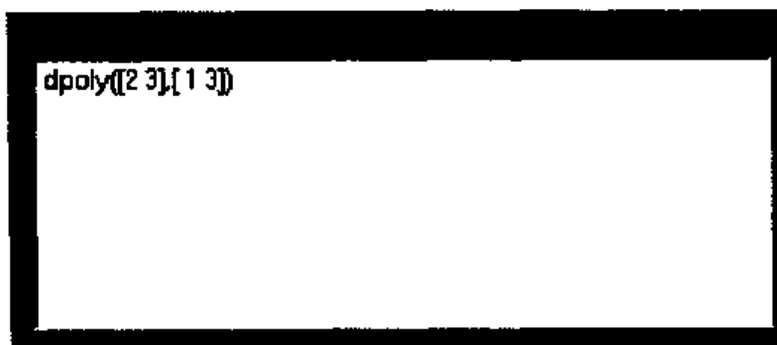


图 3.35

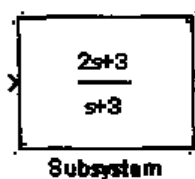


图 3.36

另外这个指令可以有下面几种不同形式:

- 显示z转换只要在指令后加上'z'即可,如果执行图3.37所示的指令,则Subsystem功能模块变为如图3.38所示的对象。

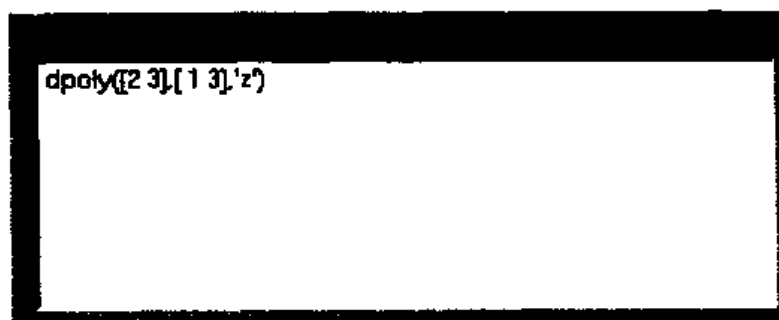


图 3.37

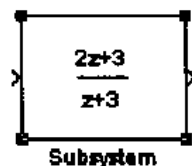


图 3.38

- 显示z倒数转换只要在指令后加上'z-'即可，如图3.39所示，则Subsystem功能模块变为如图3.40所示的对象。

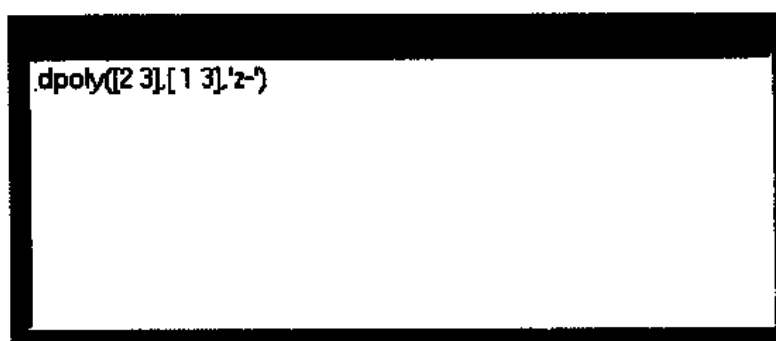


图 3.39

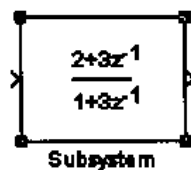
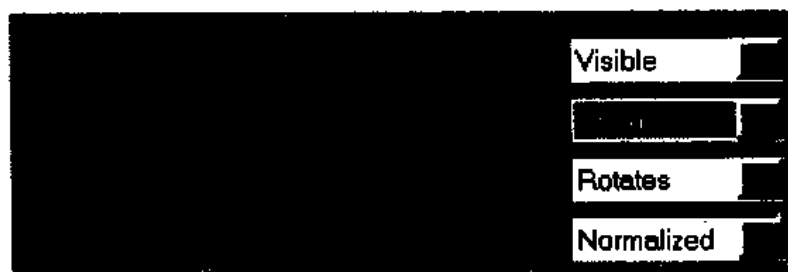


图 3.40

#### 4. 功能模块属性的运用：

在图3.41的4个下拉式列表框中可以对自定义的功能模块外观进行设计。



3.41

- Icon frame: Visible 显示外框线。Invisible: 隐藏外框线。

- Icon Transparency: Opaque 隐藏port的标签。 Transparent: 显示port的标签。
- Icon Rotation: 旋转模块, 但实际上在SIMULINK设计区中执行Ctrl+R更方便一些。
- Drawing coordinate: 画图时的坐标系。例如选择Autoscale, 如图3.42所示。



图 3.42

在 Drawing coordinates 下拉式列表框中选择 Autoscale 后的结果如图 3.43 所示。

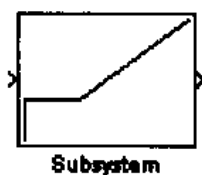


图 3.43

如果选择 Pixel, 如图 3.44 所示, 则反而看不到线了, 如图 3.45 所示。



图 3.44

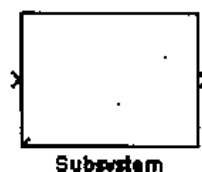


图 3.45

如果选择 Normalized, 如图 3.46 所示, 则坐标变得不精确了, 如图 3.47 所示。



图 3.46

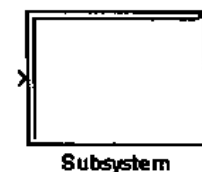


图 3.47

所以正常情形下建议使用Autoscale。

### 3.4.2 Initialization 标签页

Initialization的编辑画面如图3.48所示，主要用于设计输入提示(Prompt)以及对应的变量名称(Variable)。在prompt栏上输入变量的含义，其内容会显示在输入提示中。而variable是程序控制要用到的变量，该变量的值一直存于mask workspace中，因此可以与其他程序相互传递。如果配合在initialization commands内编辑程序，可以发挥功能模块的功能来执行特定的操作，在范例3.6与范例3.7中有针对这一方面的说明。

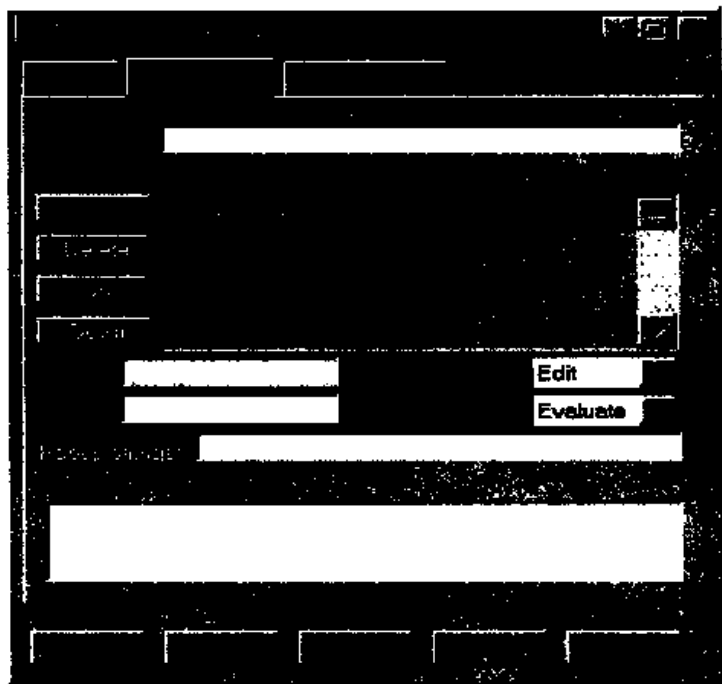


图 3.48

#### • Prompt列表

如果在Prompt编辑框中输入文字，这些文字就会出现在Prompt列表中。比如在Prompt列表中输入test，在variable列表中输入t，再单击Apply按钮后，则显示出的画面如图3.49所示。

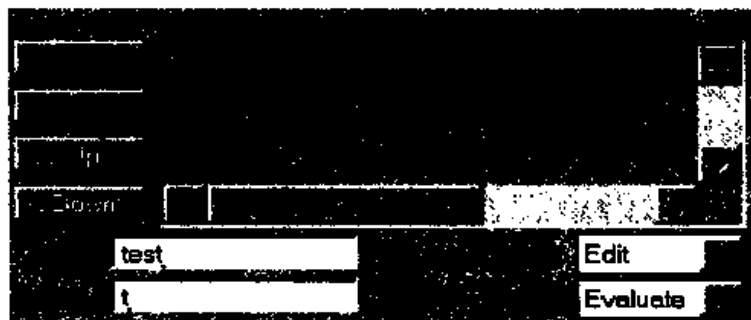


图 3.49

如果要增加项目的话，可以单击Add→Apply，会出现与刚才一样的输入框以供用户输入新的变量。如果输入velocity，其变量为v，则出现如图3.50所示的画面。

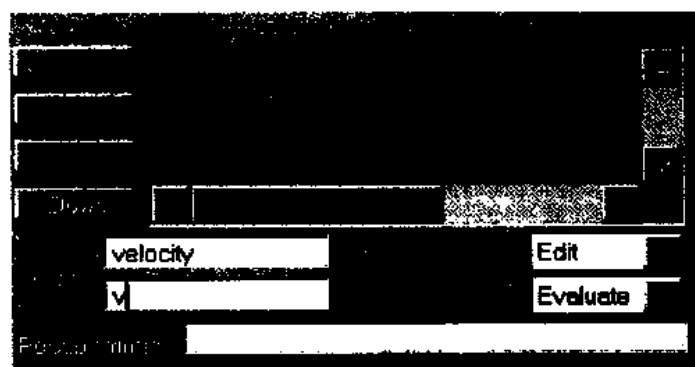


图 3.50

Up与Down按钮则用于执行项目间的位置调整。例如单击Up后，图3.50中的velocity位置会调到test之上，如图3.51所示。

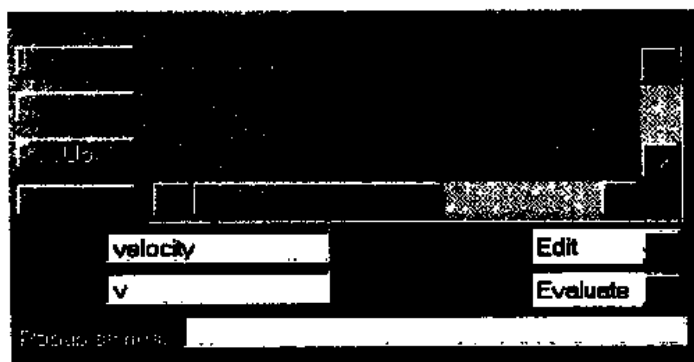


图 3.51

如果还不清楚请参考范例3.5。

#### • Control type与Assignment

Control type给用户选择设计的编辑区，有Edit, Popup和Checkbox三种。选择Edit会出现供输入的空白区域，所输入的值代表variable，如图3.52所示。

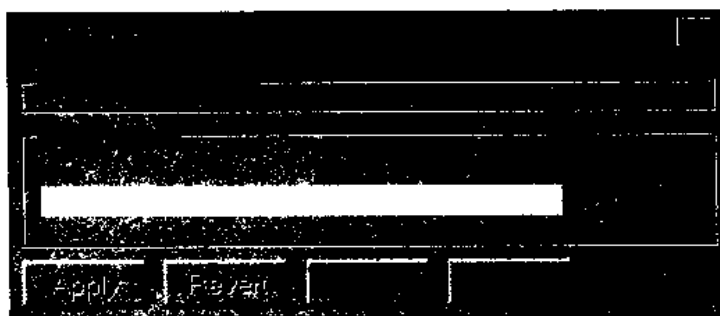


图 3.52

Popup则为用户提供可选择的列表框,所选的值代表variable。此时在下面会出现Popup strings输入框,用来设计选择的内容。如果输入a,b,c,如图3.53所示,则在双击该功能模块以后会出现下拉式列表框,如图3.54所示。



图 3.53

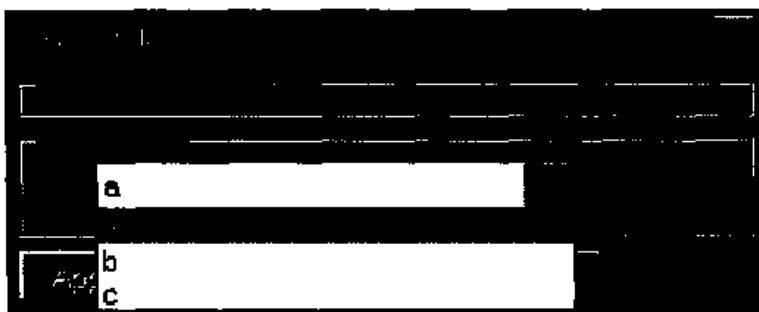


图 3.54

Checkbox则用于on与off的选择设定,如图3.55所示。

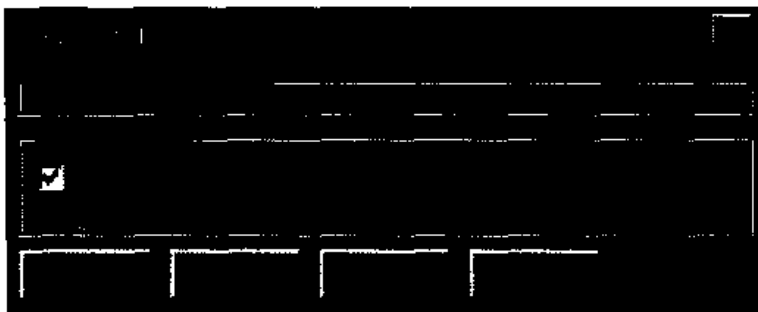


图 3.55

Assignment用于配合Control type的不同选择来提供不同的变量值,变量值有Evaluate, Literal两种,其含义如表3.7所示。

表3.7 Assignment不同参数的含义

Control type Assignment	Evaluate	Literal
Edit	输入的文字是程序执行时所用的变量值	输入的内容当作字符串处理
Popup	为选择的序号,选第一项,则输出值为1,往下依此类推	选择的内容当作字符串处理
Checkbox	输出为1或0	输出为'on'或'off'的字符串

上面说明的应用方法请参考范例3.6与范例3.7。

范例3.5 设计一个有临界值的比较器，该临界值为输入的变量。当输入信号大于0且大于临界值时输出为1，否则为0。

程序：ex343.mdl

首先到模块库中把Subsystem功能模块复制到设计区域中，再进入Subsystem中设计程序，如图3.56所示。

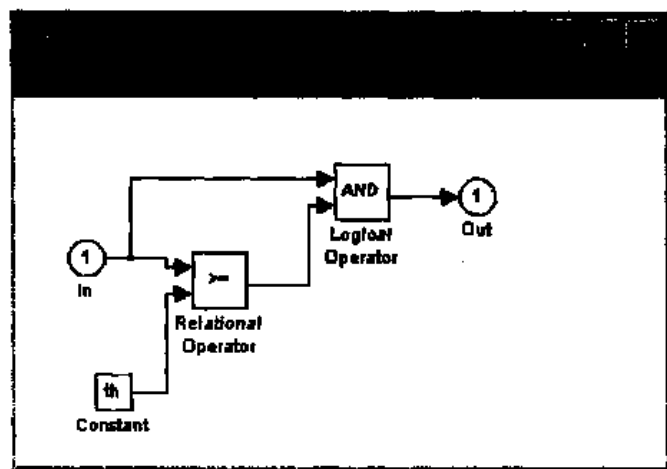


图 3.56

接着回到SIMULINK设计区域中加入信号源与示波器，并到Subsystem功能模块中设定Icon属性，结果如图3.57所示。

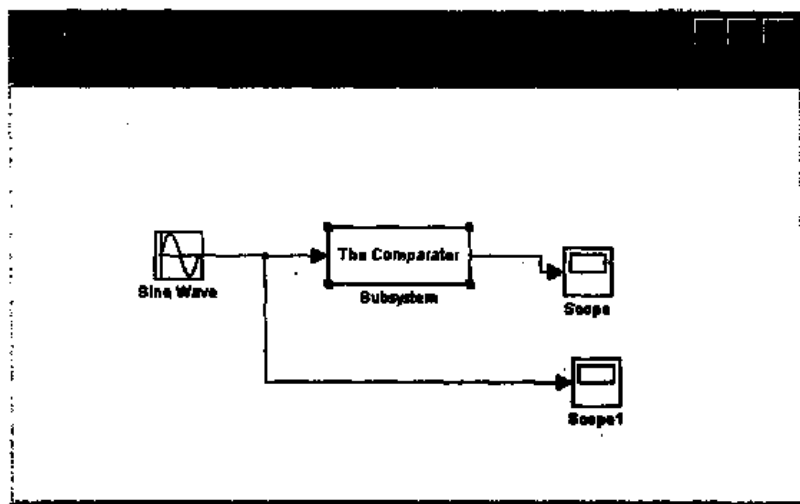


图 3.57

然后设定Initialization属性，在prompt栏中加入说明为threshold，由于在图3.56中就已经定义了constant功能模块的变量为th，所以在variable栏中也要加入th。单击Apply按钮后，出现如图3.58所示的结果，prompt列表多了一行上面设计的变量。

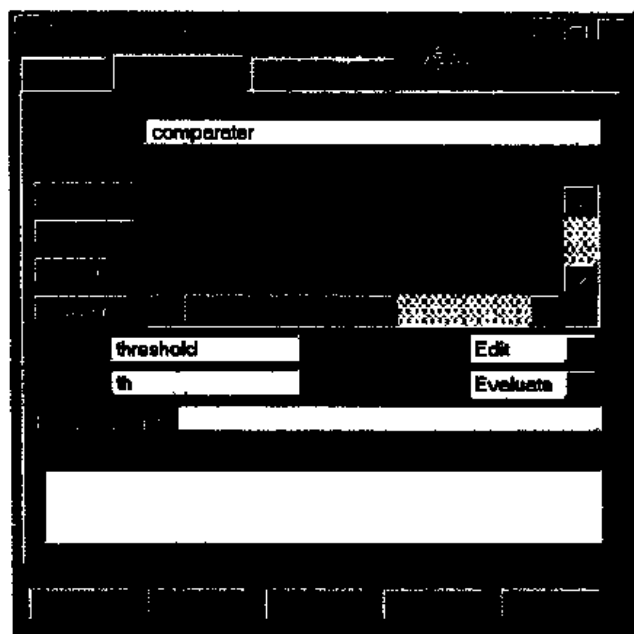


图 3.58

至此，设计阶段完成，接着就是执行程序了。先双击刚设计完成的功能模块，出现如图3.59所示的窗口供输入临界值，假设输入为0.5。最后单击Start执行程序，结果如图3.60所示。图3.61为原始信号。

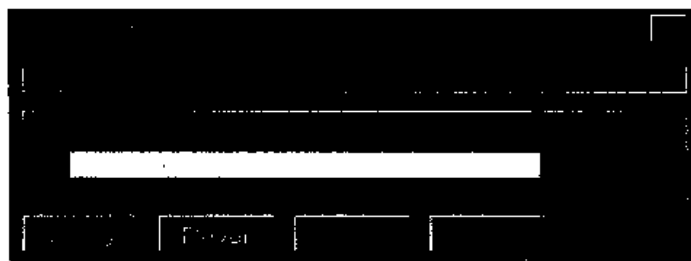


图 3.59

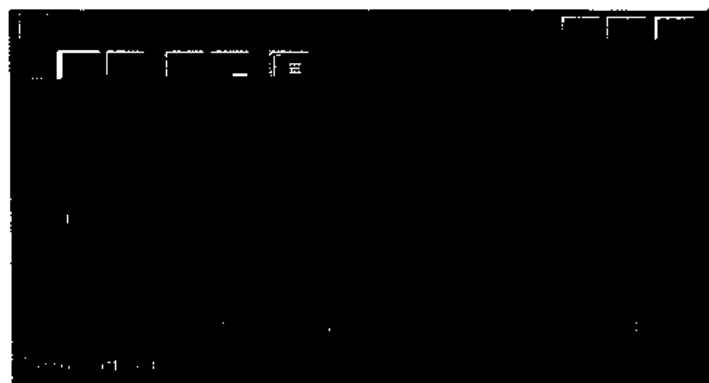


图 3.60



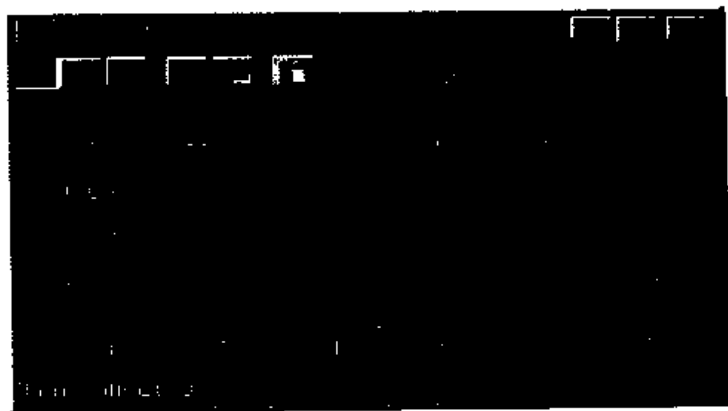


图 3.61

范例3.6 用下拉式列表框重新设计范例3.5。

程序: ex344.mdl

如图3.62所示, 把原来设定Control type的Edit改为Popup, 而Assignment初始设定的Evaluate会把数值设定为1,2,3..., 这些不是我们所需要的结果, 所以要改为Literal。但是输出又变为字符串, 因此variable内原先的数值变量th要改为字符串变量, 在这里改为t, 然后在Initialization commands里执行

```
th=str2num(t)
```

指令, 把t转换成数值。

双击comparater功能模块, 则出现如图3.63所示的下拉式列表框。如果选中0.1, 执行的结果如图3.64所示。

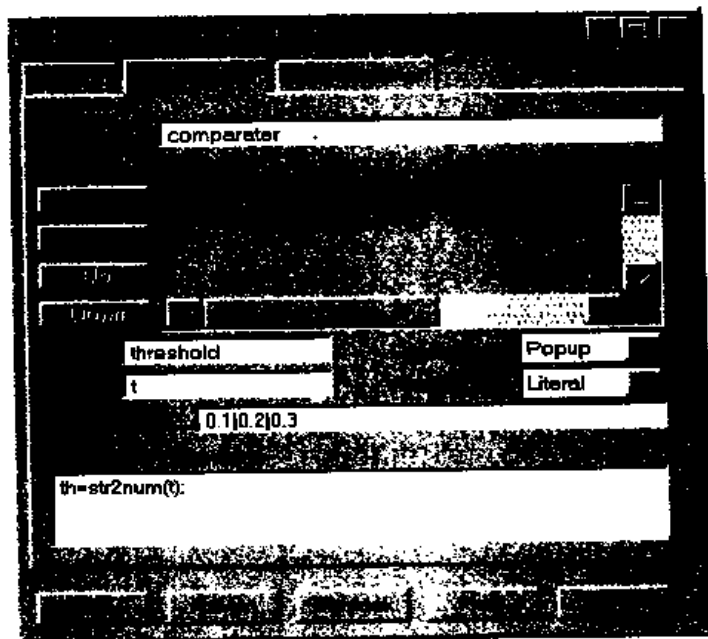


图 3.62

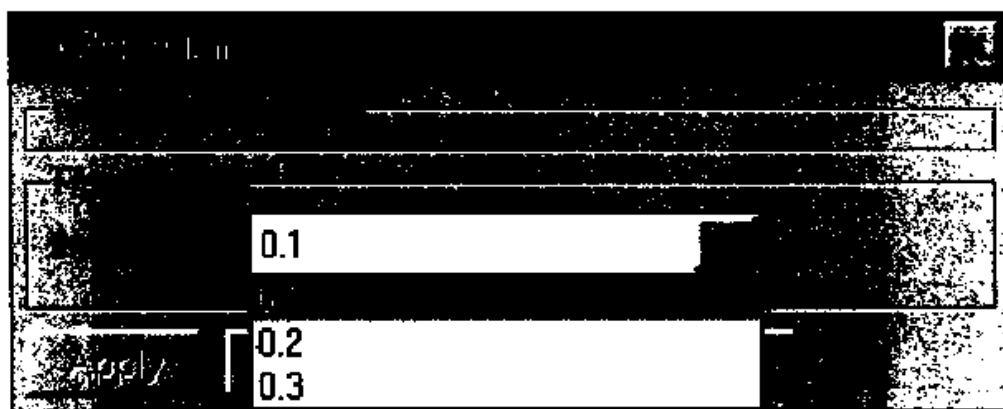


图 3.63

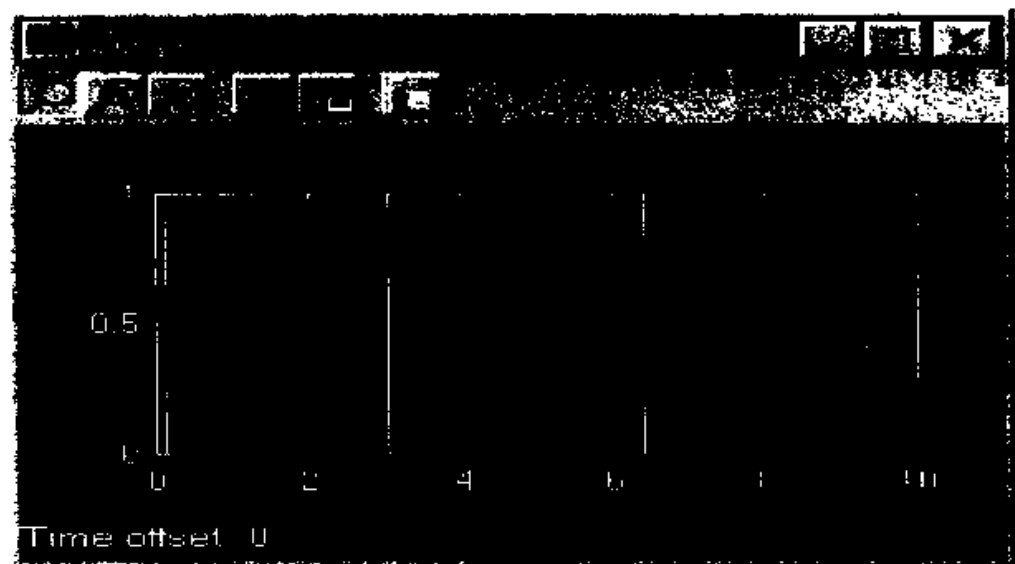


图 3.64

**范例3.7 用checkbox选择是否需要临界值，重新设计范例3.5。**

程序: ex345.mdl

按照范例3.6将Control type改为checkbox，并在Initialization commands中输入判断程序。

```

if t
    th=0.5;    如果选中checkbox，则设定临界值为0.5
else
    th=0      否则不设定临界值
end

```

执行程序，结果如图3.65所示。

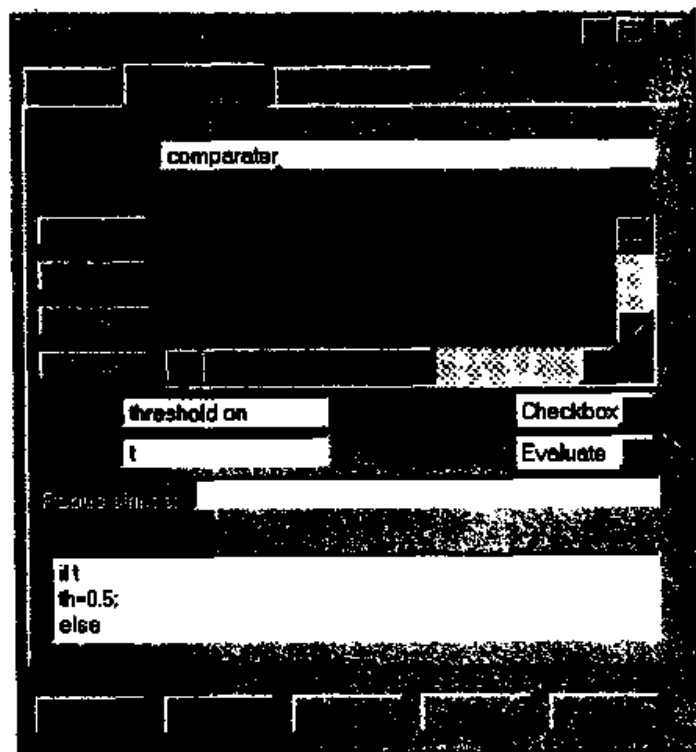


图 3.65

如果设定临界值，如图3.66所示，则执行结果如图3.60所示。

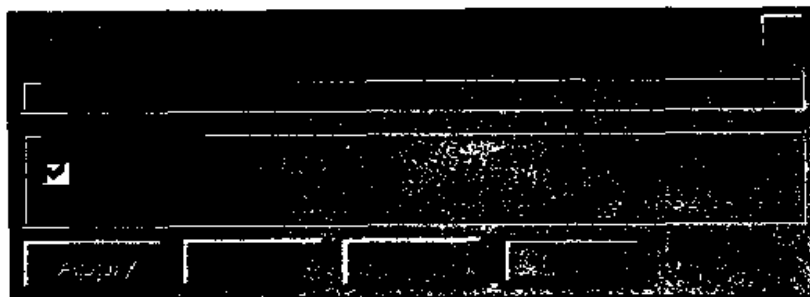


图 3.66

如果不设定临界值，如图3.67所示，则执行结果可将sin波转化为方波，如图3.68所示。

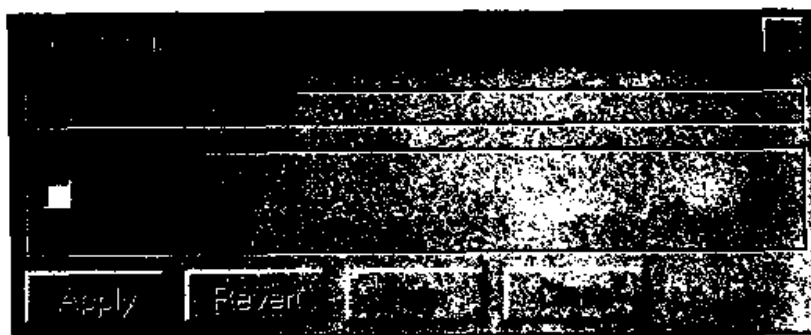


图 3.67

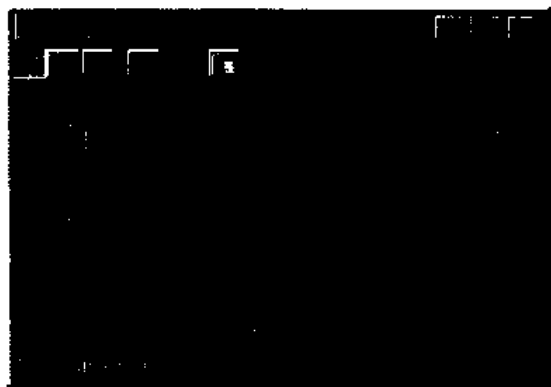


图 3.68

### 3.4.3 Documentation标签页

在这个标签页里，设计者可以针对设计完成的功能模块来编写相应的Help和说明文字。在Block description中输入的文字会出现在属性窗口的说明部分，而在Block help中输入的文字则会显示在单击属性窗口中的help按钮后浏览器所加载的HTML文件中。

以范例3.7为例，编写其Help，如图3.69所示，执行后会出现相应的说明文字，如图3.70和图3.71所示。

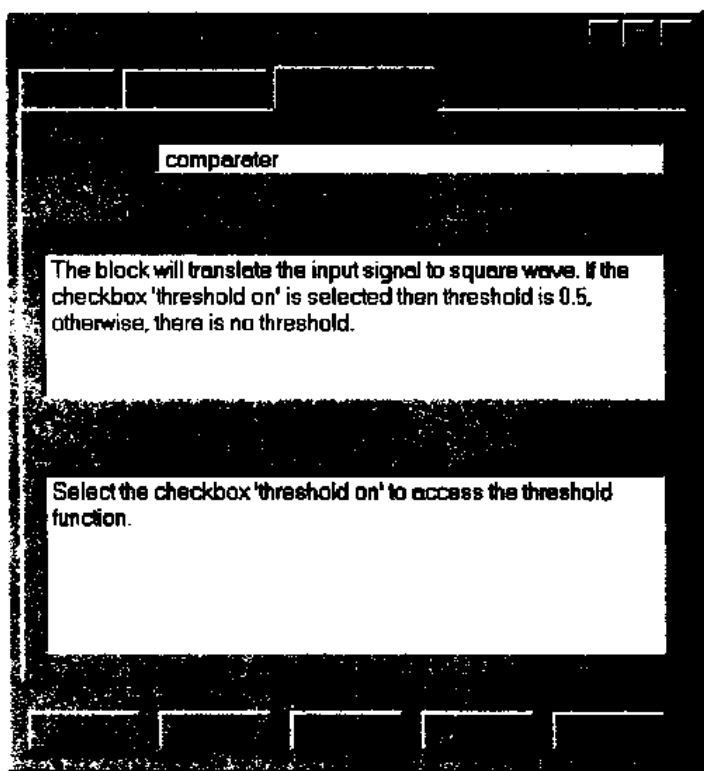


图 3.69

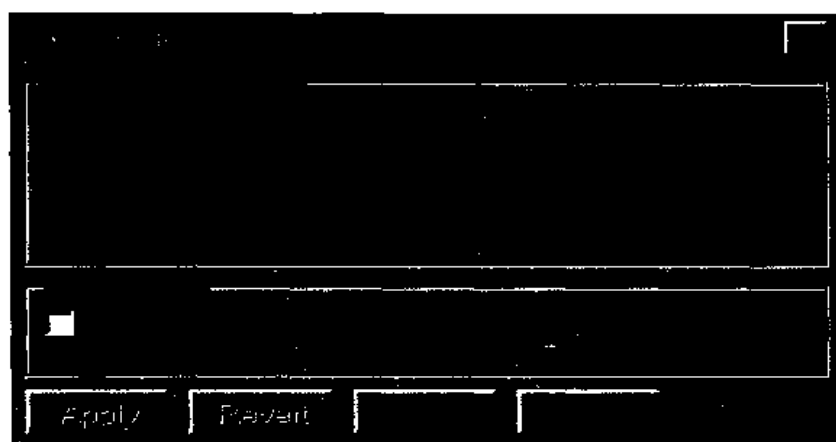


图 3.70

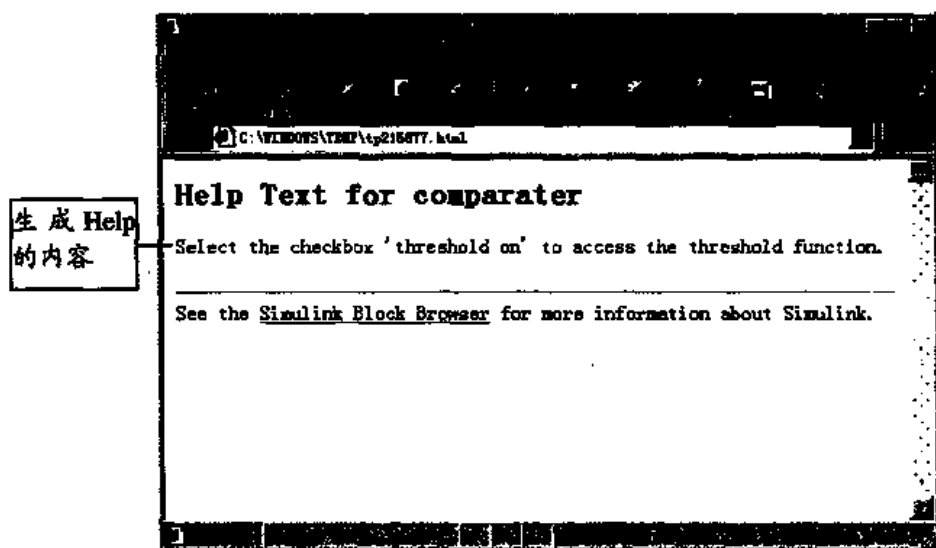


图 3.71

#### 3.4.4 Unmask标签页

对于一个Subsystem功能模块而言，通过上述的mask步骤之后，该功能模块就可以执行所设计的功能了。另一方面，也可以通过unmask对该功能模块进行修改，待修改完成后再进行mask操作，所以设计的弹性很大，给设计者的空间也很大。下面就以ex345.mdl的设计文件为例说明unmask的步骤。

选择The Comparater功能模块，执行Edit→Edit Mask，如图3.72所示，单击窗口下方的“Unmask”按钮，接着双击该功能模块，就会出现先前设计的系统，如图3.73所示。所以mask与unmask就是通过Edit菜单中的Edit mask与Mask Subsystem两个选项互相切换。

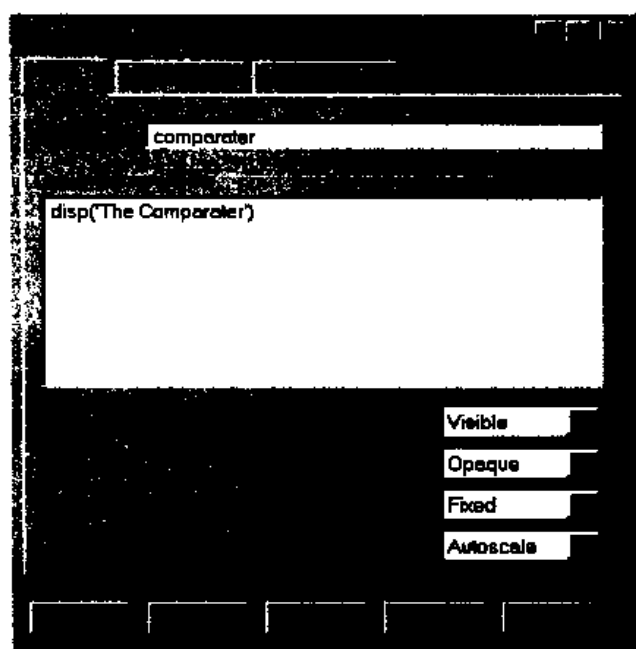


图 3.72

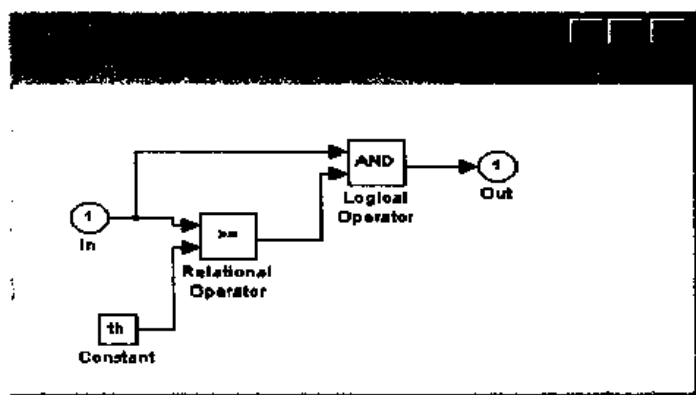


图 3.73

### 3.5 模拟参数的设定

#### 3.5.1 解题器(Solver) 参数的设定

解题器的参数设定是进行模拟工作前的准备工作，如何设定参数是根据程序设计的需求而定的，以便SIMULINK发挥最好的效果。最基本的参数设定包括起始与终止模拟的时间、模拟的步阶大小与解题器的类别等等。

进行参数设定可以直接使用simset指令，或者选择Simulation→Parameters，出现对话框后进行设定。对话框如图3.74所示，其中的选项意义如下：

- Apply用于修改参数后的确认。表示将目前改变的参数设定用于接下来的模拟。

- Revert是复原的意思,表示沿用最近一次打开对话框前未修改的参数设定用于接下来的模拟。
  - Close关闭对话框,在进行模拟时可以不关闭该对话框。
  - Help则可以显示使用方法的说明。
  - Simulation time为模拟的起止时间,不一定等于真正的时间,需要根据程序的大小与复杂程度而定。
  - Solver options包括许多选项,其中:
    - Variable-step能够在模拟过程中自动修改步阶大小(step sizes)以满足容许误差的设定与零跨越(zero crossing)的需求。对于解题器可以参考2.6.1节的说明,包括ode45, ode23, ode113, ode15s, ode23s, discrete几种,一般设定ode45为解题器,但当模型没有表示连续的状态(continuous state)时,就必须选discrete。Max step size与initial step size的定义可参考2.6.4节第4项的说明,一般默认值为auto。
- Relative tolerance与Absolute tolerance的定义请参见2.6.4节第1项的说明。
- Fixed-step固定步阶的大小,不会自动修正步阶大小以满足容许误差的设定与零跨越的需求。其显示的Solver options与Variable-step不同,如图3.75所示。其解题器共有ode5, ode4, ode3, ode2, ode1, discrete几种可选,一般采用ode4作为解题器,它等效于ode45。另外ode3等效于ode23。其step size只可以设定为fixed step size,同时也没有output options的设定问题。
- output options的第一个选项为Refine output,其Refine factor最大为4,默认值为1,数字愈大则输出愈平滑。第二个选项为Produce additional output,设定其Output times的数值。例如, [0:2] 代表在0到2秒之间,解题器会使用连续扩展公式(continuous extension formular)算出更多的输出值,与Refine factor类似,它能使输出更平滑。第三个选项是Produce specified output only,只在设定的Output times中产生输出。

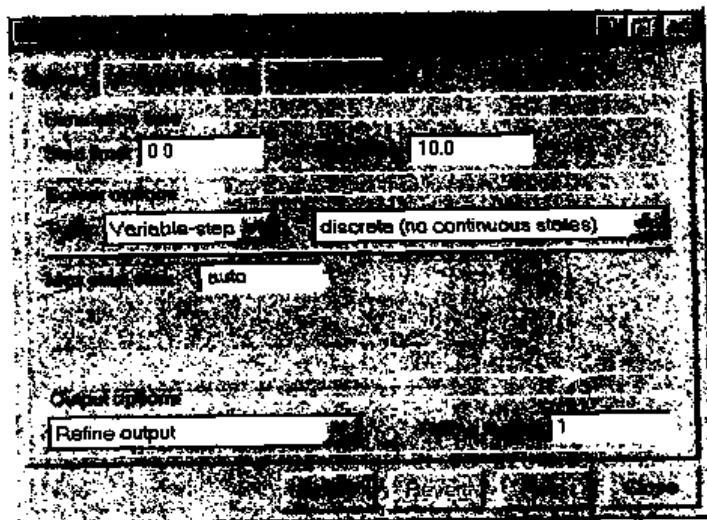


图 3.74 解题器参数设定窗口

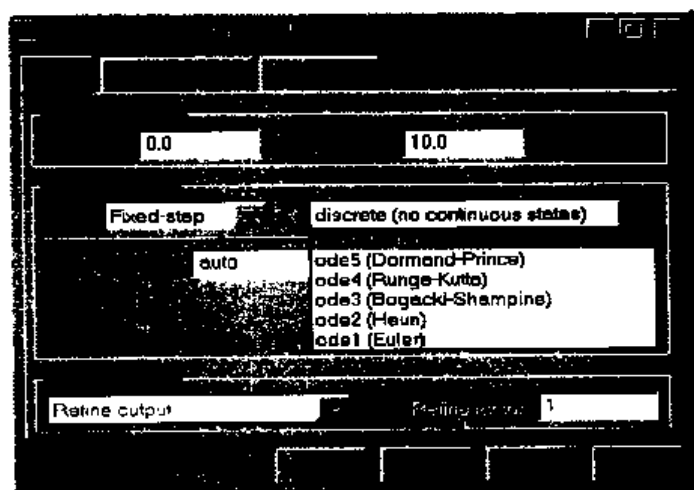


图 3.75

### 3.5.2 工作空间(Workspace)参数的设定

在Workspace I/O标签页内可以设定若干参数，如图3.76所示。与From Workspace和To Workspace功能模块类似，该标签页的主要目的是处理数据的输入与输出。可在Load from workspace中设定两个列向量的变量名称，如图3.77所示[t,u]其中t是时间，而u则是真正对应该时间的数据值。写出的部分在Save to workspace中设定。可以设定三个变量，分别为Time, States, Output。

设定State后就可以从其他地方读取状态值，或在得到稳定状态值时，将其保存起来以备下次模拟时使用。只要在Load initial及Save final中给定变量名称，便可以在下次模拟时通过调用这两个变量来使用。Limit rows to last可以限定可存取的行数。Decimation为降频的程度，降频系数的默认值为1，表示每一个点都返回状态与输出值；若设为2，则会每隔2个点返回状态与输出值，这些结果会被保存起来。

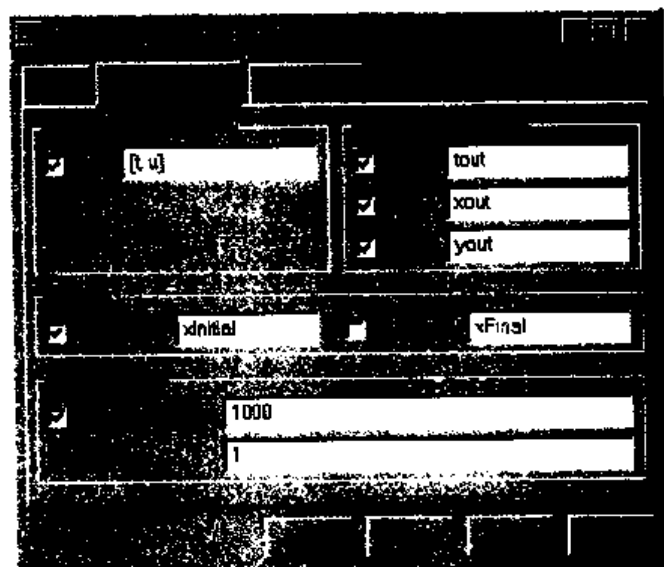


图 3.76 设定 Workspace 参数



Diagnostics标签页如图3.77所示, 用于诊断模型是否精确、效果是否很好, 或者在发生某些事件时, 设定应采取的措施。Event栏内为程序执行时可能遇到的情况, 而Action为情况发生时应执行的操作, 这些都是由用户选择设定的。

Consistency Checking用于s函数与SIMULINK解题器连接后的功能验证, 有off, warning, error三种选择, 一般都选off, 以免影响执行速度。如果选择warning或error, SIMULINK会验证输出值、零跨越值(zero crossing)、微分值和状态值是否正确。

最后一个是Disable zero crossing detection复选框, 该复选框的目的是取消对零跨越的检测, 但是如果模型内有Hit Crossing功能模块时, 最好不要选该复选框。

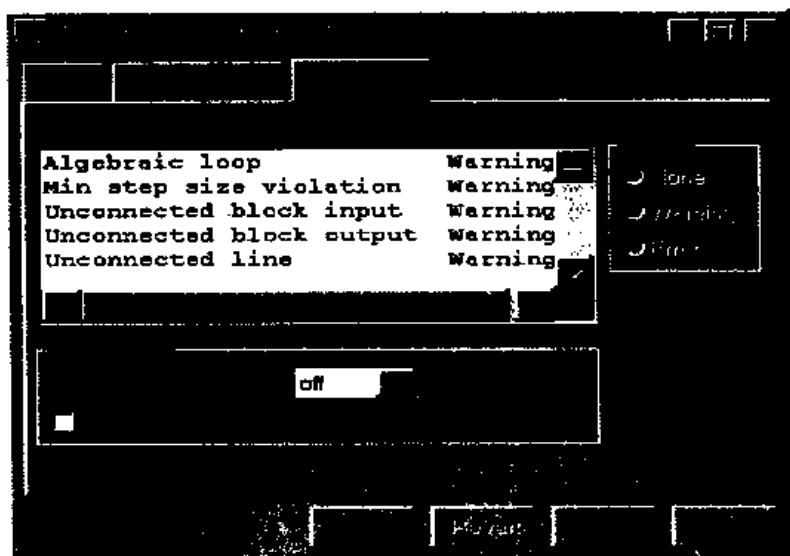


图 3.77 设定 Diagnostics

### 3.6 在命令行中执行SIMULINK

设定好SIMULINK的各项参数之后, 就可以执行模拟了。前面提到的方法是由鼠标单击执行, 本节则介绍另外一种执行SIMULINK的方法。该方法是在Command Line下执行, 并且直接设定参数以取代原先的设定。有了这种功能, SIMULINK就可以成为其他程序的子程序, 因而大大提高了程序设计的灵活性与扩充性。在主程序中调用SIMULINK的指令为sim(将1.x版本中的指令linsim、rk23、rk45、gear、adams、euler等集成在一起), 并用simset指令设定参数, 其指令格式为:

```
[t,x,y]=sim('模型名称', [t0 tf], simset('参数1', 参数值1, '参数2', 参数值2; ...))
```

其中, t0为模拟开始时间, tf为模拟终止时间。[t,x,y]为返回值, 可以省略。t为返回的时间向量值, x为返回的状态值, y为返回的输出向量值。

以上的指令格式也可以分为两部分, 先定义simset, 再把定义好的参数代入sim中。有关参数的种类与意义可以对照3.5节并参见下面的说明:

AbsTol      默认值为1e-6设定绝对误差范围(Absolute error tolerance), 用于variable-

	step的解题器。
Decimation	默认值为1降频的程度，以便决定隔几个点返回状态与输出值。
Solver	解题器的形式，可设为VariableStepDiscrete, ode45, ode23, ode113, ode15s, ode23s, FixedStepDiscrete, ode5, ode4, ode3, ode2, ode1。
MaxRows	默认值为0，表示不限制。若为大于0的数值，则表示限制输出与状态等的最大行数等于该数值。
InitialState	一个向量值，它用于设定初始状态。
FixedStep	用正数表示步阶的大小，仅用于fixed-step的解题器。
MaxStep	默认值为auto。用于variable-step的解题器，表示最大的步阶大小。

以上是常用且重要的参数，当然还可以设定很多参数，至于到底有哪些可以设定，可以用simget指令来获得。例如如果想知道ex342.mdl的各项参数，只需执行：

```
simget('ex342')
```

就可获得下列参数信息：

```

Solve      : 'VariableStepDiscrete'
RelTol     : 1.0000e-003
AbsTol     : 1.0000e-006
Refine     : 1
MaxStep    : 'auto'
InitialStep : 'auto'
MaxOrder   : 5
FixedStep  : 'auto'
OutputPoints : 'all'
OutputVariables : 'ty'
MaxRows    : 1000
Decimation : 1
InitialState : []
FinalStateName : 'xFinal'
Debug      : 'off'
Trace      : ''
SrcWorkspace : 'current'
DstWorkspace : 'current'
ZeroCross  : 'on'
```

读者可以试着改变上述参数的值，看看模型的变化，便可以了解其意义。现在以ex342.mdl为例，执行两行指令：

```

>> p=simset('solver','FixedStepDiscrete','fixedstep',2);
>> sim('ex342',[0 8],p)
```

然后在ex342.mdl中查看示波器功能模块，就会发现结果如图3.78所示，此结果不同于图3.29，其输出结果受simset指令的影响，每隔2秒运算一次，而且按照sim指令的设定执行了8秒。可见simset与sim的指令会改变原本在ex342.mdl中设定的参数值。

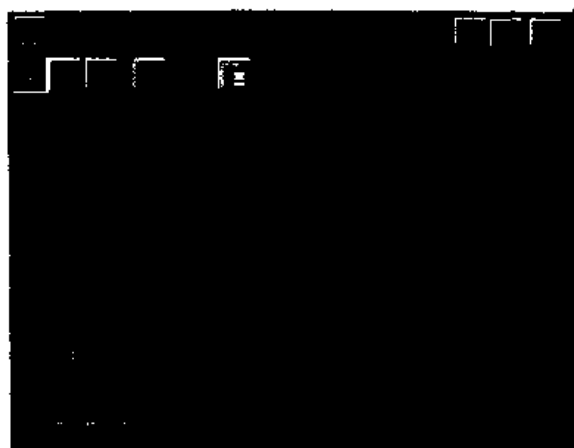


图 3.78

### 3.7 模型的线性化

非线性系统的线性化就是用一个非线性的模型找出其稳定的操作点(Operating point), 如果系统属性的变化只会在该操作点周围产生微小变化, 非线性的模型就能够以此操作点为基础, 表示成一个线性模型。以往的线性系统理论都能适用于该模型, 但是将非线性系统表示成一个线性系统会牵涉到很多复杂的数学理论, 如偏微分方程。MATLAB提供了linmod指令专门用于解决这个问题。

对于非线性系统:

$$\begin{aligned}\dot{X} &= f(x(t), u) \\ y(t) &= h(x(t))\end{aligned}$$

如果在操作点 $x_0$ 及输入 $u_0$ 的条件下, 将其表示成线性模型:

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

其指令格式为:

$$[A, B, C, D] = \text{linmod}(\text{'模型名称'}, \text{操作点向量}x_0, \text{输入向量}u_0)$$

下面举例说明如何将一个非线性系统在某个操作点上进行分解, 从而得到一个线性模型。

#### 范例3.8

非线性系统为:

$$\begin{aligned}\dot{x}_1 &= \frac{-1}{x_2} + u \\ \dot{x}_2 &= ux_1\end{aligned}$$

在操作点 $x_0 = [0 \ 1]$ ,  $u_0 = 1$ 的条件下将其分解成一个线性模型。

使用linmod的方法为：先建立表示上述非线性系统的含输入、输出端的SIMULINK模型，如图3.79所示。程序：nolinm.mdl。

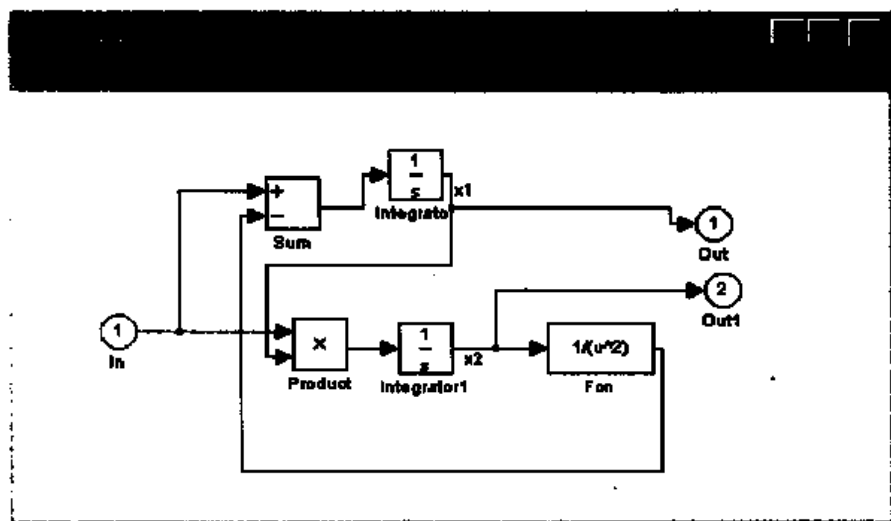


图 3.79

图中的in代表输入u，而out及out1则分别代表状态x1与x2。接着在命令窗口中执行如下指令：

```
[a,b,c,d] = linmod('nolinm', [0 1], 1)
```

即可获得线性模型的矩阵值如下：

```
a=
    0    2.0000
  1.0000    0
b=
  1.0000
    0
c=
  1.0000    0
    0  1.0000
d=
    0
    0
```

因此线性系统的状态方程为：

$$\dot{x} = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \quad x0=[0 \ 1] \quad u0=1$$

$$Y = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} X$$

linmod指令对于线性模型也是很有用的，与分解非线性系统比较，不同的地方是不必

给定操作点。对于线性系统来说,任何操作点得到的动态方程都是一样的。除非连接的方法不同才会得到不一样的动态方程。读者也许还有印象,在控制理论里面讲到了如何将传递函数转换成动态方程,方法包括串联法和并联法,根据不同的接法则得到不同的状态变量 $x$ 的定义,所以得到的状态方程外观会不同,但都表示相同的系统。比如:

**范例3.9** 将传递函数  $\frac{3}{s^2 + 3s + 2}$  分别用并联法及串联法求出状态方程。

在这里用SIMULINK及linmod指令来求解。

首先建立模型,先建立串联法的模型,程序为linm.mdl,如图3.80所示。接着执行:

```
[a,b,c,d] = linmod('linm')
```

会得到下列结果:

```
a=
    -1     3
     0    -2
b=
     0
     1
c=
     1     0
d=
     0
```

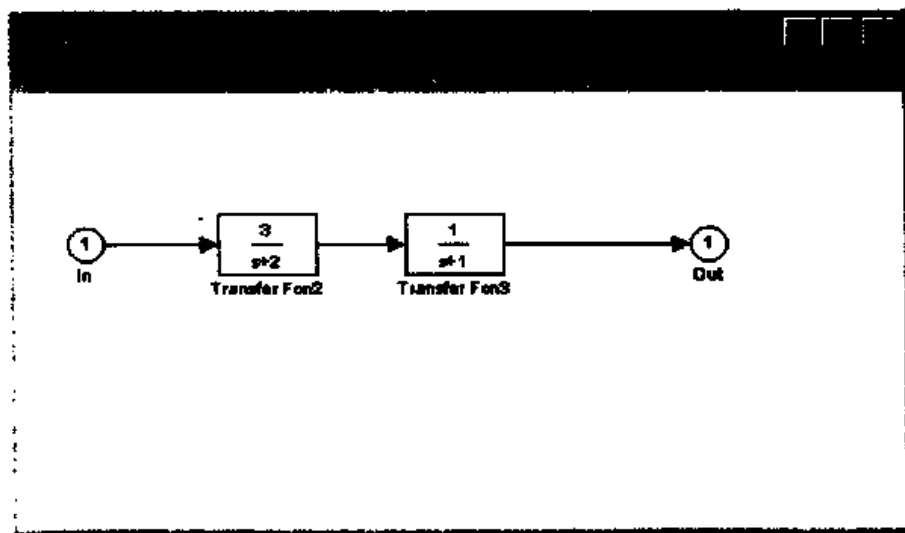


图 3.80

如果用并联法连接,由于  $\frac{3}{s^2 + 3s + 2} = \frac{-3}{s + 2} + \frac{3}{s + 1}$ , 所以可以连接,如图3.81所示,程序为linm1.mdl, 执行:

`[a,b,c,d] = linmod('linm1')`会得到下列结果:

```

a=
    -2     0
     0    -1

b=
     1
     1

c=
   -33

d=
     0
  
```

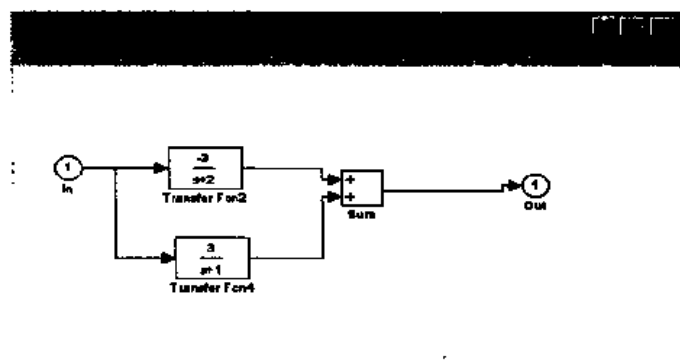


图 3.81

很显然这两种连接法得到的动态方程是不同的，但是其响应是相同的，并代表相同的系统。在上面两种连接法的系统中分别执行linmod求得a,b,c,d后，再用指令：

```
step(a,b,c,d)
```

去观察比较两者的执行结果，可以发现结果均如图3.82所示，所以很显然这两种表示法都是可以的，关键是要看设计者如何定义状态变量，也就是由连接的形式决定。

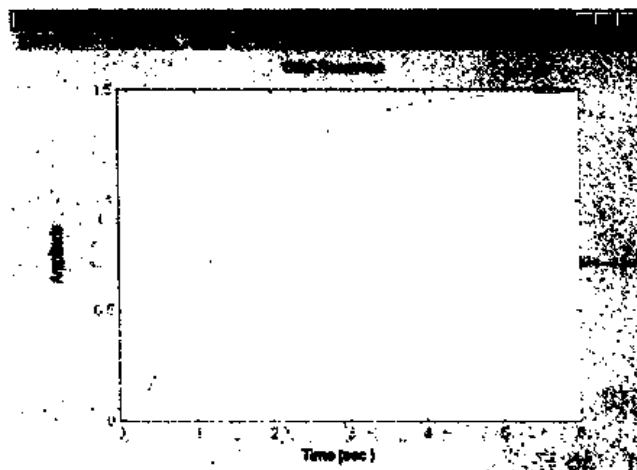


图 3.82 步阶响应均相同

### 3.8 模型的状态稳定平衡点

当一个系统的状态变化最小时,表示在该对应输入条件下是最佳平衡点。因此,对于给定状态值 $x$ 与输入值 $u$ 、输出值 $y$ 的条件下,求该最接近的平衡点 $(x,u,y,dx)$ 的指令如下:

```
[x,u,y,dx]=trim('模型名称',状态值x0,输入值u0,输入输出y0)
```

设定的值用`[]`代表不限定,更详细的内容可以执行`help trim`查阅。在这里继续说明上节的例子,求其平衡点。

执行:

```
[x,u,y,dx]=trim('linm',[1.5;0.1],2,1.5)
```

得到

```
x=
    2.1000
    0.7000
u=
    1.4000
y=
    2.1000
dx=
    1.0e-015*
    -0.4441
    0
```

如果不限定 $x,u$ 而只限定 $y$ 为1.5,则执行:

```
[x,u,y,dx]=trim('linm',[ ],[ ],1.5)
```

得到结果:

```
x=
    0.7500
    0.2500
u=
    0.5000
y=
    0.7500
dx=
    0
    0
```

所以要得到平衡点,必须先确定想得到平衡点所要接近的条件,然后用指令`trim`就可以得到所要的答案。如果想知道其解题的次数,可以通过返回`options`的第10个元素来得到,方法如下:

```
[x,u,y,dx,options] = trim('limn' [], [], 1.5]
```



查看options(10)

```
ans=
```

```
16
```

表示解题器求解了16次才算出答案。

### 3.9 建立可以激活与触发的功能模块

Simulink提供了两个功能模块——激活(Enabled)功能模块与触发(Triggered)功能模块，可以用于设计按照时序(Timing)操作的新功能模块。激活功能模块能够在输入信号大于0时激活新的功能模块，而触发功能模块可以根据给定的Rising、Falling、Either或Function-call的设定而触发新的功能模块。这两种功能模块位于Connections模块库中，模块符号为  和 。双击打开，Enable功能模块的内容如图3.83所示。

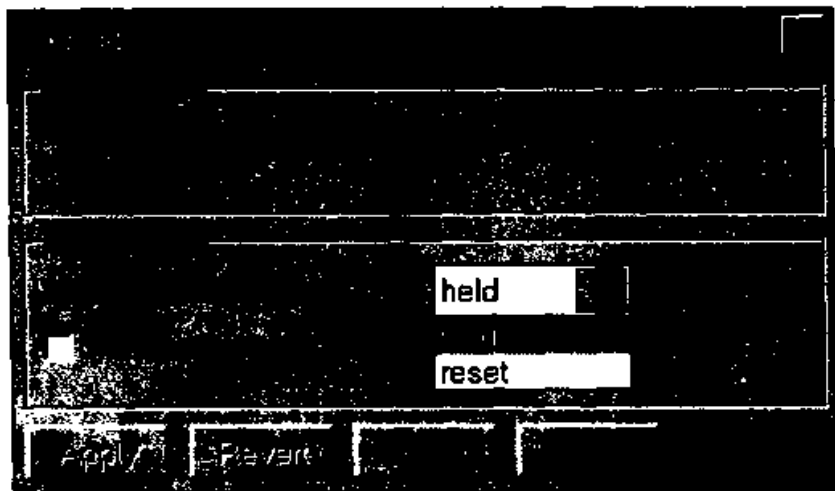


图 3.83 Enable 功能模块的设定窗口

该模块包括held, reset两个选项以及Show output port复选框:

- held: 当控制信号小于零，也就是在失效(Disable)状态时，新功能模块的输出维持最近的输出状态。
- reset: 当控制信号小于零，也就是在失效(Disable)状态时，新功能模块的输出重置为起始值。
- Show output port: 如果选中该复选框，激活功能模块会产生输出端以供设计时使用。

Trigger功能模块的内容如图3.84所示。



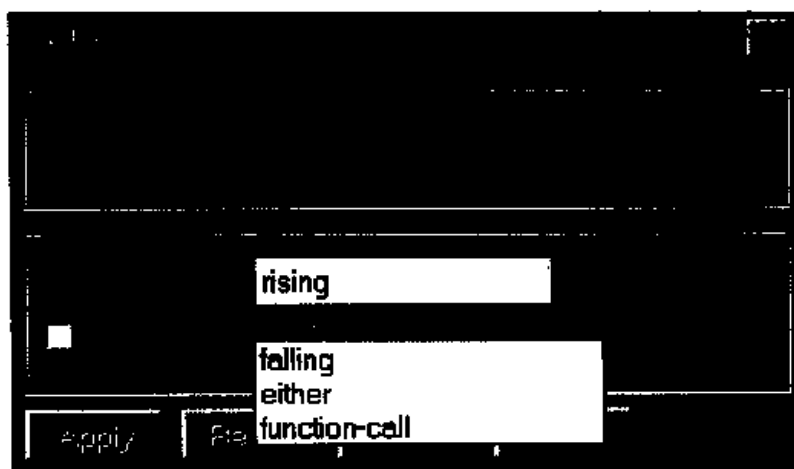


图 3.84 Trigger 功能模块

此模块中可以选择的触发条件有4种，分别是rising, falling, either, function-call。

设计方法：

第一步：先从Connections模块库中将Subsystem功能模块复制到设计区域中。由于“激活功能模块”与“触发功能模块”只能放到Subsystem的设计区域中，所以必须进入Subsystem的设计区域进行设计。

第二步：将“激活功能模块”或“触发功能模块”复制到Subsystem的设计区域中。

第三步：进行新的功能模块设计。

范例3.10 在激活状态输出sin波形，在非激活状态重量为0。

程序：ex391.mdl

根据上述的设计方法，先建立Subsystem功能模块，然后放入“激活功能模块”，如图3.85所示。

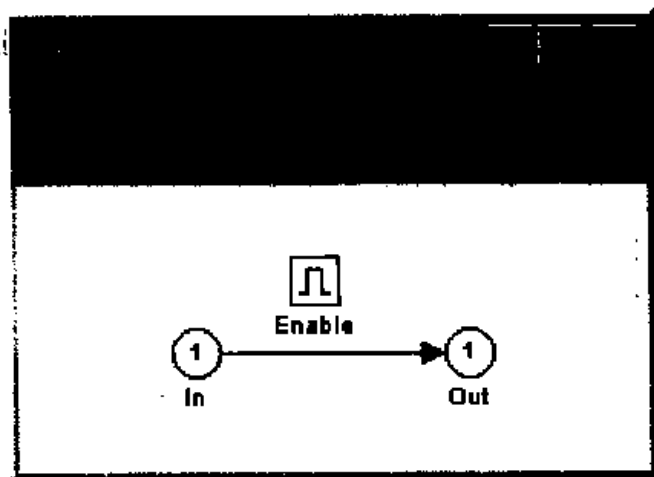


图 3.85

接着设定“激活功能模块”为reset，如图3.86所示。

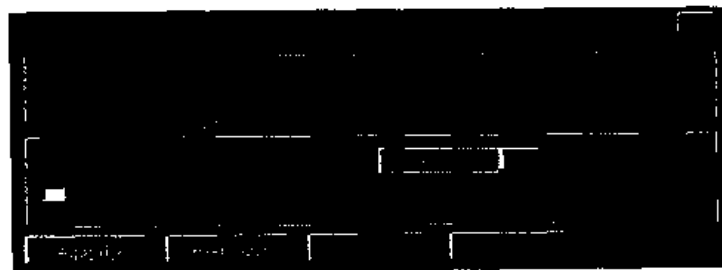


图 3.86

输出端也设定为reset，如图3.87所示。

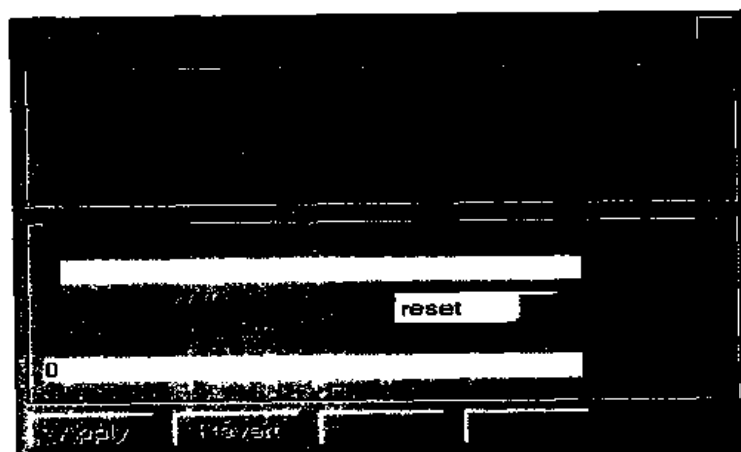


图 3.87

将该新功能模块封装 (mask)，如图3.88所示。省略该步骤并不会影响程序的执行。

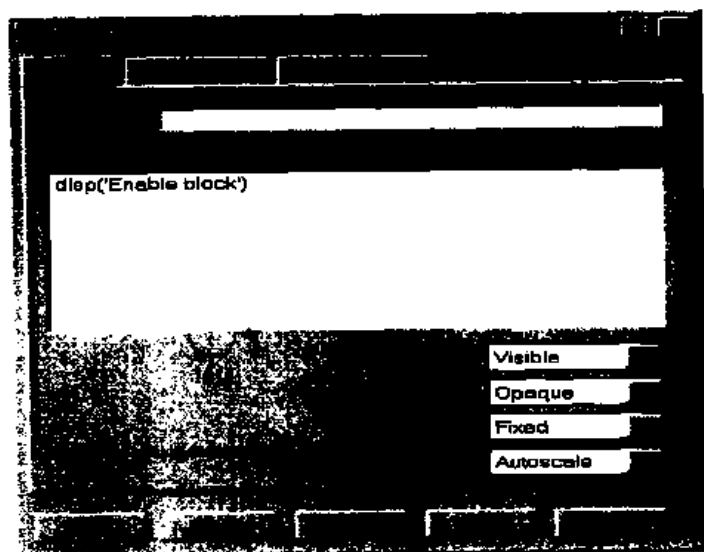


图 3.88

最后把信号源与示波器放进去。控制信号在此是脉冲发生器(Pulse generator), 如图3.89所示。

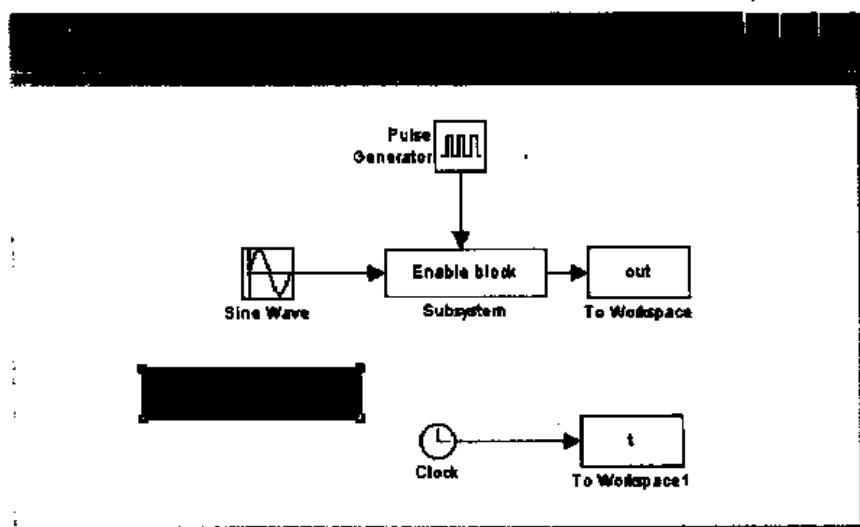


图 3.89

在非激活状态下, 输入正弦波信号被重置为零; 在激活状态下, 正弦波则维持原信号输出, 如图3.90所示。激活信号的来源是脉冲发生器。

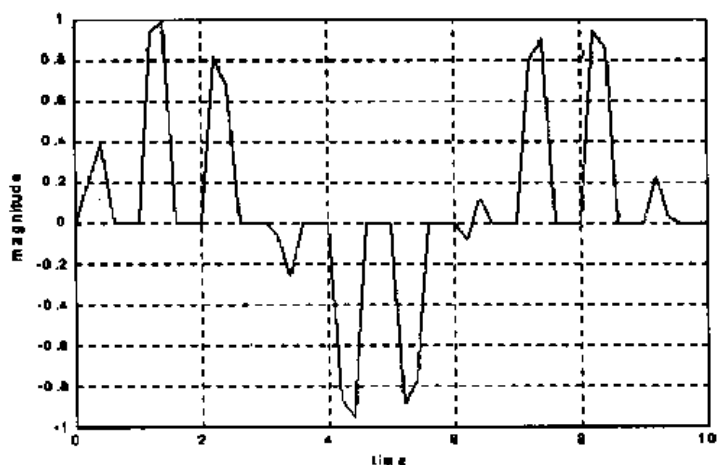


图 3.90 模拟结果

“触发功能模块”的设计步骤与“激活功能模块”的步骤相同, 请读者自行练习。

范例3.11 用激活功能模块设计RS触发器(RS Flip-Flop)。

程序: ex392.m

rsflop.mdl

RS触发器是最基本的触发器, 由两个AND门(gate)及两个NOR门组成, 如图3.91所示。

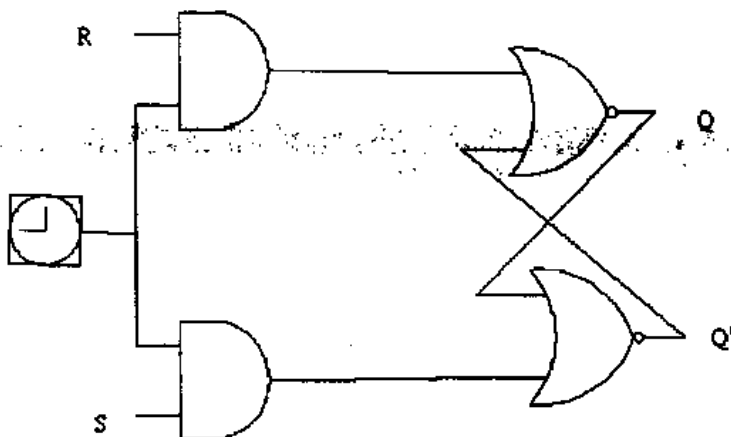


图 3.91 RS 触发器

对于此触发器, 如果  $S=1$   $R=0$ , 则  $Q$  输出为 1; 如果  $S=0$   $R=1$ , 则  $Q$  输出为 0; 如果  $S=0$   $R=0$ , 则  $Q$  保持原来状态; 如果  $S=1$   $R=1$ , 则避免发生。

SIMULINK 设计的方法是把激活功能模块当作触发器的时序控制信号, 为了使模拟不产生代数回环(Algebraic loop)的错误, 必须在反馈的地方加上加法器产生的初始值。根据 RS 触发器的结构, 从非线性模块库中把 AND 门与 NOR 门复制到子系统功能模块中, 连接完成 RS 触发器的一个新功能模块, 如图 3.92 所示。

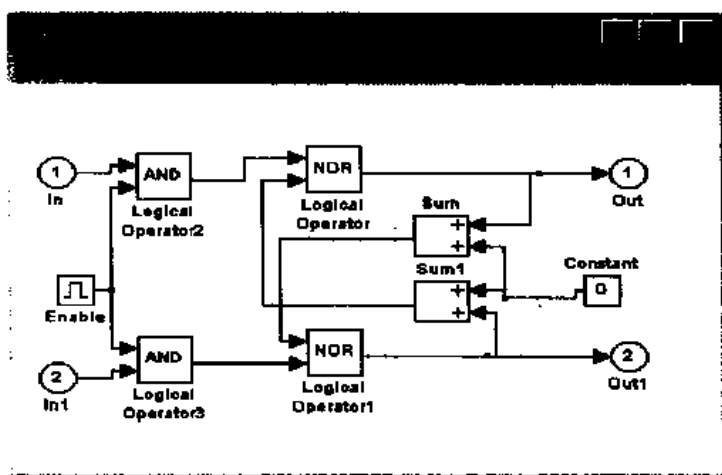


图 3.92 RS 触发器设计图

接着设计外部电路, 用“开关(Switch)功能模块”与“阶跃(Step)功能模块”组合成一个脉冲发生器, 如图 3.93 所示。在 0 至 2 秒之间  $R=S=0$ , 第 2 秒至第 4 秒之间  $S=1$ ,  $R=0$ , 第 5 秒至第 7 秒之间  $R=1$ ,  $S=0$ , 激活信号周期为 1 秒, 如图 3.95 所示。

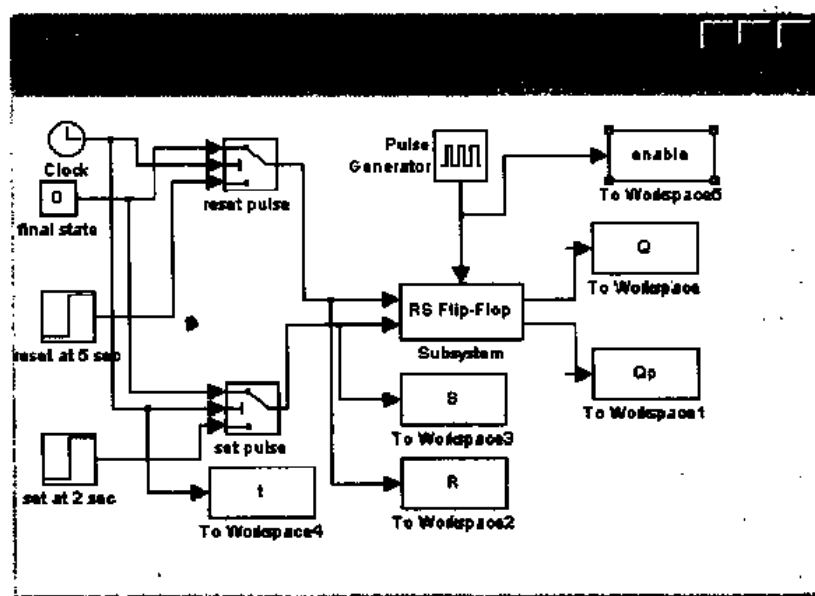


图 3.93 整体设计图

最后编写程序调用RS触发器功能模块，执行后绘出图形。程序为：

```
clear;
sim('rsflop')
figure(1)
subplot(311);
plot(t,S);
xlabel('time(sec)')
ylabel('S input');
axis( [0 10 0 2] );
grid on
subplot(312);
plot(t,R);
xlabel('time(sec)')
ylabel('R input');
axis( [0 10 0 2] );
grid on
subplot(313);
plot(t,enable);
xlabel('time(sec)')
ylabel('enable');
axis( [0 10 0 2] );
grid on
figure(2)
subplot(211);
plot(t,Q);
```

```
xlabel('time(sec)')
ylabel('Q output')
axis( [0 10 0 2] );
grid on
subplot(212);
plot(t,Qp);
xlabel('time(sec)')
ylabel('Q_inverse output')
axis( [0 10 0 2] );
grid on
```

模拟结果如图3.94所示, 输出Q在第2秒被置为1, 维持到第5秒后, 被重置为0。图3.95所示的是模拟的输入信号R、S与激活信号。

其他的触发器设计原理也是相同的。注意要避免产生代数回环错误, 给定适当的初始值。通过该例的RS触发器可以组合出更复杂的逻辑电路, 来验证所设计的逻辑电路是否正确。

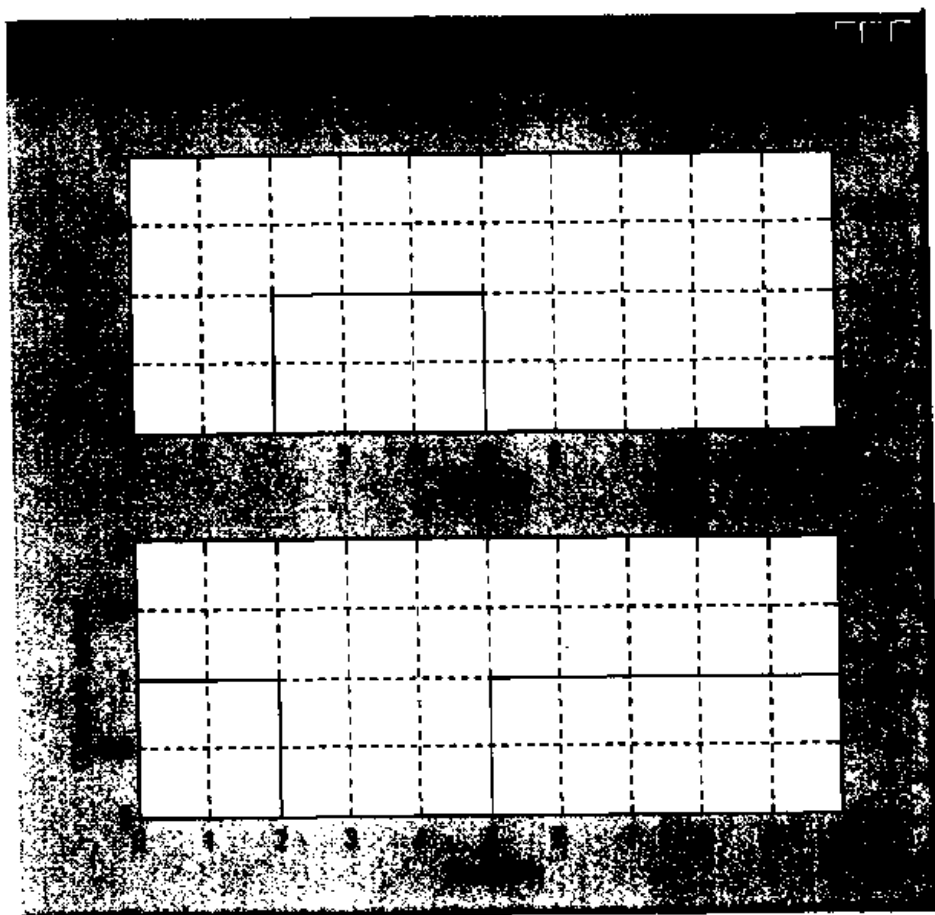


图 3.94 模拟结果

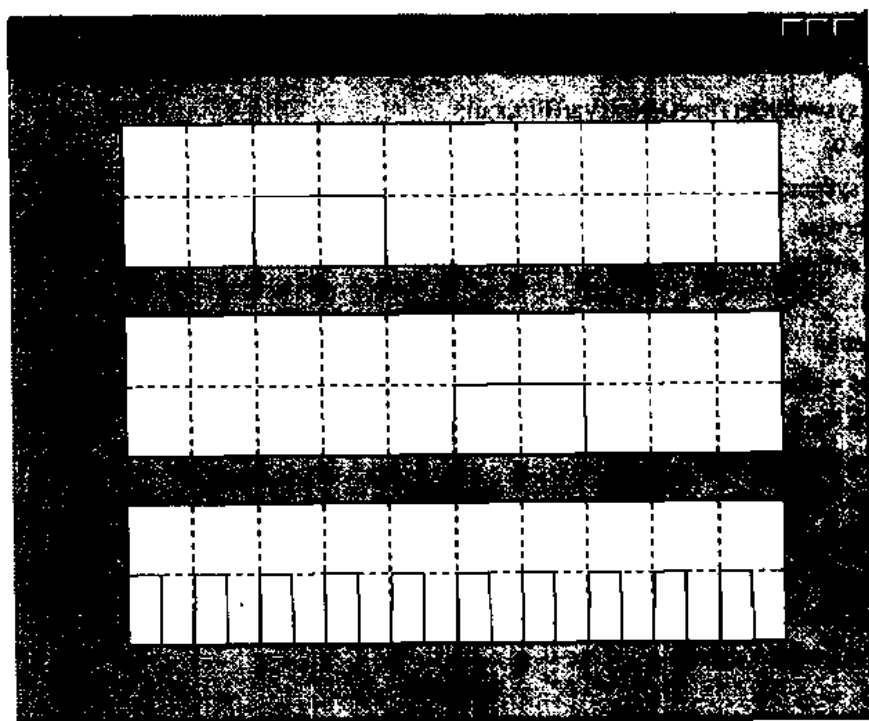


图 3.95 模拟信号的输入

### 3.10 S-function的设计

S-function是System Function的简称，其功能是通过MATLAB或C语言程序，设计出可实现所需功能的功能模块。因此S-function非常重要，如果学会了S-function，就能充分发挥SIMULINK的功能。否则只能用模块库中的模块去拼出系统。

S-function的运行是由sys参数控制的，所以第一步是先写程序。程序的写法有一定的格式，在MATLAB 4.x中必须自己编写，而MATLAB 5.x提供了一个模板(Template)程序，只要在必要的子程序中编写程序并输入参数即可。第二步就是到非线性模块库中将S-function系统功能模块复制进来，然后输入程序文件名，以供调用。S-function的设计步骤很简单，只有两步，但是可以实现非常复杂的功能，所以可以说它是SIMULINK的精华。

模板程序位于toolbox/simulink/blocks目录下，文件名为sfuntmpl.m，为便于说明将其注释部分省略简化后摘录如下：

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;
case 1,
    sys = mdlDerivatives(t,x,u);
case 2,
    sys = mdlUpdate(t,x,u);
case 3,
```

```

    sys=mdlOutputs(t,x,u);
case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
case 9,
    sys=mdlTerminate(t,x,u);
otherwise
    error( ['Unhandled flag = ',num2str(flag)] );
end
function [sys,x0,str,ts] =mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 0;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
sys = [];
function sys=mdlUpdate(t,x,u)
sys = [];
function sys=mdlOutputs(t,x,u)
sys = [];
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1; %Example, set the next hit to be one second later.
sys = t + sampleTime;
function sys=mdlTerminate(t,x,u)
sys = [];

```

通过上面的说明可以看出，主程序本身是SIMULINK模型的一个子程序，只提供分支标志 (Flag)，再通过设定分支标志来处理另外6个必备的子程序。这6个子程序的内容，就是使用者需要设计的部分，表3.8是关于6个子程序的说明。

表3.8 (摘自SIMULINK 用户指南)

mdlInitializeSizes(flag=0)	<p>在S-function运行之前，必须先设定其运行的结构。此结构用sizes来存储，然后再用simsizes指令去提取。其结构的内容有6项：</p> <ul style="list-style-type: none"> <li>• sizes.NumContStates连续状态个数</li> <li>• sizes.NumDiscStates离散状态个数</li> <li>• Sizes.NumOutputs输出个数</li> <li>• sizes.NumInputs输入个数</li> </ul>
----------------------------	--



(续表)

	<ul style="list-style-type: none"> <li>• <code>sizes.DirFeedthrough</code>是否设定<code>direct feedthrough</code>, 取决于输出是否为输入的函数, 或者取样频率是否为输入的函数</li> <li>• <code>sizes.NumSampleTimes</code>一般设定为1</li> </ul>
<code>mdlDerivatives(flag=1)</code>	输出值 <code>sys</code> 为状态值的微分
<code>mdlUpdate(flag=2)</code>	输出值 <code>sys</code> 为状态值在下一时刻的更新值
<code>mdlOutputs(flag=3)</code>	输出值 <code>sys</code> 为输入值与状态值的函数
<code>mdlGetTimeOfNextVarHit(flag=4)</code>	输出值 <code>sys</code> 为下一次被触发的时间。例如: <code>sys=t+u(1)</code> , 则输入项 <code>u(1)</code> 的值为下一次被触发的时间
<code>mdlTerminate(flag=9)</code>	模拟结束

SIMULINK调用S-function的输入参数有4个, 分别为`t,x,u,flag`:

- `t`为时间。
- `x`为状态向量。
- `u`为输入向量。
- `flag`则是SIMULINK执行何种操作的阶段标记。

返回的参数值也有4个, 分别为`sys,x0,str,ts`:

- `sys`为S-function根据`flag`的值, 运算得出来的解。
- `x0`为初始状态。
- `str`对`m`型文件的S-function来说是空矩阵。
- `ts`为两列的向量, 定义取样时间及偏移量(offset)。

对于`mdlInitializeSizes`, `mdlDerivatives`, `mdlOutputs`三个子程序, 一定要非常熟练地运用。至于`mdlGetTimeOfNextVarHit`, `mdlUpdate`则用于离散系统或比较复杂的系统, 需要时再参考SIMULINK用户指南里面提供的详细说明和范例。下面以如何维持波形的峰值为例进行说明。

### 范例3.12

程序: `ex3101.m`

`holdp.mdl`

S-function:`sfun_h.m`

通过范例2.7的例子对原始波形进行设计, 前段程序同`ex231_3.m`一样, 后半段则调用SIMULINK模型。模型中的S-function是主要的部分, 由模板程序复制过来之后修改`mdlInitializeSizes`, `mdlOutputs`部分, 程序如下:

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

```

case 2,
    sys=mdlUpdate(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
case 9,
    sys=mdlTerminate(t,x,u);
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; % at least one sample time is needed
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
sys = [];
function sys=mdlUpdate(t,x,u)
sys = [];
function sys=mdlOutputs(t,x,u)
m=u(1);
n=u(2);
sys = max(m,n);
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1; %Example, set the next hit to be one second later.
sys = t + sampleTime;
function sys=mdlTerminate(t,x,u)
sys = [];

```

输入参数个数值

在此编写程序

完成S-function之后，接着建立SIMULINK模型，将功能模块“S-function”复制到设计区域中，打开其属性页，输入S-function的文件名sfun\_h，如图3.96所示。完成的模型如图3.97所示。

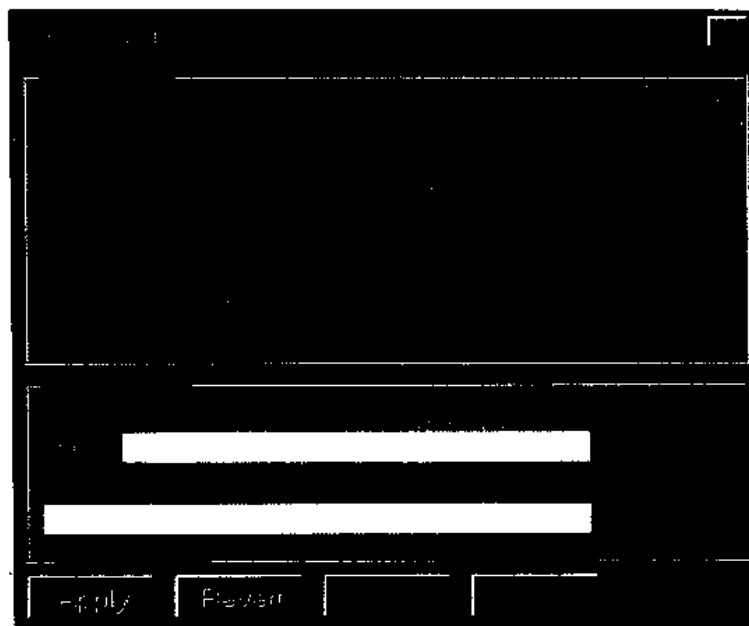


图 3.96 S-function 功能模块

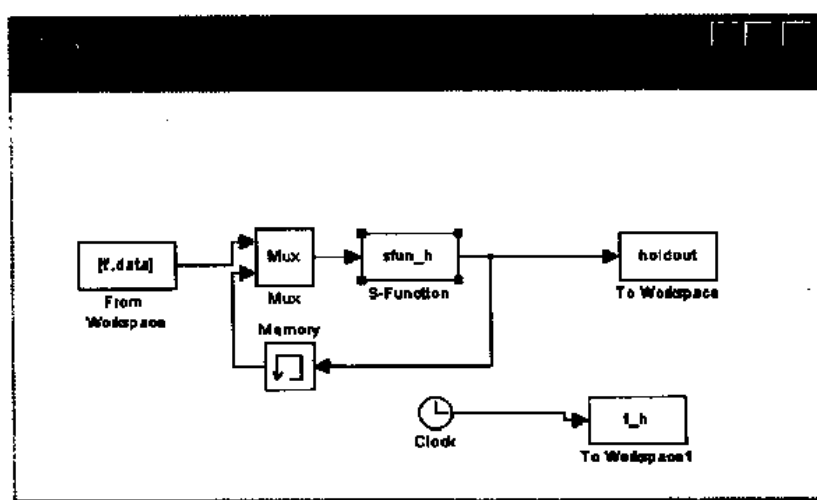


图 3.97 集成设计

最后编写主程序，读取波形并调用SIMULINK模型进行运算处理，程序如下：

```
%ex3101.m hold the peak value
%data i/o;
clear;
fid=0;%initialize file identifier
while fid<1
    filename='ex231data.txt';
    [fid,message] =fopen(filename,'r');%open file from disk
    if fid==-1
        disp(message)
```

```

end
end
data=fscanf(fid, '%g');%read data from file
%get the time point reference
for u=1:length(data);
    t(u)=u;
end;
%By SIMULINK model 'sfunc_h' to hold the peak value
sim('holdp');
plot(t,data);hold on
plot(t_h,holdout, 'r: ');

```

执行的结果如图3.98所示。

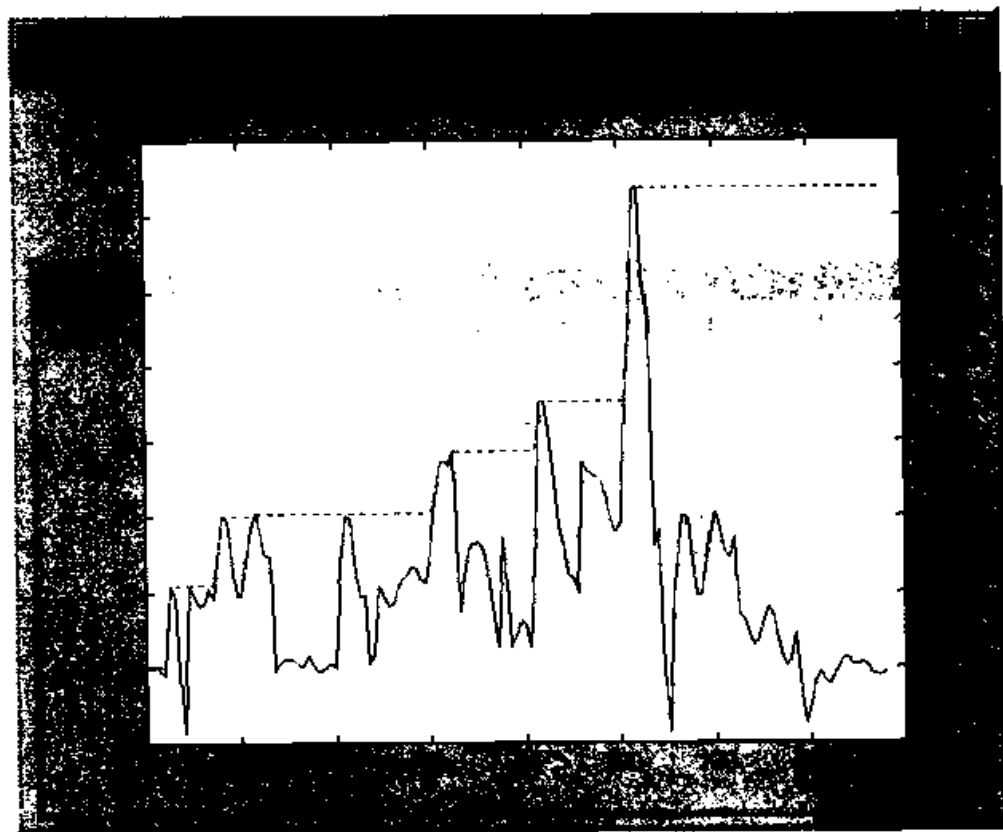


图 3.98 模拟结果

**范例3.13** 用SIMULINK设计两节机械手臂(见图3.99)定位控制, 其初始条件为 $q_1=0^\circ$ ,  $q_2=0^\circ$ , 希望到达的位置为 $q_{d1}=60^\circ$ ,  $q_{d2}=90^\circ$ 。

本范例根据Slotine and Li所提出的比例微分法(proportional derivative简称P.D.)来进行机械臂的定位控制。简述如下:

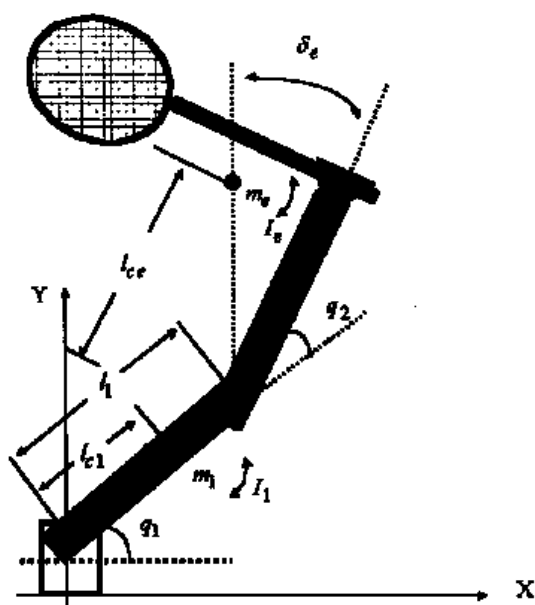


图 3.99 两节机械手臂

对于图3.99所示的机械手臂，其控制量为角度向量，即 $q = [q_1, q_2]^T$ ，输入为 $\tau = [\tau_1, \tau_2]^T$ ，系统方程为：

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau$$

也就是：

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -h\dot{q}_2 & -h(\dot{q}_1 + \dot{q}_2) \\ h\dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

其中

$$H_{11} = a_1 + 2a_3 \cos q_2 + 2a_4 \sin q_2$$

$$H_{12} = H_{21} = a_2 + a_2 \cos q_2 + a_4 \sin q_2$$

$$H_{22} = a_2$$

$$h = a_3 \sin q_2 - a_4 \cos q_2$$

$$a_1 = I_1 + m_1 L_{cl}^2 + I_e + m_e l_{ce}^2 + m_e l_1^2$$

$$a_2 = I_e + m_e l_{ce}^2$$

$$a_3 = m_e l_1 l_{ce} \cos \delta_e$$

$$a_4 = m_e l_1 l_{ce} \sin \delta_e$$

参数值为：

$$m_1 = 1 \quad l_1 = 1 \quad m_e = 2 \quad \delta_e = 30^\circ \quad I_1 = 0.12 \quad l_{cl} = 0.5 \quad I_e = 0.25 \quad l_{ce} = 0.6$$

由Lyapunov定理及能量守恒定理可知:

$$\frac{1}{2} \frac{d}{dt} [\dot{q}^T H \dot{q}] = \dot{q}^T \tau$$

如果选择

$$\tau = -K_p \tilde{q} - K_D \dot{q}$$

作为控制律(control law), 关节角度误差(joint angles erro)为:

$$\tilde{q} = q - q_d$$

其中 $q_d$ 代表期望的角度(desired joint angles)向量; 而微分与比例的增益值为:

$$K_D = 100I$$

$$K_p = 20K_D$$

在此情况下, 总能量为:

$$V = \frac{1}{2} [\dot{q}^T H \dot{q} + \tilde{q}^T K_p \tilde{q}]$$

其能量 $V$ 的微分为:

$$\dot{V} = \dot{q}^T (\tau + K_p \tilde{q}) = -\dot{q}^T K_D \dot{q} \leq 0$$

根据Lyapunov定理可知, 系统保持稳定。所以控制律是正确并可行的。现在以SIMULINK为工具, 建立这个模型, 如图3.100所示。

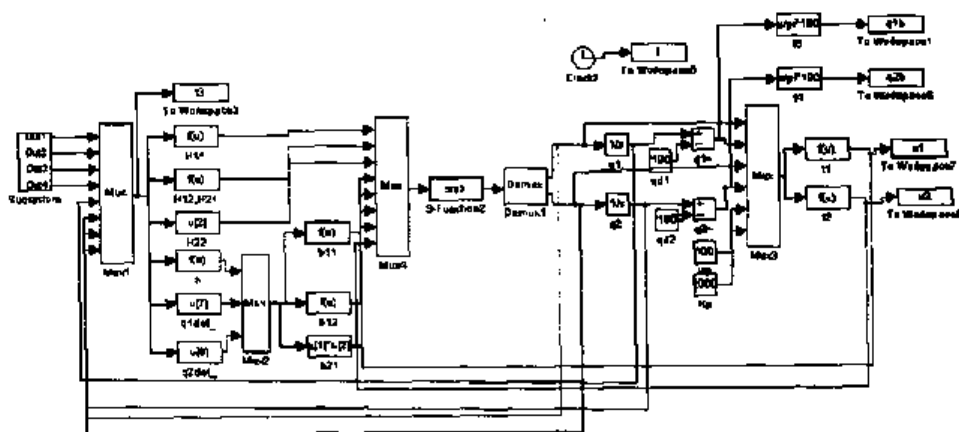


图 3.100 设计图

模型说明: SIMULINK模型文件: pd1.mdl

前半段为建立机械臂的模型, 中间则为S-function功能模块, 编写的程序为srob.m。在该程序中用到的是微分输出与输出两个子程序, 在微分输出中求的值为状态的微分, 接着再通过输出子程序输出该状态。因此在S-function设计中, 常会配合Mux的设计, 固定以u来表示该输出的向量, 并用Fcn功能模块来对u做一些数学运算, 例如机械臂的控制律为:

$$\tau = -K_p \tilde{q} - K_D \dot{q}$$

对于角度q1而言, 在模型中是用t1功能模块来进行运算的, 由Mux3功能模块来看, u(5)、u(6)均为控制的增益量, u(1)为角度的微分量, u(2)为角度的误差量, 得到q1的控制律为:

$$-u(6)*u(2)-u(5)*u(1)$$

建立这一公式的结果如图3.101所示。

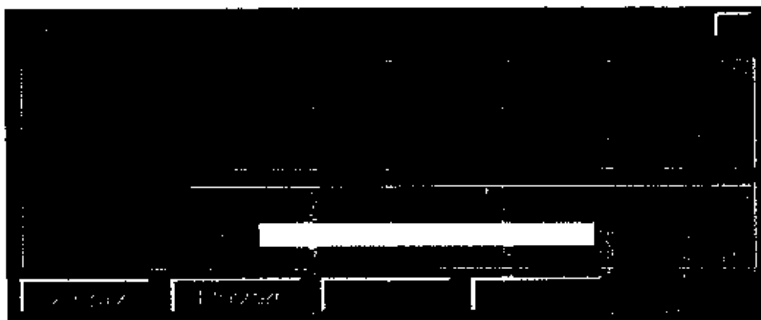


图 3.101 建立控制律公式

其他的fcn功能模块同理, 只要把u的每个向量所代表的意义搞清楚, 然后再按照公式依次代入就可以了。对于S-function的程序编写技巧, 读者最好多找其他的例子来练习, 这样才能有更大的收获。模型的其他部分就比较简单了, 这些部分只是把结果放到workspace中, 以便主程序能够画出图形。

下面是S-function的程序, 程序文件名为srob.m。

```
function [sys,x0,str,ts] = srob(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 2,
    sys=mdlUpdate(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
case 9,
```

```

    sys=mdlTerminate(t,x,u);
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 = 8;
sizes.DirFeedthrough = 1;
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)
a = [u(1) u(2);u(2) u(3)] ;
b = [u(4) u(5);u(6) 0] ;ui= [u(7);u(8)] ;
sys=-inv(a)*b*x+inv(a)*ui;
function sys=mdlUpdate(t,x,u)
sys = [] ;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1; %Example, set the next hit to be one second later.
sys = t + sampleTime;
function sys=mdlTerminate(t,x,u)
sys = [] ;

```

求状态的微分

输出为角度的微分状态

通过下面的程序完成画出结果的工作。画出的结果如图3.102和图3.103所示。

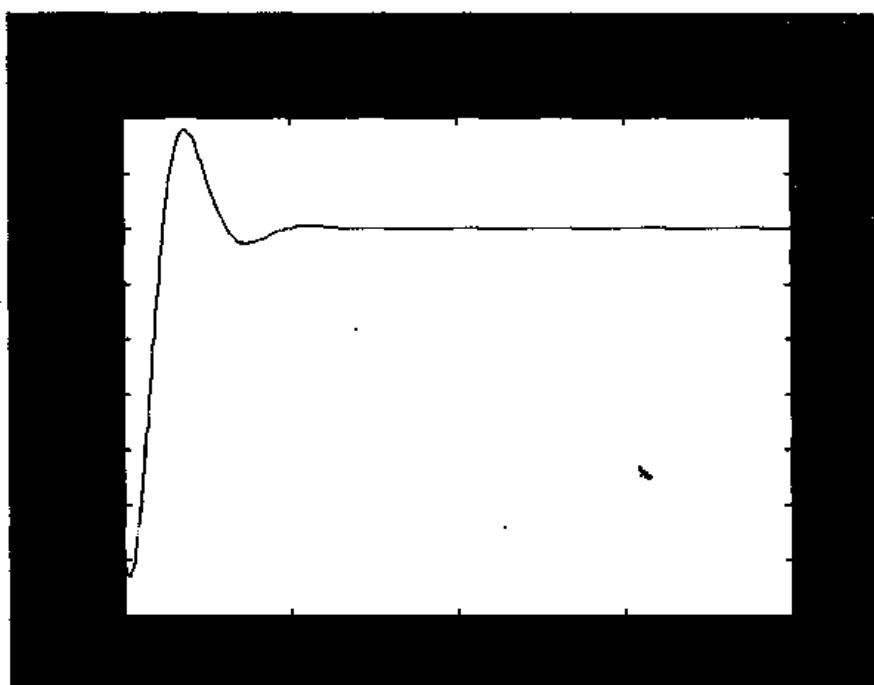
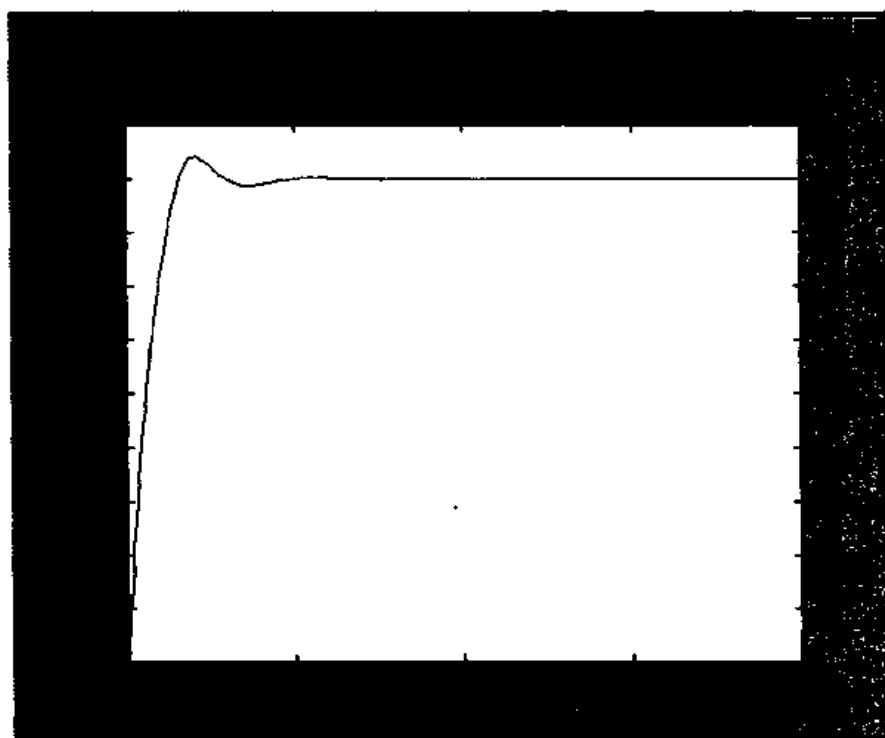
主程序: ex3102.m

```

clear;
sim('pdl')
plot(t,q1b)
xlabel('time(sec)')
ylabel('Q1 joint angle error(degree)')
figure(2)
plot(t,q2b)
xlabel('time(sec)')
ylabel('Q2 joint angle error(degree)')

```



图 3.102  $q1$  输出结果图 3.103  $q2$  输出结果

如果要简化SIMULINK的功能模块，可以利用S-function的功能，将部分功能模块建立到S-function程序中。由此也可以认识到，我们可以无限发挥S-function的功能。下面就沿用范例3.13，将机械臂的角度微分量、误差量及模型参数变化，都用S-function来编写，只

剩下control law使用SIMULINK来建立。注意比较一下两者之间执行速度的差异。

范例3.14 沿用范例3.13的两节机械手臂，用S-function改写定位控制程序，其初始条件为 $q_1=0^\circ$ ， $q_2=0^\circ$ ，希望到达的位置为 $q_{d1}=60^\circ$ ， $q_{d2}=90^\circ$ 。

在图3.100中，SIMULINK的功能模块前半部分、角度 $q_1$ 和 $q_2$ 的微分量及误差量计算部分，可以改写为S-function。剩下的部分如图3.104所示，显然只剩下control law和输出及输入部分。还要在S-function功能模块中将S-function name修改为新的程序名称，在本例中为srob2，如图3.105所示。

SIMULINK程序：pd2.mdl

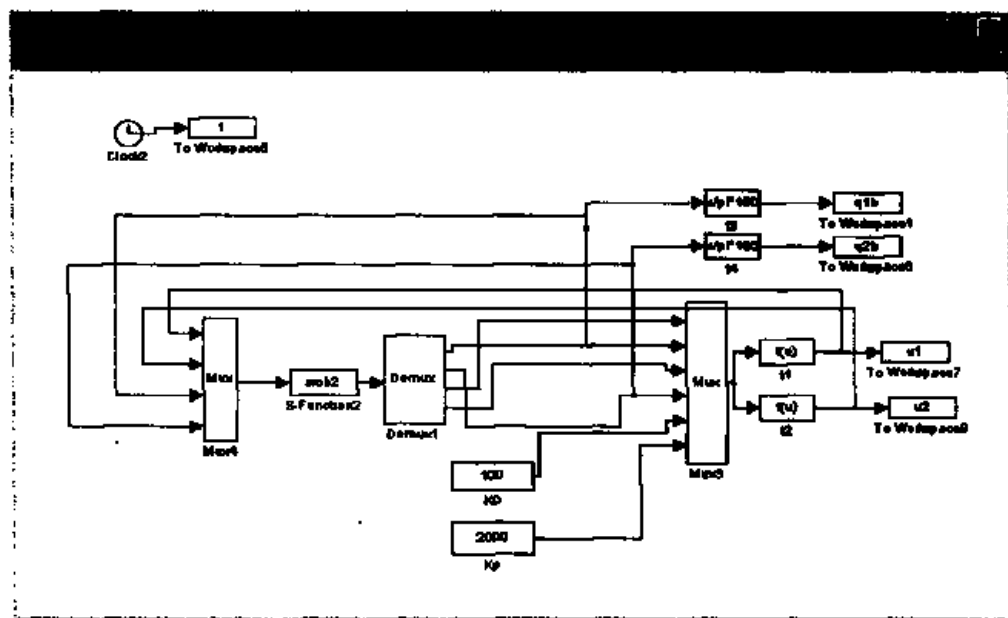


图 3.104 经过 S-function 简化后的功能模块图

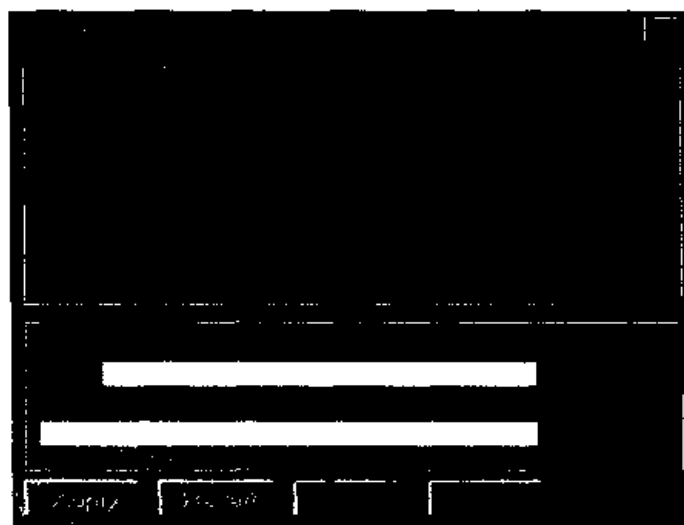


图 3.105 S-function 功能模块

S-function部分说明如下:

- 在mdlDerivatives的function中, 加入参数的运算, 并将SIMULINK模型Pd1.mdl(如图3.100所示)左半部输入参数功能模块合并进来运算。
- 在mdlInitializeSizes部分, 修改4个输入:  
u(1)、u(2)为控制输入; u(3)、u(4)为q1角度与q2角度的误差量。  
并修改4个输出:  
x(1)、x(2)为q1角度与q2角度的误差量; x(3)、x(4)为q1角度与q2角度的微分量。  
再加入初始条件x0的值:  
 $x0 = [-60/180 \cdot \pi; -90/180 \cdot \pi; 0; 0];$
- 在mdlOutputs部分, 直接以x状态为输出。S-function的输出一定为sys, 所以直接以 $sys = x$ 来表示。其他没有用到的部分, 设定 $sys = []$ 。

S-function程序: srob2.m

```
function [sys,x0,str,ts] = srob2(t,x,u,flag)
    switch flag,
    case 0,
        [sys,x0,str,ts] = mdlInitializeSizes;
    case 1,
        sys=mdlDerivatives(t,x,u);
    case 2,
        sys=mdlUpdate(t,x,u);
    case 3,
        sys=mdlOutputs(t,x,u);
    case 4,
        sys=mdlGetTimeOfNextVarHit(t,x,u);
    case 9,
        sys=mdlTerminate(t,x,u);
    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 = 4;
        sizes.DirFeedthrough = 0;
        sizes.NumSampleTimes = 1; % at least one sample time is needed
        sys = simsizes(sizes);
        x0 = [-60/180*pi; -90/180*pi; 0; 0]; —— 设定初始条件
        str = [];
        ts = [0 0];
```

```

function sys=mdlDerivatives(t,x,u)
q1b=u(3);%/180*pi;
q2b=u(4);%/180*pi;
q1d=60/180*pi;
q2d=90/180*pi;
q1=q1b+q1d;
q2=q2b+q2d;
m1=1;
l1=1;
me=2;
deltae=30/180*pi;
i1=0.12;
lc1=0.5;
ie=0.25;
lce=0.6;
a1=i1+m1*lc1^2+ie+me*lce^2+me*l1^2;
a2=ie+me*lce^2;
a3=me*l1*lce*cos(deltae);
a4=me*l1*lce*sin(deltae);
h11=a1+2*a3*cos(q2)+2*a4*sin(q2);
h12=a2+a3*cos(q2)+a4*sin(q2);
h21=h12;
h22=a2;
h=a3*sin(q2)-a4*cos(q2);
a= [h11 h12;h21 h22] ;
inva=inv(a);
inva1=inva(1,:);
inva2=inva(2,:);
x(1)=q1b;
x(2)=q2b;
sys(1)=x(3);
sys(2)=x(4);
sys(3)=-inva1* [-h.*x(4) -h.*(x(3)+x(4));h.*x(3) 0] * [x(3);x(4)] +inva1* [u(1); u(2)] ;
sys(4)=-inva2* [-h.*x(4) -h.*(x(3)+x(4));h.*x(3) 0] * [x(3);x(4)] +inva2* [u(1); u(2)] ;
function sys=mdlOutputs(t,x,u)
sys=x;
function sys=mdlUpdate(t,x,u)
sys = [] ;
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1; %Example, set the next hit to be one second later.

```

合并pd1.mdl输入参数  
的功能方块

```

sys = t + sampleTime;
function sys=mdlTerminate(t,x,u)
sys = [];

```

最后编写主程序，模拟设计的结果。

主程序：ex3103.m

```

clear;
sim('pd2')
plot(t,q1b)
xlabel('time(sec)')
ylabel('Q1 joint angle error(degree)')
figure(2)
plot(t,q2b)
xlabel('time(sec)')
ylabel('Q2 joint angle error(degree)')

```

执行结果与图3.102、图3.103是一样的，但本方法的执行速度显然比范例3.13快很多。用pentium 200MHz的PC执行，范例3.13约花费20秒，而范例3.14则只花费5秒即执行完毕。所以能够简化的功能模块尽量用S-function来写，虽然程序更复杂，但这样可以加快程序的执行速度。

范例3.15 用S-function进行状态空间矩阵的设计。

对于一个单输入单输出(SISO)的稳定线性系统：

$$\begin{aligned}
 \dot{x} &= \begin{bmatrix} -3 & 1 & 1 \\ 5 & -15 & 5 \\ -9 & 3 & 17 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \\
 y &= [1 \ 0 \ 0]x
 \end{aligned}$$

初始状态为：

$$x_0 = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}$$

用最简单的状态反馈法设计封闭循环控制器。

$$u = [1 \ 0 \ 0] x$$

可以使系统输出稳定。

主程序：ex3104.m

```

clear;
sim('stspace')
plot(t,yout)

```

```
xlabel('time(sec)')
ylabel('output')
```

下面为所设计的SIMULINK模型(见图3.106)及S-function程序。在S-function程序中,一开始就定义矩阵a,b,c的大小,所以在后面的function程序中,就必须包括对参数a,b,c的调用。另外不同于前面的写法是没有使用到的case{2,4,9}均设为sys= []。

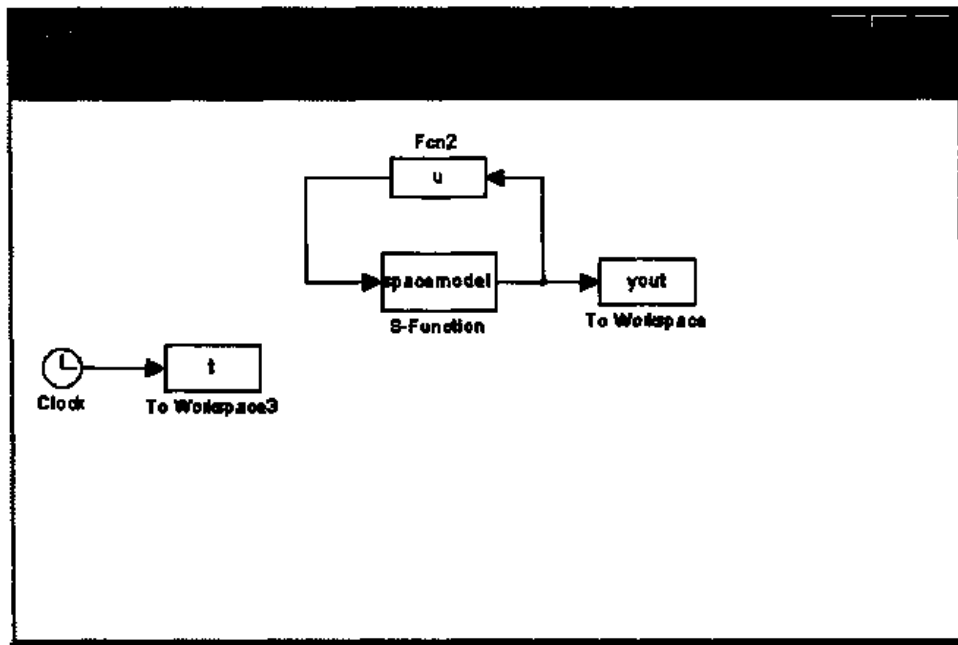


图 3.106 状态空间设计

S-function程序: spacemodel.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
a = [-3 1 1; 5 -15 5; -9 3 -17]; 输入矩阵
b = [0;0;1];
c = [1 0 0];
switch flag,
case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(a,b,c);
case 1,
    sys=mdlDerivatives(t,x,u,a,b,c);
case 3,
    sys=mdlOutputs(t,x,u,a,b,c);
case {2,4,9} 没有使用到的case,sys设为 []
    sys= [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts] = mdlInitializeSizes(a,b,c)
sizes = simsizes;
```

```

sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes(sizes);
x0 = [10;0;0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u,a,b,c)
sys = a*x+b*u(1);
function sys=mdlOutputs(t,x,u,a,b,c)
sys = c*x;

```

要包括参数a,b,c

执行结果如图3.107所示，系统输出可以很快地收到0。

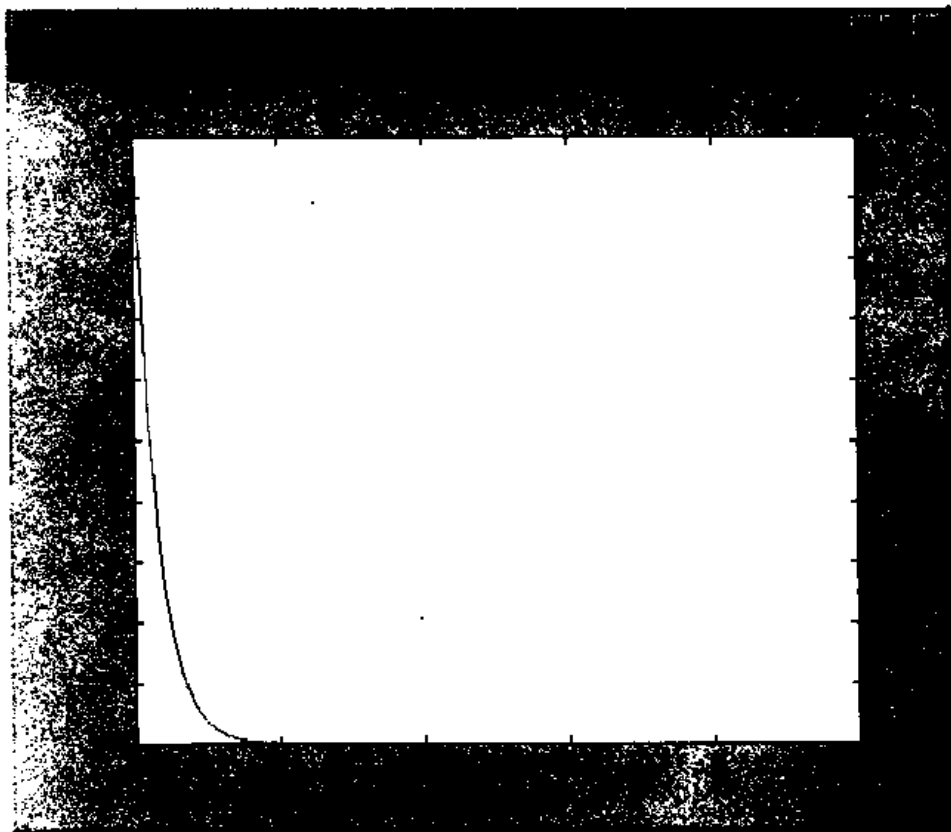


图 3.107 模拟结果

### 3.11 4.x版的S-function设计

在MATLAB 4.x中所用的指令大部分都能在5.x版中执行，即使不能执行也会出现提示信息，请用户修改部分指令，对于S-function的设计方法也不例外。MATLAB 4.x(即

SIMULINK 1.x版)的设计方法在MATLAB 5.x(即SIMULINK2.x版)中一样适用, 虽然写法不同, 但其运行的原理是一样的。SIMULINK 1.x版比较简洁, 但设计思路比较清晰; SIMULINK 2.x版复杂, 但初学者容易学习。所以本节将说明1.x版的S-function设计方法, 除了可以加强对S-function设计原理的了解之外, 也可以使读者多一种选择。以前用4.x版所写的程序在5.x版中一样可以执行, 不过速度会慢4倍左右。

1. 首先说明flag的值所表示的含义:

flag=0定义初始状态。

通常用sys= [n1,n2,n3,n4,n5,n6] 来表示:

- n1: 连续状态的个数。
- n2: 离散状态的个数。
- n3: 输出的个数。
- n4: 输入的个数。
- n5: 非连续根的个数。
- n6: 用于代数回路时, 设定为1。

flag=1求状态微分。

flag=2求离散的下一状态。

flag=3求输出状态。

flag=4求下一时间间隔的更新离散状态值。

2. 最后就是了解其程序的结构, 程序结构如下:

```
function [sys,x0] =程序名称(t,x,u,flag);  
if flag==1;  
    运算微分状态的设计程序  
elseif flag==3;  
    运算输出状态的设计程序  
elseif flag==0;  
    设定sys及x0的值  
else  
    sys= [];  
end;
```

下面将上节的例子改为MATLAB 4.x的写法, 请比较两者在写法上的差异, 以及其执行速度是否比用5.x版所写的程序慢。

程序: sfun\_41.m

```
function [sys,x0] =sfun_41(t,x,u,flag);  
if flag==1;  
    q1b=u(3);%/180*pi;  
    q2b=u(4);%/180*pi;
```



```

q1d=60/180*pi;
q2d=90/180*pi;
q1=q1b+q1d;
q2=q2b+q2d;
m1=1;
l1=1;
me=2;
deltae=30/180*pi;
i1=0.12;
lc1=0.5;
ie=0.25;
lce=0.6;
a1=i1+m1*lc1^2+ie+me*lce^2+me*l1^2;
a2=ie+me*lce^2;
a3=me*l1*lce*cos(deltae);
a4=me*l1*lce*sin(deltae);
h11=a1+2*a3*cos(q2)+2*a4*sin(q2);
h12=a2+a3*cos(q2)+a4*sin(q2);
h21=h12;
h22=a2;
h=a3*sin(q2)-a4*cos(q2);
a=[h11 h12;h21 h22];
inva=inv(a);
inva1=inva(1,:);
inva2=inva(2,:);
x(1)=q1b;
x(2)=q2b;
sys(1)=x(3);
sys(2)=x(4);
sys(3)=-inva1*[-h.*x(4)-h.*(x(3)+x(4));h.*x(3) 0]*[x(3);x(4)]+inva1*[u(1);u(2)];
sys(4)=-inva2*[-h.*x(4)-h.*(x(3)+x(4));h.*x(3) 0]*[x(3);x(4)]+inva2*[u(1);u(2)];
elseif flag==3;
    sys=x;
elseif flag==0;
    sys=[4,0,4,4,0,1];
    x0=[-60/180*pi;-90/180*pi;0;0];
else
    sys=[];
end;

```

模拟时, 将图3.104与图3.105中的S-function功能模块的S-function name改为sfun\_41(见如图3.108与图3.109), 再执行ex3102→Enter即可。执行结果与图3.102、图3.103相同。

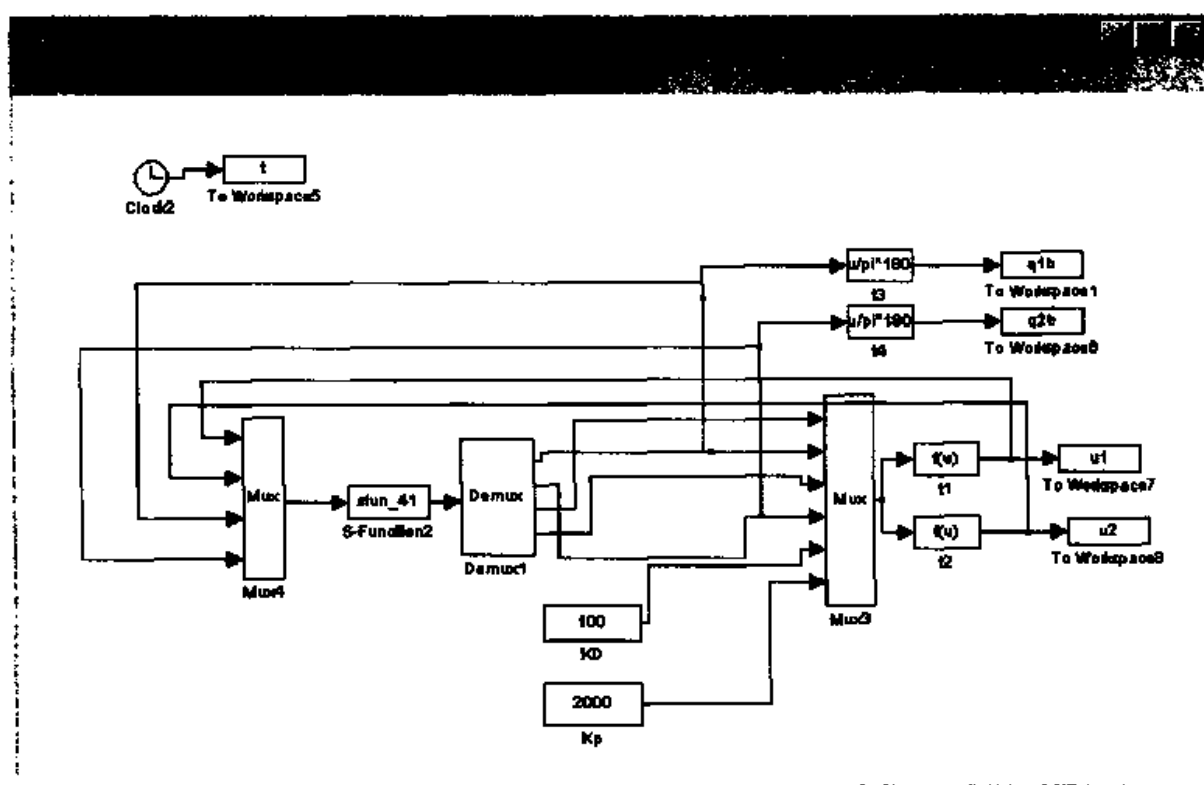


图 3.108 以 sfun\_41 作为 S-function

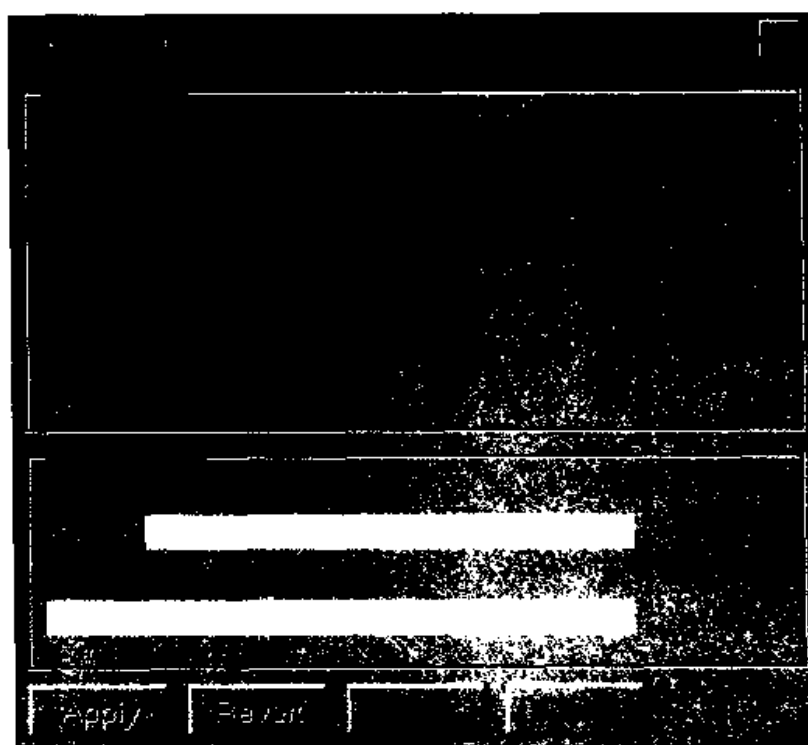


图 3.109 输入 sfun\_41

## 第4章 Fuzzy Toolbox 设计

本章先介绍Fuzzy的基本概念，然后说明Fuzzy Toolbox的应用方法，最后说明ANFIS的概念及设计方法。

### 4.1 Fuzzy概述

在现实生活中，人们碰到的问题大多具有不确定性，不同的人对同一问题的看法往往不尽相同。我们常常会说“大概”、“约”之类的话，这些不能用明确的数字或模型表达的事物，必须借助一种表示程度的语法来形容，因此就产生了模糊理论(fuzzy theorem)，以便将事物的属性或性质用模糊形态或模糊群组来表现和处理。

模糊理论的创始人是美国柏克莱大学的Zadeh教授，他提出的fuzzy set观念最早应用于蒸汽涡轮引擎。这种设计就是将人类对烧煤炭的经验认知过程，转化为模糊的判断规则，并根据某种语法的数学函数，将其进行量化，最后就可以推论出控制量的大小。

fuzzy理论经过近30年的发展，目前已广泛应用于人类生活的方方面面。除了主流的模糊控制利用fuzzy设计控制器(如焊接机器人、自动炼钢、家电用品、自动汽车等)，还应用到管理学、社会学、医学、运输管理学、生态学、环保预测、地震研究、谈判等方面。

当面对一个复杂的非线性、随时间变化且难以精确测量的系统时，利用fuzzy理论来设计控制器是很好的做法。因为需要解决的系统属性正是fuzzy能解决的。

模糊逻辑控制器(Fuzzy Logic Controller)简称FLC，其基本结构如图4.1所示。它包括5个部分：

- rule base包含许多fuzzy if-then规则。
- database定义隶属函数(Membership Function)的形式与范围。
- decision-making unit执行模糊规则的推论(inference)。
- fuzzification interface将明确的输入(crisp inputs)转换为对应隶属函数的模糊语言值(linguistic values)。
- defuzzification interface将模糊的计算结果转换为明确的输出(crisp output)。

模糊规则的形式为IF A is a THEN B is b, A与B是语言变量(linguistic variables)，而a和b是隶属函数映射到的语言值(linguistic values)。这些模糊规则都可以表示成模糊伴随记忆(Fuzzy Associated Memory)，简称FAM。

在模糊规则上进行模糊推论的流程示意图，如图4.2所示。其步骤为：

1. 每个输入变量在前提中(premise part)对应隶属函数可以得到一个对语言标签(linguistic label)而言的程度值。
2. 通过T-norm过程(通常取min)，对每一个模糊规则在前提得到触发强度(firing strength)，即权重(weight)。

3. 根据每个模糊规则的权重, 产生每个模糊规则的推论(consequent)。
4. 通过推论的解模糊化, 产生明确的输出。

解模糊通常使用中心法(center of area)来实现, 另外也有用加权平均法(weighted average)的。这两种方法各有特点, 应根据要求采用不同的方法。中心法使用max的方法, 将推论求和后算出中心, 其特点是稳态效果较好。而加权平均法则是将输入变量线性组合(linear combination)后, 乘以各个模糊规则推论的权重再平均, 其特点是适合做网络调适与训练, 属于Sugeno形式的模糊推论系统, 在4.6节中涉及到的适应性网络模糊推论系统(ANFIS)就是这一类型。

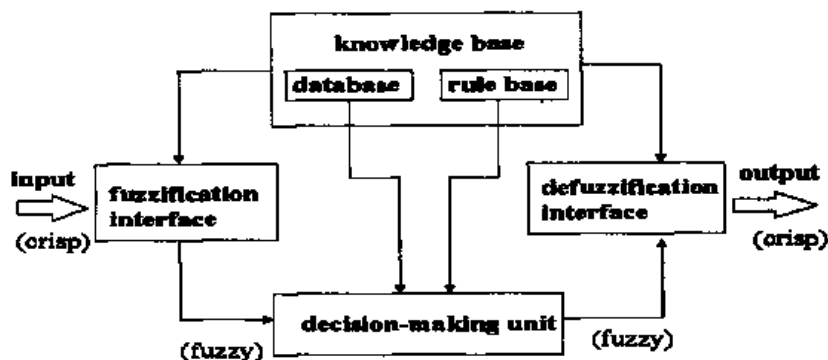


图 4.1 模糊逻辑控制器的结构

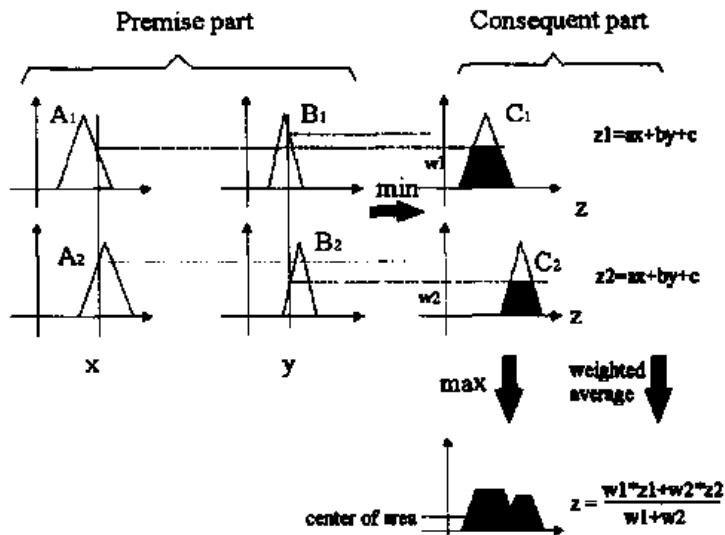


图 4.2 模糊推论示意图

## 4.2 建立Fuzzy推论系统

第一步：在命令窗口中执行

fuzzy→enter

就会出现fuzzy的编辑画面，如图4.3所示。

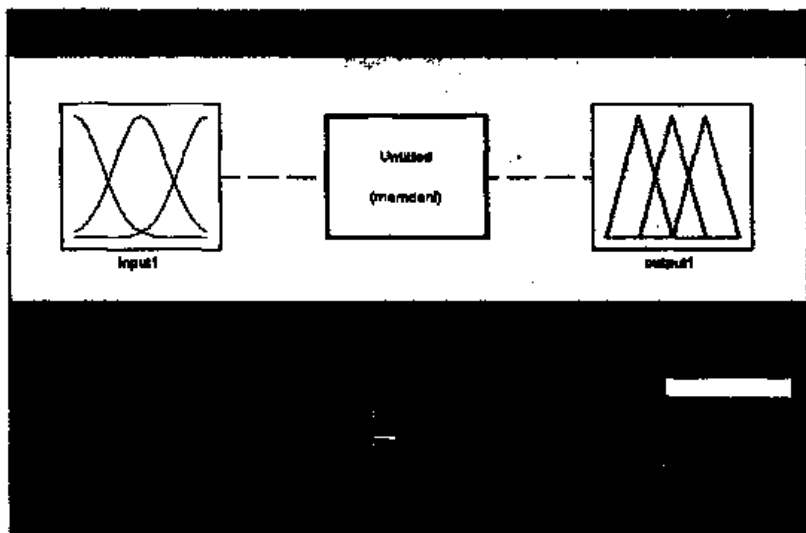


图 4.3 fuzzy toolbox 的初始窗口

第二步：接着执行Edit→Add Input以便增加输入项，如图4.4所示。

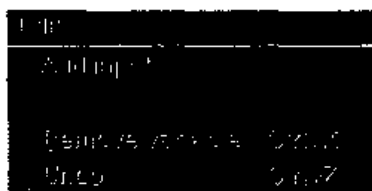


图 4.4 增加输入项的菜单

第三步：双击input1，即可对输入项1的隶属函数(membership function)进行编辑，如图4.5所示。其他项的隶属函数也用同样的方法进行处理。

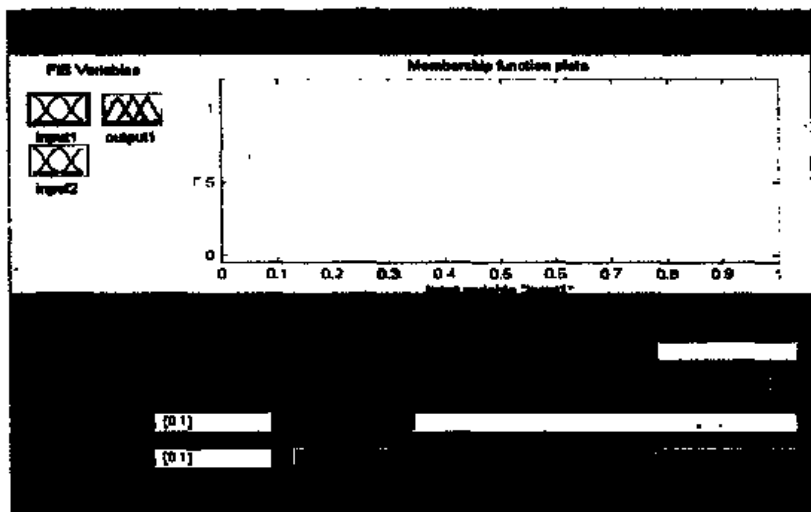


图 4.5 隶属函数的编辑窗口

第四步：进入隶属函数的编辑窗口后，执行Edit→Add MFs定义其个数，如图4.6及图4.7所示。在图4.6中输入‘3’，则图4.7中出现3个MF。

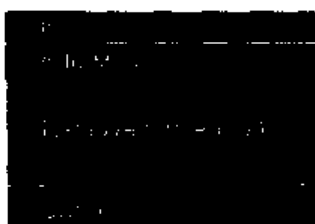


图 4.6 增加隶属函数的菜单

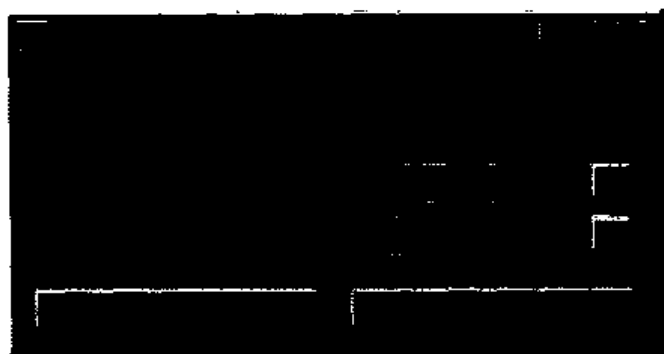


图 4.7 增加三个隶属函数

第五步：针对每个隶属函数(选中的函数用红色的粗线表示)，可以设定其名称、范围、形状等属性，如图4.8所示。在Range项内可以看到  $[0 \ 1]$ ，它代表坐标轴的范围；在Name项内可以看到mf1，它代表这个隶属函数的名称；在Type项内可以选择多种不同的隶属函数形状，trimf代表三角形的隶属函数；Params为  $[-0.5 \ 0 \ 0.5]$ ，代表该隶属函数转角的X轴坐标。

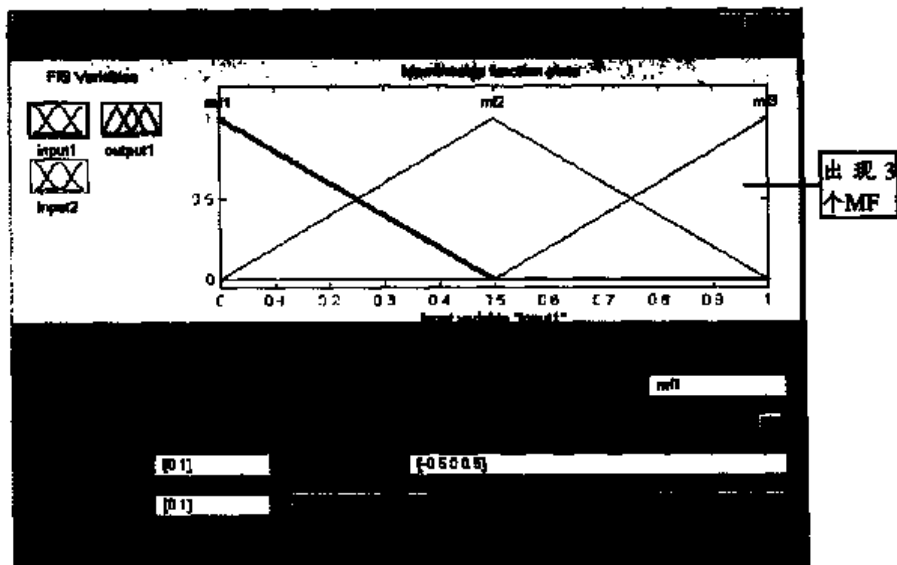


图 4.8 定义隶属函数的窗口

第六步：设定好隶属函数之后，单击close回到图4.3所示的主窗体。在第二步时增加了一个input，所以input变成两个了，如图4.9所示。

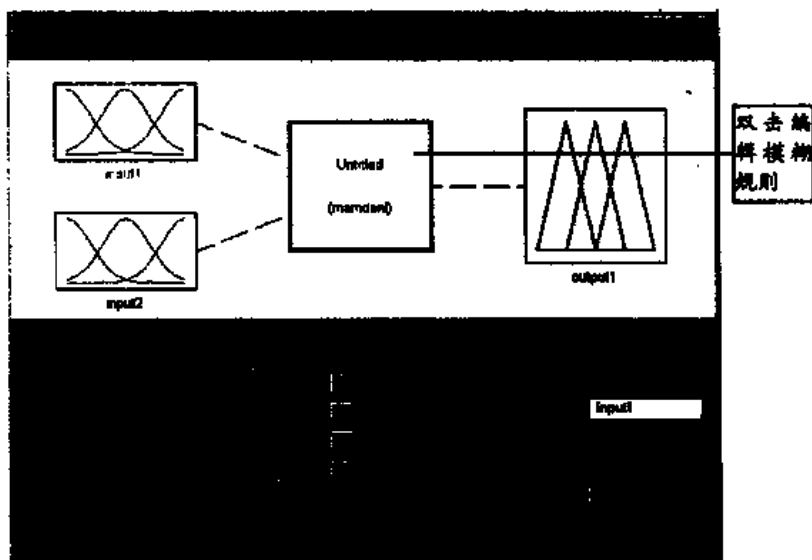


图 4.9 两输入一输出的主窗口

第七步：双击中间的模块，编辑模糊规则。因为只是举例说明，所以只输入了三条规则，实际应用中可以再增加，如图4.10所示。在Rule Format中有三项可供选择，分别为verbose, symbolic, indexed，读者可以选择一种比较喜欢的方式。下面就是这三种不同方法的显示形式：

- verbose法：

1. If (input1 is mf1) and (input2 is mf2) then (output1 is mf1)(1)
2. If(input1 is mf2) and (input2 is mf3) then (output1 is mf2)(1)
3. If (input1 is mf3) and (input2 is mf1) then (output1 is mf3)(1)

- symbolic法：

1. (input1==mf1) & (input2==mf2)=>(output1==mf1) (1)
2. (input1==mf2) & (input2==mf3)=>(output1==mf2) (1)
3. (input1==mf3) & (input2==mf1)=>(output1==mf3) (1)

- indexed法：

- 1 2,1 (1):1
- 2 3,2 (1):1
- 3 1,3 (1):1

建议使用indexed法，因为只要输入数字即可。其格式是用逗号分隔的输入与输出的隶属函数，数字代表隶属函数从左算起的次序，后面连接(1):1即完成一个规则的建立。另外两种方法的规则必须一一建立，可以先建立一条规则，再用鼠标选中它，单击鼠标右键，选取“复制”，再到要复制的区域单击鼠标右键选择“粘贴”，即可复制多条规则，最后再到每条规则内修改数字即可。

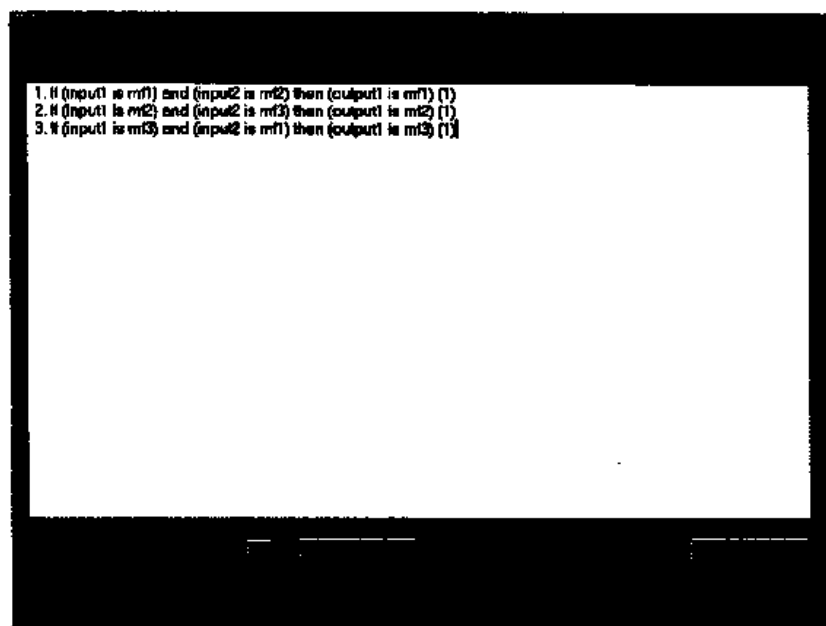


图 4.10 模糊规则编辑器

第八步：保存文件，在这里文件名为fzy1，文件类型固定是.FIS。

### 4.3 Fuzzy推论系统的调试

对于建立的模糊推论系统，必须能够验证其功能是否与期望一致，因此必须进行调试(Debug)工作。首先打开刚才建立的fzy1.fis，然后在View菜单下执行View rules，观察规则的推论是否正确；或者执行View surface，观察输入与输出的关系是否正确，如图4.11所示。

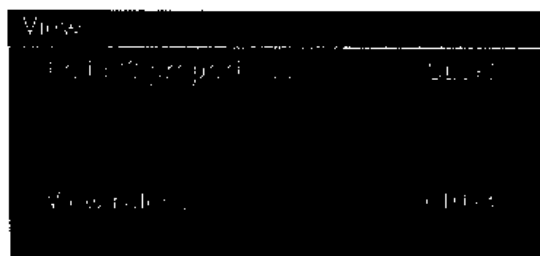


图 4.11 规则调试的菜单

如果选择View rules，就会出现可以调整输入值并即时观察输出变化的窗口，如图4.12所示。调整输入值的方法有两种，一种是用鼠标直接拖曳红线到指定位置，另一种是在input的输入框中输入数字，如果输入[0.1 0.8]，如图4.13所示，可以发现推论出来的输出结果为0.392。

如果选择View surface，就会出现输出结果的三维立体图，如图4.14所示。用鼠标可以直接在上面改变视角。



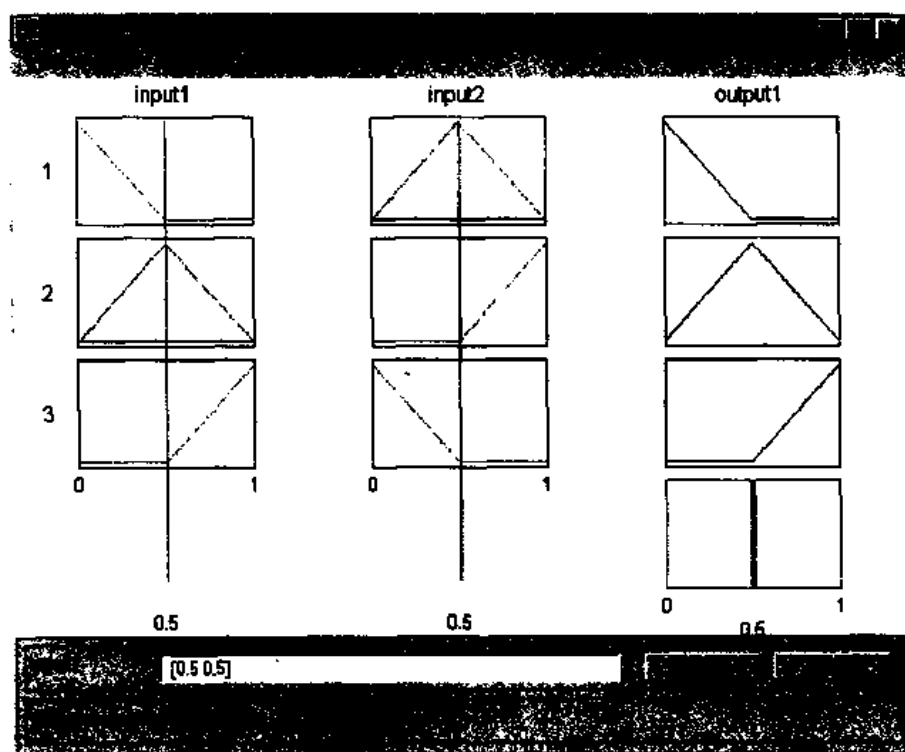


图 4.12 规则调试窗口

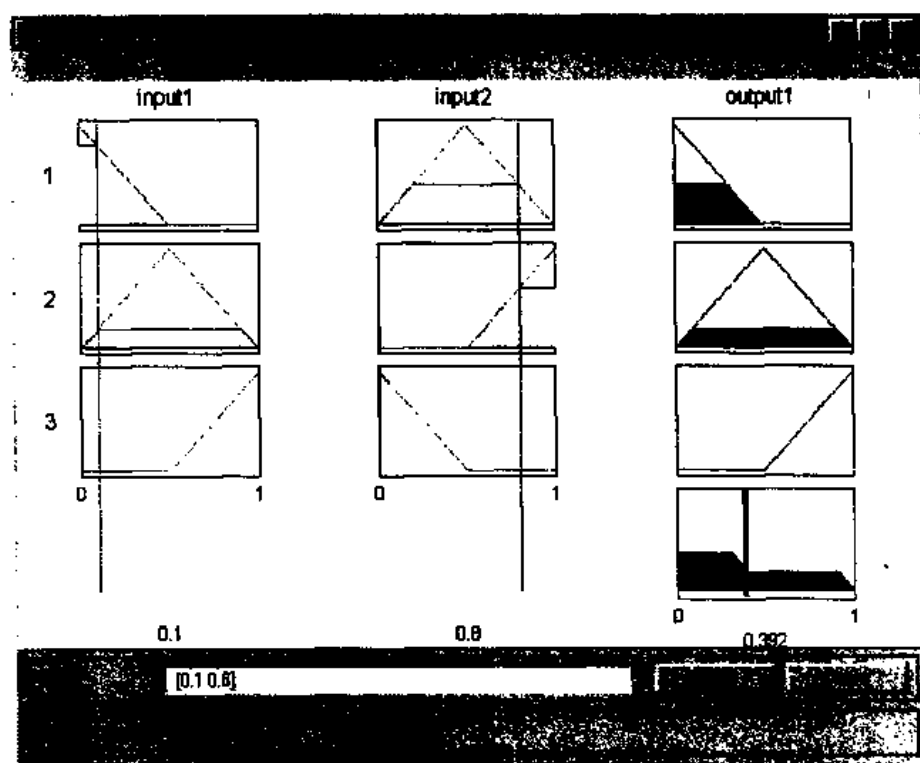


图 4.13 设定输入值得到推论的输出值

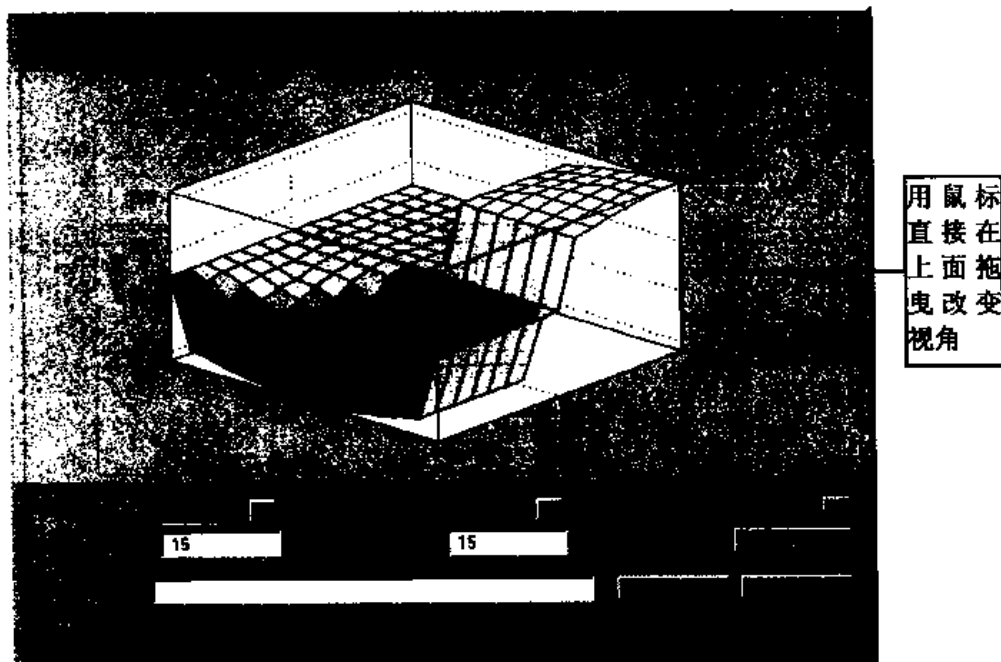


图 4.14 推论后的三维空间图

#### 4.4 Fuzzy推论系统的模拟

建立好隶属函数以及规则后，便完成了一个fuzzy推论系统，接着就是要进行模拟。模拟是通过SIMULINK来进行的，首先用readfis指令(指令格式参见4.5节)将fuzzy推论系统读到MATLAB的workspace里，然后再到SIMULINK的Blocksets &Toolboxes模块库的SIMULINK fuzzy模块库中将Fuzzy Logic Controller功能模块复制进来，并输入读到的.fis文件的文件名。下面的范例说明应用的方法。

**范例4.1** 用fuzzy推论系统执行XOR功能。

fuzzy文件: fzy2.fis

**第一步：**首先采用上一节的方法，进入fuzzy toolbox后建立一个两输入一输出的fuzzy结构，如图4.15所示。为了方便说明，只建立两种输入及输出的隶属函数，即high及low，如图4.16所示。接着配合XOR门的逻辑关系建立模糊规则，XOR的逻辑关系是当输入均为high或low时，输出为0，否则为1，如表4.1所示。建立的模糊规则为：

- If(input1 is low)and (input2 is low)then(output1 is low)
- If(input1 is low)and (input2 is high)then(output1 is high)
- If(input1 is high)and(input2 is low)then (output1 is high)
- If(input1 is high)and(input2 is high)then(output1 is low)

如图4.17所示。接着将其保存，文件名取为fzy2.FIS。

按4.3节的方法，经过模糊推论验证，与XOR门功能相同，如图4.18所示。

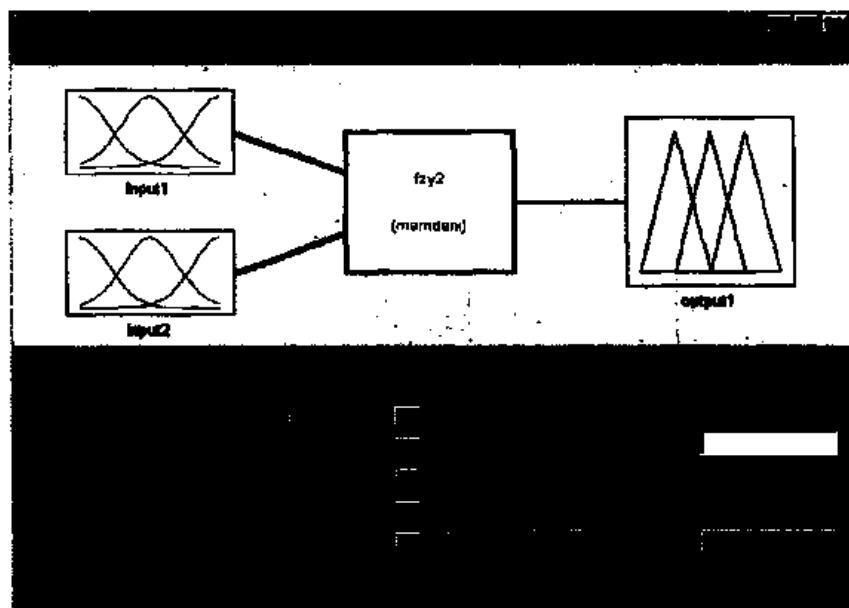


图 4.15 XOR gate 模糊推论系统结构图

表4.1 XOR gate I/O关系图

input1	input2	output
0	0	0
0	1	1
1	0	1
1	1	0

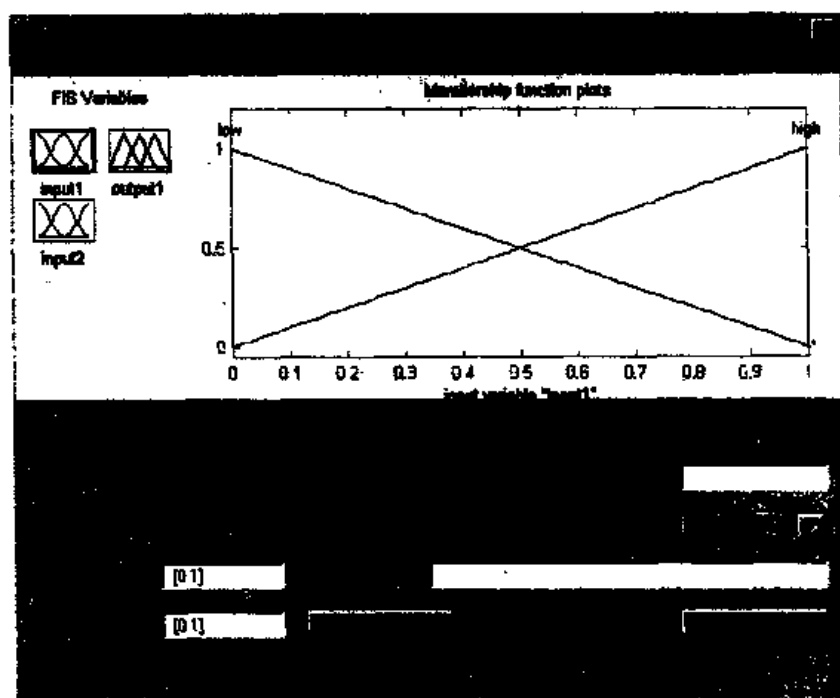


图 4.16 输入的隶属函数

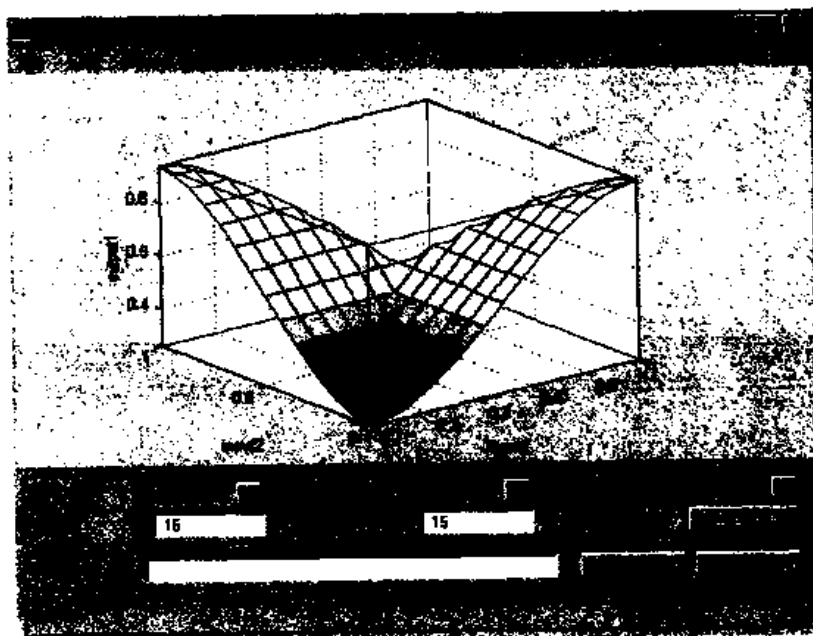


图 4.17 模糊规则编辑图

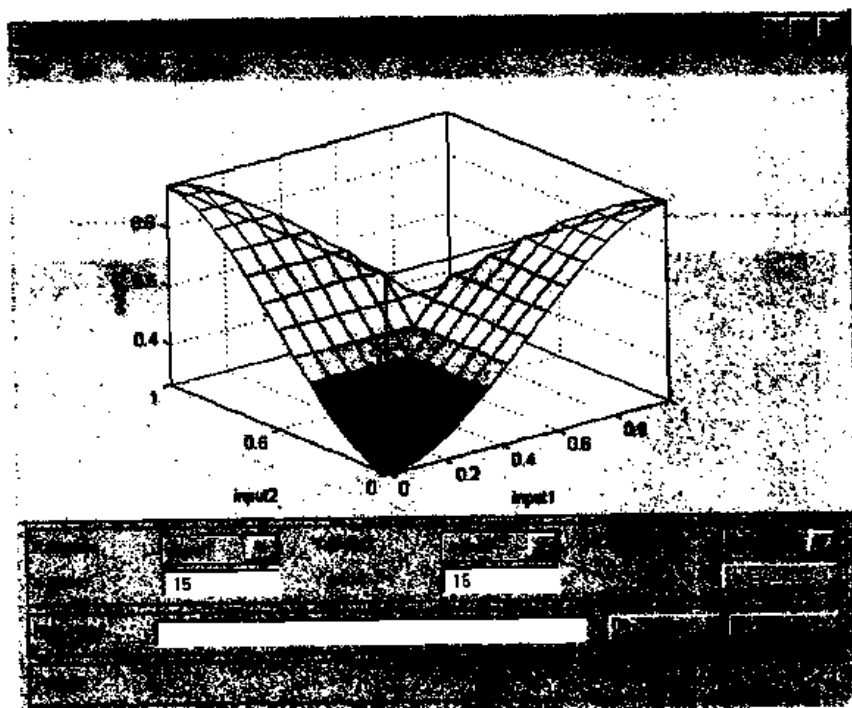


图 4.18 推论结果

SIMULINK文件: fzy2.mdl

第二步: 利用readfis指令把fuzzy推论系统的.fis文件转换成模糊推论矩阵:

```
fzy2_mat=readfis('fzy2')
```

以备功能模块调用。接着到SIMULINK的Blocksets&Toolboxes模块库中的

SIMULINK fuzzy中将fuzzy logic controller功能模块复制进来，并在该功能模块中输入模糊推论矩阵的名称fzy2\_mat，如图4.19所示。完成的SIMULINK模型如图4.20所示。在fuzzy logic controller的前端必须接mux，u(1)代表input1，u(2)代表input2。

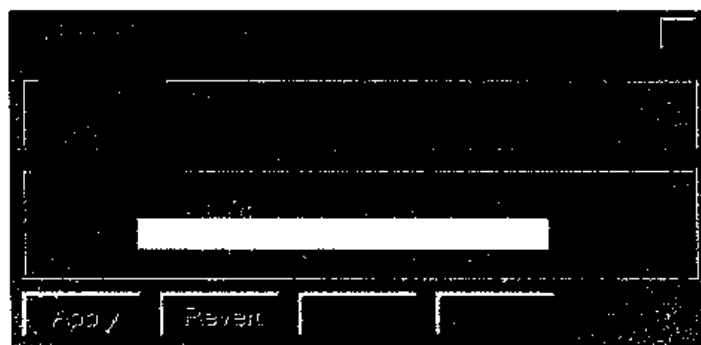


图 4.19 输入模糊推论矩阵

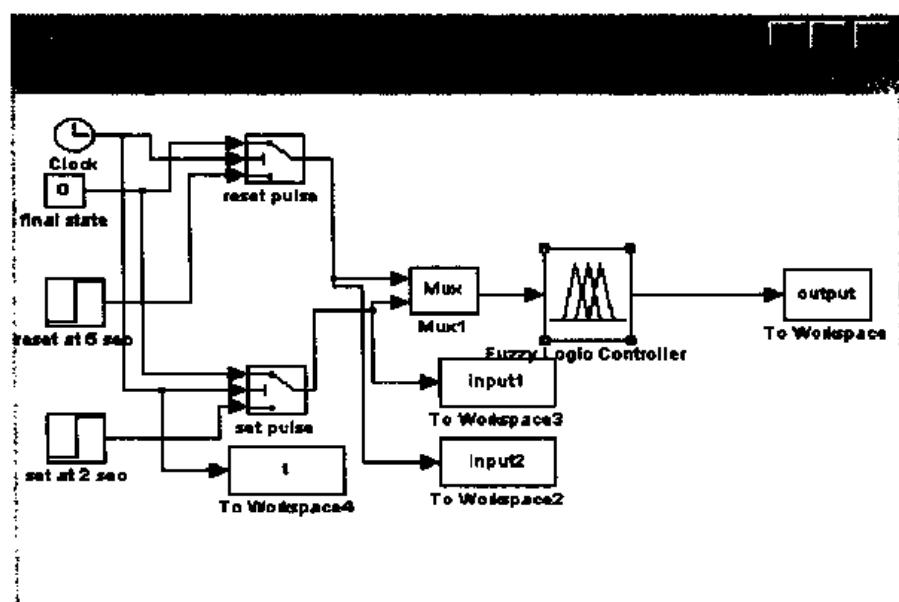


图 4.20 SIMULINK 设计图

**第三步：**编写主程序读入fuzzy文件并模拟SIMULINK，最后再画出结果，如图4.21所示。可以发现输出的结果并不是0或1，而是介于两者中间的值，这就是模糊推论的属性。程序如下：

程序：ex441.m

```
clear;
fzy2_mat=readfis('fzy2')
sim('fzy2');
figure(1)
```

```
subplot(311);  
plot(t,input1);  
xlabel('time(sec)')  
ylabel('input1');  
axis( [0 10 0 2] );  
grid on  
subplot(312);  
plot(t,input2);  
xlabel('time(sec)')  
ylabel('input2')  
axis( [0 10 0 2] );  
grid on  
subplot(313);  
plot(t,output);  
xlabel('time(sec)')  
ylabel('output')  
axis( [0 10 0 2] );  
grid on
```

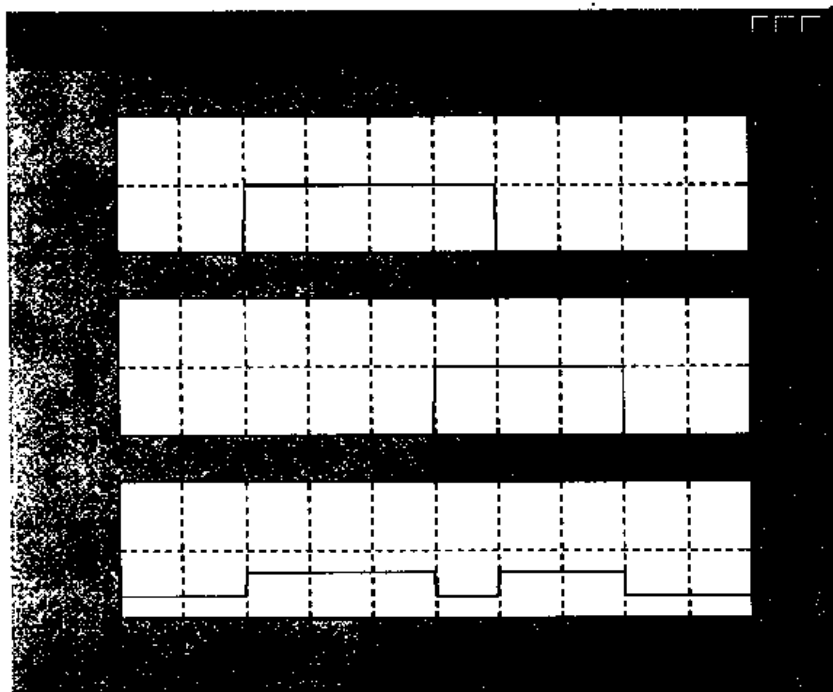


图 4.21 模拟结果图

#### 范例4.2

1. 设计fuzzy controller控制汽车由静止启动，追赶200m外时速90Km的汽车并与其保持30m的距离。
2. 前车速度改为时速110Km时，仍与其保持30m距离。

### 3. 前车速度改为时速70Km时，仍与其保持30m距离。

本题是典型的模糊控制应用，其设计的最基本理论就是通过模糊规则与隶属函数的关系将系统误差的相平面(phase plane)属性轨迹逼近原点，以达到控制稳态误差为0的目的。以下是设计的步骤：

#### 第一步：建立模型。

设计结构如图4.22所示。根据题意可知，目标车的速度曲线图如图4.24所示，积分后得目标车的行进距离a1。设计的目的是要用fuzzy controller控制本车的速度y，使两者的相对距离e能保持30m。

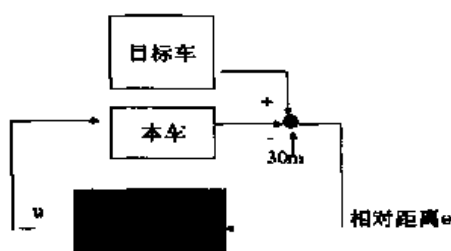


图 4.22 设计结构图

模拟的时候假设两车均为理想状态，即  $\frac{Y(s)}{U(s)} = \frac{4}{s^2 + 2 * 0.7 * 2s + 4}$ ，Y为速度，U为油门控制输入。由于希望与前车保持30m，所以将两车的距离相减后再减掉30，用fuzzy控制使其趋近于0。又因为实际的汽车速度存在极限，所以在simulink中建立的模型中加入saturation block进行模拟。整体的控制设计如图4.23所示，文件为car1.mdl。

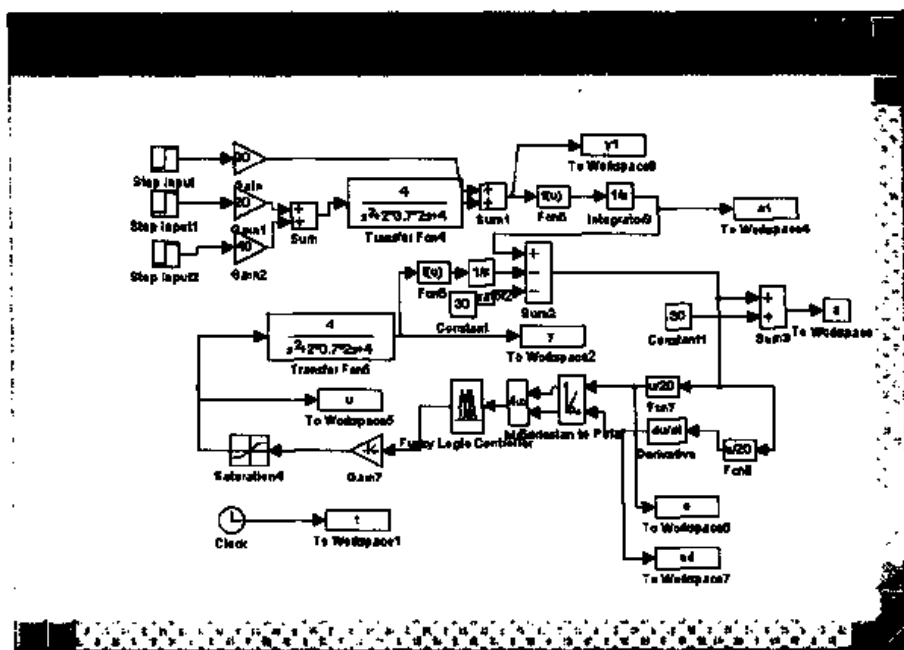


图 4.23 simulink 设计图

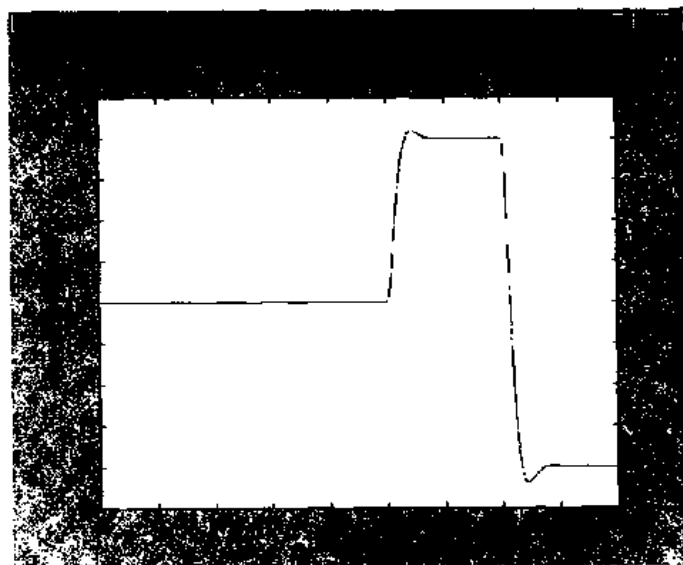


图 4.24 目标车的速度曲线图

第二步: fuzzy controller 规则的设计: 文件为carf2.fis。

首先, 将误差 $e$ 以及误差微分 $\dot{e}$ 标准化除以20, 以便将输出范围限定在10左右。接着将 $e$ 以及 $\dot{e}$ 转换成极坐标 $r$ 与 $\theta$ 并输入, 这样可以将所有规则库填满, 使受控对象不至于因为无效条件而失控。把误差量与误差对时间的变化量当作输入进行模糊规则设计, 得到的控制量为全输出, 属于位置型的fuzzy控制, 也就是模拟的传统PD控制。如果将得到的控制量当作输出的差量, 则为速度型的fuzzy控制, 也就是模拟的传统PI控制。本例采用位置型的Fuzzy控制器进行设计, 转换成极坐标后如图4.25所示, 所以输入项变为 $r$ 与 $\theta$ , 输出项为 $u$ 。

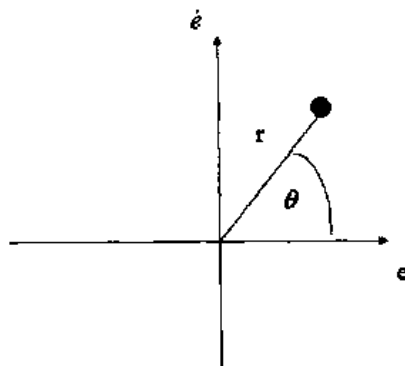


图 4.25 极坐标图

表4.2的FAM(Fuzzy associated memory)转换后, 也得到表4.3的填满且数目变少的FAM控制准则。根据FAM,  $r$ 分成三等分,  $\theta$ 分成五等分, 因此 $3 \times 5 = 15$ 共15条规则。每一条规则均说明, 当轨迹到达该位置时, 控制量应该是多少, 而输出的控制量分为五等分, 至于



应该取哪一份,要看如何将轨迹逼近原点而定。举例来说,当 $r$ 很大为PB(Positive Big),  $\theta$ 为NB(Negative Big)时,代表 $e$ 为负,即系统目前轨迹在第三或第四象限,斜率为负,且离原点很远,所以控制量用NB去拉它。反之当 $r$ 为PB或PM(Positive Medium),  $\theta$ 为PM或ZE(Zero)时,代表 $e$ 为正,即系统目前轨迹在第一或第二象限,斜率为正,且离原点仍很远,所以控制量用PB或PM去推它。只有在 $r$ 为ZE时,不管 $\theta$ 为多少,都表示系统轨迹慢慢接近原点了,因此,控制量不推也不拉,用ZE即可。这类控制规则在很多文献中都可以看到,同时也是最常用且最实用的方法,只要按照上述的判断原则,就不难掌握规则设计的精髓。

表 4.2 FAM

$e \setminus e$	NB	NM	ZE	PM	PB
PB			PB		
PM			PM		
ZE	NB	NM	ZE	PM	PB
NM			NM		
NB			NM		

表 4.3 极坐标的 FAM

$\theta \setminus r$	NB	ZE	PB
PB	ZE	NM	NB
PM	ZE	PM	PB
ZE	ZE	PM	PB
NM	ZE	NM	NB
NB	ZE	NM	NB

rule有15个:

- If( $r$  is pb)and(th is pb)then( $y$  is nb)(1)
- If( $r$  is pb)and(th is pm)then( $y$  is pb)(1)
- If( $r$  is pb)and(th is ze)then( $y$  is pb)(1)
- If( $r$  is pb)and(th is nm) then( $y$  is nb)(1)
- If( $r$  is pb)and(th is nb)then( $y$  is nb)(1)
- If( $r$  is ze)and(th is pb)then( $y$  is nm)(1)
- If( $r$  is ze)and(th is pm)then( $y$  is pm)(1)
- If( $r$  is ze)and(th is ze)then( $y$  is pm)(1)
- If( $r$  is ze)and(th is nm)then( $y$  is nm)(1)
- If( $r$  is ze)and (th is nb)then( $y$  is nm)(1)
- If( $r$  is nb)and(th is pb)then( $y$  is ze) (1)
- If( $r$  is nb)and(th is pm)then( $y$  is ze)(1)
- If( $r$  is nb)and(th is ze)then( $y$  is ze)(1)
- If( $r$  is nb)and(th is nm)then( $y$  is ze)(1)
- If( $r$  is nb)and(th is nb)then( $y$  is ze)(1)

第三步：隶属函数的设计。

为配合第二步规则设计的需要，Fuzzy的输入项及输出项的隶属函数定为：

$r$  分为PB ZE NB

$\theta$  分为PB PM ZE NM NB

$u$  分为PB PM ZE NM NB

以上的隶属函数均取三角函数，其数值范围与大小如图4.26及图4.27所示。

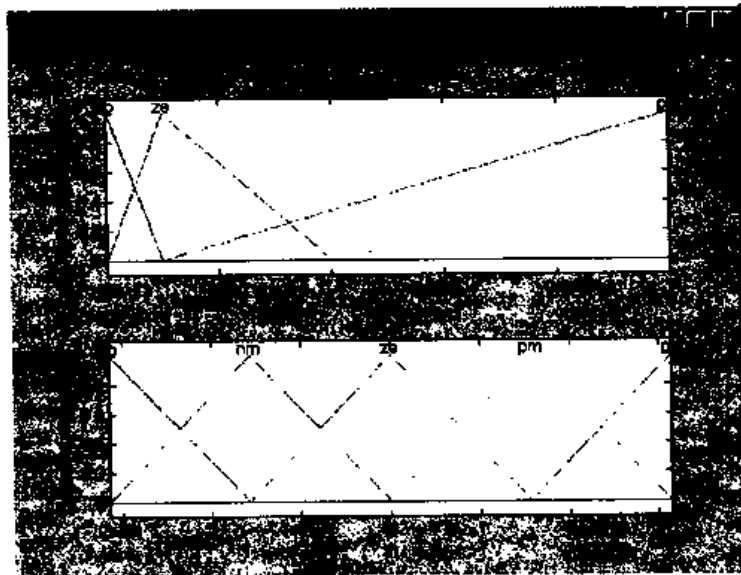


图 4.26 input 的隶属函数

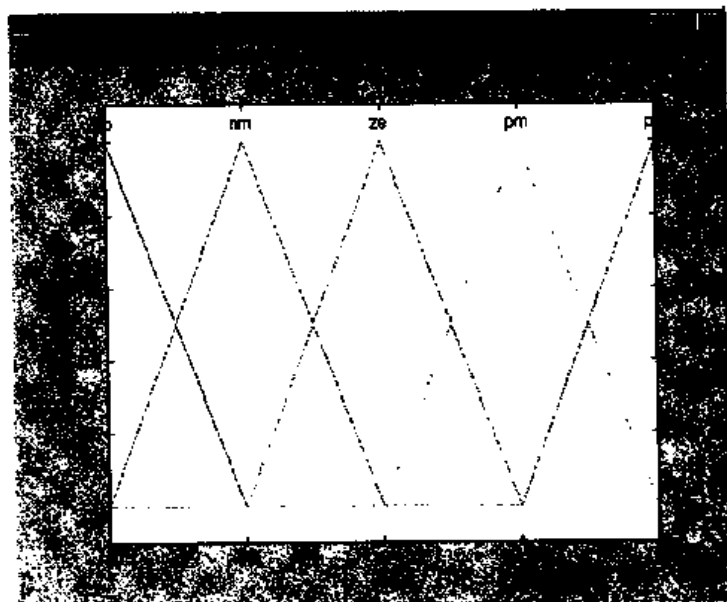


图 4.27 output 的隶属函数

第四步：推论的结果。

由MATLAB Fuzzy toolbox可以看出 $r$ 在0到1之间的变化,  $\theta$ 在 $\pm \pi$ 之间变化时, 推论的结果如图4.28所示。观察其是否与所要的FAM一致, 以决定设计是否完成。可以发现, 图4.28与表4.2对应一致, 所以Fuzzy控制器设计完成。接着进行验证, 如果验证结果可以满足, 则设计到此为止, 如果验证结果不满足, 就要看是哪一部分不符合, 根据该部分去修改模糊规则, 再设计隶属函数, 调整其宽度或形状。这部分对控制的结果有很大的影响, 所以很多人在这方面下功夫进行研究, 有的借助神经网络方法(Neural Network), 有的采用自适应(Adaptive)法, 有兴趣的读者请自行查阅资料。

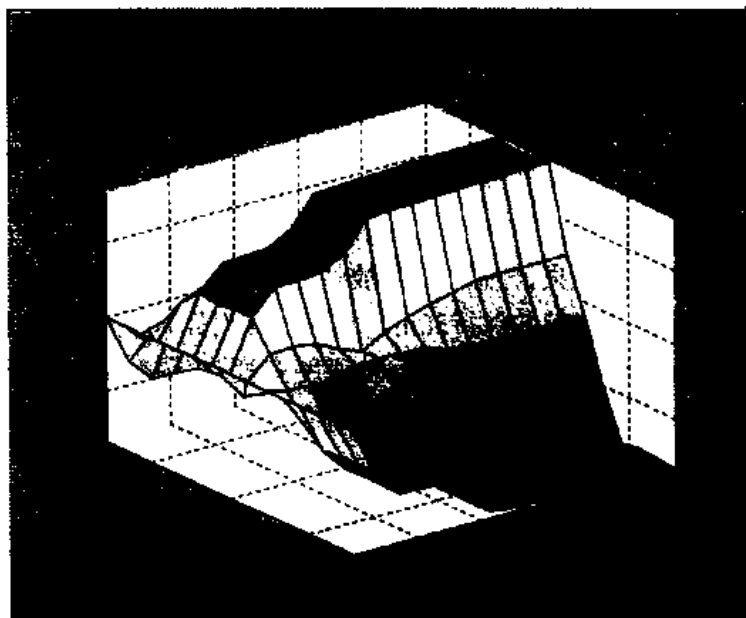


图 4.28 推论结果立体图

设计结果:

程序: ex442.m

```
%Tracking car simulation
clear;
pidf=readfis('carf2');
subplot(211)
plotmf(pidf,'input',1);
subplot(212)
plotmf(pidf,'input',2);
figure
plotmf(pidf,'output',1);
figure
gensurf(pidf)
%simulation by SIMULINK
sim('car1');
```

```
figure
plot(t,y)
xlabel('time(sec) ');
ylabel('velocity(km/hr) ');
figure
plot(t,y1)
xlabel('time(sec) ');
ylabel('target car velocity(km/hr) ');
figure
plot(t,a)
xlabel('time(sec) ');
ylabel('distance(m) ');
figure
plot(t,u)
xlabel('time(sec) ');
ylabel('control input');
```

根据SIMULINK的模拟结果, 油门的踩放结果如图4.29所示, 追赶车的速度如图4.30所示, 在15秒左右即可追上前车并保持30m, 与前车相对距离图如图4.31所示。

由图4.31可知, 虽然目标车在25s及35s时速度有变化(图4.24), 但经fuzzy控制后仍能有效的维持相对距离为30m。由此可见, 用极坐标作为输入的fuzzy controller具有很好的适应性。

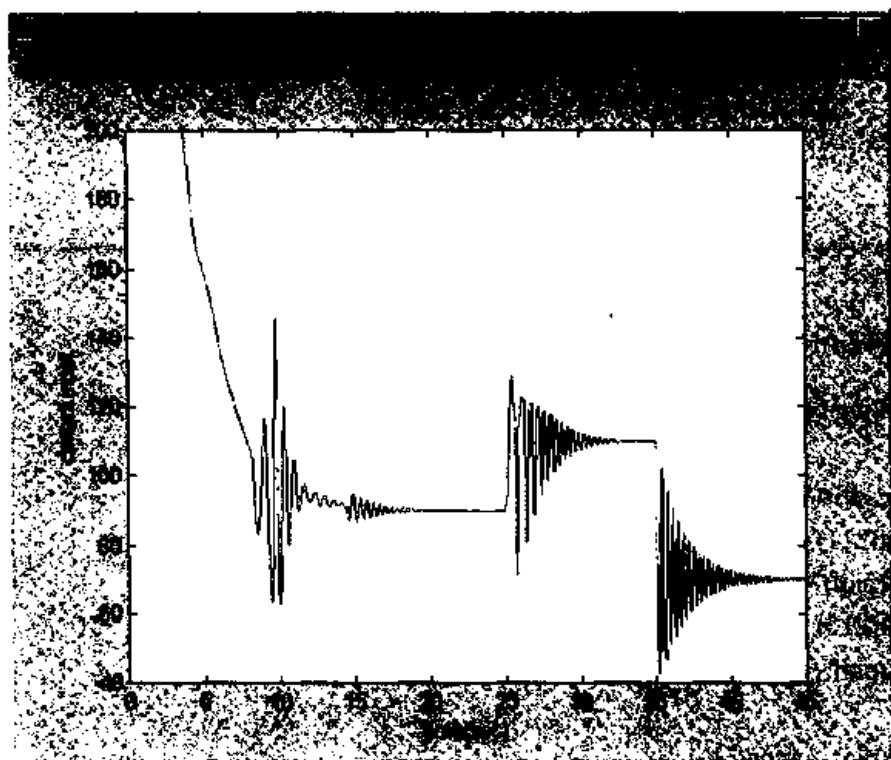


图 4.29 油门踩放图

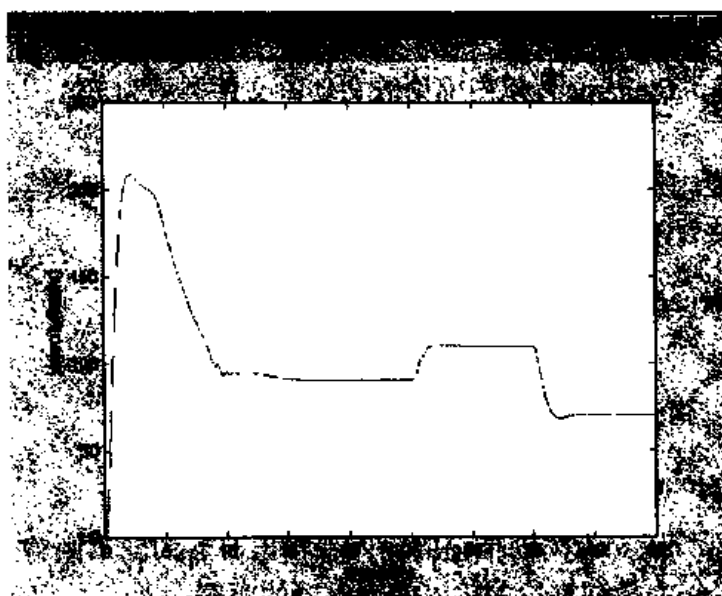


图 4.30 追赶车速度图

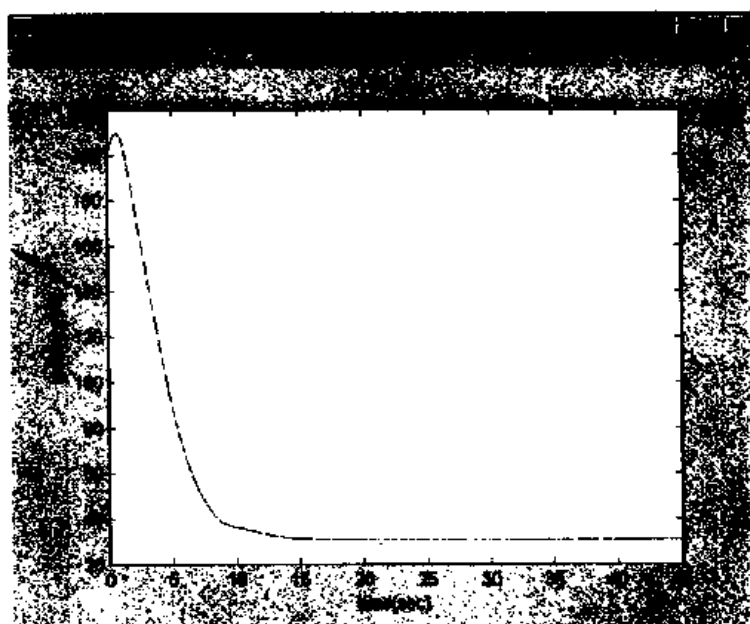


图 4.31 与前车相对距离图

## 4.5 fuzzy常用指令介绍

前面所用的fuzzy推论系统是借助图形界面(GUI)来设计的,但是因为有时候要画图或是希望自己写程序,所以最好能熟练地使用几个重要的指令。本节就来介绍它们,对于其他没提到的指令,读者在用到时可以用help去查询。

### 1. readfis

前面已经用过了,它是应用程序必须最先执行的操作,其格式为:

```
a=readfis('filename')
```

a为推论矩阵, filename为模糊推论的文件, 文件类型为.FIS。

## 2. writefis

与readfis功能相反, 这个指令的目的是将生成的推论矩阵保存到一个文件中, 通常与ANFIS指令配合使用。其指令格式为:

```
writefis(a, 'filename')
```

## 3. plotmf

这个指令可以画出隶属函数, 其指令格式为:

```
plotmf(a, 'mf_io', no)
```

a为推论矩阵, mf\_io为模糊推论的输入或输出, 若为输入则输入input, 若为输出则输入output。no为模糊推论的输入或输出项的顺序。

例如: 对范例4.1来说, 执行

```
plotmf(fzy2_mat, 'input', 1)
```

表示画出fzy2\_mat推论矩阵的第1个输入隶属函数, 得到图4.32所示的结果。

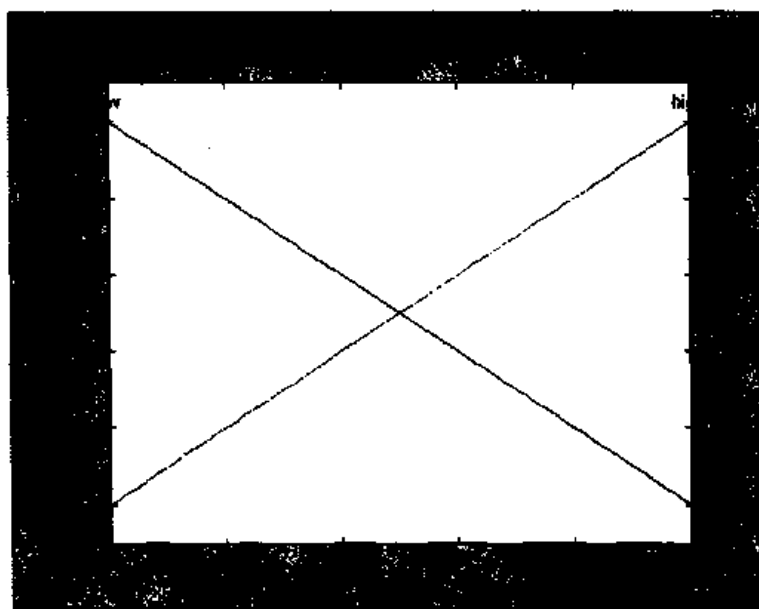


图 4.32 模糊隶属函数

## 4. showrule

这个指令显示模糊规则, 其指令格式为:

```
showrule(a, 范围, '形式')
```

例如:

```
showrule(fzy2_mat, 1:3, verbose)
```

ans=

- If(input1 is low)and(input2 is high)then(output1 is high)(1)
- If(input1 is low)and (input2 is low)then(output1 is low)(1)
- If(input1 is high)and(input2 is low)then(output1 is high)(1)

5. `getfis`获得模糊推论系统的属性，例如执行`getfis(fzy2_mat)`可以得到下列属性值：

```
Name = fzy2
Type = mamdani
NumInputs = 2
InLabels =
    input1
    input2
NumOutputs = 1
OutLabels =
    output1
NumRules = 4
AndMethod = min
OrMethod = max
ImpMethod = min
AggMethod = max
DefuzzMethod = centroid
```

6. `gensurf`可以画出输出对输入的立体图。例如：`gensurf(fzy2_mat)`得到图4.33。

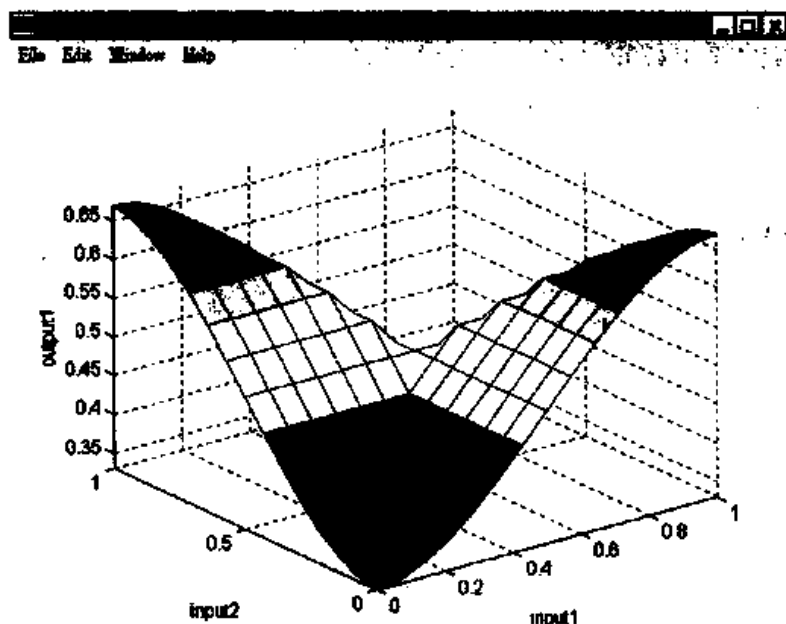


图 4.33 用 `gensurf` 画出的三维图形

7. `genfis1`根据输入的data，隶属函数的个数及形式来决定模糊关系矩阵。其指令格式为：

```
fismat=genfis1(data,numMFs,mfType)
```

8. evalfis通过给定的输入向量及模糊关系矩阵算出输出。其指令格式为:

```
output=evalfis(input,fismat)
```

#### 4.6 ANFIS指令的应用

自适应网络模糊推论系统(Adaptive-Network-Based Fuzzy Inference System)简称ANFIS。ANFIS能够改善传统模糊控制设计中必须靠人的思维一次又一次地调整隶属函数才能达到减小误差、增进效能的缺点。在MATLAB中已经有了这个指令,就叫做anfis,能够通过训练与自适应解决上述的问题。它以复合式的学习过程(hybrid learning procedure)为基础,建立起一套if-then的规则,并慢慢地调配出适当的隶属函数来满足所要的模糊推论输入输出关系。例如一个两输入一输出的网络构成Takagi and Sugeno形式的模糊推论系统,如图4.34所示。其意义为:

Rule 1: If  $x$  is  $A_1$  and  $y$  is  $B_1$ , then  $h_1 = p_1x + q_1y + r_1$ ;

Rule 2: If  $x$  is  $A_2$  and  $y$  is  $B_2$ , then  $h_2 = p_2x + q_2y + r_2$ ;

其中 $A_i, B_i, i=1, 2$ 为语言标签;  $x, y$ 为模糊推论的假设;  $p_i, q_i, r_i, i=1, 2$ 为模糊推论的结论。

使用加权平均法(weighted averaged method)解模糊, 则输出 $h$ 为:

$$h = \frac{w_1}{w_1 + w_2} h_1 + \frac{w_2}{w_1 + w_2} h_2$$

其中 $w_i$ 为第 $i$ 个节点的输出权重。

所以由Takagi and Sugeno形式 $h_i = p_i x + q_i y + r_i$ 可以写成

$$\begin{aligned} h &= \bar{w}_1 h_1 + \bar{w}_2 h_2 \\ &= (\bar{w}_1 x) p_1 + (\bar{w}_1 y) q_1 + (\bar{w}_1) r_1 + (\bar{w}_2 x) p_2 + (\bar{w}_2 y) q_2 + (\bar{w}_2) r_2 \end{aligned}$$

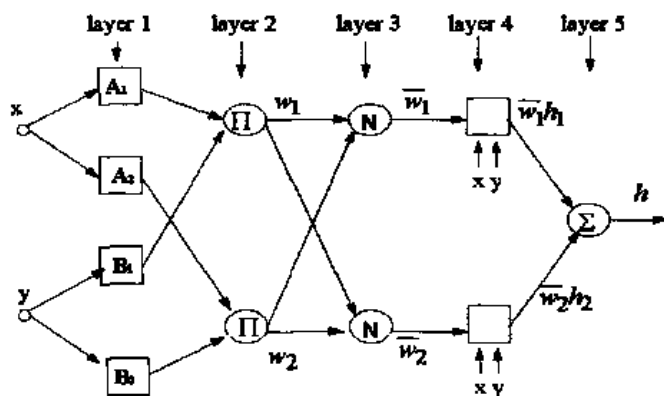


图 4.34 ANFIS 结构图



在复合式的学习过程中, 前向学习(forward pass)到图4.34的第四层, 而推论的参数 $p_i$ ,  $q_i, r_i$ 则由最小二乘估计法(least squares estimate)简称LSE求得, 通过反向学习(backward pass)的最大梯度法(gradient descent approach), 返回误差变化率以更新前提的参数 $x, y$ 。在改变这些参数的过程中, 各种对应 $A_i$ 与 $B_i$ 的适当的钟形隶属函数也就出现了。由于前提与推论的参数已经解耦(decoupled), 且ANFIS又是放射状网络(radial basis-function network), 所以它的学习速度比神经网络快。以上只是概略地描述了一下ANFIS的原理, 有兴趣的读者可以参阅相关资料。

在fuzzy toolbox中提供了运算ANFIS的指令, 只要给定训练的数据、训练次数、验证点等数据, 就可以求出模糊关系矩阵, 并显示训练的相关信息。其指令格式为:

```
[fismat1, trnerr, ss, fismat2, chkerr] = anfis(trndata, fismat, numepochs, NaN, chkdata);
```

上式在实际应用中, 只要设定被训练的成对输入输出数据trndata、先行产生的一组被训练的模糊关系矩阵fismat、训练的次数numepochs以及验证点chkdata等, 即可产生训练完成的模糊关系矩阵fismat1。下面的范例说明如何用ANFIS得到更好的隶属函数, 以便执行XOR的逻辑运算, 从而改进范例4.1。

#### 范例4.3 用ANFIS设计XOR门的模糊控制器。

根据4.4节的说明, 了解XOR门的运算关系后, 编写ANFIS的训练文件。

程序: anfis\_ap1.m

```
%ANFIS training file
```

```
data= [...
```

```
    0 0 0
```

```
    0 1 1
```

```
    1 0 1
```

```
    1 1 0];
```

```
numpts=length(data(:,1));
```

```
trndata=data(1:1:numpts,:);
```

```
chkdata=data(2:2:numpts,:);
```

```
figure(1);
```

```
plot3(trndata(:,1),trndata(:,2),trndata(:,3),'o',chkdata(:,1),chkdata(:,2),chkdata(:,3),'x');
```

```
title('train data and check data before training');
```

```
nummfs=2
```

```
mftype='gbellmf';
```

```
fismat=genfis1(trndata,nummfs,mftype);
```

```
[x,mf-x] = plotmf(fismat,'input',1);
```

```
[y,mf-y] = plotmf(fismat,'input',2);
```

```
figure(2);
```

被训练的成对输入输出数据

产生被训练的模糊关系矩阵

```

subplot(311)
plot(x,mf-x, 'o');hold on;
plot(y,mf-y, 'x');hold on;
title('The input membership function before training')
%Begin to train the membership function by ANFIS
numepochs=30;
[fismat1,trnerr,ss,fismat2,chkerr] =anfis(trndata,fismat,numepochs,
NaN,chkdata);
[ x,mf-x] =plotmf(fismat1, 'input',1);
[ y,mf-y] =plotmf(fismat1, 'input',2);
subplot(312)
plot(x,mf-x);hold on;
title('The input1 membership function after training')
subplot(313)
plot(y,mf-y);hold on;
title('The input2 membership function after training')
%Evaluate the fis matrix
out=evalfis( [ trndata(:,1) trndata(:,2)] ,fismat1);
figure(3);
trndata=data(1:1:numpts,:);
chkdata=data(2:2:numpts,:);
plot3(trndata(:,1),trndata(:,2),trndata(:,3), 'o',chkdata(:,1),chkdata(:,2),chk
data(:,3), 'x');hold on;
plot3(trndata(:,1),trndata(:,2),out);
title('line represents that train data and check data after training');

```

用ANFIS开始训练

当ANFIS的训练程序完成后，接着建立SIMULINK模型，与范例4.1的模型相同，也是fzy2.mdl（见图4.20）。最后写主程序来执行上面所说的ANFIS训练文件及SIMULINK模型。主程序如下：

程序：ex461.m

```

clear;
close all;
anfis_ap1
writefis(fismat1, 'fzy3');
fzy2_mat=readfis('fzy3')
sim('fzy2');
figure
subplot(311);
plot(t,input1);

```

```

xlabel('time(sec)')
ylabel('input1');
axis( [0 10 0 2] );
grid on
subplot(312);
plot(t,input2);
xlabel('time(sec)')
ylabel('input2')
axis( [0 10 0 2] );
grid on
subplot(313);
plot(t,output);
xlabel('time(sec)')
ylabel('output')
axis( [0 10 0 2] );
grid on

```

执行后, 在训练时产生的信息及模糊推论矩阵fismat1如下:

```

nummfs =
    2
ANFIS info:
    Number of nodes:21
    Number of linear parameters:12
    Number of nonlinear parameters:12
    Total number of parameters:24
    Number of training data pairs:4
    Number of checking data pairs:2
    Number of fuzzy rules:4
Warning:number of data is smaller than number of modifiable parameters
Start training ANFIS ...
    1   4.61235e-007   4.57832e-007
    2   4.55842e-007   4.52898e-007
    3   4.50252e-007   4.47452e-007
    4   4.45208e-007   4.42721e-007
    5   4.40233e-007   4.37737e-007
Step size increases to 0.011000 after epoch 5.
    6   4.35828e-007   4.33839e-007
    7   4.31136e-007   4.29199e-007
    8   4.26516e-007   4.2482e-007
    9   4.22464e-007   4.20969e-007

```

Step size increases to 0.012100 after epoch 9.

```
10  4.1822e-007  4.16909e-007
11  4.14242e-007  4.12958e-007
12  4.10197e-007  4.09171e-007
13  4.06498e-007  4.05486e-007
```

Step size increases to 0.013310 after epoch 13.

```
14  4.03126e-007  4.0251e-007
15  3.99547e-007  3.98973e-007
16  3.96313e-007  3.9584e-007
17  3.93007e-007  3.92528e-007
```

Step size increases to 0.014641 after epoch 17.

```
18  3.90203e-007  3.89698e-007
19  3.87288e-007  3.86928e-007
20  3.84369e-007  3.83909e-007
21  3.81939e-007  3.81635e-007
```

Step size increases to 0.016105 after epoch 21.

```
22  3.79336e-007  3.79239e-007
23  3.76982e-007  3.76696e-007
24  3.74845e-007  3.74679e-007
25  3.72798e-007  3.72677e-007
```

Step size increases to 0.017716 after epoch 25.

```
26  3.70955e-007  3.70891e-007
27  3.69194e-007  3.68963e-007
28  3.67576e-007  3.67424e-007
29  3.66007e-007  3.66053e-007
```

Step size increases to 0.019487 after epoch 29.

```
30  3.64481e-007  3.64413e-007
```

Designated epoch number reached      ANFIS training completed at epoch 30.

Minimal training RMSE = 0.000000

Minimal checking RMSE = 3.64413e-007

完成的模糊推论矩阵为:

fzy2\_mat =

102.0000	122.0000	121.0000	51.0000	0	0	0
115.0000	117.0000	103.0000	101.0000	110.0000	111.0000	0
2.0000	1.0000	0	0	0	0	0
2.0000	2.0000	0	0	0	0	0
4.0000	0	0	0	0	0	0
4.0000	0	0	0	0	0	0
112.0000	114.0000	111.0000	100.0000	0	0	0
109.0000	97.0000	120.0000	0	0	0	0
112.0000	114.0000	111.0000	100.0000	0	0	0
109.0000	97.0000	120.0000	0	0	0	0

119.0000	116.0000	97.0000	118.0000	101.0000	114.0000	0
105.0000	110.0000	112.0000	117.0000	116.0000	49.0000	0
105.0000	110.0000	112.0000	117.0000	116.0000	50.0000	0
111.0000	117.0000	116.0000	112.0000	117.0000	116.0000	0
0	1.0000	0	0	0	0	0
0	1.0000	0	0	0	0	0
0	1.0000	0	0	0	0	0
105.0000	110.0000	49.0000	109.0000	102.0000	49.0000	0
105.0000	110.0000	49.0000	109.0000	102.0000	50.0000	0
105.0000	110.0000	50.0000	109.0000	102.0000	49.0000	0
105.0000	110.0000	50.0000	109.0000	102.0000	50.0000	0
111.0000	117.0000	116.0000	49.0000	109.0000	102.0000	49.0000
111.0000	117.0000	116.0000	49.0000	109.0000	102.0000	50.0000
111.0000	117.0000	116.0000	49.0000	109.0000	102.0000	51.0000
111.0000	117.0000	116.0000	49.0000	109.0000	102.0000	52.0000
103.0000	98.0000	101.0000	108.0000	108.0000	109.0000	102.0000
103.0000	98.0000	101.0000	108.0000	108.0000	109.0000	102.0000
103.0000	98.0000	101.0000	108.0000	108.0000	109.0000	102.0000
103.0000	98.0000	101.0000	108.0000	108.0000	109.0000	102.0000
108.0000	105.0000	110.0000	101.0000	97.0000	114.0000	0
108.0000	105.0000	110.0000	101.0000	97.0000	114.0000	0
108.0000	105.0000	110.0000	101.0000	97.0000	114.0000	0
108.0000	105.0000	110.0000	101.0000	97.0000	114.0000	0
0.3220	2.0332	-0.0710	0	0	0	0
0.3220	2.0332	1.0710	0	0	0	0
0.3220	2.0332	-0.0710	0	0	0	0
0.3220	2.0332	1.0710	0	0	0	0
0.0038	0.0038	-0.0076	0	0	0	0
0.0000	0.5076	0.5075	0	0	0	0
0.5076	0.0000	0.5075	0	0	0	0
-0.0063	-0.0063	-0.0025	0	0	0	0
1.0000	1.0000	1.0000	1.0000	1.0000	0	0
1.0000	2.0000	2.0000	1.0000	1.0000	0	0
2.0000	1.0000	3.0000	1.0000	1.0000	0	0
2.0000	2.0000	4.0000	1.0000	1.0000	0	0

训练前的成对输入输出，如图4.35所示。o代表训练点，有4个点；x代表验证点，有两个点。比较训练前后的隶属函数，如图4.36所示，最上面的一个图表明训练前两个输入隶属函数均相同，中间的和下面的图则为训练后的输入隶属函数图。训练后针对原来的成对输入输出模拟的输出，如图4.37所示。所画的线均经过训练点，表示训练已经完成，经SIMULINK模拟XOR门后的结果，如图4.38所示，其结果比范例4.1更准确了。这是因为隶属函数被训练过后，推论的结果更能符合预期的效果，这样可以避免靠人一直不停修改隶属函数的缺点。

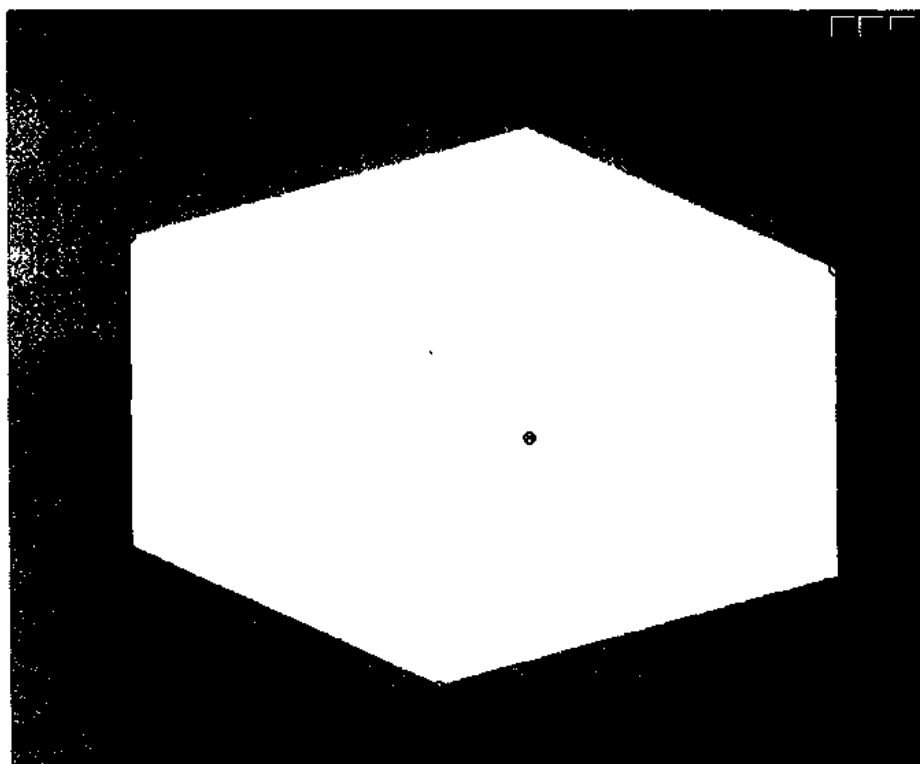


图 4.35 训练前的训练点(o)与验证点(x)图

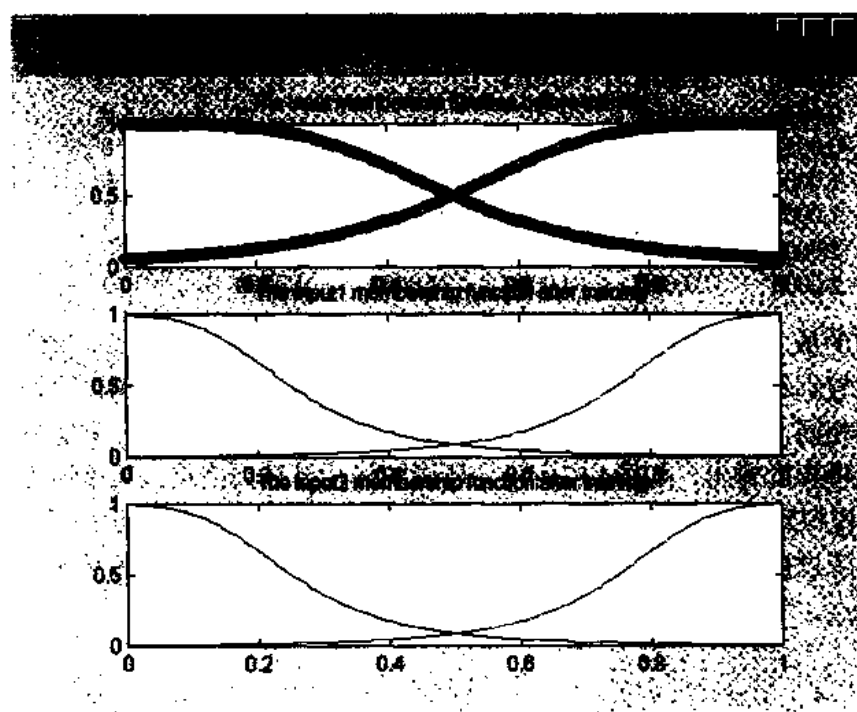


图 4.36 训练前后的隶属函数比较

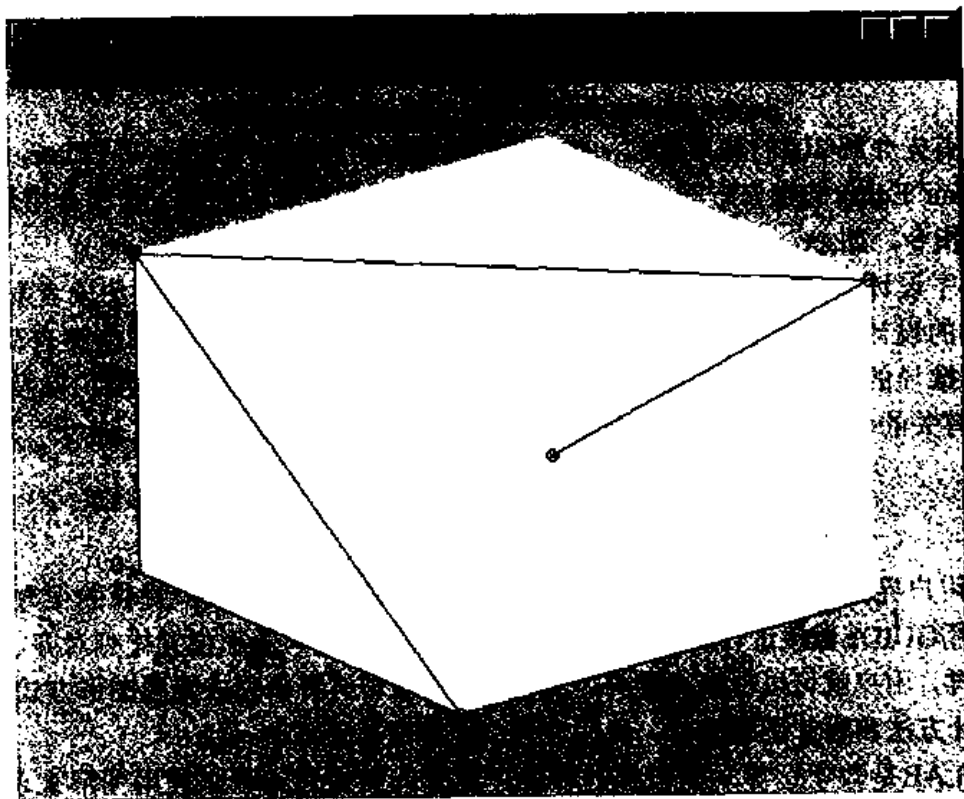


图 4.37 训练后的训练点(o)与验证点(x)图

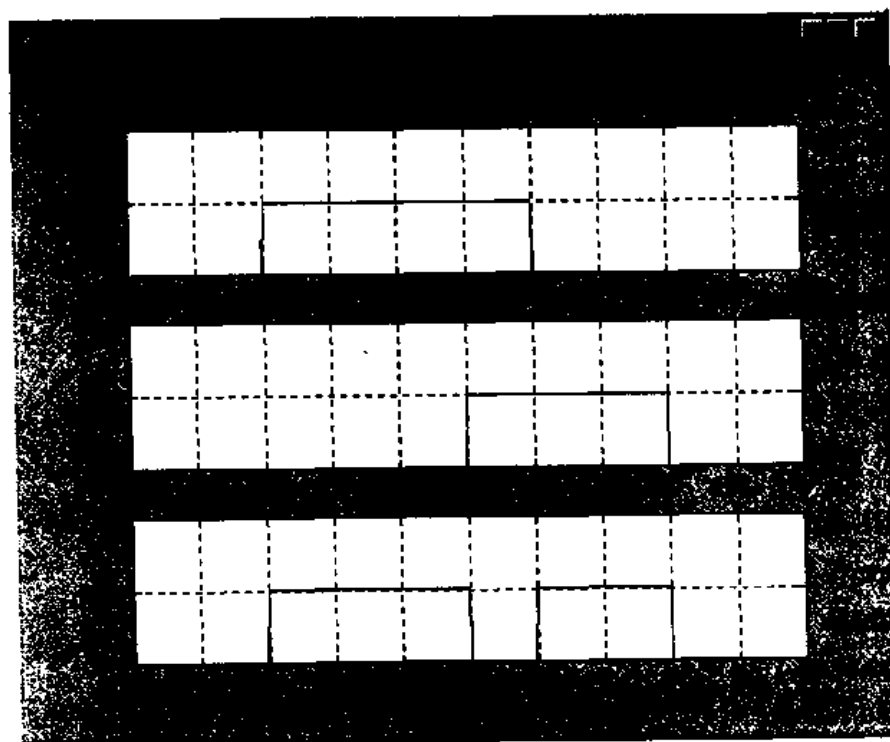


图 4.38 模拟结果图

## 第5章 图形用户界面(GUI)的应用

本章首先介绍GUI的基本概念,然后说明Property Editor, Callback Editor, Alignment Tool, Menu Editor的操作方法,以及pushbutton, text等对象的功能。其次介绍几个处理对象的重要指令,如set, get, findobj等的应用,以及程序设计的技巧。

在程序设计方面,先举一个简单的三维空间控制视角的例子,使读者对回调程序(Callback)的设计先有一点认识;然后介绍动态图形设计方法,使图形能够自动旋转。并采用范例推导的方式,由浅入深地说明一些与鼠标有关的指令的应用,以及与slider和checkbox有关的程序设计技巧,希望能与读者分享GUI的精髓。

### 5.1 GUI概述

图形用户界面(Graphical User Interface),简称GUI,是MATLAB®提供的很方便的工具,它可以利用GUIDE直接进行功能按钮编排、控件属性设定、图形位置排列对齐、操作区设计等。这样,用户就可以根据实际的需求以及个人爱好,将研究成果用图形方式表示出来。GUI的设计方法和设计步骤简单,表达力强,程序也易于维护。

MATLAB是数学运算功能很强的软件,运算的结果通常都必须用图形来表示,其绘图能力同样很强。GUI更易于将图形表现的多元化,不需要一行行的用程序指令去设定,只要用鼠标选择即可。在回调(Callback)处理模式下,也可以用程序设计方式将各个相关的处理程序集成为单一的处理系统,使研究成果表现得更好,这就是使用GUI的目的。

GUI的设计很简单,只需用鼠标选择即可增减对象;并且可将数个图形合并到一个图形上,增强了可视性,强化了展示(Demo)的功能。利用GUI开发的程序同样可以在MATLAB命令窗口环境下执行,所以GUI的兼容性很强。图形表示还具有便于理解和引人注意的特点。

进入GUI的方法是在命令窗口下输入指令:

`guide→Enter`

即可进入辅助控制面板(Guide Control Panel),如图5.1所示。

辅助控制面板主要用于对象的属性设定、回调程序(Callback)的编辑、位置的排列及菜单的设计。它共有三个部分可供用户使用,分别是:

- 辅助工具(Guide Tool)

- Property Editor

- Callback Editor

- Alignment Tool

- Menu Editor



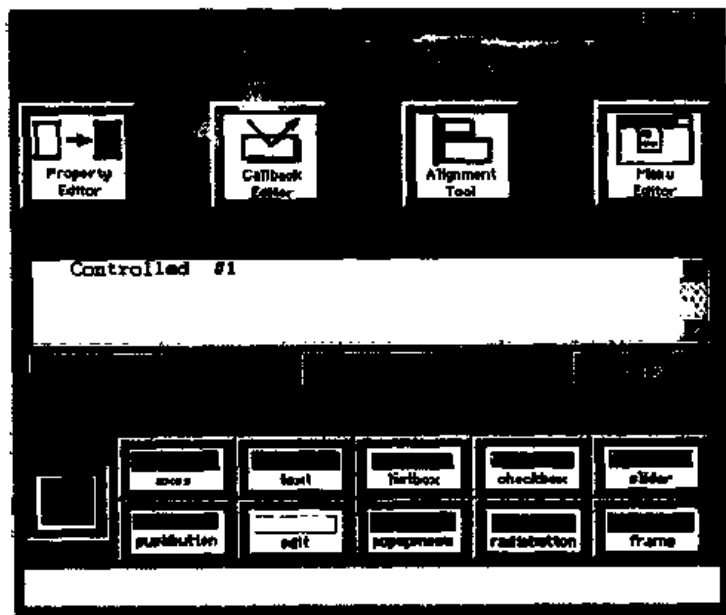


图 5.1 辅助控制面板

- 辅助控制图形列表(Guided Controlled Figure List)显示可供处理的图形(Figure), 并用于修改状态与作用状态之间的切换。
- 新建对象面板(New Object Palette)  
提供样板供用户建立对象, 包括按钮、坐标轴、编辑输入框、文本框、列表框等等。

在下一节中将会详细介绍它们的使用方法。

## 5.2 辅助控制面板

下面将分别说明四个编辑器以及辅助控制图形列表与新建对象面板。

### 5.2.1 Property编辑器

MATLAB中的图形(Figure)称为Handle Graphics™, 这样称呼是因为在屏幕上的每个对象(Object)都可以在被识别的情况下加以处理, 这种处理称为Handle。MATLAB允许用户随时修改处理图形, 所以Handle Graphics不只是一门技术, 更是一门艺术。Handle Graphics的树形结构如图5.2所示。

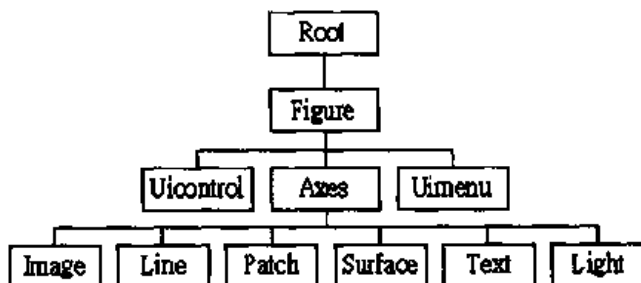


图 5.2 Handle Graphics 树形结构

属性编辑器(Property Editor)是在上述结构下进行浏览的好工具,它可以很容易地设定任何一个对象的属性值。

可以使用get与set两个指令获得对象,而指令的相关参数有:

- gcf : 表示get current figure, 获得当前显示图形的属性。
- gca : 表示get current axes, 获得当前坐标轴的属性。
- gco : 表示get current object, 获得当前鼠标选中对象的属性。
- gcbf: 表示get callback figure, 回调程序(Callback)获得图形属性。
- gcba: 表示get callback axes, 回调程序(Callback)获得坐标属性。
- bcbo: 表示get callback object, 回调程序(Callback)获得对象属性。

例如: 执行set(gcf,'visible','on')指令, 表示设定当前所显示图形的'visible'属性为'on'。返回属性编辑器将会发现所显示的属性列表(Property List)中的'visible'属性变为了'on', 如图5.3所示。

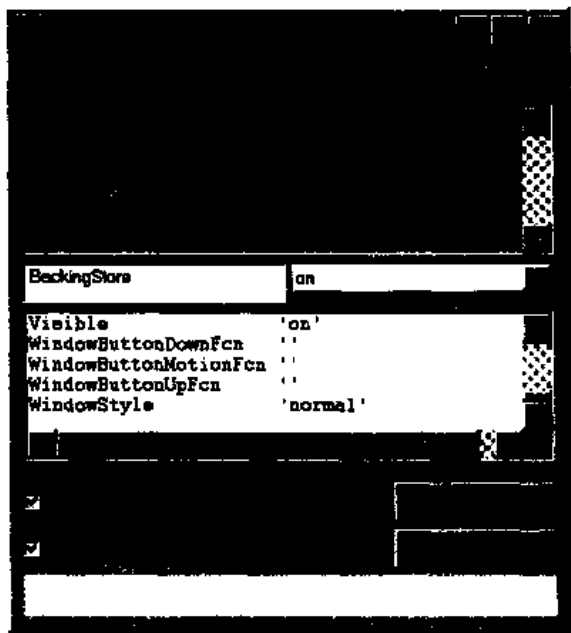


图 5.3

可以用鼠标在列表中移动并选中某个属性, 然后在输入框中直接输入属性值。或者在命令窗口中输入:

```
get(gcf)
```

以便获得该图形的属性值, 或者输入:

```
get(gco)
```

获得该对象的属性值如下:

```
BackgroundColor = [0.752941 0.752941 0.752941]
Callback =
```

```
Enable = inactive
Extent = [0 0 2.48276 2.48276]
FontAngle = normal
FontName = MS Sans Serif
FontSize = [8]
FontUnits = points
FontWeight = normal
ForegroundColor = [0 0 0]
HorizontalAlignment = center
ListboxTop = [1]
Max = [1]
Min = [0]
Position = [289.862 94.3448 62.6897 18.6207]
String =
Style = text
SliderStep = [0.01 0.1]
Units = points
Value = [0]

ButtonDownFcn= ctfpanel SelectMoveResize
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
Interruptible = off
Parent = [1]
Selected = on
SelectionHighlight = on
Tag = StaticText1
Type = uicontrol
UserData = []
Visible = on
```

不同的Handle Graphics会有不同的属性值，因此显示出来的结果不见得和上面所列的一样，上面的信息只是为了方便说明，并给读者一个大概的印象。下面就对较常用的属性加以说明：

- **BackgroundColor**: 设定值为1×3的矩阵，内部数字由0到1表示 [R(红)G(绿)B(蓝)] 的色度，可以设定背景颜色。
- **ButtonDownFcn**: 是一个字符串，表示鼠标按下后该执行的操作。在后面章节中会说明其用法。

- **Callback:** 这是最重要的属性，是一段程序，可由Callback Editor编辑完成。
- **Enable:** 通常设定为on，如果设为off，当鼠标选中该对象时，回调程序就不会产生作用。
- **ForegroundColor:** 也是RGB的矩阵，用来设定对象上文字或图标的颜色。
- **HorizontalAlignment:** 有left, center和right三种选择，用来决定对象上的文字是左对齐、居中还是右对齐。
- **Max, Min, Value:** 只对某些对象有用，用来表示该对象目前是否被选中。以check box对象为例，Min固定为0，Max固定为1，当check box被选中时，就将Max(即1)赋给Value；反之就将Min(即0)赋给 Value。这非常适用于if...then...回调程序设计。
- **Position:** 用于表示对象在图形上的位置及大小。这个属性实际上不需要设定，只要用鼠标拖曳即可，但仍有必要了解其代表的意义。Position用1×4的矩阵 [left bottom width height] 表示，如图5.4所示。

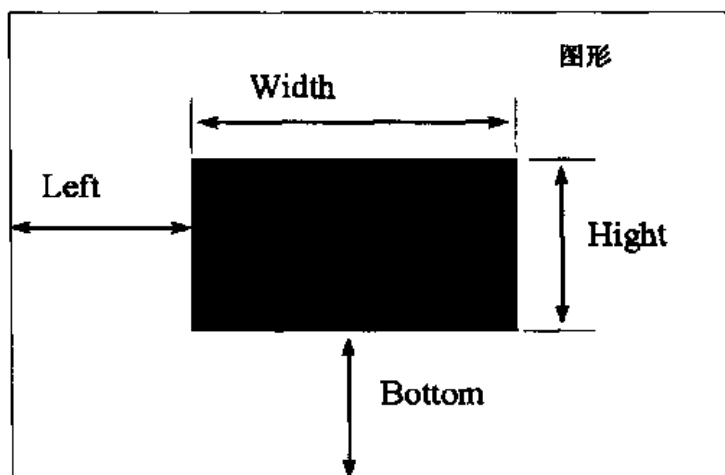


图 5.4 Position 表示的意义

- **String:** 与字面的意思一样，此属性表示该对象的名称，并显示在对象上面。
- **Unit:** 位置大小的单位。
- **Interruptible:** 有on, off两种选择。初始设定为off，表示现在处理的回调程序不受其他事件的影响而中断。初始设定为on，就可以中断回调程序去处理其他事件。
- **Tag:** 也是很重要的调用参数，表示每一个对象的标签，如同每个人都有姓名一样。
- **Visible:** 初始设定为on，以便让用户看到对象，有时候因为设计的需要要隐藏对象时，这个属性参数的设定就很有用了。
- **UserData:** 用作数据储藏区，以供后续处理时用。通常用法是先将数据装载进来（假设放到Temp中），然后用：

```
set(gcf, 'UserData', temp)
```

就完成了对UserData的设定。

### 5.2.2 Callback编辑器

回调程序编辑器(Callback Editor)是用来供用户对各个不同对象输入操作指令的区域,如图5.5所示。

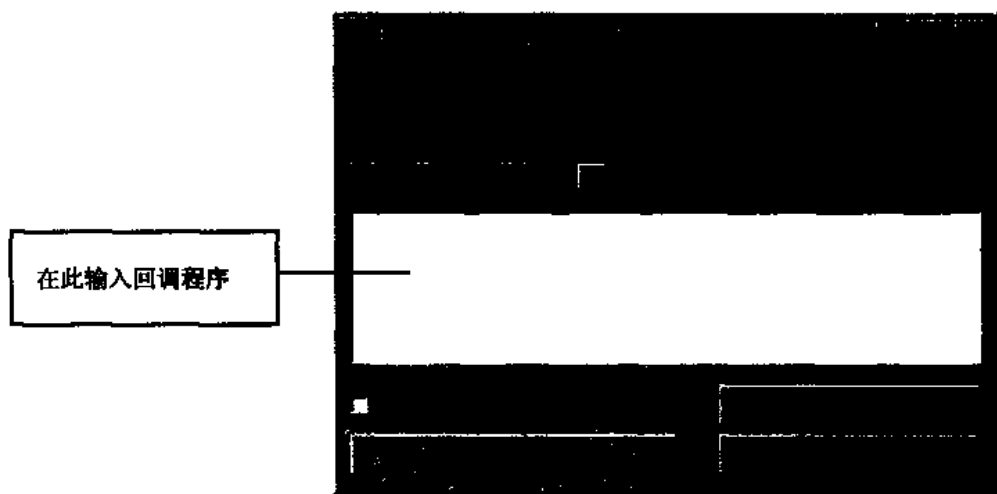


图 5.5 回调程序编辑器

中间的空白区域就是输入回调程序的位置。输入完成后,单击“Apply”按钮,即可完成编辑的动作。

举例来说,如果建立一个名为“grid on”按钮对象,然后进入该对象的回调程序编辑区,输入指令:

grid on → 单击“Apply”

则回调程序编辑完成。这样,以后每当用户单击“grid on”按钮时,程序便执行grid on的操作。

### 5.2.3 Alignment工具

当对象放到图形上以后,为了使用的方便及整体的美感,必须对其位置做适当的安排,对齐工具(Alignment Tool)就为用户提供了这方面的设计功能。只要将需排列的对象,用鼠标+Shift键选中后,再使用“排列工具”就可对这些对象进行下列的排列操作:

- 垂直方向可做的排列(Align)有顶端对齐、中间对齐、底端对齐的排列(Align),并可以进行上下分布(Distribute)和设定间距(Set Spacing)。
- 水平方向可做的排列(Align)有左对齐、居中对齐、右对齐的排列(Align),并可以进行左右分布(Distribute)及设定间距(Set Spacing)。

以上说明请参照图5.6。排列工具包括三个区域,Align, Distribute和Set Spacing。上面一排进行垂直方向的排列,下面一排进行水平方向的排列。

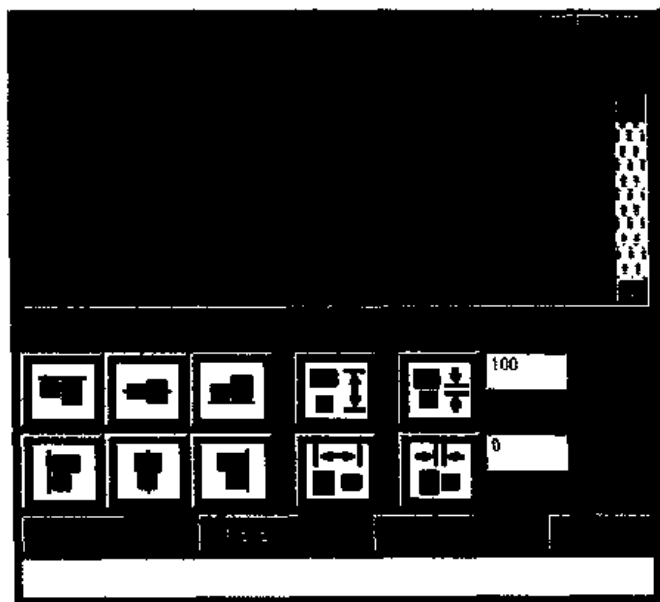


图 5.6 排列工具区

### 5.2.4 Menu编辑器

菜单编辑器(Menu Editor)用来在图形上方添加除原有的基本菜单(File, Edit, Options, Tools)以外的用户自定义的菜单。使用的方法在后面章节中再加以说明, 不过基本的建立步骤为: New Menu→排列菜单的层级→输入Label→输入Tag→进入Callback编辑回调程序→Apply。如图5.7所示。

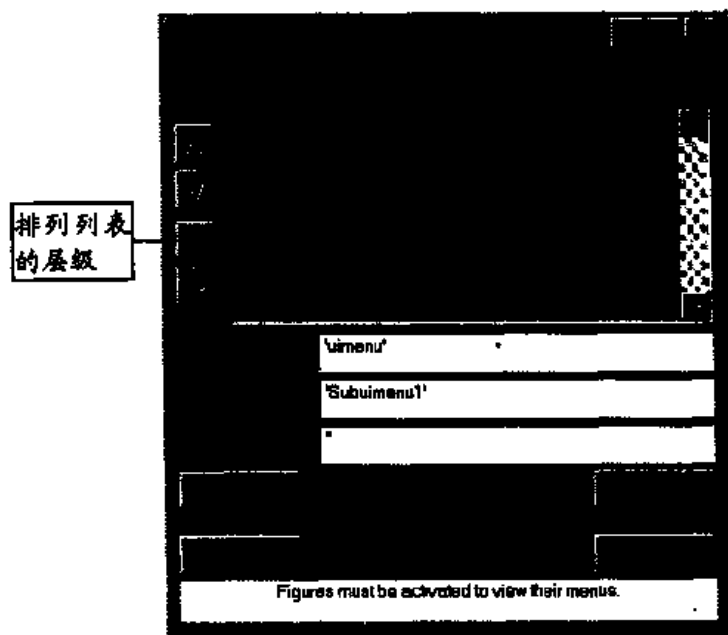


图 5.7 菜单编辑区

如果希望图形上的菜单全部不显示, 只需在图形的属性编辑区内将MenuBar设为none即可。

### 5.2.5 辅助控制列表

辅助控制列表(Guide-Controlled Figure List)列出目前的图形状态,共有两种,一种是编辑状态,另一种是作用状态。正常情况下,图形处于作用状态,当希望对图形中的对象做移动、新增或删除操作以及做菜单编辑时,就必须进入编辑状态。在做属性设定、回调程序编辑以及排列编辑时,不论处于何种状态,都可直接进入辅助控制面板进行修改。

在图形1为编辑状态时,辅助控制列表会显示:

Controlled #1

且新建对象面板为白色,可以使用。

在图形1为作用状态时,辅助控制列表会显示:

Active #1

且新建对象面板为灰色,无法使用。

这两种状态的切换很简单,只要用鼠标在状态列表项上单击,再单击“Apply”即可(当由编辑状态进入作用状态时,系统会提示是否要保存文件)。两种状态如图5.8、图5.9所示。

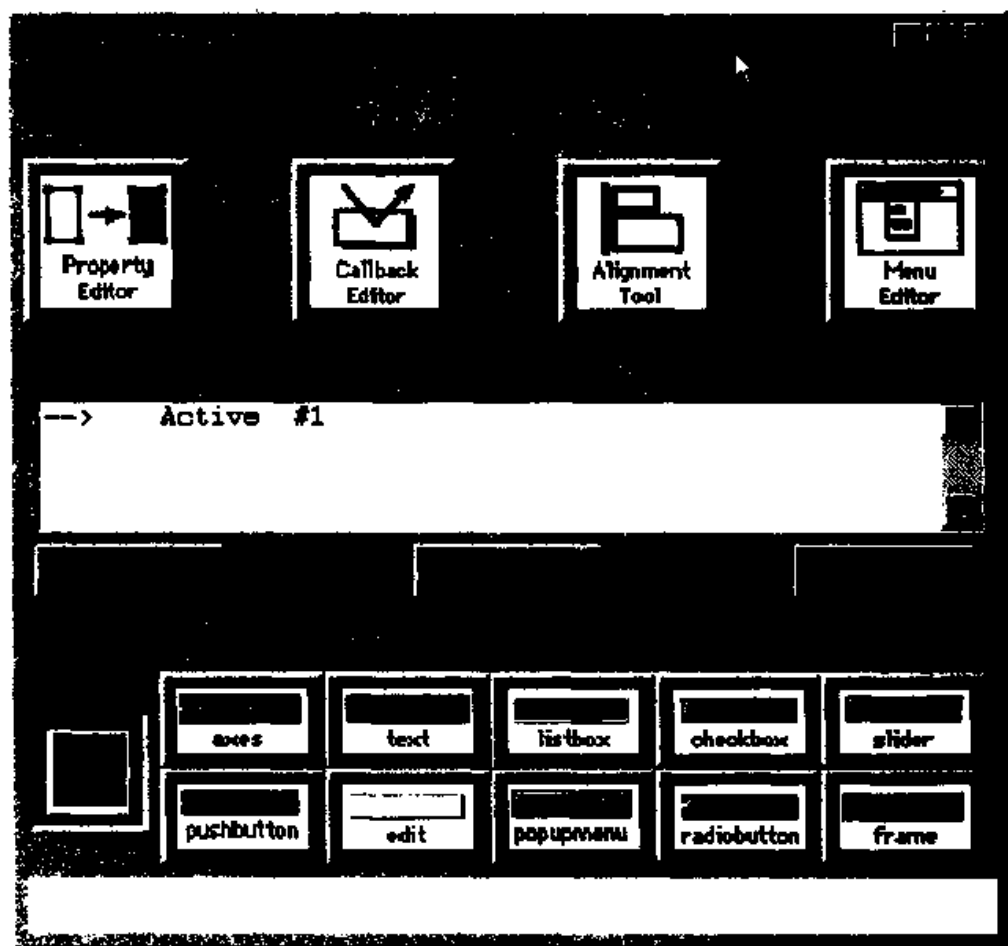


图 5.8 状态切换(作用状态)

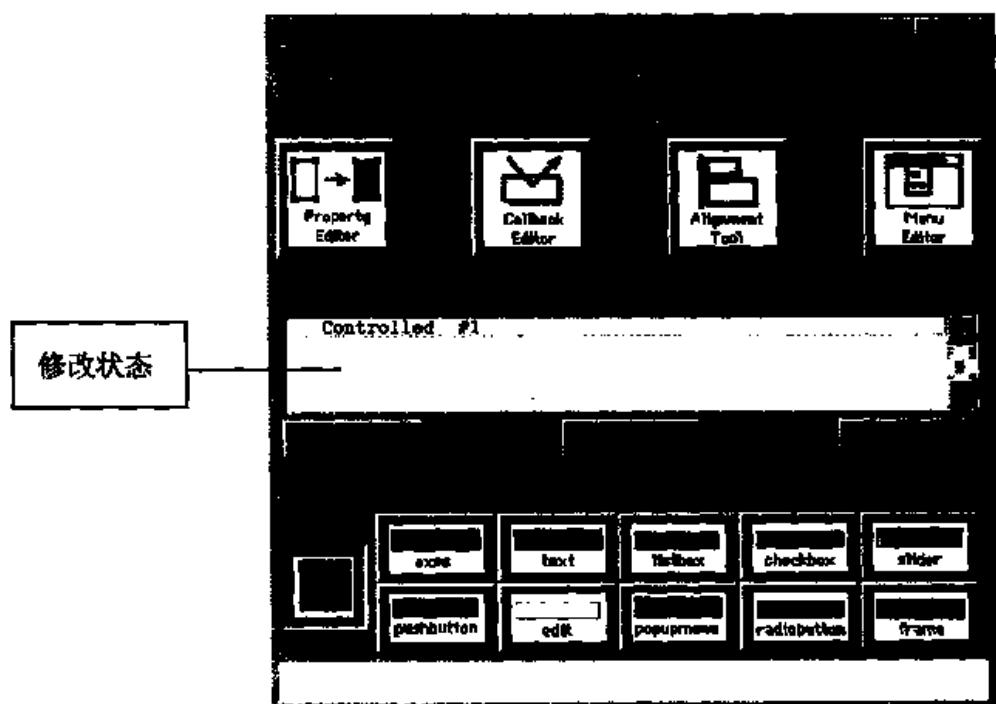


图 5.9 状态切换(修改状态)

### 5.2.6 新建对象面板

新建对象面板(New Object Palette)提供了10种可以放到图形上的对象, 下面分别进行说明:

- axes: 提供绘图的坐标区。
- pushbutton: 使用最多的对象, 可在其上建立名称, 编辑回调程序并执行特定的操作。
- text: 用于在图形上建立文字说明, 无法直接在上面输入文字。
- edit: 供用户在其中输入数字或变量, 以便程序执行时去提取。
- listbox: 建立选择列表, 供用户选取, 每个列表项都可以执行特定的操作。
- popupmenu: 与listbox类似, 建立的方法完全相同, 只不过显示的方式不同, 它只有在鼠标单击时, 才会列出列表项。
- checkbox: 用于设定on及off状态, 用鼠标选择切换状态。当被选中时, 其属性“Value”会被设为1, 表示on, 再单击一下, 就会将“Value”设为0, 表示off。
- radio button: 与checkbox功能相同, 只不过显示出来的图形不同。
- slider: 通过鼠标拖拉可以设定数据范围(range)。
- frame: 将属性相同或使用频率高的按钮用框架集中起来, 配合颜色的设定, 有利于识别。这在按钮很多或操作程序复杂时很重要。

以上10种对象可以根据需要用鼠标选中之后建立在图形的适当位置上。后面几节会再次说明其用法。



### 5.3 初级设计技巧

既然GUIDE是处理图形的辅助工具,那么它处理的对象就是图形(Figure)、坐标轴(Axes)以及对象(Object)。处理的目的是利用它来设定这些处理对象的属性(Property),进而通过回调(Callback)技巧来程序化这些属性,以达到设计要求。因此必须先确定处理的对象,然后一一对其属性做适当的设定,再根据需要加入对象,对每一个对象编写回调程序(Callback Program),最后集成GUI应用程序。下面将一步步说明设计的步骤。

**范例5.1 设计按钮以改变图形视角。**

**第一步: 建立图形。在命令窗口下输入下面的指令:**

```
z=peaks(30);  
mesh(z);
```

建立图形,结果如图5.10所示。

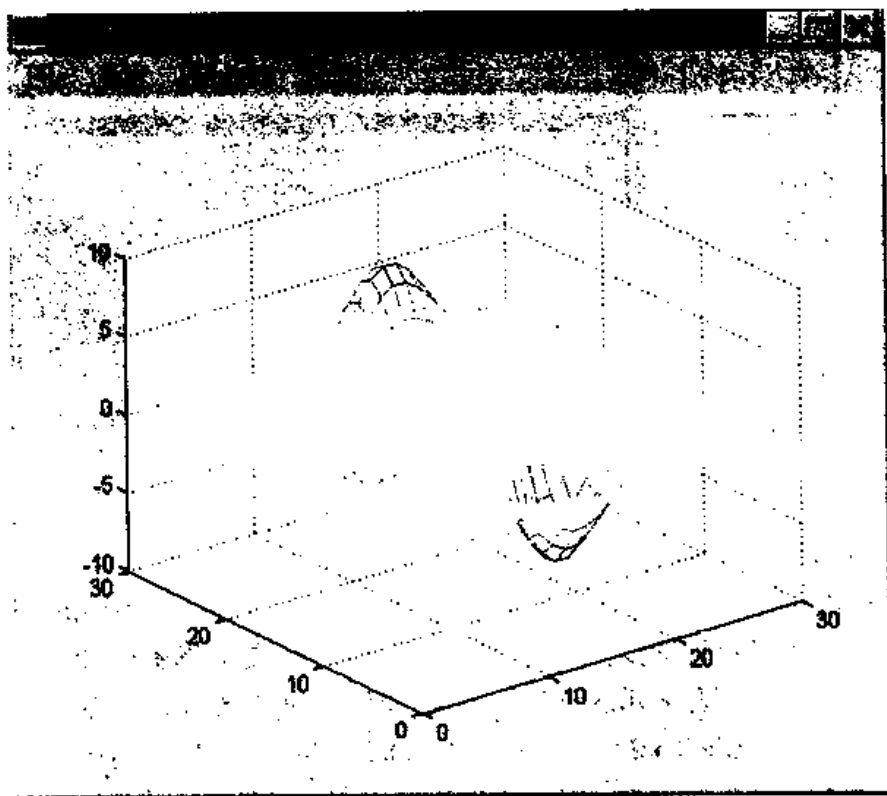


图 5.10

**第二步: 建立对象。先在命令窗口中输入:**

```
guide
```

进入辅助控制面板(Guide Control Panel)。

假设现在需要建立一个按钮以控制观看图形的视角,可以在Guide Control Panel中的New Object Palette中找到Pushbutton对象,用鼠标单击,然后在图

5.10中所希望的位置上拖曳出按钮的大小。

**第三步：命名对象。**对于刚才建立的对象，我们给它取个名字，叫做View 30。

双击刚才建立的对象“Pushbutton”，进入Property Editor中，选中最下面的Show Property List选项，此时会出现该“Pushbutton”的属性列表，再用鼠标拖曳到“String”的位置，此时处于空白状态，用鼠标单击，在列表上方的编辑区内输入：

View 30

单击“close”即可完成按钮的命名步骤。如图5.11所示。

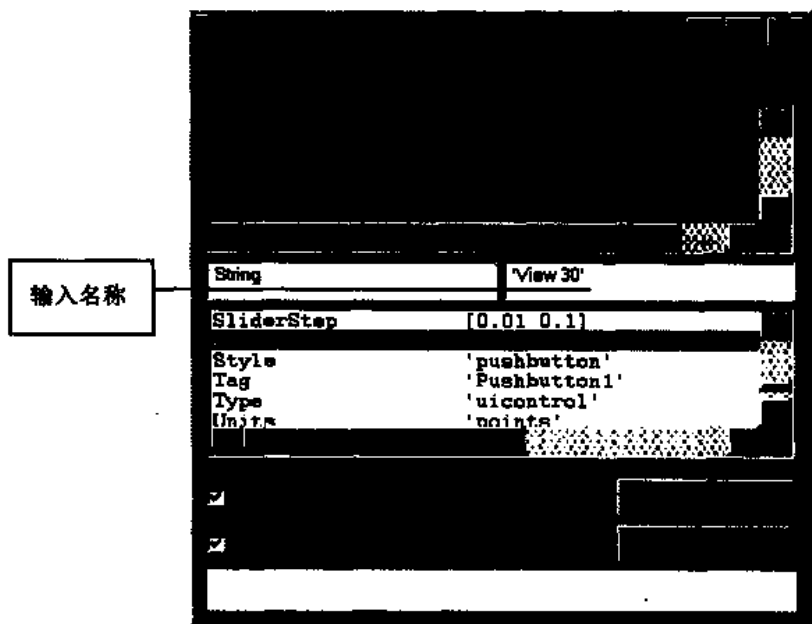


图 5.11 命名按钮

**第四步：编辑回调程序。**

按钮命名后还不能执行操作，必须利用“回调程序”才能调用MATLAB的指令，以便执行该按钮的功能。“回调程序”的编辑是在“CallbackEditor”中执行的，因此先用鼠标选取名为“View 30”的按钮，然后在辅助控制面板上的“Guide Tool”中单击“Callback Editor”，此时出现空白的编辑区，在此输入指令view(30,30)，单击“Apply”→“Close”即可，如图5.12所示。采用同样的方法，再建立“reset”按钮，在其回调程序内输入：

```
view(-30,10)
```

用于恢复视角。

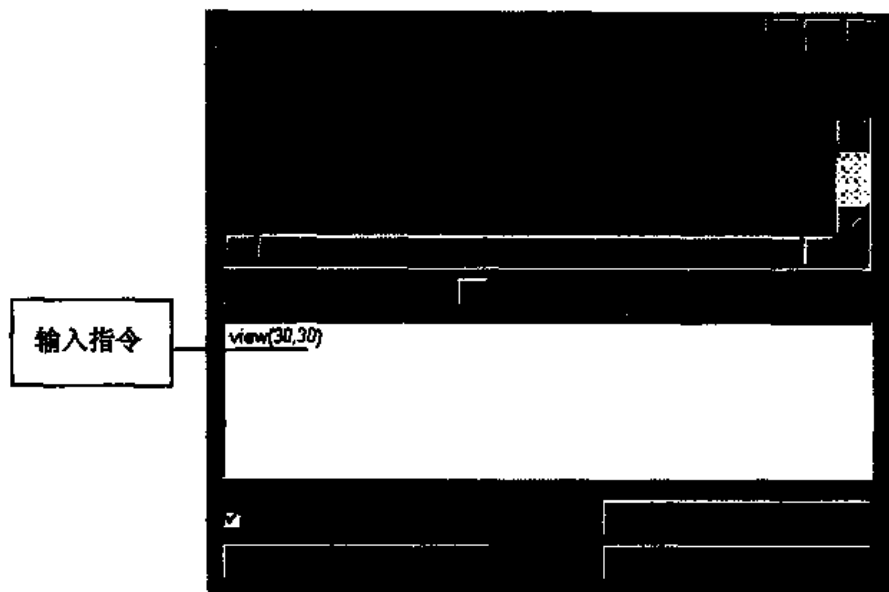


图 5.12 编辑回调程序

**第五步：保存并执行。**

所有的对象均做完属性设定、回调程序编辑后，就可以将其保存并执行了。在辅助控制面板中间的“Guide-Controlled Figure List”上，用鼠标单击“Controlled #1”，使其变成“Active #1”，再单击下方的“Apply”按钮，系统会询问是否要保存文件。做完保存文件的工作后，即可在设计完成的图形中执行刚才设计的GUI了，如图5.13所示。

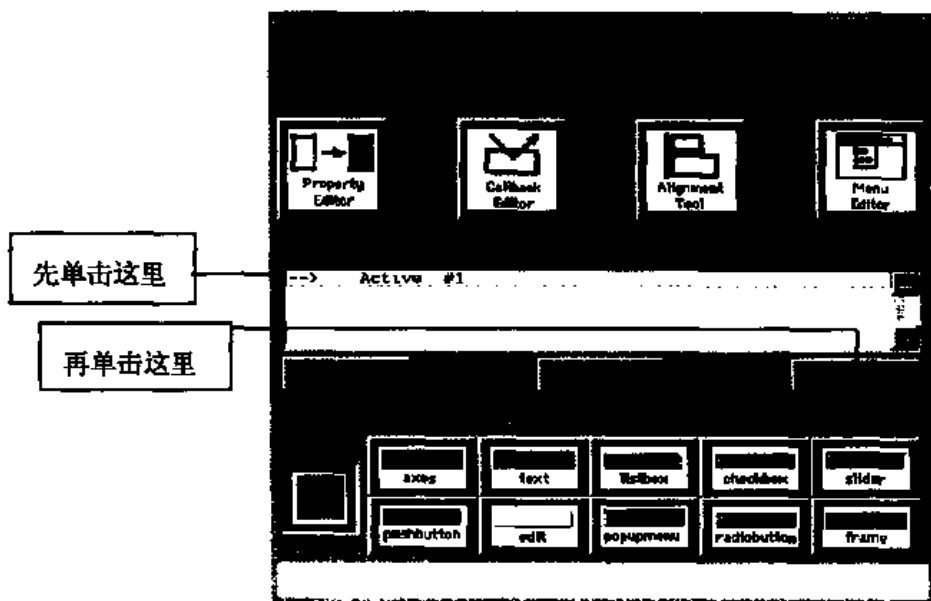


图 5.13 保存 GUI

执行时，用鼠标单击“View 30”，图形就可以按照回调程序执行视角的设定，如图5.14所示。

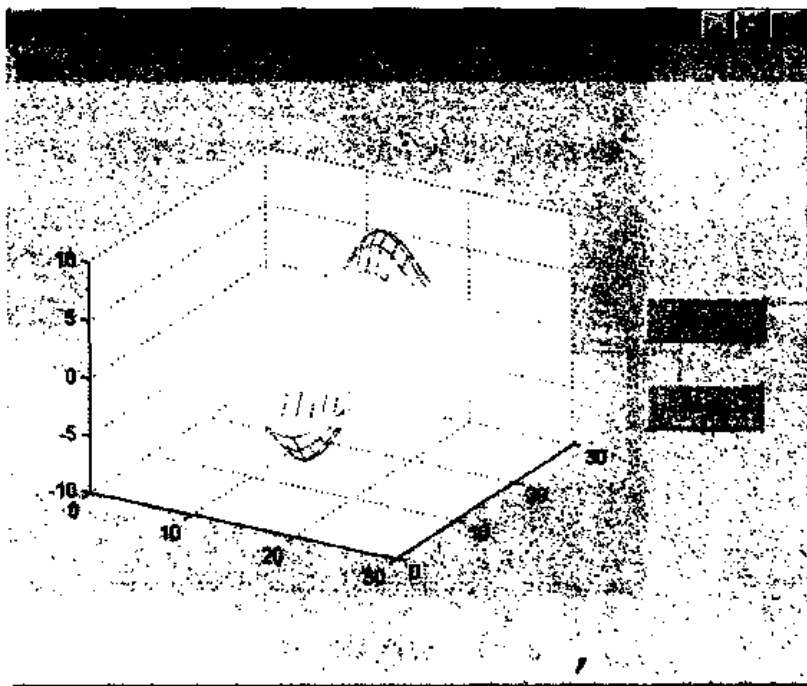


图 5.14 设计结果

在命令窗口中输入`ex1`→`Enter`即可执行本例程序。

## 5.4 对象处理指令介绍

本节介绍如何集成并应用各种不同的对象。其中最重要的有三个指令，分别为`get`、`set`和`findobj`。`get`可以获得鼠标目前选中对象的某个属性值，`set`用于设定鼠标目前选中对象的属性值。`findobj`指令用于获取目标对象。这三个指令的格式说明如下：

```
variable_name=get(handle_object,'property_name')
```

`variable_name`是程序内的变量名称，`handle_object`是`gcbo`或通过`findobj`指令所获取的对象。`gcbo`表示`get callback object`的意思。`property_name`可以是该对象属性列表中的任意项，因此这条指令可以获取属性列表中的任何一项属性值。例如，如果在`Callback Editor`中执行如下指令：

```
name1=get(gcbo,'String');
```

如果该对象的`String`项内的值为`'OK'`，则执行后`name1`的值也就是`'OK'`。

```
set(handle_object,'property_name',data)
```

`set`的操作刚好和`get`相反，将`data`值放到`handle_object`的`property_name`中。`data`可以是单个的数值或者是`data set`的文件名，也可以是一个运算式。`handle_object`则是`gcbf`(`get callback figure`)或是通过`findobj`指令所获取的对象，因此可以设定`figure`中的任何一个对象的属性值。例如，如果在`Callback Editor`中执行如下指令：

```
set(gcf,'String','OK');
```

如果该对象的String项初始为空白, 则执行后, 到该figure的property list中去查看String的值, 将变为'OK'。

```
handle_object=findobj(gcf,'property_name','care_name')
```

findobj的作用就是在gcbf中查找property\_name与care\_name相同的对象。如果有相同的, 就把找到的对象的名称赋给handle\_object。findobj指令必须配合set与get运用, 不然找到了对象也毫无意义。例如:

```
handle1=findobj(gcf,'String','OK')
```

表示在gcbf中查找property list中的一个String为'OK'的对象, 其名称叫做handle1。然后就可以利用set去改变其属性值, 如果执行:

```
set(handle1,'String','Wrong')
```

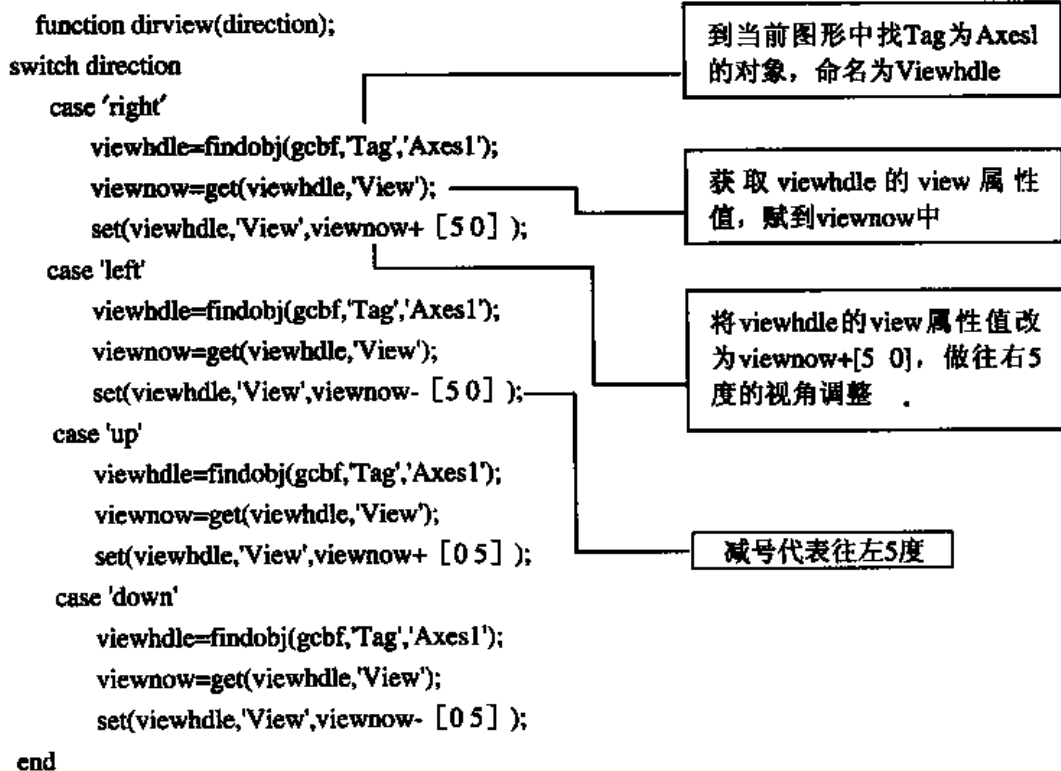
则该对象的String值变为'Wrong'。

下面提供了一个简单而实用的例子, 请大家熟练这三个指令的用法。

### 范例5.2 设计图形旋转。

第一步: 首先编写一个程序, 以备回调程序使用。内容如下:

程序: dirview.m获取viewhdle的view属性值, 赋到viewnow中。



说明: 本程序先建立4种旋转的模式, 并在每一个模式中定义好视角的调整角度。因此当单击按钮时, 就会去执行该按钮的回调程序所定义的操作。

**第二步：建立回调程序。**

在4个Pushbutton中的“Callback Editor”中输入回调程序。

```
dirview right  
dirview left  
dirview up  
dirview down
```

例如，在up view按钮的“Callback Editor”中输入：

```
dirview up
```

如图5.15所示。

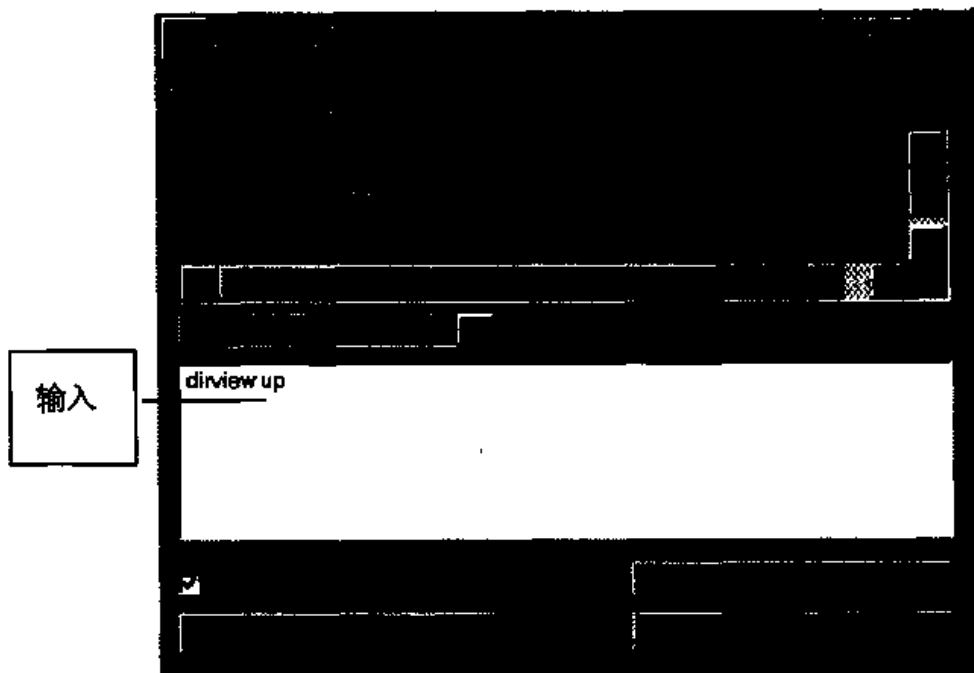


图 5.15 设计回调程序

按照同样的方法，在每个按钮的“Callback Editor”中输入回调程序。

**第三步：保存文件。**

按照上节所介绍的方法将其保存并执行。结果如图5.16所示，每按一下按钮，图形即向所希望调整的方向旋转5°。

本例的程序可以在命令窗口中输入ex2→Enter即可执行。

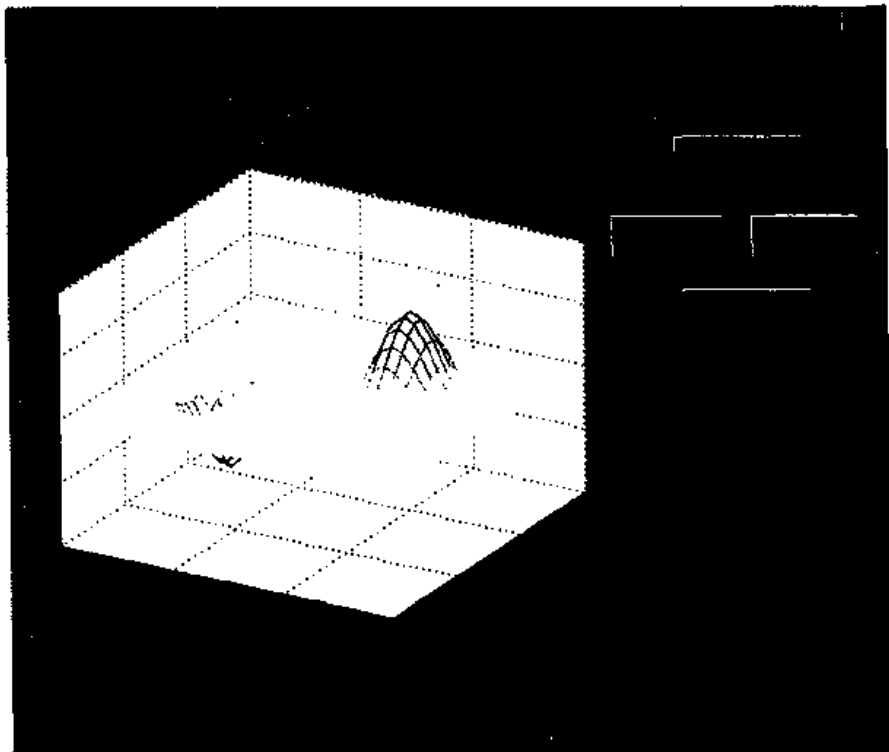


图 5.16 设计结果

### 5.5 动态图形设计技巧

上节的图形还需要用鼠标单击，本节要介绍通过程序设计让图形自动旋转。所用到的关键指令为：

`drawnow`

配合for loop的程序设计，可以让图形自动按照用户的要求去旋转，而不必一直用鼠标单击。这是个简单的例子，读者可以将其加入自己的设计结果，再加上自己的一些设计创意，一定可以设计出更完善更精致的动态图形程序。

#### 范例5.3 设计连续旋转图形。

先将“New Object Palette”中的“Text”及“Edit”加到图形上，并且在“Text”的“String”中输入所要标示的文字：

“Keyin desired rotation degree”

“Text”的功能不在于执行什么操作，而是仅将设计者所要表明的一些想法或按钮的名称显示在图形上。

“Edit”提供一个空白区域，供用户随意输入用于调用的参数。

最后，把4个操作按钮改为：

Up Rotate

Left Rotate

Right Rotate

Down Rotate

这4个按钮是“回调程序”执行的部分，因此每个按钮都对应一个“回调程序”。本节的回调程序是一个共用模块，在程序里面用：

switch

指令来区分4个不同的操作。为了便于switch进行区分，在“回调程序”的后面就必须输入操作的参数，例如下面所介绍的rotview up，其中up就是操作参数。图5.17即为完成的图形，接着介绍程序设计的技巧。

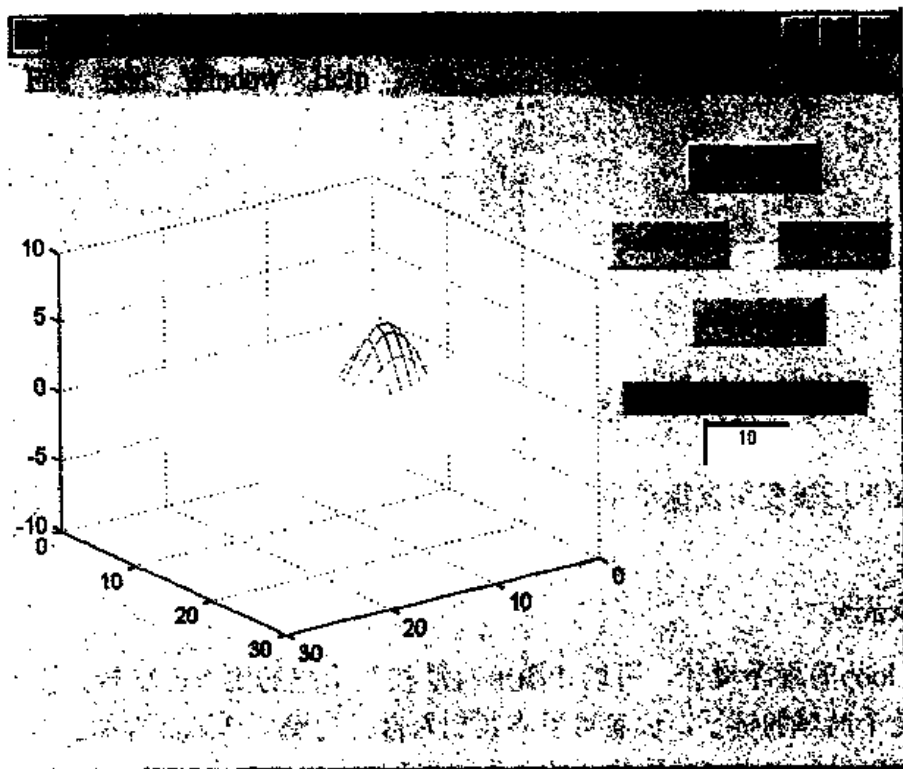


图 5.17 设计结果

回调程序为rotview.m，内容如下：

```
function rotview(direction);
```

```
%get the value of edit text
```

```
rot_rag=findobj(gcf,'Tag','EditText1');
```

```
stop_range=str2num(get(rot_rag,'String'));
```

将editText1内的字符串转换为数字

```
range=1:1:stop_range;
```

设定旋转范围

```
%execute the rotation of desired direction
```

```
switch direction
```

```
case 'right'
```

```
viewhdle=findobj(gcf,'Tag','Axes1');
```



```

viewnow=get(viewhdlg, 'View');
for i=1:length(range);
    set(viewhdlg, 'View',viewnow+ [range(i) 0] );
    drawnow _____ 屏幕绘图指令
end
case 'left'
viewhdlg=findobj(gcf, 'Tag','Axes1');
viewnow=get(viewhdlg, 'View');
for i=1:length(range);
    set(viewhdlg, 'View',viewnow- [range(i) 0] );
    drawnow _____ 屏幕绘图指令
end
case 'up'
viewhdlg=findobj(gcf, 'Tag','Axes1');
viewnow=get(viewhdlg, 'View');
for i=1:length(range);
    set(viewhdlg, 'View',viewnow+ [0 range(i)] );
    drawnow _____ 屏幕绘图指令
end
case 'down'
viewhdlg=findobj(gcf, 'Tag','Axes1');
viewnow=get(viewhdlg, 'View');
for i=1:length(range);
    set(viewhdlg, 'View',viewnow- [0 range(i)] );
    drawnow _____ 屏幕绘图指令
end
end
end

```

说明：在本段程序中，先由findobj去获取“Edit”中所输入的数值，作为旋转多少角度的依据；接着使用switch指令确定要向哪个方向旋转，再由for loop以及drawnow指令去完成屏幕显示的工作。

接着在4个按钮的“Callback Editor”中输入其回调程序的名称：

```

rotview up
rotview down
rotview left
rotview right

```

其中的rotview right如图5.18所示。

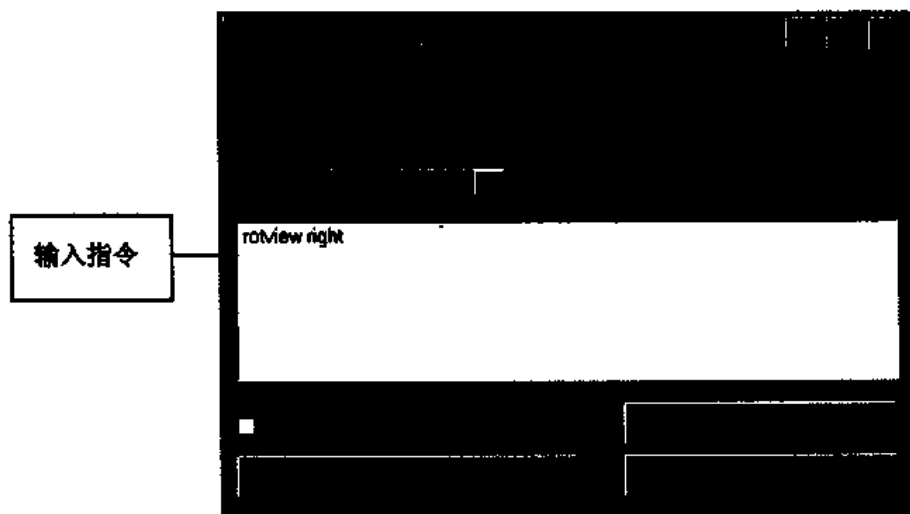


图 5.18 编辑回调程序

在命令窗口中执行本范例的程序ex3.m就会出现图5.17所示的画面，单击任何一个按钮，图形就会按指定的方向旋转。

## 5.6 列表框(Listbox)设计技巧

本节学习列表框的设计方法，列表框可由设计者根据需要先行建立项目名称，项目名称通常是先建立好以备回调程序调用的。本节将承接上例，加上列表框对象后，在列表内设定颜色以改变图形的颜色。

### 范例5.4 设计列表框改变图形的颜色。

在命令窗口下执行

`guide`→`Enter`

进入辅助控制面板后，在新建对象面板中，用鼠标选中“listbox”拖曳到图形上，画出期望的大小。因为希望建立的是一组向量，因此先到命令窗口中输入：

```
ma={'hsv';'hot';'gray';'prism'}
```

接着回到“Listbox”上双击鼠标，进入属性编辑器，到“property list”找到“String”，输入：

```
ma
```

如图5.19所示，请注意不要加单引号。

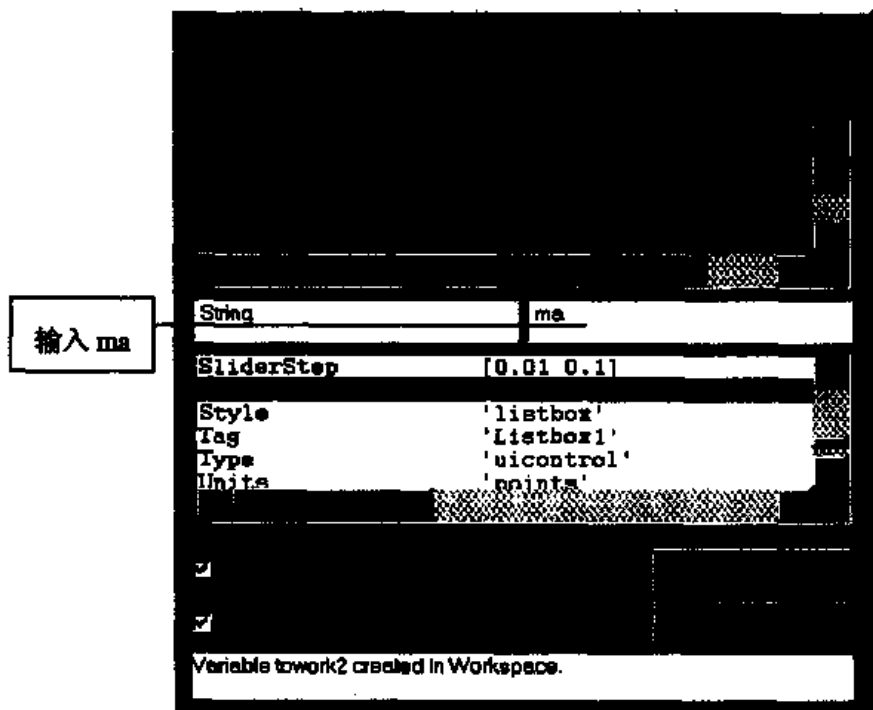


图 5.19 列表框项目的建立

继续讲述范例5.3, 进入“Callback Editor”编辑回调程序, 程序内容为:

<code>color_value=get(gcbo,'Value')</code>	获得当前对象的value值
<code>color_set=get(gcbo,'String')</code>	获得当前对象的string值
<code>colormap(color_set{color_value})</code>	执行colormap指令, 参数为color_set向量中的第color_value个值。

要注意最后一行的括号中有大括号。

完成编辑后, 如图5.20所示。

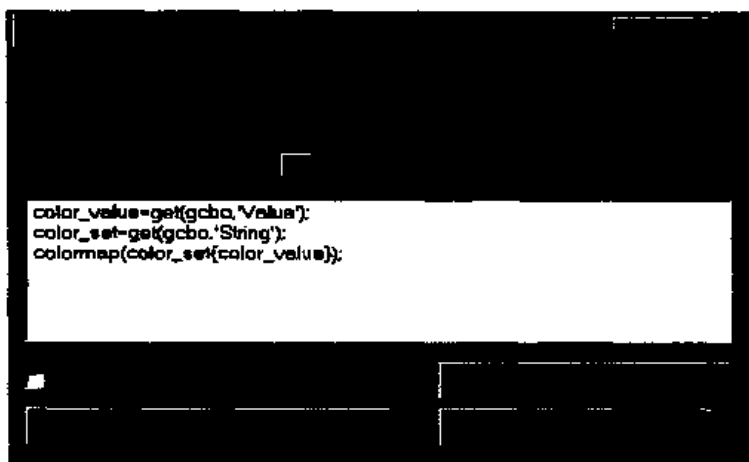


图 5.20 编辑回调程序

最后即可依次执行Apply→Close→选择Controlled #1→Active #1→保存文件。执行的结果是随时可以变换图形的颜色, 如图5.21所示。

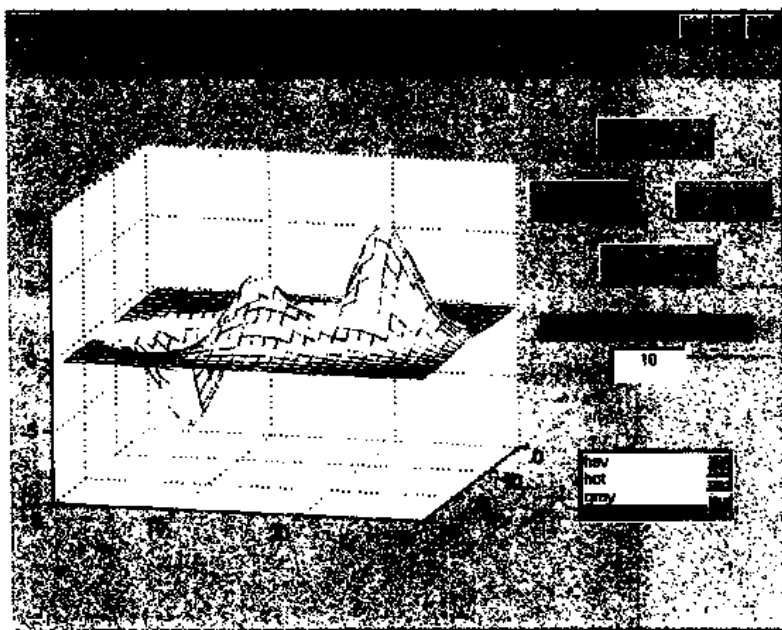


图 5.21 设计结果

本例的文件名为ex4.m。

## 5.7 充分利用ButtonDownFcn指令

如果希望做一些可用鼠标切换的操作，ButtonDownFcn指令就非常适合，在此特别加以介绍。ButtonDownFcn在鼠标按下后就会执行指令。与“回调程序”的不同之处在于回调程序只有在鼠标选到的按钮上单击，才会执行操作，而ButtonDownFcn则与按钮无关，它仅与鼠标有关，只要鼠标按下就会产生操作，因此ButtonDownFcn通常都会是设定在Axes区域内的操作。在5.8节的画板设计技巧中，ButtonDownFcn就是设定在Axes区域内的，如果配合WindowButtonMotionFcn以及WindowButtonUpFcn两个属性，就可以让鼠标的动作连贯起来，也就是按下（ButtonDownFcn）→移动（WindowButtonMotionFcn）→放开（WindowButtonUpFcn）。下面就用一个简单的例子来说明ButtonDownFcn是如何发生作用的。

### 范例5.5 用鼠标变换字体的大小。

设计的步骤为：(继续范例5.4)

1. 建立“Text”对象。
2. 进入“Callback Editor”，将“Callback”切换到Button-DownFcn，并在下面输入：

```
set(gcbo,'fontsize',16-get(gcbo,'fontsize'))
```

执行结果是可以使其在字体大小4与12之间切换。上述指令的意义是在当前对象上，先得到（用get指令）的字体大小，用16减掉后，再赋给当前的对象字体大小属性值。但是为什么能在4与12之间进行切换呢？假设当前的字体大小为4，用16作为上限，令其 $16-4=12$ ，

当前字体大小就变为12；如果鼠标再切换，则为 $16-12=4$ ，所以又变回4，如此循环就能用于设计字体大小切换的功能。回调程序设计区如图5.22所示。

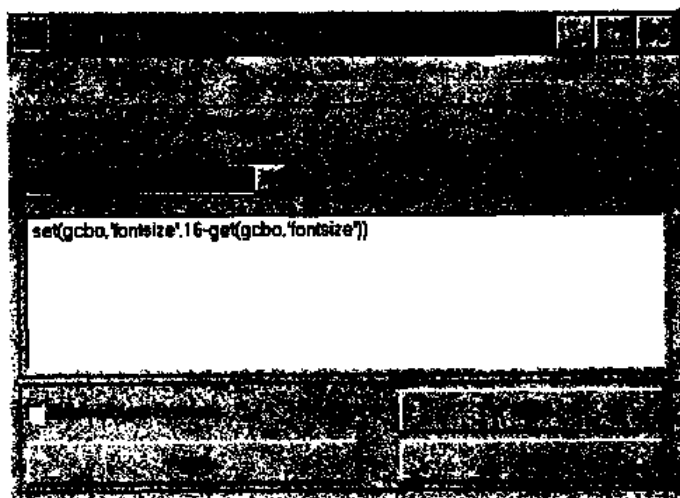


图 5.22 回调程序设计

执行结果如图5.23所示，鼠标只要单击文字区域，其字体大小就会变换。

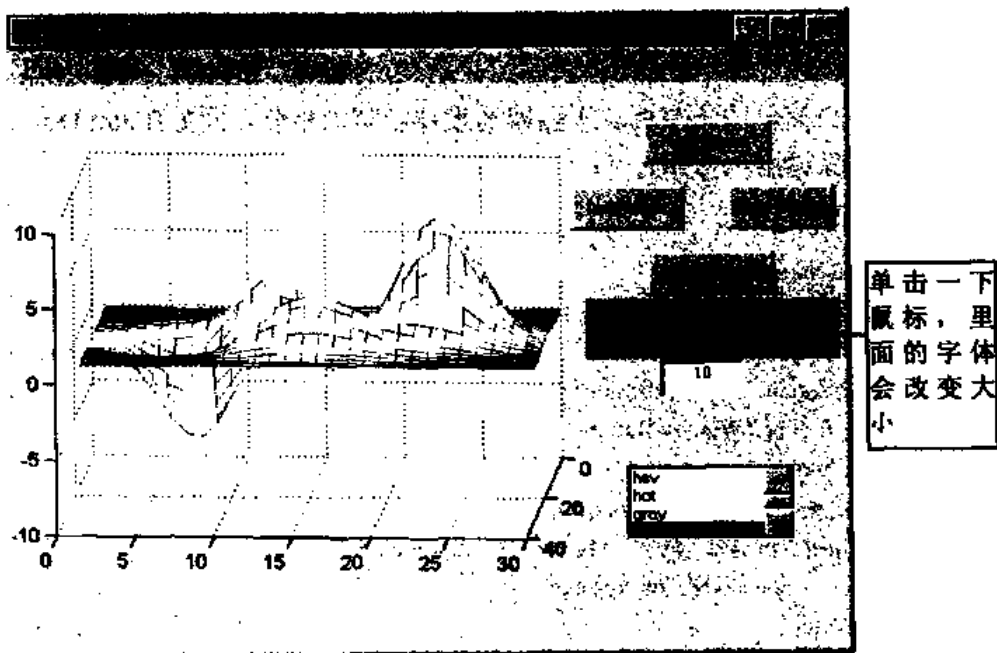


图 5.23 执行结果

本范例的程序为ex5.m。

## 5.8 鼠标属性指令设计技巧

如果想利用鼠标处理图形上的对象，可以使用 WindowButtonDownFcn 以及 WindowButtonUpFcn 两个属性值，并配合上节介绍的 Button-DownFcn 的回调程序。

WindowButtonMotionFcn以及WindowButtonUpFcn两个属性值是Figure的属性，作用是在鼠标移动或释放时产生动作，至于要做什么动作，则由ButtonDownFcn的回调程序来决定。也就是说当鼠标一按下去，ButtonDownFcn首先决定要做什么动作；当鼠标移动时，程序会去执行WindowButtonMotionFcn的动作；放开鼠标，则执行WindowButtonUpFcn的动作，如此循环下去。

最常用的是利用鼠标的位置(其属性值为Currentpoint)去设计应用程序。Currentpoint为 $2 \times 3$ 的矩阵，用来表示鼠标在三维空间中的位置。第一行是鼠标离观察者较远的坐标位置，而第二行是鼠标离观察者较近的位置，通常用x与y坐标即可表示屏幕的位置，例如：

```
currentpoint=[1 1 1;2 2 0]
```

表示在坐标系中鼠标由[1 1]移动到[2 2]的位置。

另外还有erasemode参数，它用于设定对象的清除模式，共有4种模式可供设定：

- none: 对象移动时不被清除。
- normal: 一般用途，对象的表示精确度最好。
- xor: 对象的产生与清除是通过XOR逻辑产生的，大都用于动画设计中。
- background: 与xor相同，但用于不希望影响颜色的情况。

在设计动画时，通常设定为xor。

#### 范例5.6 利用鼠标属性设计画板。

本例设计一个简单的画图板。下面即为设计步骤。首先，写好ButtonDownFcn执行的程序，供Axes对象调用，内容如下：

程序：dirmov.m

```
function dirmov(direction);
switch direction
case 'start'
    set(gcf,'WindowButtonMotionFcn','dirmov move');
    set(gcf,'WindowButtonUpFcn','dirmov stop')
case 'move'
    c_p=get(gca,'currentpoint');
    line(c_p(:,1),c_p(:,2),'Marker','*','clipping','on',...
        'erasemode','background')
case 'stop'
    set(gcf,'WindowButtonMotionFcn','');
    set(gcf,'WindowButtonUpFcn','')
end
```

本程序根据不同操作分别执行三种动作在鼠标按下时执行第一种：

```
dirmov start
```

在第二种情况下，执行WindowButtonMotionFcn。

`dirmov move`

在第三种情况下，执行WindowButtonUpFcn。

`dirmov stop`

因此当鼠标移动时，会在dirmov.m程序中执行第二种动作，获得鼠标的坐标位置，并画出图形。clipping设为on使图形能够被剪掉，所以画的图不会超过方框边界。erasemode设为background可使图形显示速度加快，色彩变好且不会闪烁。当放开鼠标按钮时，会到dirmov.m程序中执行第三种动作，清除WindowButtonMotionFcn及WindowButtonUpFcn的设定。写好程序后，可以设计GUI面板。和前几节的方法相同，进入辅助控制面板后，加入“Axes”及“Pushbutton”两个对象。

在“Pushbutton”的回调程序编辑区中输入：

`cla`

指令，并在属性编辑区中输入：

`clear`

指令。接着对“Axes”进行编辑，选择回调程序编辑器的Button-DownFcn选项并输入：

`dirmov start`

以便使鼠标按下时能够执行dirmov.m回调程序，如图5.24所示。

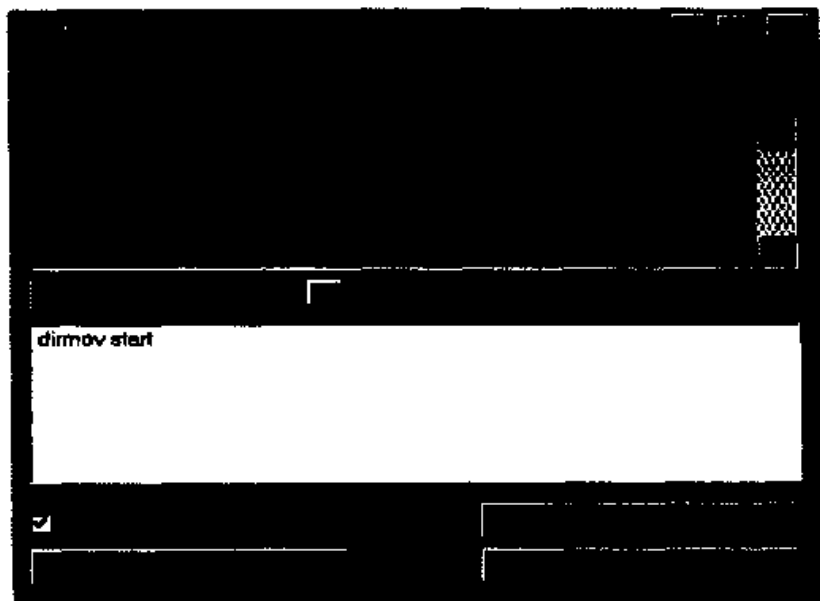


图 5.24 编辑回调程序

保存文件并执行，结果如图5.25所示，可以用鼠标在上面画图，如果要清除图形，按一下“Clear”按钮即可。

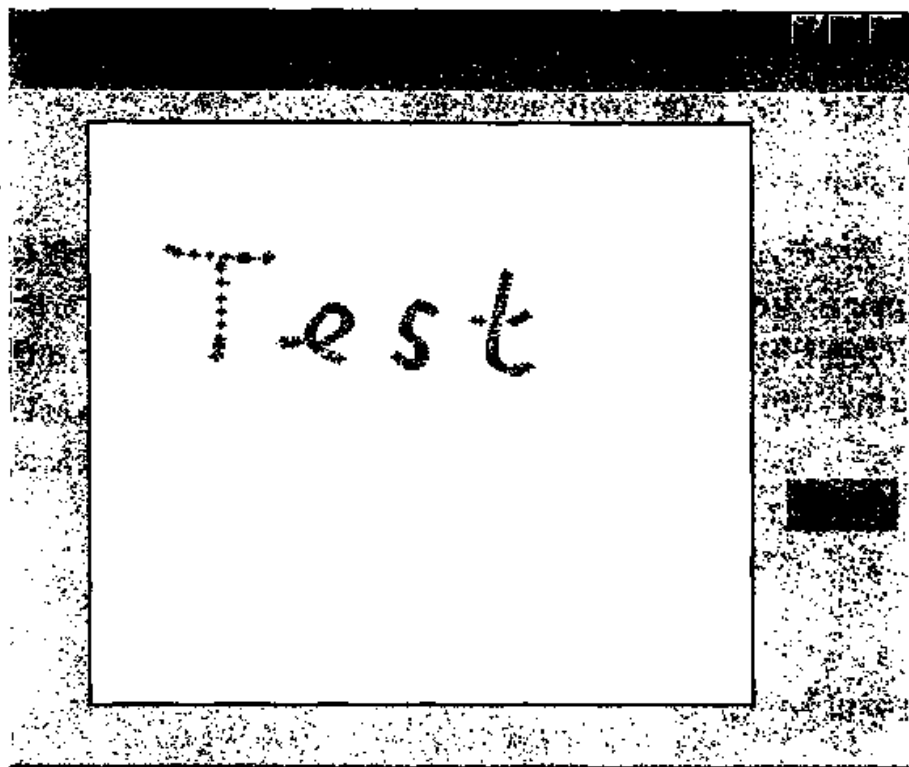


图 5.25 执行结果

本范例的程序为ex6.m。

本范例虽然简单，但若灵活运用WindowButtonMotionFcn， WindowButtonUpFcn及控制回调程序，并充分利用Currentpoint的变量值，就可设计出许多好的GUI应用程序。

## 5.9 Menu设计技巧

在图形上可以通过辅助控制面板上的菜单编辑器(Menu Editor)进行菜单(Menu)格式的设计。

**范例5.7 加入一个用于调整背景颜色的菜单。**

设计菜单名称为Figure，下面包括color，color又分为blue， white， gray三种颜色。建立该菜单的方法为：

单击“Menu Editor”后，单击“New Menu”按钮，分别输入所希望的名称和回调程序。如图5.26所示的是设定白色背景菜单项的输入信息，其回调程序为：

```
set(gcf,'color',[1 1 1])
```

[1 1 1]代表RGB(red green blue)的色度，最大值为1，最小值为0，在这里设为[1 1 1]，均为最大值，所以组合出来的颜色为白色。

在上节的例子中加入Figure菜单，如图5.27所示。



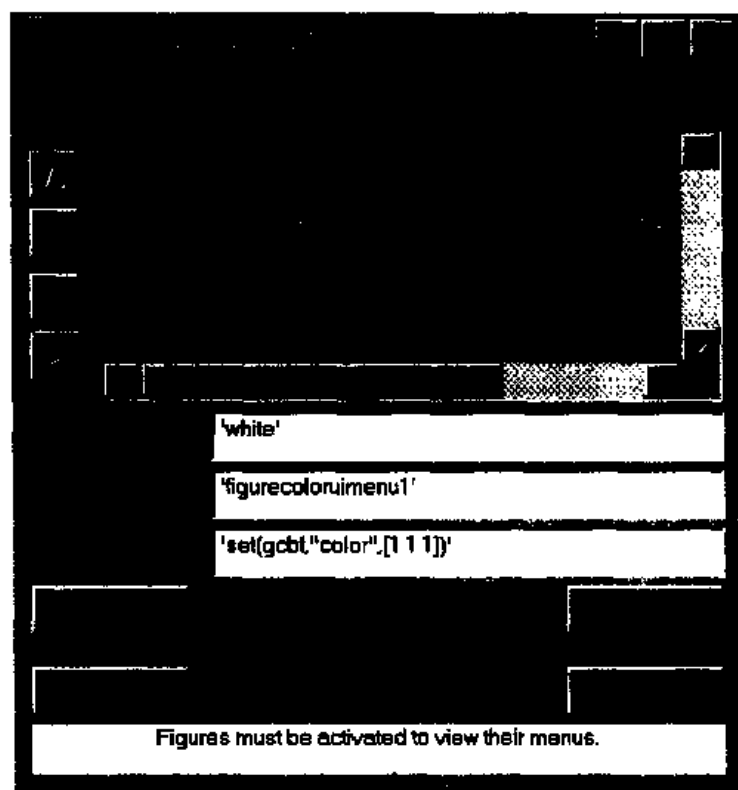


图 5.26 菜单设计

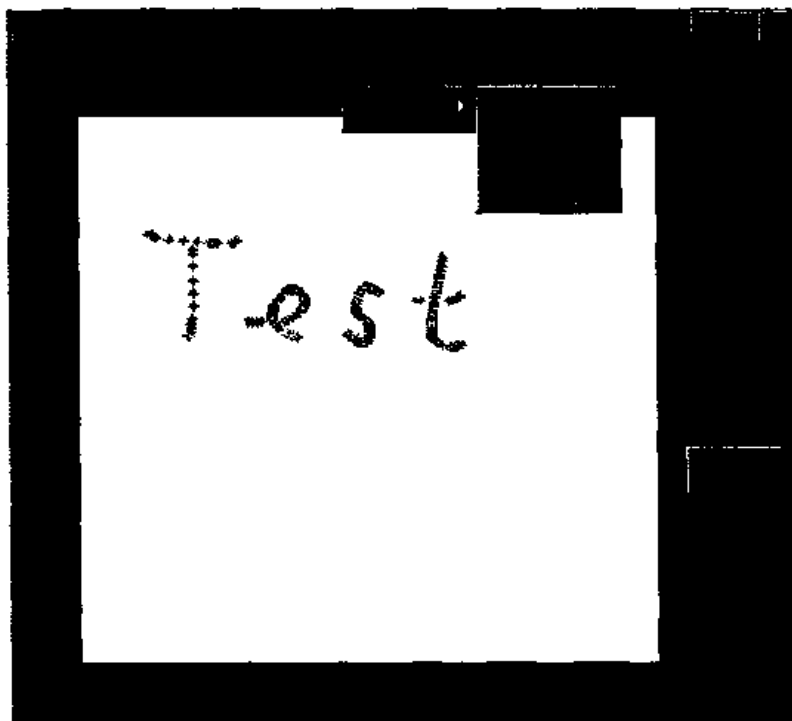


图 5.27 设计结果

本范例的程序为ex7.m。

## 5.10 中文对象设计

假如想设计一个汉化的GUI，必须在完成所有设计后，直接进入程序去改，不能通过辅助控制面板来设定，否则其处理过的属性值会转换成乱码，导致程序无法执行。只要电脑安装了中文字体，GUI就可以显示出来。

### 范例5.8 将Text改为中文字体。

以上节的例子来说，先到辅助控制面板上加入“Text”对象，然后到“Property Editor”中随便输入一个英文名称，然后退出，在命令窗口中打开相应的.m文件，找到String项，修改刚才输入的英文名称，假设叫做“中文画板”且其字体为宋体，而原来的“Clear”按钮也改为中文“清除”，则这部分程序内容如下：

```
b = uicontrol('Parent',a, ...  
    'Units','points', ...  
    'Callback','cla', ...  
    'FontAngle','oblique', ... 更改字体  
    'FontName','宋体',...  
    'FontSize',12, ... 字体大小  
    'Position', [359.379 109.862 52.1379 24.8276] , ...  
    'String','清除', ... 名称  
    'Tag','Pushbutton1');  
b = uicontrol('Parent',a, ...  
    'Units','points', ...  
    'BackgroundColor', [0.752941 0.752941 0.752941] , ...  
    'FontAngle','oblique', ...  
    'FontName','宋体', ...  
    'FontSize',16, ...  
    'Position', [97.4483 283.034 168.207 19.2414] , ...  
    'String','中文画板', ...  
    'Style','text', ...  
    'Tag','StaticText1');
```

保存好上面的程序后，即可在命令窗口中执行它，结果如图5.28所示。

本范例的程序为ex8.m。

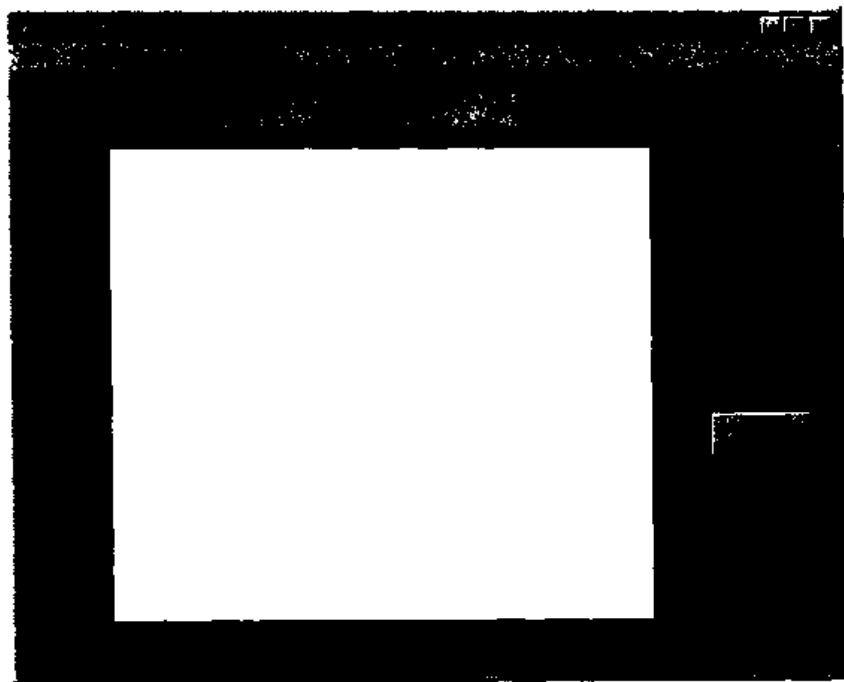


图 5.28 设计结果

### 5.11 Slider及Frame的设计技巧

本节主要说明slider及frame的用法。slider最主要的功能是使用户能够通过鼠标移动滑动杆来设定数值，而不必用键盘输入。该数值为0~1之间的实数，保存在该slider对象的“Value”属性值中。因此在设计时，可以根据需要由程序到该属性值中去获取“Value”的值。所用的命令在前面已经介绍过了，是findobj和get。frame则用于提供按钮的背景，改善应用程序的视觉效果。

下面就是沿用前面设计的画板例子，说明如何用slider设定画笔的颜色，并用frame将调色的slider合并起来，表示出这三个slider是一体的。

#### 范例5.9 调色板设计。

首先，到“新建对象面板”中点取slider移到图形上，接着点取frame移到图形上。用Alignment Tool调好其相关位置，然后保存文件。接着编写程序dircolr.m，内容如下：

程序：dircolr.m

```
function dircolr(direction);
switch direction
case 'start'
    %%%%Action of drawing
    set(gcf,'WindowButtonMotionFcn','dircolr move');
    set(gcf,'WindowButtonUpFcn','dircolr stop')
case 'move'
    %%%%set the pen color
```

```

s1=findobj(gcf,'Tag','Slider1');
s2=findobj(gcf,'Tag','Slider2');
s3=findobj(gcf,'Tag','Slider3');
r_s=get(s1,'Value');
g_s=get(s2,'Value');
b_s=get(s3,'Value');
c_p=get(gca,'currentpoint');
line(c_p(:,1),c_p(:,2),'marker','*','color',[r_s g_s b_s],'clipping','on',...
'erasemode','background')
case 'stop'
set(gcf,'WindowButtonMotionFcn','');
set(gcf,'WindowButtonUpFcn','')
end

```

寻找对象并设定色度

将设定值存入属性中

说明：与前面的dirmov.m不同之处在于增加了set pen color部分，先将slider1对象命名为s1，slider2命名为s2，slider3命名为s3。然后再用get指令去设定r-s、g-s、b-s的值，每当程序执行line指令的时候，由于其属性值color为[r-s g-s b-s]也就会跟着改变，从而实现了调色的目的。

写好程序之后，别忘了到“回调程序编辑器”中修改回调程序，修改后如图5.29所示。

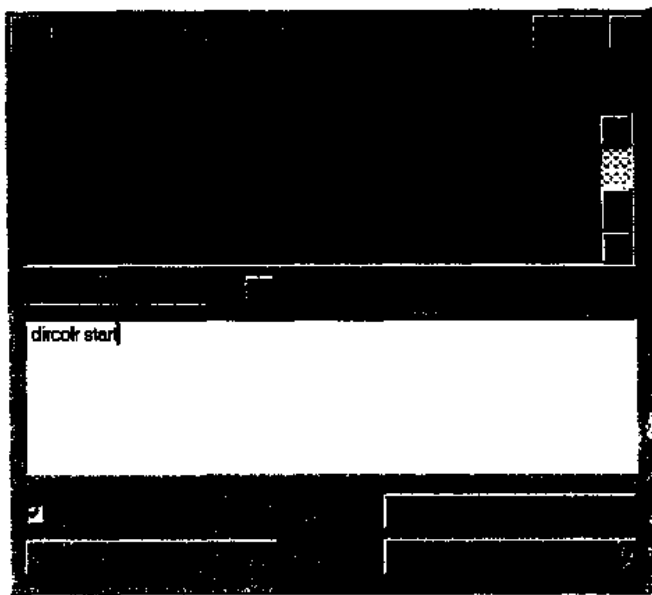


图 5.29 编辑回调程序

程序的执行结果如图5.30所示，读者可以根据自己的爱好来调整画笔的颜色，slider往右值愈大，最大为1，色度也最浓。从本节简单的例子中，读者一定已经了解到了slider的设计概念。如果将来需要设计更复杂的程序时，只要记得灵活运用属性值“Value”即可。例如，也可以将其当作一个比例系数来设计坐标轴的范围，当增大滚动条的值时，使其坐标刻度变大，这部分请读者找机会练习一下。使用时，先单击clear清除画板的内容，

然后用鼠标在上面写所要的内容即可。如图5.31所示。

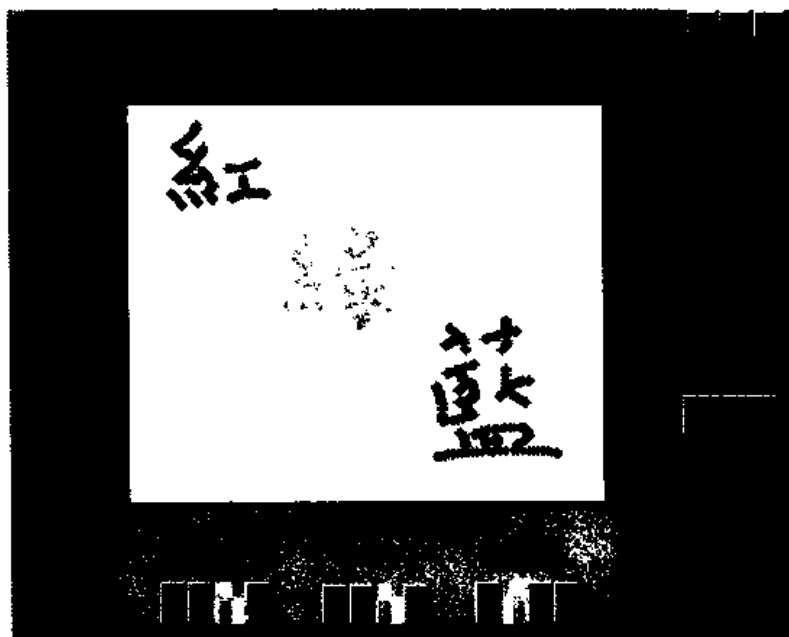


图 5.30 执行结果

本范例程序为ex9.m。

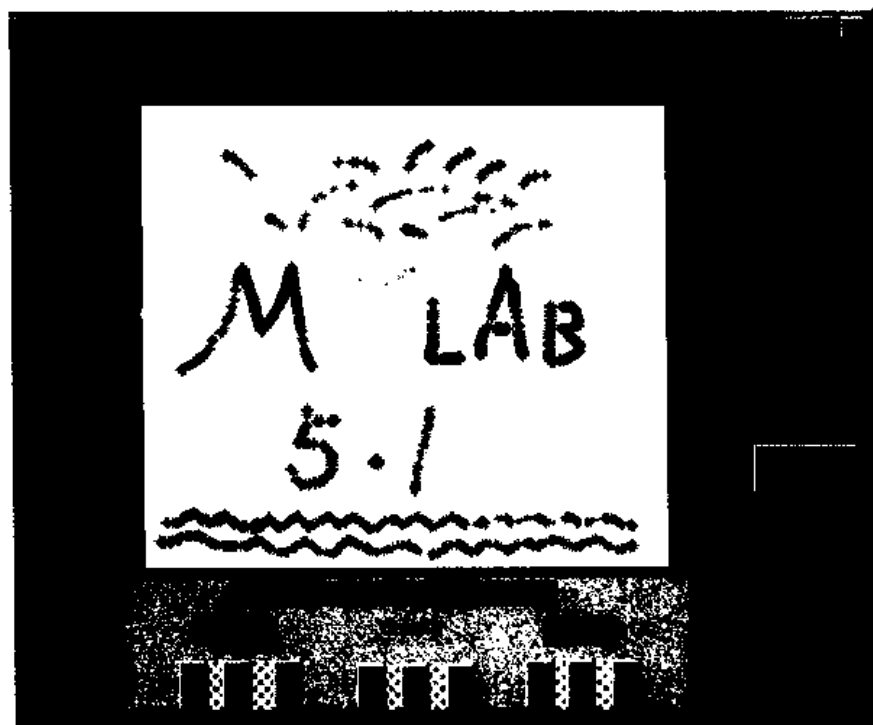


图 5.31 另一种执行结果

本范例的程序为ex9\_1.m。

## 5.12 Checkbox设计技巧

checkbox和radiobutton的设计技巧大同小异，两者的主要功能是做状态的判别。也就是说，程序会去验证用户有没有单击这个对象，如果单击了，则其属性值Value设为1，代表true；否则为0，代表false。在设计中通常使用if then来编写。

比如下面的程序结构：

```
if get(object_name, 'Value');
```

做值为true时的操作

```
else
```

做值为false时的操作

```
end
```

### 范例5.10 弹跳皮球设计。

本节以设计一个弹跳的皮球为例，当选中checkbox时，皮球弹跳；如果没有选中，皮球就原地不动。首先在图形上建立好Axes, pushbutton和checkbox，命名各个对象后，编写名为start的pushbutton的回调程序，如图5.32所示。

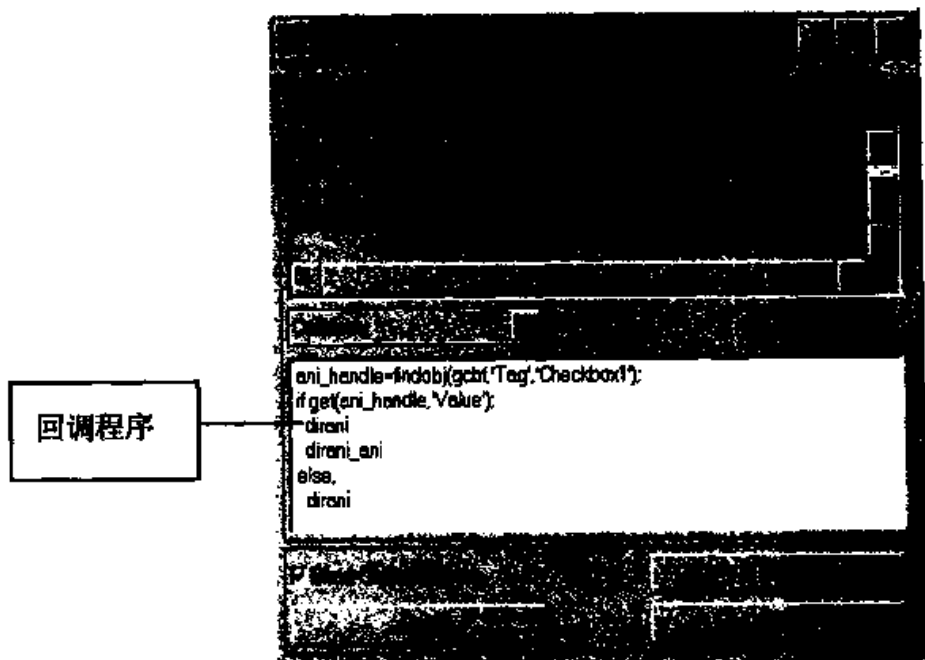


图 5.32 编辑回调程序

根据图5.32的回调程序可知，当checkbox被选中时，其value值为1，所以执行dirani和dirani\_ani两个程序使皮球能够跳动。如果checkbox没有被选中，则只执行dirani，皮球固定不动。所以接下来要编写这两个程序，内容如下：

程序: dirani.m

```
clear all
cla;
x = -pi:(pi/16):pi;
an_line=patch(0.3*sin(x),0.6*cos(x), 'r');
set(gca,'xtick', [], 'ytick', []);
set(gca, 'xticklabel', '', 'yticklabel', '');
```

画出皮球

清除坐标刻度

说明: 本程序的主要目的是画出固定的圆形皮球。

程序: dirani\_ani.m

```
echo on
pause_f=0
set(an_line, 'erasemode', 'xor');
for alpha= 0:pi/64:4*pi
    y_inc=get(an_line, 'ydata');
    set(an_line, 'ydata', y_inc, y_inc+0.1*cos(alpha));
    drawnow;
    if pause_f==1
        break
    end
end
echo off
```

缓和图形的显示

获取皮球的当前高度值

设定皮球弹跳的幅度

判断是否跳出

说明:

1. 用get指令获取皮球的当前高度, 用set指令调整皮球的高度, 在循环中进行运算。
2. 如果中途要跳出, 可以加入另一个循环, 判断是否已经设定了pause\_f, 如果已经设定, 就靠break指令跳出整个循环。

前面提到的pause\_f, 其值设定与否, 要看名为stop的pushbutton是否按下, 因此必须在其回调程序中输入:

```
pause_f=1
```

如图5.33所示。

整个设计完成后, 执行结果如图5.34所示。

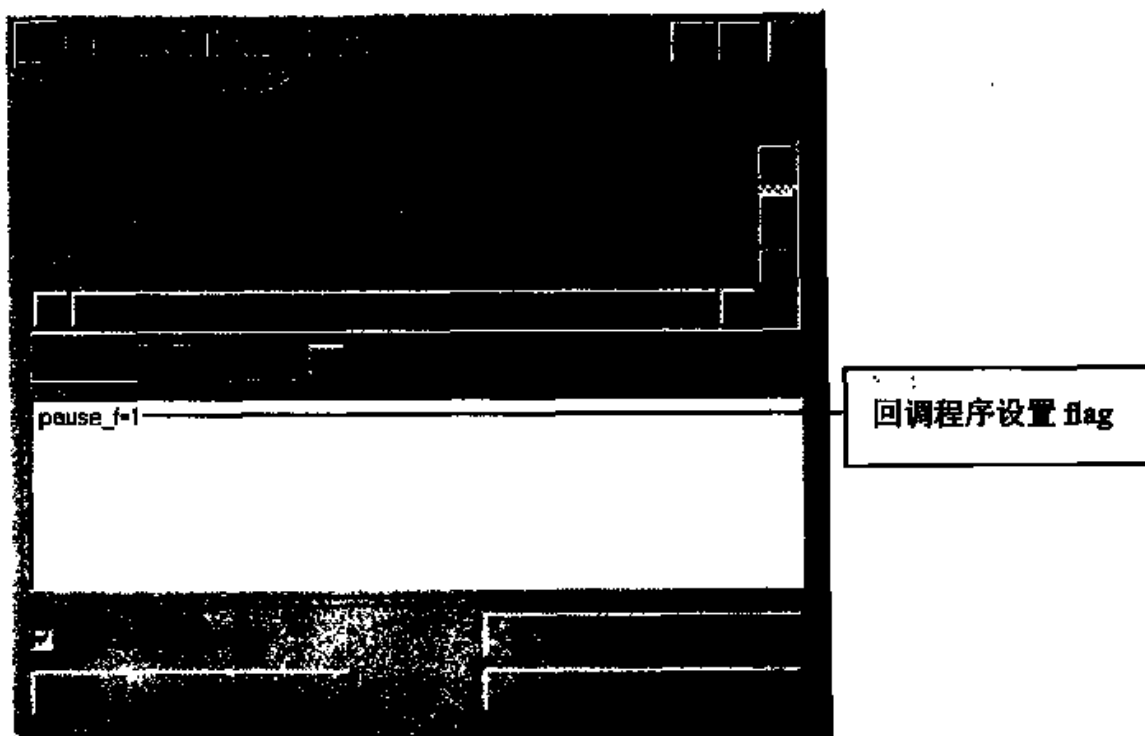


图 5.33 编辑回调程序

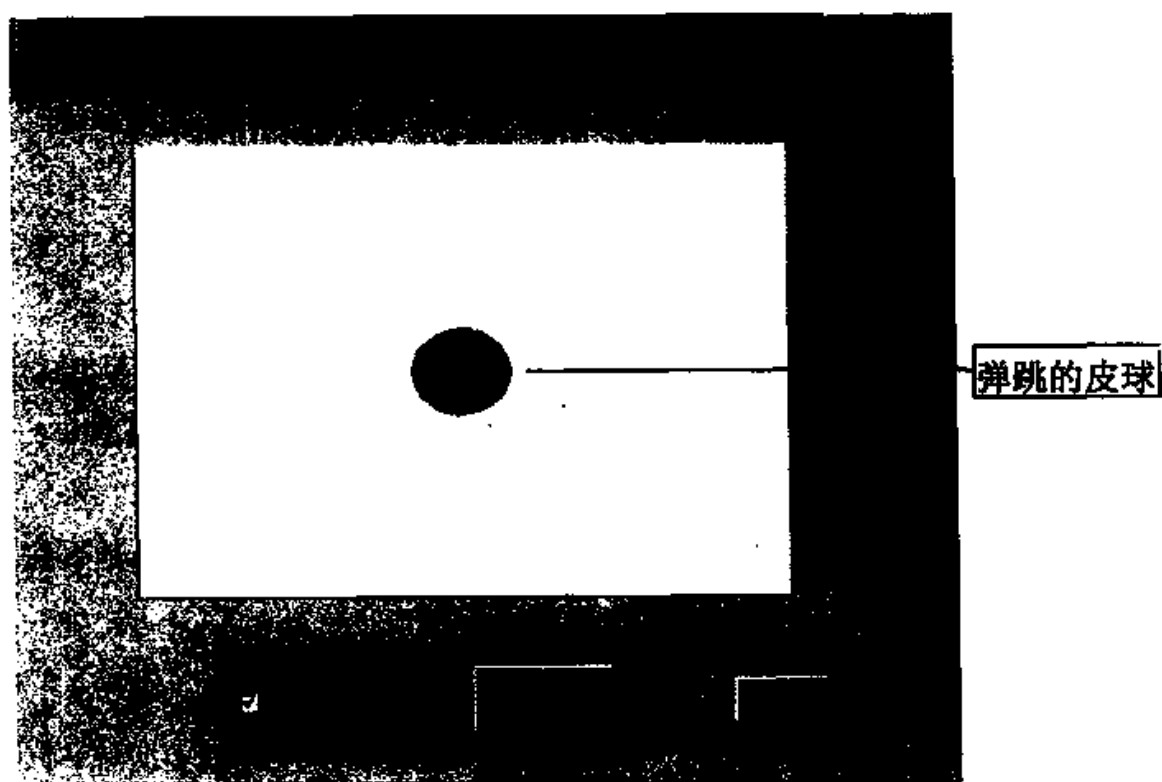


图 5.34 设计结果

本范例的程序为ex10.m。



## 第6章 图形处理技巧

本章主要介绍如何将设计的结果转化为精美的图形并表现出来。

### 6.1 基本的平面绘图

plot是最基本的绘图指令，也是最常用的指令，它的功能是把一组成对的数据分别画到x轴和y轴上。因此本章从plot开始介绍。

#### 6.1.1 plot的指令格式

`plot(x,y,'颜色+线型+标示')`

颜色：

黄	y
红	r
绿	g
蓝	b
白	w
黑	k
紫	m
青	c

线型：

实线	-
破折线	--
虚线	:
点划线	-.

符号：

正号	+
圆形字母	"o"
星号	*
点	.
交叉字母	"x"
方形单词	"square"

例如：

```
t=1:20
```

```
plot(t,sin(t),'r:square')
```

r为颜色，:为线型，square为符号

就可以画出图6.1。

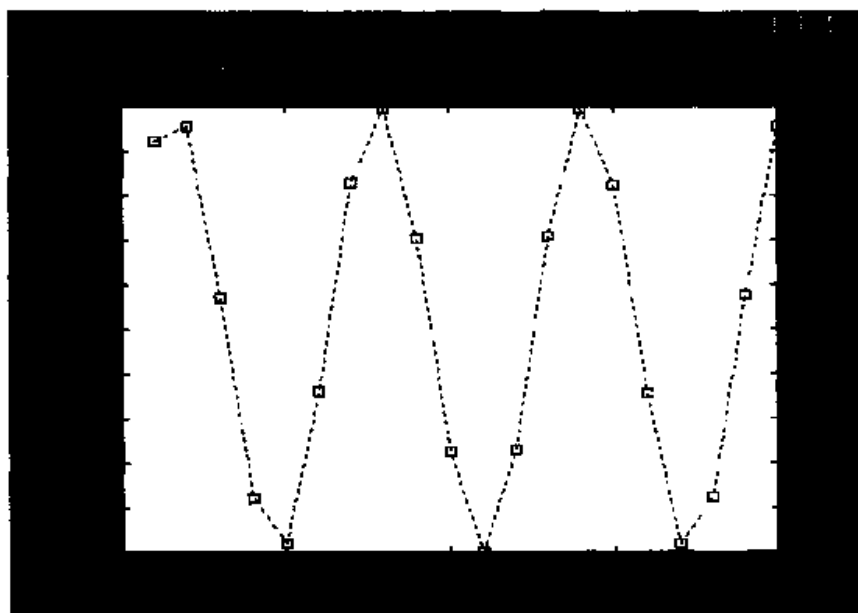


图 6.1

### 6.1.2 axis指令的格式

`axis( [xmin xmax ymin ymax] )`

针对图6.1, 输入指令:

`axis( [0 10 -1 1] )`

则x轴被限定在0与10之间, y轴被限定在-1与1之间, 如图6.2所示。

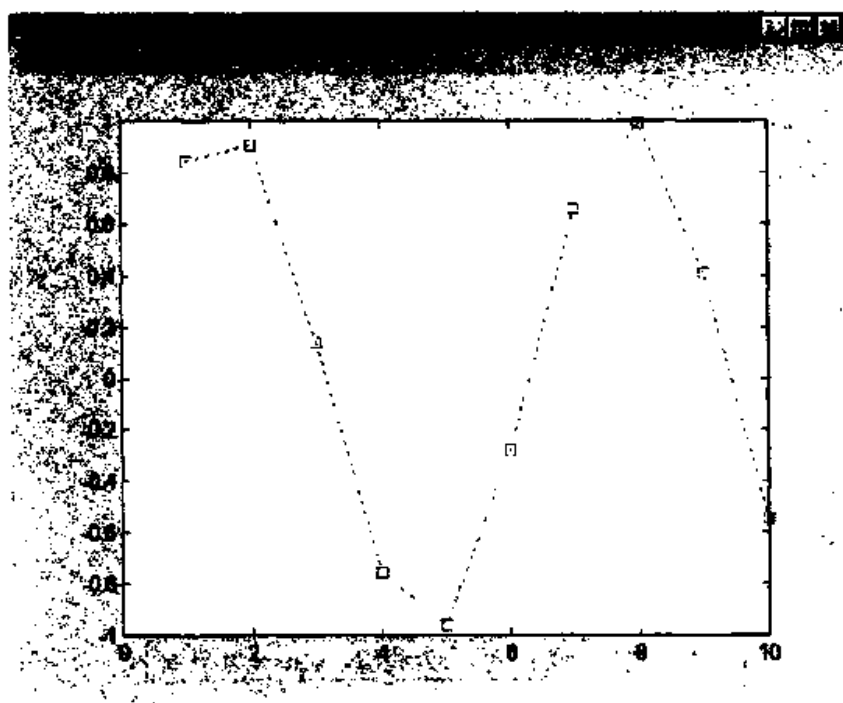


图 6.2

### 6.1.3 标示坐标刻度

利用上一章介绍过的set指令，设定xtick或ytick两个属性值。其指令格式为：

```
set(gca,'xtick',[1 2 5 10])
```

执行后，如图6.3所示。

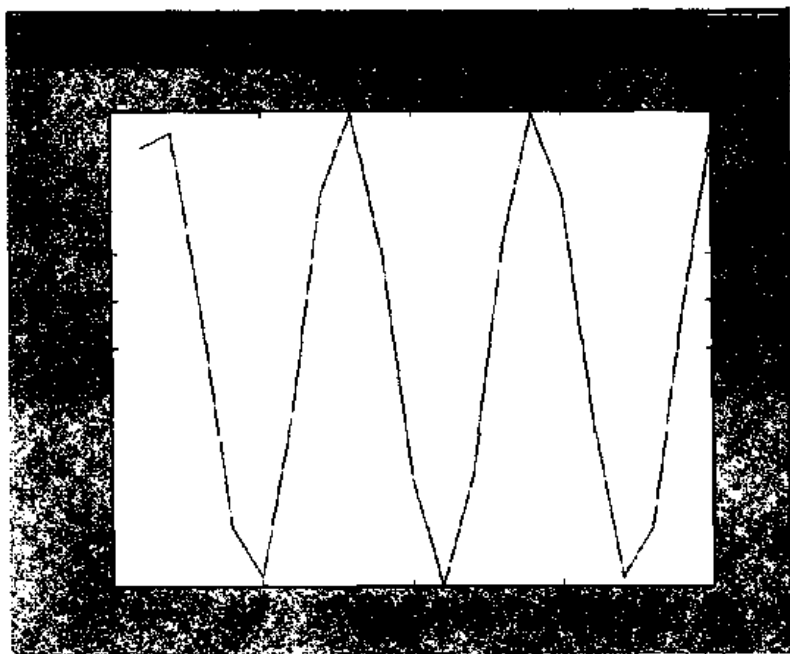


图 6.3

下面介绍yticklabel的用法：

先set(gca,'ytick',[0 1 2])，之后接着对应其刻度值执行：

```
set(gca,'yticklabel',{'a','b','c'})
```

其中：

0对应a

1对应b

2对应c

显示在y轴的刻度上。同理xtick, ztick, xticklabel, zticklabel的用法也是相同的。下面举例来说明。

在命令窗口中执行下面的指令：

```
t=1:20
```

```
plot(t,sin(t))
```

```
set(gca,'ytick',[-1 0 0.2 0.4 1])
```

```
set(gca,'yticklabel',{'-1|0|cutoff|0.4|1'})
```

0.2对应显示cutoff

结果如图6.4所示。

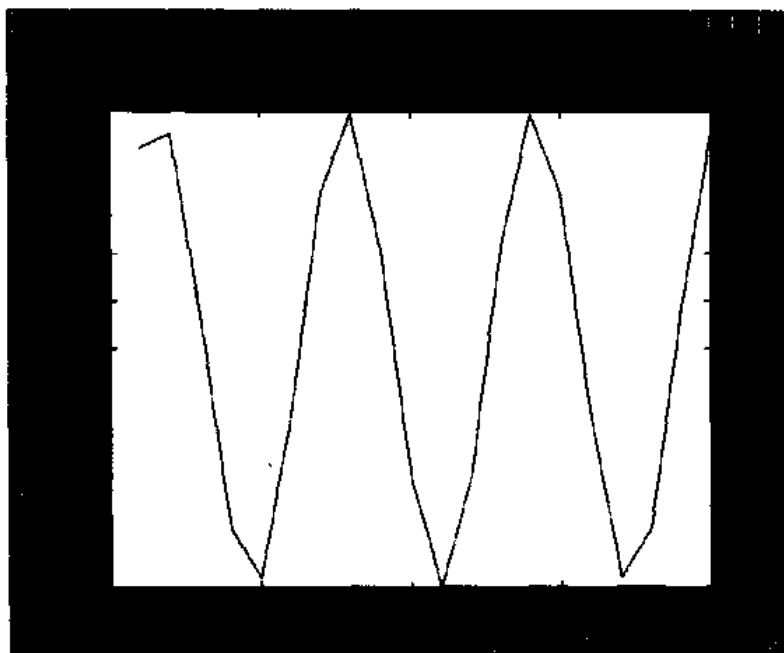


图 6.4

#### 6.1.4 文字标示

有关图形的标题、轴线说明和内容标示等的指令有：

title	图形标题
xlabel	x轴名称
ylabel	y轴名称
zlabel	z轴名称
text	创建说明文字
gtext	用鼠标在特定位置输入文字

其格式都必须用两个单引号(')括起来。输入特定的文字需要用反斜杠(\)开头，用法为：

字体italy	\it后面接{}, 在括号里面输入文字
符号 $\pi$	\pi
符号 $\alpha$	\alpha
符号 $\beta$	\beta
左箭头 $\leftarrow$	\leftarrow
右箭头 $\rightarrow$	\rightarrow
点号 $\bullet$	\bullet

可以通过下面两个参数设定符号相对于文字的方向：

- HorizontalAlignment 可以设定为left(默认值), center, right。
- VerticalAlignment 可以设定为middle(默认值), top, cap, baseline, bottom。

接着用几个例子来说明其使用方法。

### 范例6.1

程序: ex5601.m

```
t=0:0.1:3*pi
plot(t,sin(t));
xlabel('t(deg) ');
ylabel('magnitude');
title('\it{sine wave from zero to 3\pi}'); ———— 设定标题内容
```

执行结果如图6.5所示, 图上的标题处出现'sine wave from zero to  $3\pi$ '。

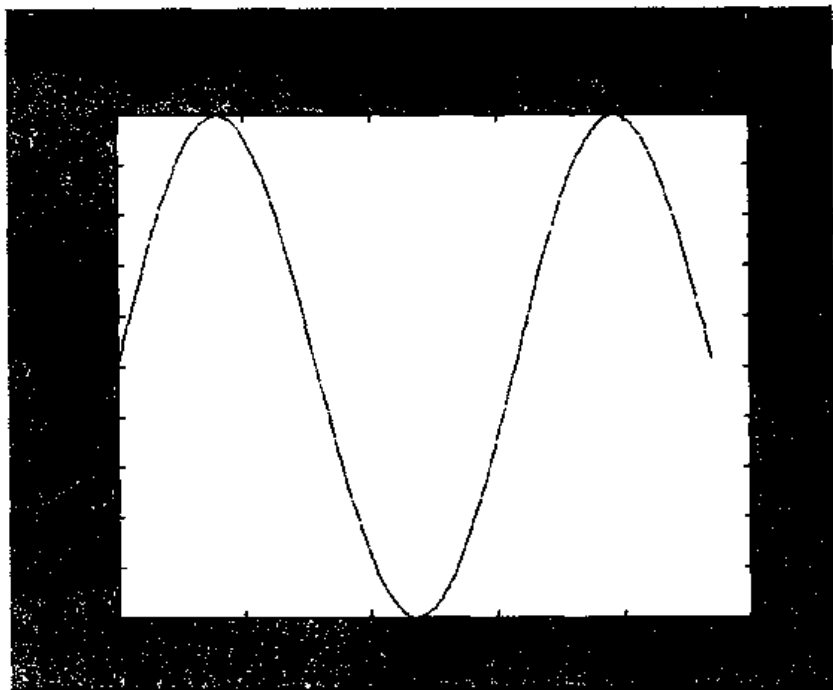


图 6.5

如果要在对象上标示文字, 可以使用text指令, 并配合反斜杠(\)设定文字样式。参见下面程序的说明。

### 范例6.2

程序: ex5602.m

```
t=0:0.1:3*pi
plot(t,sin(t));
xlabel('t(deg) ');
ylabel('magnitude');
title('\it{sine wave from zero to 3\pi}');
text(2,sin(2), '\bullet\leftarrow\fontname{times} The {\itsin(t)} at
{\itt}=2')
```

执行结果如图6.6所示。

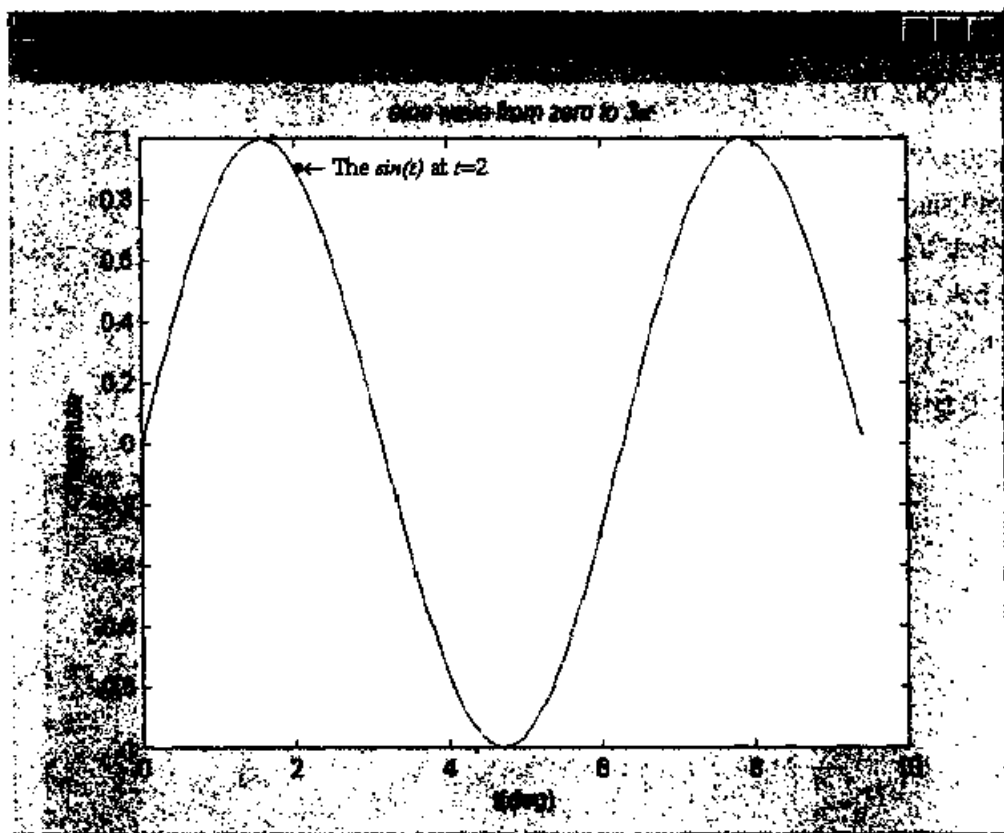


图 6.6

在本例中, fontname 设为 times, 表示字型为 Times New Romans。默认值是把对象放在文字的左边, 所以要先写 \bullet\leftarrow, text 指令才会依次把文字显示在图形上。假如希望标示在  $t=6$  时, 把对象放到文字右边, 可以将 HorizontalAlignment 的值改为 right, 并且一定要把 \rightarrow\bullet 放到最后, 参见下例。

### 范例 6.3

程序: ex5603.m

```
t=0:0.1:3*pi;
plot(t,sin(t));
xlabel('t(deg)');
ylabel('magnitude');
title('\it{sine wave from zero to 3\pi}');
text(6,sin(6), '\fontname{times}The {\itsin(t)} at
{\itt}=6\rightarrow\bullet', 'HorizontalAlignment','right');
```

执行结果如图6.7所示。

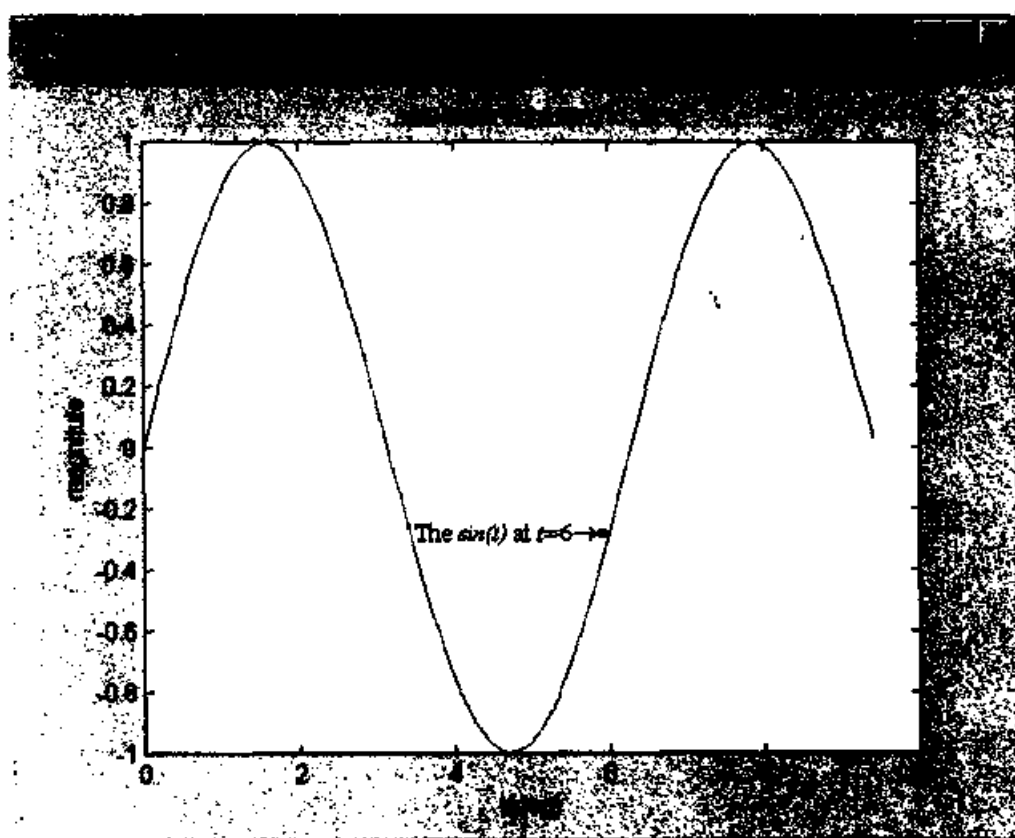


图 6.7

范例6.4 显示特殊符号的方法。

程序: ex5604.m

```
t=0:0.1:3*pi;
alpha=0:0.1:3*pi;
plot(t,sin(t), 'r-');hold on
plot(alpha,3*exp(-0.5*alpha), 'b: ');
xlabel('t(deg)');
ylabel('magnitude');
title('\it{sine wave and {\itAe}^{-\alpha{\it t}} wave from zero to 3\pi}');
text(6,sin(6), '\fontname{times}The {\itsin(t)} at
{\it t}=6\rightarrow\bullet',
'HorizontalAlignment','right');
text(2,3*exp(-0.5*2), '\bullet\leftarrow\fontname{times}The
{\it3e}^{-0.5{\it t}} at {\it t}=2',
'HorizontalAlignment','left');
```

显示特殊符号  $\alpha$

执行结果如图6.8所示。

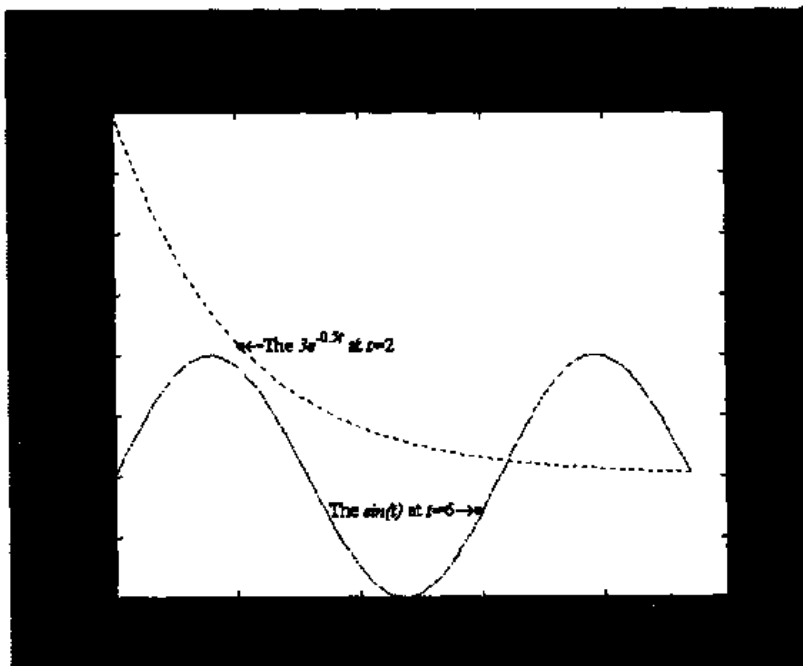


图 6.8

通过本例读者不难发现，与范例6.3的不同之处在于此程序增加了说明特殊符号的显示方法。而且在画两个以上的图形时，一定要加上

```
hold on
```

指令。如果要取消，执行

```
hold off
```

即可。

### 6.1.5 legend指令的格式

legend可用于区分图形上的线，这些线代表不同的含义，格式为：

```
legend('string1','string2')
```

string1表示第一个出现的线条。

string2表示第二个出现的线条。

例如范例6.4，显示两条线可以用legend指令，在图上显示图例，并且可以用鼠标来移动该图例的位置。

#### 范例6.5

程序：ex5605.m

```
t=0:0.1:3*pi
alpha=0:0.1:3*pi;
plot(t,sin(t),'r-');hold on
```



```

plot(alpha,3*exp(-0.5*alpha), 'b: ');
xlabel('t(deg) ');
ylabel('magnitude');
title('\it{sine wave and {\it{Ae}}^{-\alpha{\it{t}}} wave from zero
to 3\pi}');
text(6,sin(6), '\fontname{times} Value=',num2str(sin(6)), ' at
{\it{t}}=6\rightarrow\bullet' ,...
    '\HorizontalAlignment','right');
text(2,3*exp(-0.5*2), ['\bullet\leftarrow\fontname{times} The
{\it{3e}}^{-0.5{\it{t}}} at {\it{t}}=2' ],...
    '\HorizontalAlignment','left');
legend('sin(t) ','{\it{Ae}}^{-\alpha{\it{t}}}');

```

输入的字符串会依序显示在图形上

执行结果如图6.9所示。

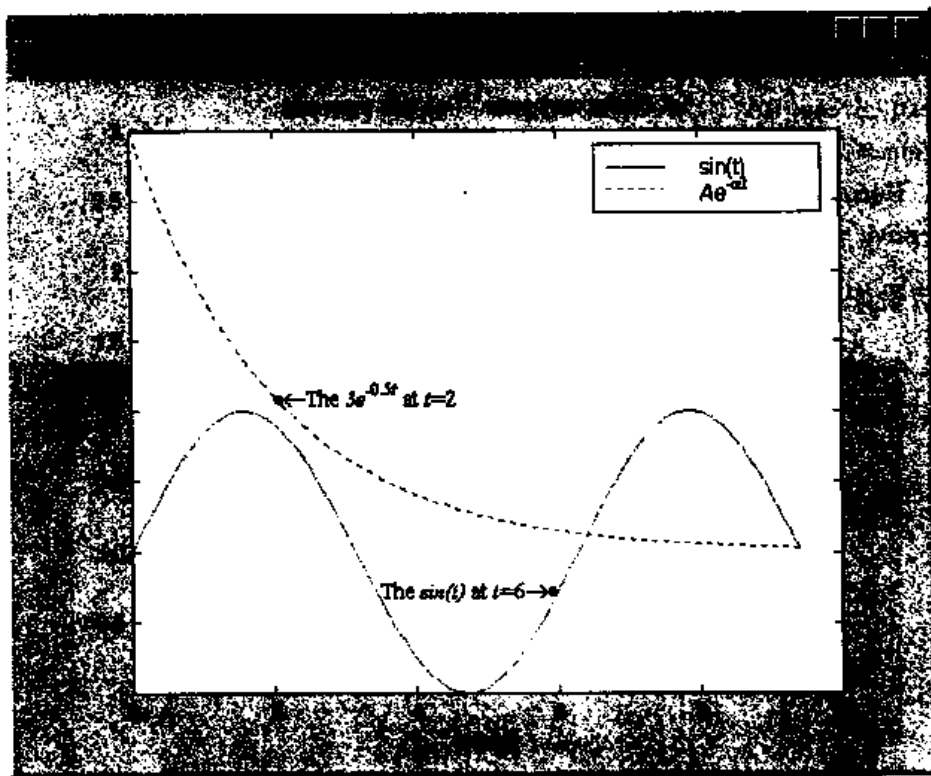


图 6.9

### 6.1.6 num2str指令的格式

如果希望在图形上显示一段固定的字符串和一段变量，就可以利用num2str指令。其格式为：

```
a=num2str(b)
```

执行之后就可以把数字b转换成字符串a。配合使用text指令，必须加上中括号"[]"，把表示的文字括起来。格式如下：

```
text( ['string1',num2str(number), 'string2'] )
```

继续范例6.5，把显示的值表示出来。

### 范例6.6

程序: ex5606.m

```
t=0:0.1:3*pi
alpha=0:0.1:3*pi;
plot(t,sin(t), 'r-');hold on
plot(alpha,3*exp(-0.5*alpha), 'b: ');
xlabel('t(rad) ');
ylabel('magnitude');
title('\it{sine wave and {\it{Ae}}^{-\alpha{\it{t}}} wave from zero
to 3\pi}');
text(6,sin(6), ['\fontname{times} Value=',num2str(sin(6)), 'at
{\it{t}}=6\rightarrow\bullet'], 'HorizontalAlignment','right');
text(2,3*exp(-0.5*2), ['\bullet\leftarrow\fontname{times}
Value=',num2str(3*exp(-0.5*2)), 'at {\it{t}}=2'], ...,
'HorizontalAlignment','left');
legend('sin(t)', '\it{Ae}^{-\alpha t}');
```

在这里输入

执行结果如图6.10所示。

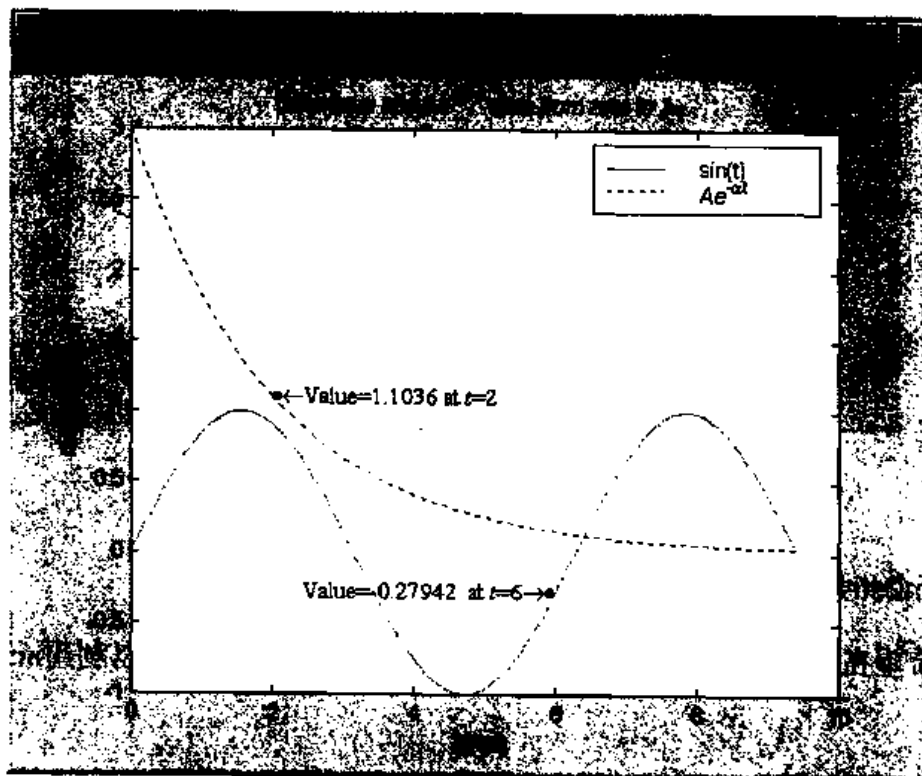


图 6.10

### 6.1.7 fill指令的格式

fill指令可以在封闭区域内涂满颜色，其指令格式为：

`fill(x,y,'color_style')`

下面的程序说明了如何画出布满颜色的圆形和方形。

#### 范例6.7

程序：ex5607.m

```
x = [1 1 5 5 1];  
y = [5 1 1 5 5];  
t = 0:0.01:2*pi;  
fill(5+x,5+y,'r');hold on  
fill(5*cos(t),5*sin(t),'b')  
axis([-20 20 -20 20]);
```

执行结果如图6.11所示。

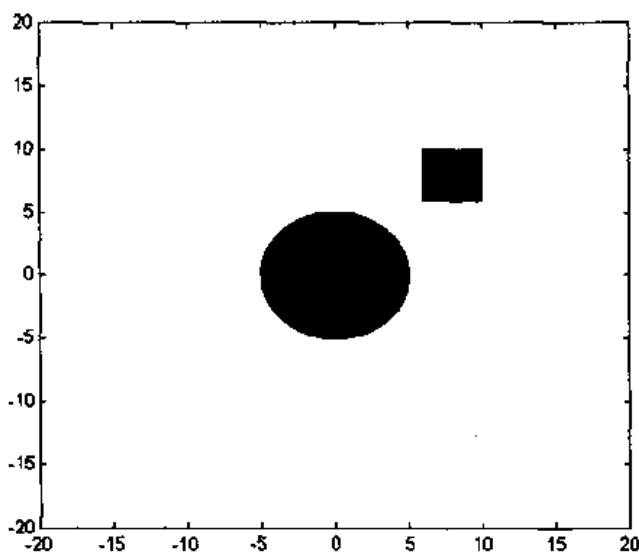


图 6.11

### 6.1.8 subplot的用法

如果希望在一个图形区域里表示多个图形，以便进行分析比较，就可以使用subplot指令。其格式为：

`subplot(mnk)`

m: 上下分割的个数。

n: 左右分割的个数。

k: 分割后的子图编号。

下面的范例说明了子图在图形中出现的顺序，同时使用：

**gtext**

指令通过鼠标输入图形文字。

### 范例6.8

程序：ex5608.m

```
figure(1);  
subplot(221);  
gtext('this is subplot(221) ');  
subplot(222);  
gtext('this is subplot(222) ');  
subplot(223);  
gtext('this is subplot(223) ');  
subplot(224);  
gtext('this is subplot(224) ');
```

在第一个子图中用鼠标输入字符串

执行结果如图6.12所示。

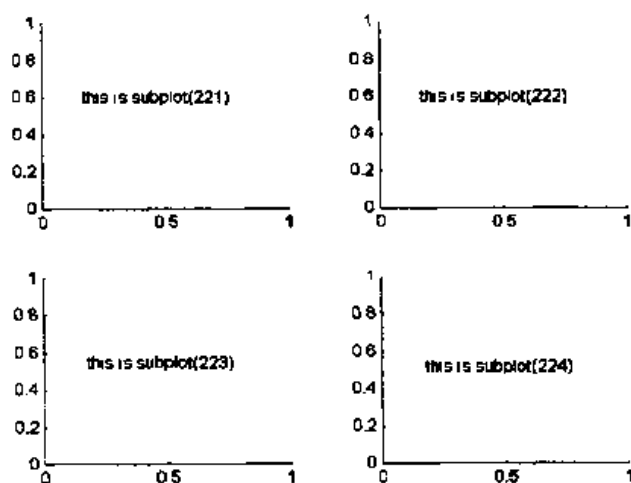


图 6.12

### 6.1.9 fplot指令的格式

对于以x为输入，以y为输出的函数关系式

$$y=f(x)$$

的绘图，可以使用fplot指令将某个函数区间的变化图画出来。其指令格式为：

**fplot('函数运算式', [xmin xmax])**

例如，要画出

$$y = \frac{1}{x^3 - 2x + 4}$$

其中x在0至10之间变化的图形,可以在命令窗口中执行:

```
fplot('1/(t.^3-2.*+4)', [0 10])
```

即可得到如图6.13所示的结果。

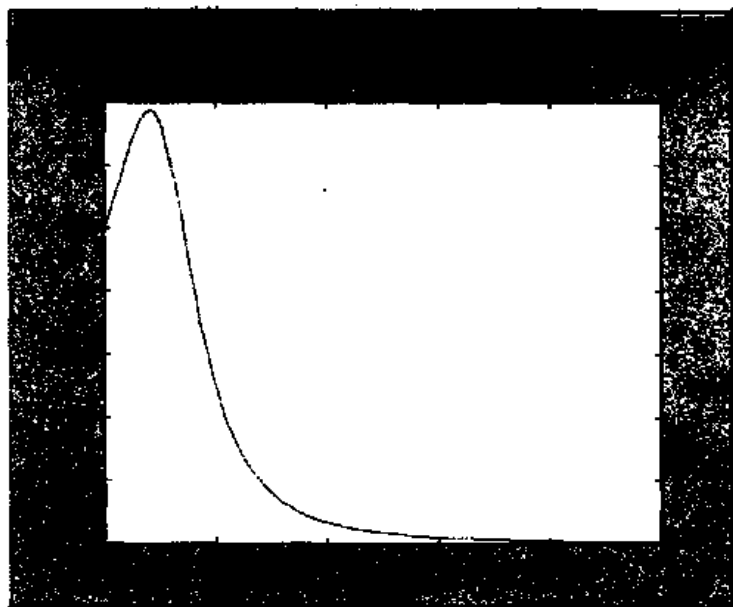


图 6.13

#### 6.1.10 应用绘图指令

MATLAB提供了许多绘图指令,可用于数值统计分析或离散数据处理,比如:

<code>bar(x,y)</code>	绘制对应于每个输入x的输出y的高度条形图(如范例6.9所示)。
<code>hist(y,x)</code>	绘制y在以x为中心的区间中分布的个数条形图(如范例6.10所示)。
<code>stairs(x,y)</code>	y对应于x的梯形图(如范例6.11所示)。
<code>stem(x,y)</code>	y对应于x的散点图(如范例6.12所示)。
<code>pie(vector,explode)</code>	显示各统计项目百分比(vector)分布的饼图,同时用explode凸出显示其中某一项(如范例6.13所示)。

#### 范例6.9

程序: ex5609.m

```
x = [1 2 3 4 5 6 7 8 9 10];
hight = [5 6 3 4 8 1 10 3 5 6]
```

```
bar(x,hight);
```

执行结果如图6.14所示。

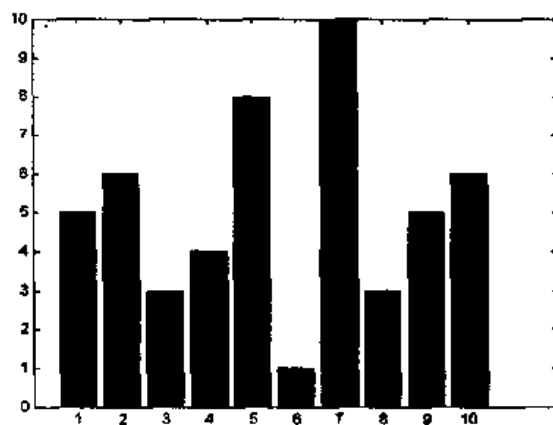


图 6.14

#### 范例6.10

程序: ex5610.m

```
hight=randn(1,1000)
```

```
y=-3:0.1:3;
```

```
hist(hight,y);
```

绘制1000个正态分布的随机数在[-3,3]区间的分布图

执行结果如图6.15所示。

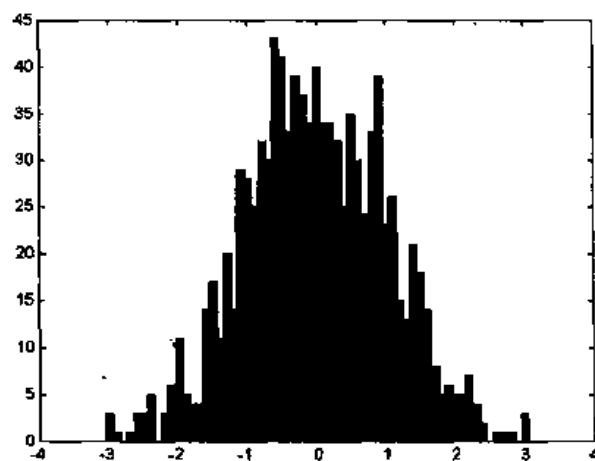


图 6.15

#### 范例6.11

程序: ex5611.m

```
x=0:0.1:10;
```

```
y=1./(x.^3-2.*x+4);
```

```
stairs(x,y);
```

执行结果如图6.16所示。

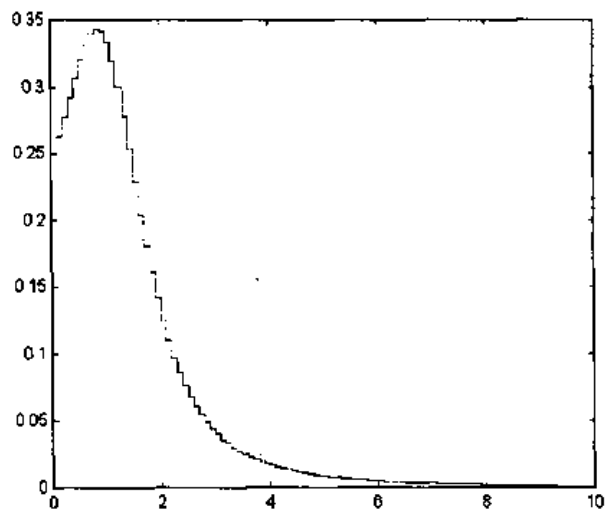


图 6.16

#### 范例6.12

程序: ex5612.m

```
x=0:0.1:10;  
y=1./(x.^3-2.*x+4);  
stem(x,y);
```

执行结果如图6.17所示。

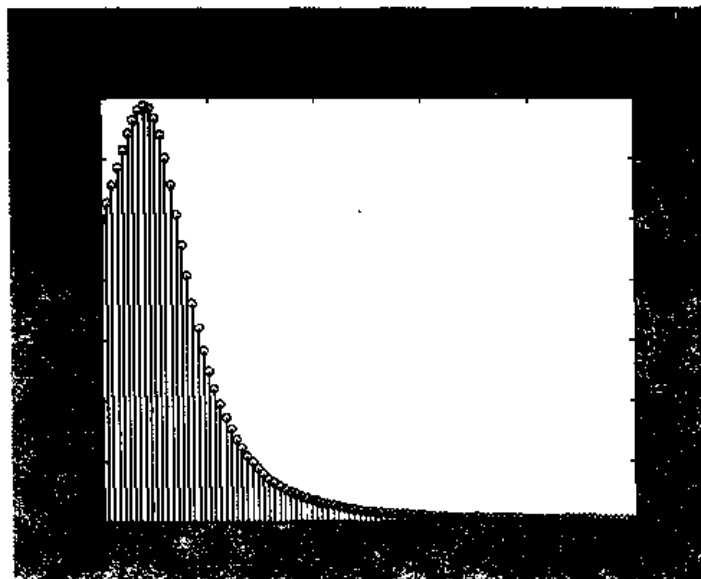


图 6.17

## 范例6.13

程序: ex5613\_1.m

```
clear
x= [5 20 38 22 15] ;
f1=pie(x);%plot figure
%%set the names
percent_num=findobj(f1,'type','text');%find the text content
percent_str=get(percent_num,{ 'string'});%get the text strings
names=str2mat('Petter','Alicce','John ','Merry','Hanson');%set names
new_str=strcat(names,percent_str);%concatenate the names and number
set(percent_num,{ 'string'},new_str);
%%adjust the position
offset1=0
offset2=-0.15%tune to left
offset3=0
offset4=0.2%tune to right
offset5=0.15
num_pos=get(percent_num,{ 'position'})%get current position
num_pos{1,1}(1,1)=num_pos{1,1}(1,1)+offset1
num_pos{2,1}(1,1)=num_pos{2,1}(1,1)+offset2
num_pos{3,1}(1,1)=num_pos{3,1}(1,1)+offset3
num_pos{4,1}(1,1)=num_pos{4,1}(1,1)+offset4
num_pos{5,1}(1,1)=num_pos{5,1}(1,1)+offset5
set(percent_num,{ 'position'},num_pos);%set new position
```

在本段程序的开始, 如果只使用pie(x), 画出来的结果则只是单纯的百分比图, 没有显示数值的归属, 如图6.18(a)所示。经过后面的程序处理后, 就可以显示出图形的百分比归属, 并可以调整其位置, 如图6.18(b)所示。此程序的处理用到了第5章介绍过的findobj, set, get指令, 这里就不再赘述。调整位置以及获得名称的部分运用到了单元数组(Cell Array)方面的指令, 详细内容可以到Help Desk中进行查询。如strcat是连接字符串(string concatenate)的意思, 目的是将strcat(a,b)中的a与b所对应的行向量连接起来表示。例如上面程序中的:

```
names=
Petter
Alicce
John
Merry
Hanson
```

而



```
percent_str=  
    '5%'  
    '20%'  
    '38%'  
    '22%'  
    '15%'
```

所以执行:

```
new_str=strcat(names,percent_str)
```

结果为:

```
new_str=  
    'Petter5%'  
    'Alice20%'  
    'John38%'  
    'Merry22%'  
    'Hanson15%'
```

接着要把new\_str写到饼图上。利用set指令把它填进对象percent\_num的string里面,但是位置会有偏差,所以要设定offset值来调整左边的宽度,负值会往左,正值则往右。注意,位置的校正变量是用Cell的方式表示的,所谓Cell就是以封装的方式将一些矩阵或字符串表示成一个数组,而其内容不会直接显示出来,比如:

```
num_pos=  
    [1x3 double]  
    [1x3 double]  
    [1x3 double]  
    [1x3 double]  
    [1x3 double]
```

其指标索引是用中括号{}表示的。对num\_pos这个数组而言,它只包含5个1×3的矩阵,指标索引为{1,1}到{5,1}。我们所关心的矩阵内容是左边的指标索引,所以取(1,1)来作偏移,表示出来的形式就是:

```
num_pos{1,1}(1,1)
```

代表num\_pos单元数组中第{1,1}位置的矩阵中的第(1,1)位置的值。

执行结果如图6.18(a)和图6.18(b)所示。

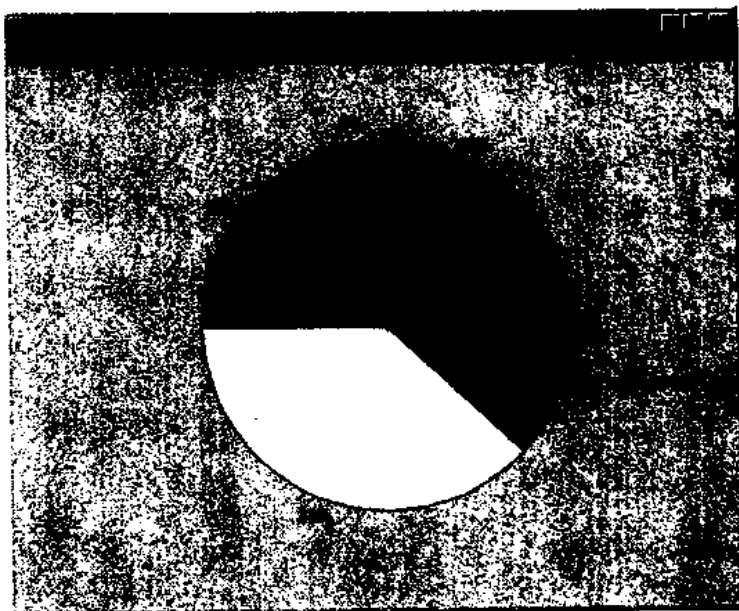


图 6.18(a) 未显示归属的饼图

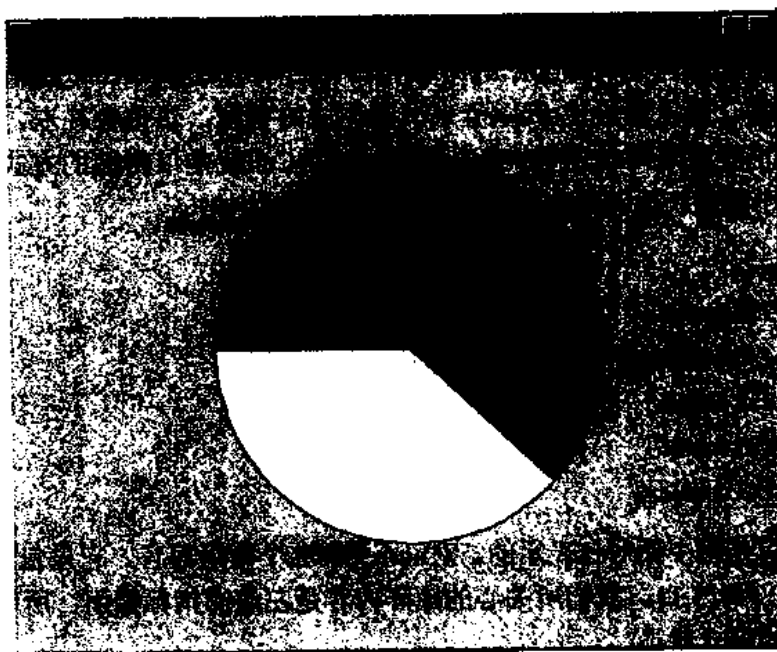


图 6.18(b) 显示归属的饼图

下面的程序其实与ex5613\_1完全一样，只是在pie指令中加入了凸出的参数而已，由 $x = [5 \ 20 \ 38 \ 22 \ 15]$ 可以知道，第三个占的比例最多，如果将其凸出0.1个单位，则凸出位置向量 $exp = [0 \ 0 \ 0.1 \ 0 \ 0]$ 。

程序：ex5613\_2.m

```
clear  
x = [5 20 38 22 15];
```

```
exp= [0 0 0.1 0 0] ;  
f1=pie(x,exp);%plot figure  
%%set the names  
percent_num=findobj(f1, 'type','text');%find the text content  
percent_str=get(percent_num, { 'string' });%get the text strings  
names=str2mat('Petter','Allice','John ','Merry','Hanson');%set names  
new_str=strcat(names,percent_str);%concatenate the names and number  
set(percent_num, { 'string' },new_str);  
%%adjust the position  
offset1=0  
offset2=-0.15%tune to left  
offset3=0  
offset4=0.2%tune to right  
offset5=0.15  
num_pos=get(percent_num, { 'position' })%get current position  
num_pos{1,1}(1,1)=num_pos{1,1}(1,1)+offset1  
num_pos{2,1}(1,1)=num_pos{2,1}(1,1)+offset2  
num_pos{3,1}(1,1)=num_pos{3,1}(1,1)+offset3  
num_pos{4,1}(1,1)=num_pos{4,1}(1,1)+offset4  
num_pos{5,1}(1,1)=num_pos{5,1}(1,1)+offset5  
set(percent_num, { 'position' },num_pos);%set new position
```

执行结果如图6.19所示。

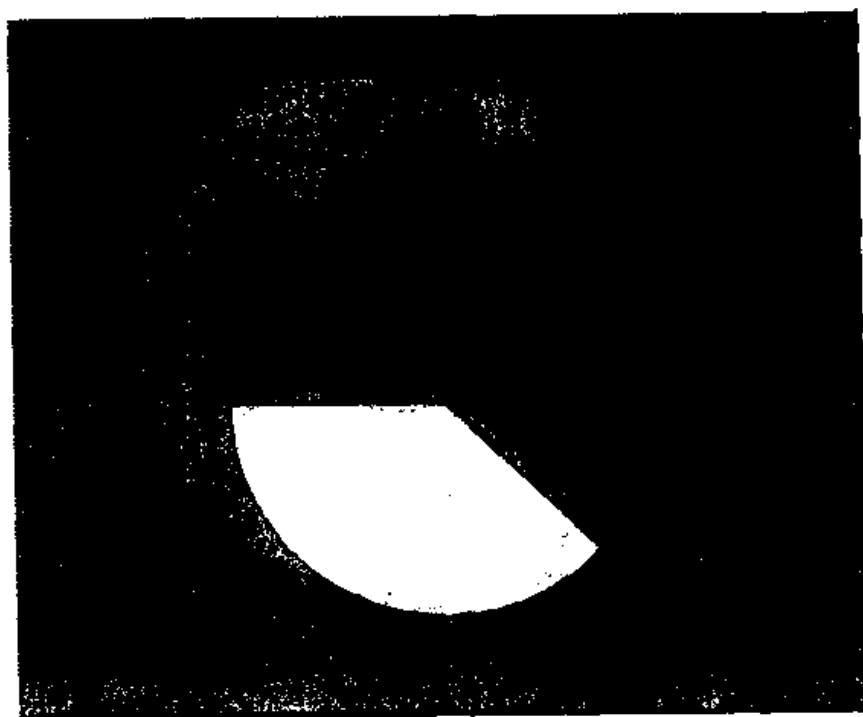


图 6.19 凸出的饼图

### 6.1.11 向量绘图指令

如果要做出表现切线、梯度或矢量大小等概念的图形,可以使用下面的指令来实现:  
**\*\*(x,y)**为笛卡尔坐标:

- compass(x,y)** 绘制x,y对原点的向量图形(参见范例6.14)。  
**feather(x,y)** 绘制x,y在水平线上的向量图形(参见范例6.15)。  
**quiver(x,y,u,v,s)** 在x,y位置上画出向量为(u,v)、大小为s的箭头。通常配合三维。图形的**contour**指令一起使用,在下节还会介绍。在本例中,用它来画出二维方程的切线向量图。如范例6.16所示。

#### 范例6.14

程序: ex5614.m

```
direction= [45 90] /180*pi;  
magnitude= [5 30] ;pol2cart  
[u,v] =pol2cart(direction,magnitude);  
compass(u,v);
```

在45° 方向上大小为5, 在90° 方向上大小为30

将极坐标转化为笛卡尔坐标

执行结果如图6.20所示。

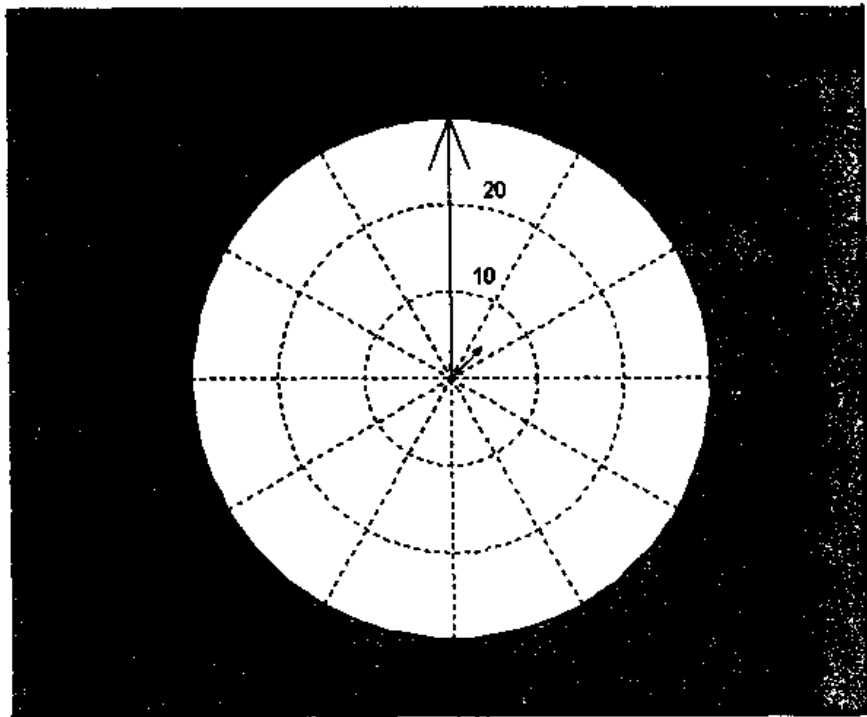


图 6.20

## 范例6.15

程序: ex5615.m

```
direction= [90:-10:0] /180*pi;  
magnitude=ones(size(direction));  
[u,v] =pol2cart(direction,magnitude);  
feather(u,v);  
axis equal
```

执行结果如图6.21所示。

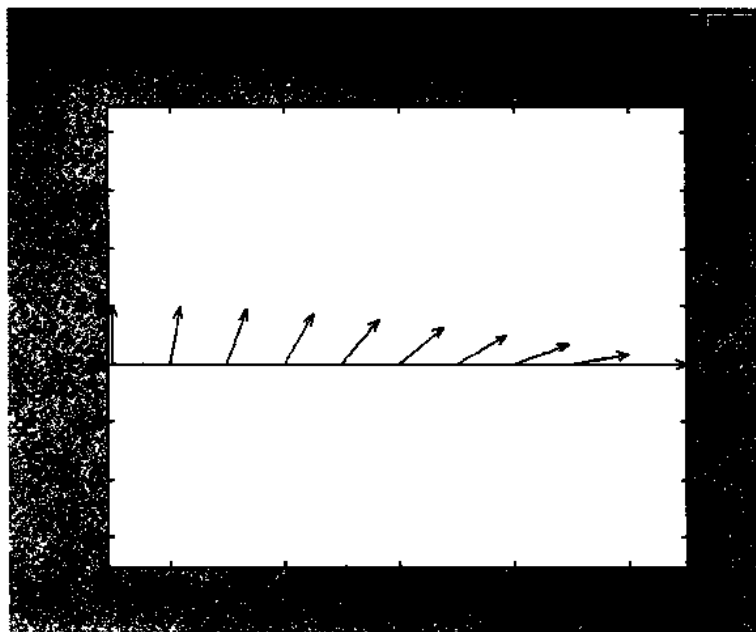


图 6.21

## 范例6.16

程序: ex5616.m

```
%2D quiver plot  
clear  
x=0:0.1:10;  
y=1./(x.^3-2.*x+4);  
u=gradient(x,0.1);  
v=gradient(y,0.1);  
quiver(x,y,u,v,0.2);hold on  
plot(x,y);
```

执行结果如图6.22所示。

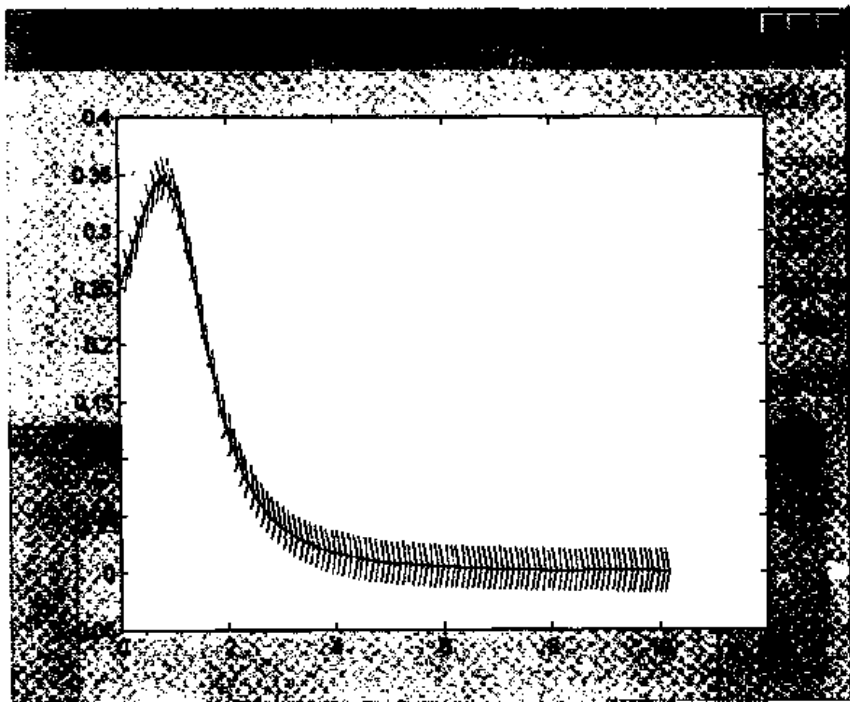


图 6.22

### 6.1.12 其他绘图指令

- loglog**      双对数坐标轴绘图。
- semilogx**     $x$ 为对数坐标轴,  $y$ 为线性坐标轴。
- semilogy**     $y$ 为对数坐标轴,  $x$ 为线性坐标轴。
- plotyy**      以同一个 $x$ 轴对应两个分别在图形左右二边的 $y$ 轴。格式为:  
`plotyy(x1,y1,x2,y2)`
- polar**       绘出极坐标图。
- line**        画线指令, 与`plot`相同。

以上的绘图指令除了`plotyy`之外, 其他指令与`plot`的格式相同, 均为:

绘图指令( $x$ 向的数值,  $y$ 向的数值, '颜色\_形式')

区别只是坐标的刻度与单位不同, 读者可以在命令窗口下执行:

**help 指令**

查看相关指令的格式。

## 6.2 基本的三维绘图

当所要绘制的数据有三组时, 就必须用三维图形来表示。最基本的应用就是画出 $z$ 值相对于 $xy$ 平面的高度图或平面图, 所用的指令就是`plot3`。画图的时候, 要很了解矩阵相对

关系的表示方法。因此，先介绍linspace，再依次介绍其他指令的用法。

### 6.2.1 产生三维图的参考点

所用的指令格式为：

`linspace(x1,x2,n)`

表示在x1与x2之间产生n个点，如果不指定n，就默认产生100个点。

另外还可以使用meshgrid指令，其格式为：

`[X,Y]=meshgrid(x,y)`

表示将 $1 \times m$ 的x向量与 $1 \times n$ 的y向量，复制成 $n \times m$ 的X矩阵，及 $n \times m$ 的Y矩阵。

例如：

`[X,Y]=meshgrid([1 2 3],[4 5 6 7])`

则执行后的结果为：

X=

1 2 3

1 2 3

1 2 3

1 2 3

其行数为4，等于y转置的行

Y=

4 4 4

5 5 5

6 6 6

7 7 7

其列数为3，等于x的列数

### 6.2.2 plot3 指令的格式

`plot3(x,y,z)`

和plot指令相同，只是绘图的维数增加为三维。如范例6.17所示。

范例8.17

程序：ex5617.m

```
%3D plot3 plot
clear
x=0:0.1:10;
y=1./(x.^3-2.*x+4);
z=0:0.1:10;
plot3(x,y,z);
xlabel('x');
```

```

ylabel('y');
xlabel('x');
grid on
box on

```

执行结果如图6.23所示。

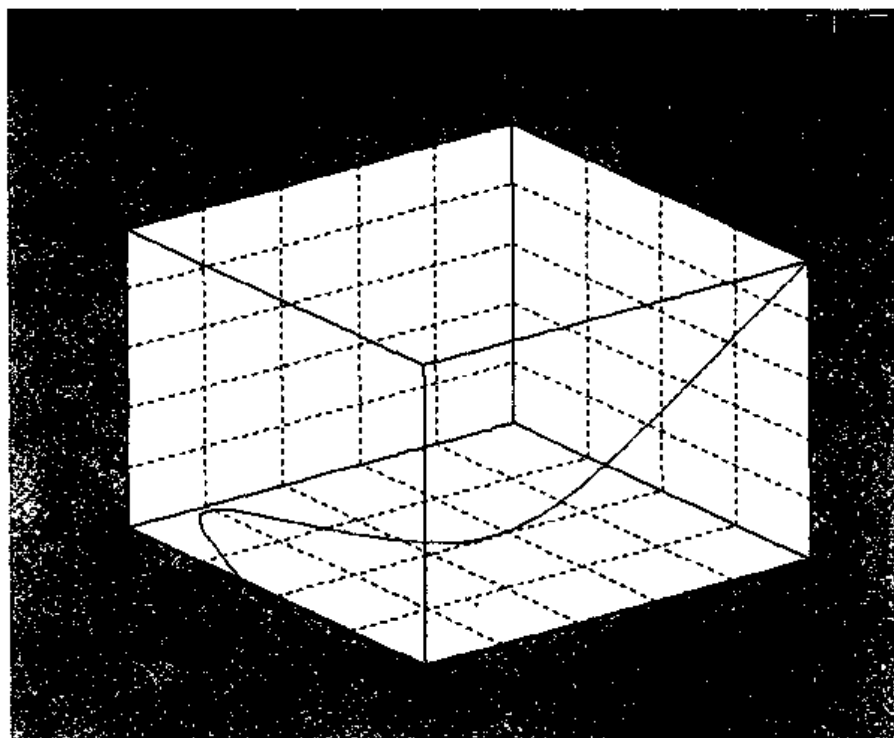


图 6.23

### 6.2.3 mesh及meshz的指令格式

`mesh(x,y,z)`

在三维空间中画出  $(x,y,z)$  表示的平面。把上节画二维图形的例子表现在三维图形上，如范例6.18所示。

`meshz(x,y,z)`

除了具有mesh的功能之外，还可以画出上下高度线。如范例6.19所示。

范例6.18

程序: ex5618.m

```

%3D mesh plot
clear
x=0:0.1:10;
[x,y]=meshgrid(linspace(0,10),linspace(0,10));

```

在0和10之间等分100个



```
z=(1./(x.^3-2.*x+4))+(1./(y.^3-2.*y+4));  
mesh(x,y,z);  
xlabel('x');  
ylabel('y');  
zlabel('z');  
grid on  
box on
```

执行结果如图6.24所示。

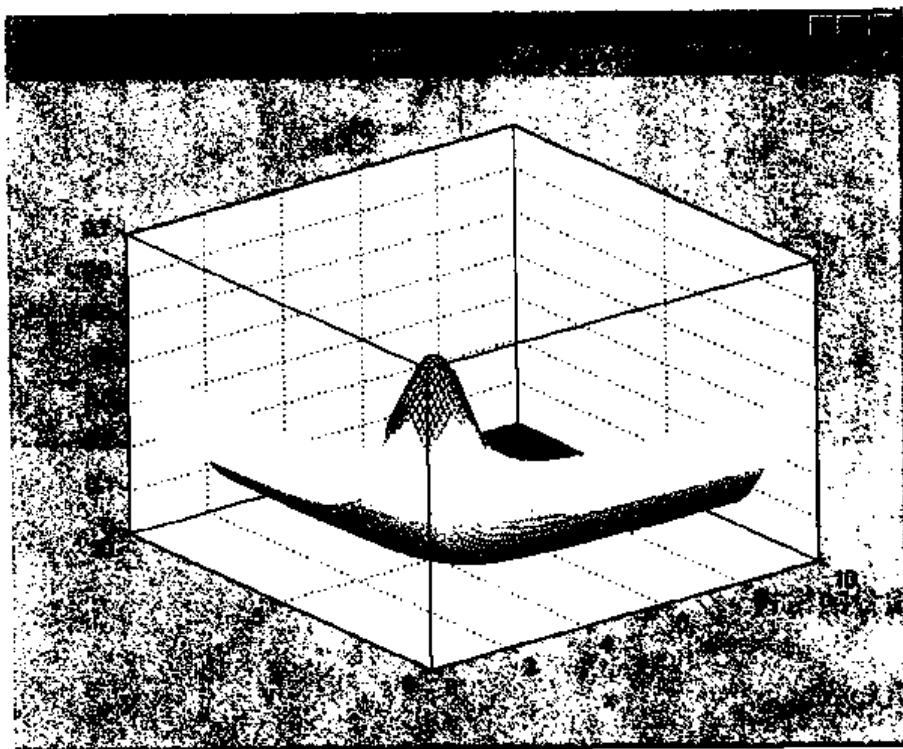


图 6.24

### 范例6.19

程序: ex5618\_1.m

```
%3D meshz plot  
clear  
x=0:0.1:10;  
[x,y]=meshgrid(linspace(0,10),linspace(0,10));  
z=(1./(x.^3-2.*x+4))+(1./(y.^3-2.*y+4));  
meshz(x,y,z);  
xlabel('x');  
ylabel('y');  
zlabel('z');  
grid on  
box on
```

执行结果如图6.25所示。

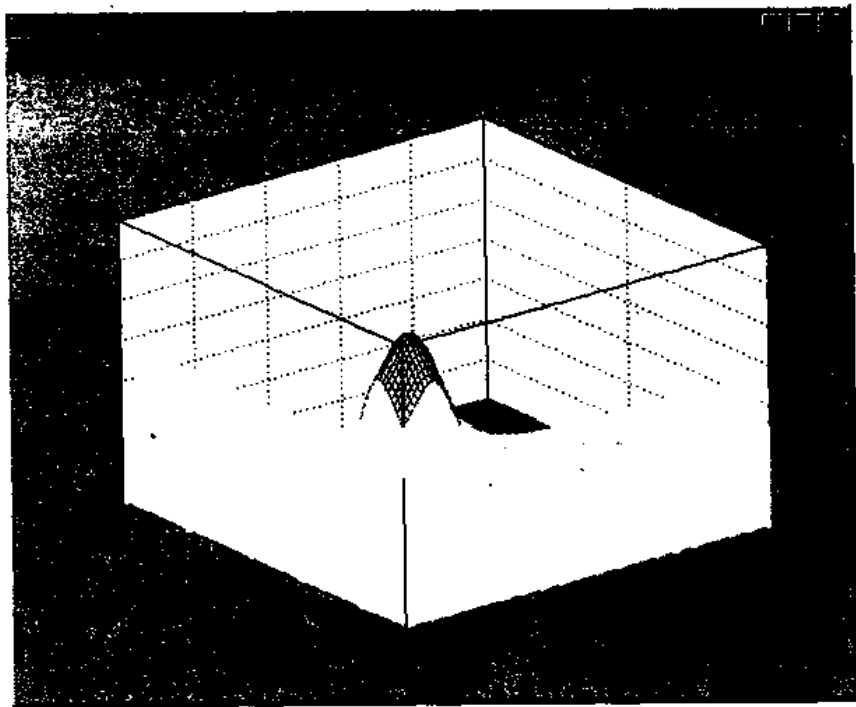


图 6.25

#### 6.2.4 surf指令的格式

`surf(x,y,z)`

也是画三维图形的指令，与`mesh`指令的差别在于`surf`表现出彩色的面，而`mesh`则表现出彩色的线，如范例6.20所示，注意与范例6.18的差异。

如果加上下列指令，就可以使`surf`的图形格线消失。如范例6.21所示。

`shading flat(interp)`

##### 范例6.20

程序: ex5619.m

```
%3D surf plot
clear
x=0:0.1:10;
[x,y]=meshgrid(linspace(0,10),linspace(0,10));
z=(1./(x.^3-2.*x+4))+(1./(y.^3-2.*y+4));
surf(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z');
```

```
grid on
```

```
box on
```

执行结果如图6.26所示。

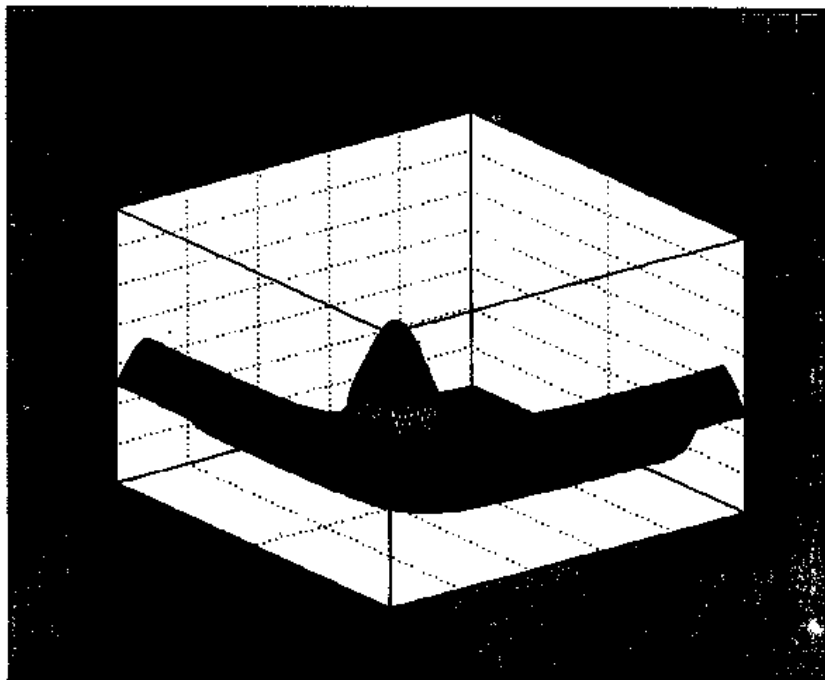


图 6.26

#### 范例6.21

程序: ex5619\_1.m

```
%3D surf plot plus shading command  
clear  
x=0:0.1:10;  
[x,y]=meshgrid(linspace(0,10),linspace(0,10));  
z=(1./(x.^3-2.*x+4))+(1./(y.^3-2.*y+4));  
surf(x,y,z);  
shading interp;  
xlabel('x');  
ylabel('y');  
zlabel('z');  
grid on  
box on
```

执行结果如图6.27所示。

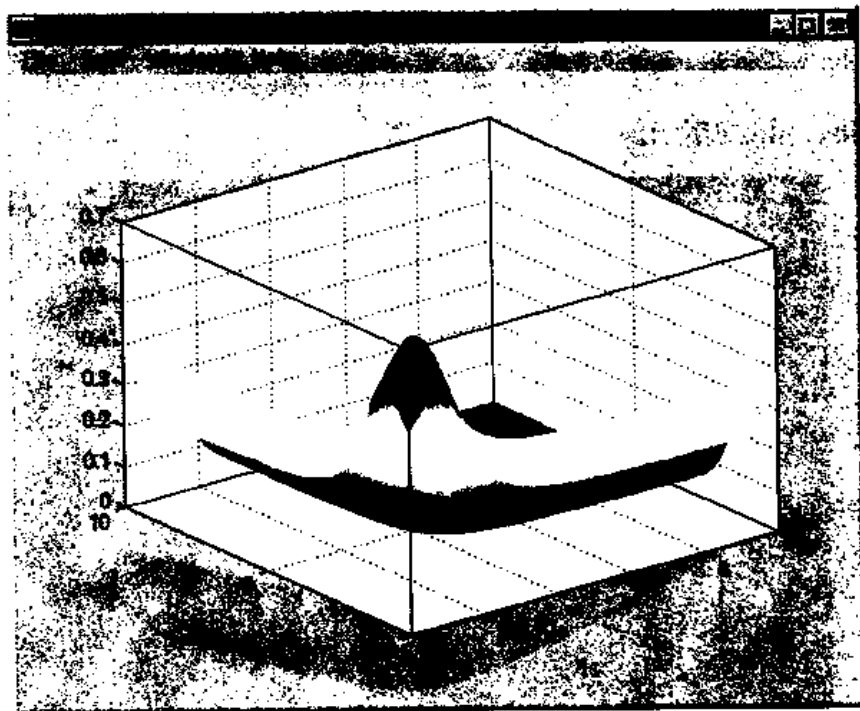


图 6.27

### 6.2.5 contour, contour3及contourf的指令格式

`contour(x,y,z,n)`

表示将空间图形水平切割所得的等高线,表现在x-y平面上,切割次数为n。如范例6.22所示。

`contour3(x,y,z,n)`

表示将空间图形切割后仍表现在空间上,切割次数为n。如范例6.23所示。

`contourf(x,y,z,n)`

表示将contour的图形涂满色彩,如范例6.24所示。

#### 范例6.22

程序: ex5620.m

```
%3D contour plot
clear
x=0:0.1:10;
[x,y]=meshgrid(linspace(0,10),linspace(0,10));
z=(1/(x.^3-2.*x+4))+(1/(y.^3-2.*y+4));
```

```
contour(x,y,z,20);  
xlabel('x');  
ylabel('y');  
zlabel('z');  
grid on  
box on
```

执行结果如图6.28所示。

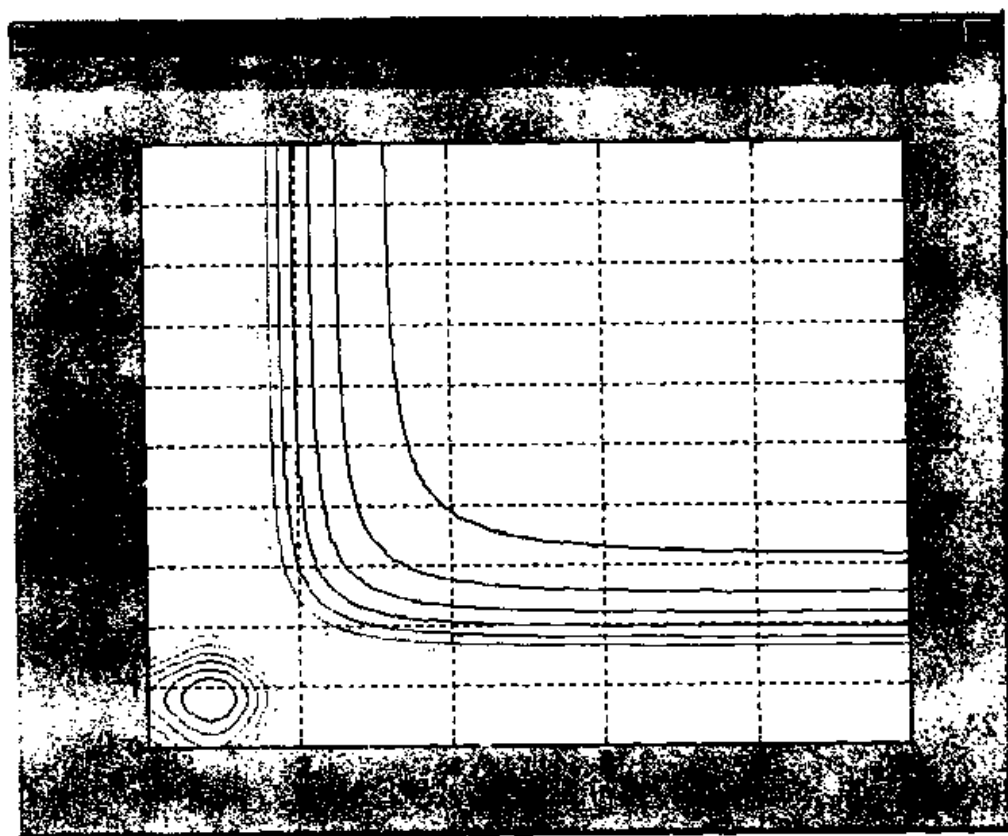


图 6.28

### 范例6.23

程序: ex5620\_1.m

```
%3D contourf3 plot  
clear  
x=0:0.1:10;  
[x,y]=meshgrid(linspace(0,10),linspace(0,10));  
z=(1./(x.^3-2.*x+4)).+(1./(y.^3-2.*y+4));  
contourf(x,y,z,20);  
xlabel('x');  
ylabel('y');
```

```
zlabel('z');
```

```
grid on
```

```
box on
```

执行结果如图6.29所示。

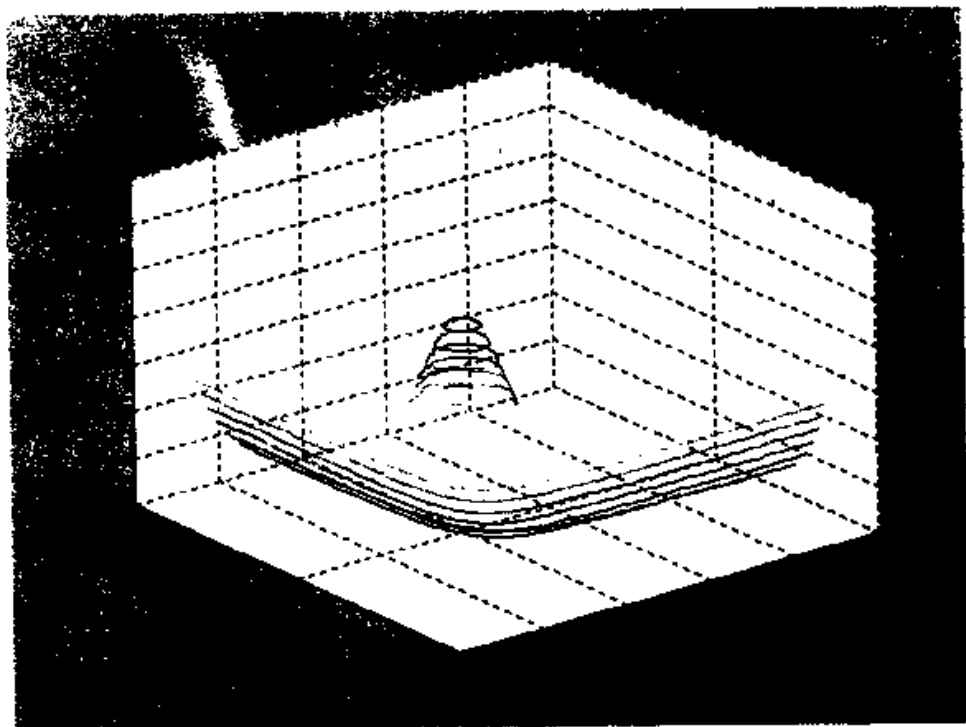


图 6.29

#### 范例6.24

程序: ex5620\_2.m

```
%3D contourf plot  
clear  
x=0:0.1:10;  
[x,y]=meshgrid(linspace(0,10),linspace(0,10));  
z=(1./(x.^3-2.*x+4))+(1./(y.^3-2.*y+4));  
contourf(x,y,z,20);  
xlabel('x');  
ylabel('y');  
zlabel('z');  
grid on  
box on
```

执行结果如图6.30所示。

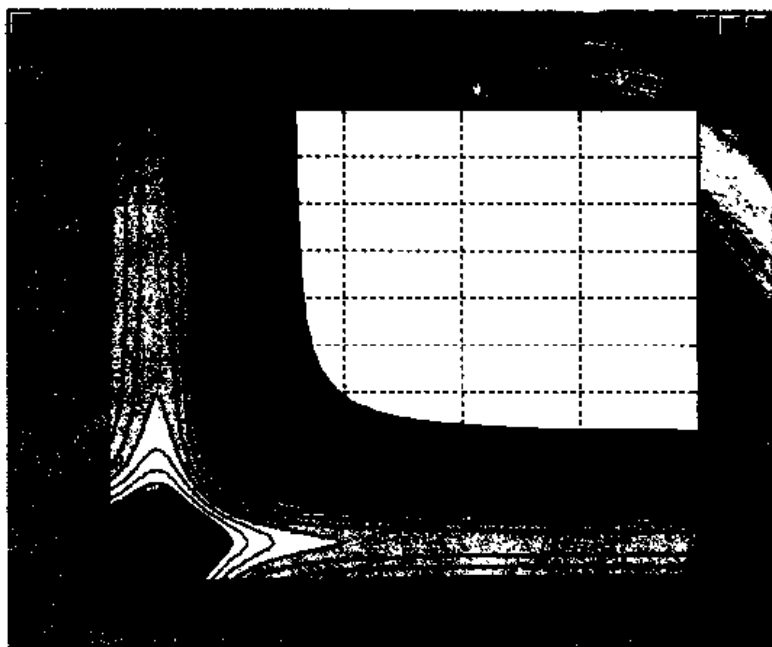


图 6.30

### 6.2.6 quiver及quiver3的用法

在6.1.11小节讨论了quiver用于平面绘图的技巧，同样也可以用它来画contour的向量图，如范例6.25所示。quiver3则可以画出空间中的向量变化图形。其格式为：

`quiver3(x,y,z,u,v,w,s)`

与quiver相同，只是多了z，w和s参数，它们用来表示箭头的大小。根据范例6.17画出如范例6.26所示的向量图。

#### 范例6.25

程序：ex5621.m

```
%3D contour and quiver plot
clear
%x=0:0.1:10;
[x,y]=meshgrid(linspace(0,10,20),linspace(0,10,20));
z=(1./(x.^3-2.*x+4)).+(1./(y.^3-2.*y+4));
contour(x,y,z,20);
hold on  ————— 别忘了，先保留已有的图形
[u,v]=gradient(z,0.5);
quiver(x,y,u,v,0.5);  ————— 画出向量图形
xlabel('x');
ylabel('y');
zlabel('z');
grid on
box on
```

执行结果如图6.31所示，可以比较与范例6.22的差别。

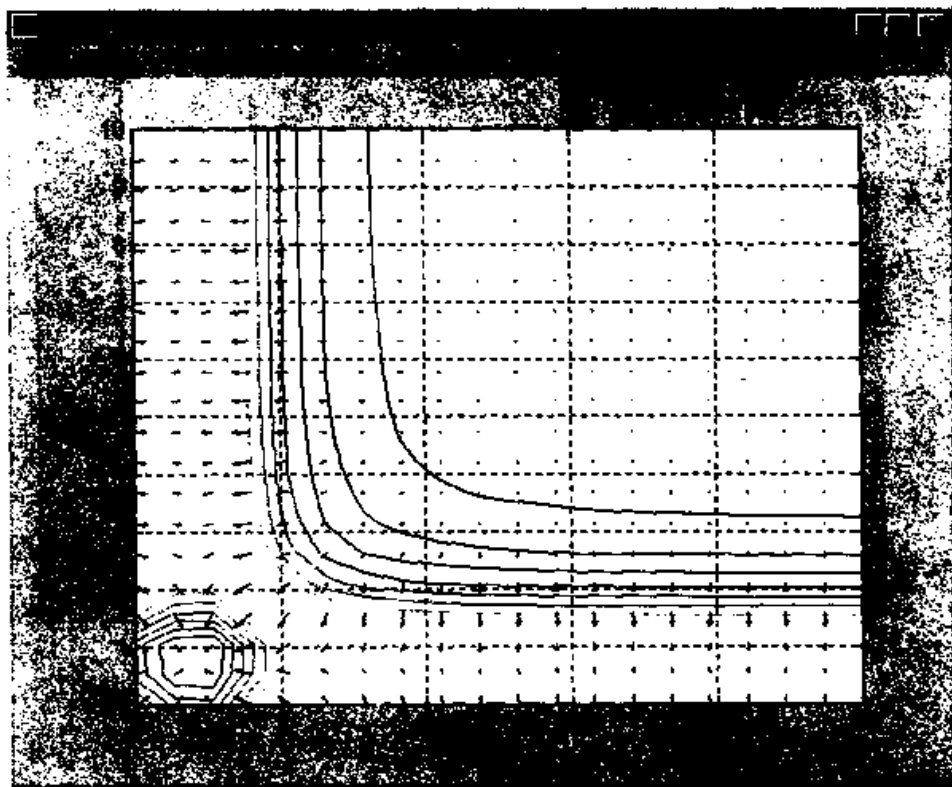


图 6.31

### 范例6. 26

程序: ex5621\_1.m

```
%3D plot3 and quiver3 plot
clear
x=0:0.1:10;
y=1./(x.^3-2.*x+4);
z=0:0.1:10;
u=gradient(x);
v=gradient(y);
w=gradient(z);
quiver3(x,y,z,u,v,w,0.3);
xlabel('x');
ylabel('y');
zlabel('z');
grid on
box on
```

执行结果如图6.32所示，可以比较一下与范例6.17的差别。



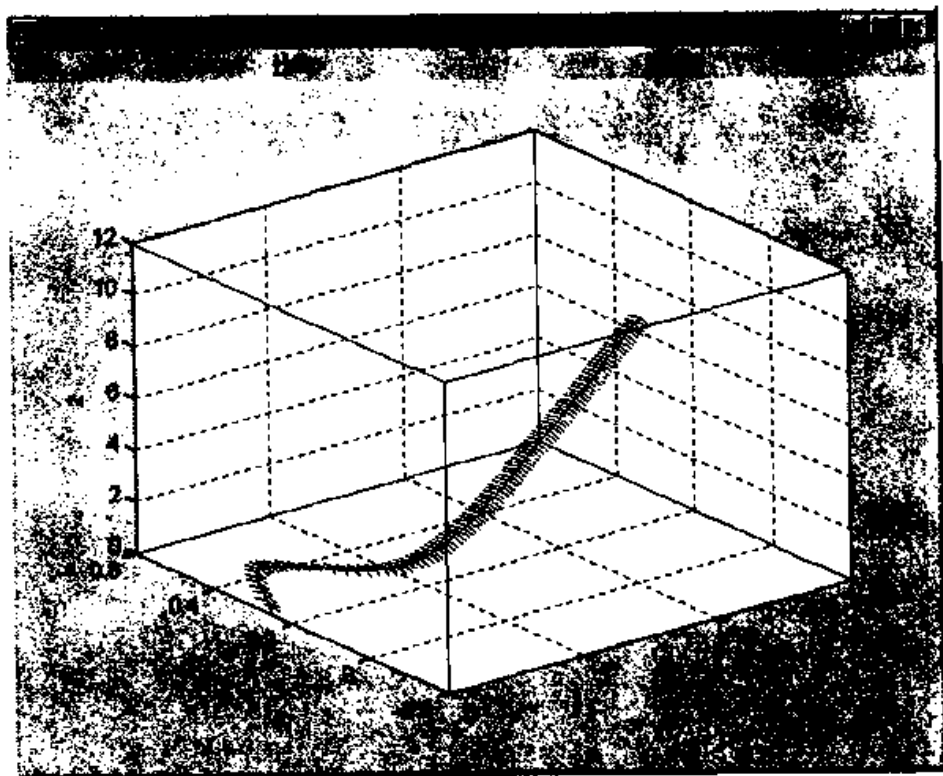


图 6.32

### 6.2.7 meshc与surf的格式

类似mesh与surf的用法, meshc与surf只是同时加上了contour的功能, 如范例6.27所示。

#### 范例6.27

程序: ex5622.m

```
%3D meshc and surf plot
clear
x=0:0.1:10;
[x,y]=meshgrid(linspace(0,10),linspace(0,10));
z=(1/(x.^3-2.*x+4))+(1/(y.^3-2.*y+4));
meshc(x,y,z);
figure(2)
surf(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z');
grid on
box on
```

执行结果如图6.33、图6.34所示。

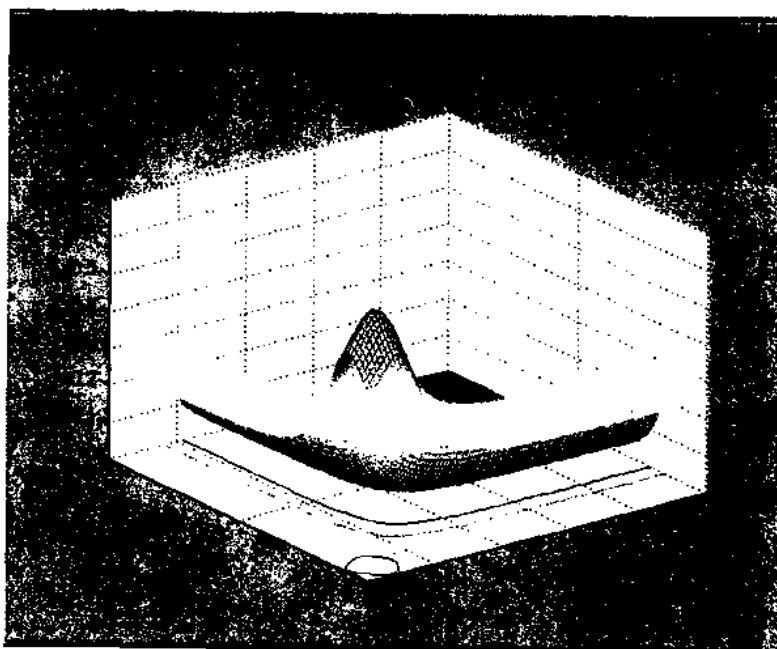


图 6.33

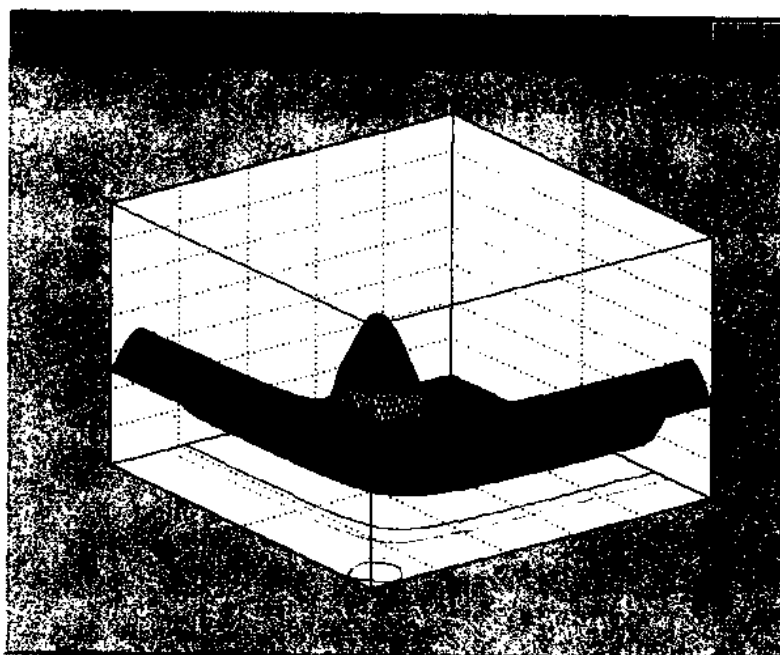


图 6.34

### 6.2.8 cylinder与sphere的格式

cylinder可以产生圆柱体，其格式为：

`[x,y,z]=cylinder(r)`

$r$ 为在每个高度值处的半径值，使用时必须配合`mesh`或`surf`等指令才能画出其图形。  
如果半径值设为 $y$ ，且高度变化为 $x$ 的关系式为：

$$y = \frac{1}{x^3 - 2x + 4}$$

画出的图形如范例6.28所示。

#### 范例6.28

程序：ex5623.m

```
%3D cylinder plot
```

```
clear
```

```
x=0:0.1:10;
```

```
y=1./(x.^3-2.*x+4);
```

```
z=0:0.1:10;
```

```
[u,v,w]=cylinder(y);
```

```
mesh(u,v,w);
```

```
xlabel('x');
```

```
ylabel('y');
```

```
zlabel('z');
```

```
grid on
```

以 $y$ 为半径，产生圆柱向量

绘出结果

执行结果如图6.35所示。

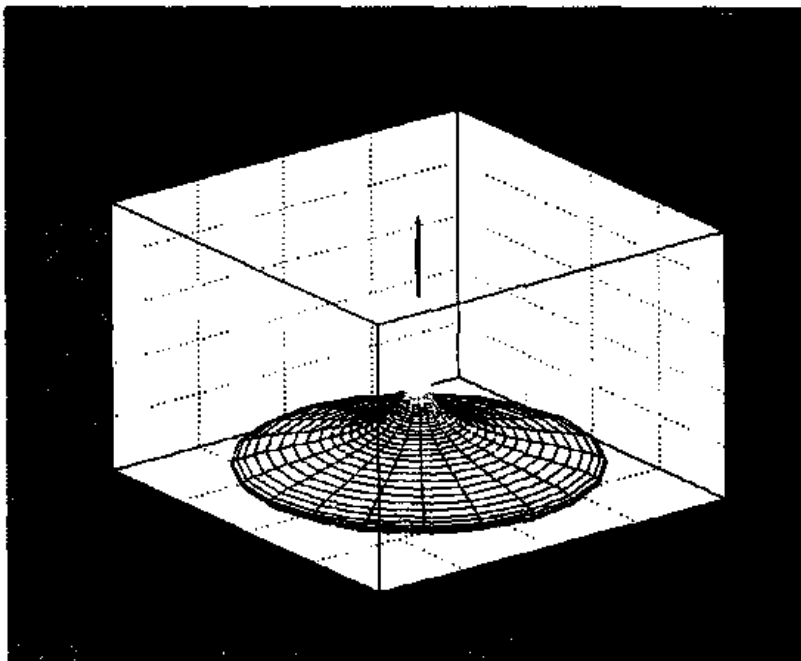


图 6.35

`sphere`可以产生单位球体，其指令格式为：

`[x,y,z] = sphere(n)`

自动产生  $(n+1) \times (n+1)$  的矩阵, 用于产生圆球体。用  $n=40$  来画圆球体, 如范例6.29所示。

### 范例6.29

程序: `ex5623_1.m`

```
%3D sphere plot
clear
x=0:0.1:10;
y=1./(x.^3-2.*x+4);
z=0:0.1:10;
[u,v,w] = sphere(40);
mesh(u,v,w);
xlabel('x');
ylabel('y');
zlabel('z');
grid on
box on
```

执行结果如图6.36 所示。

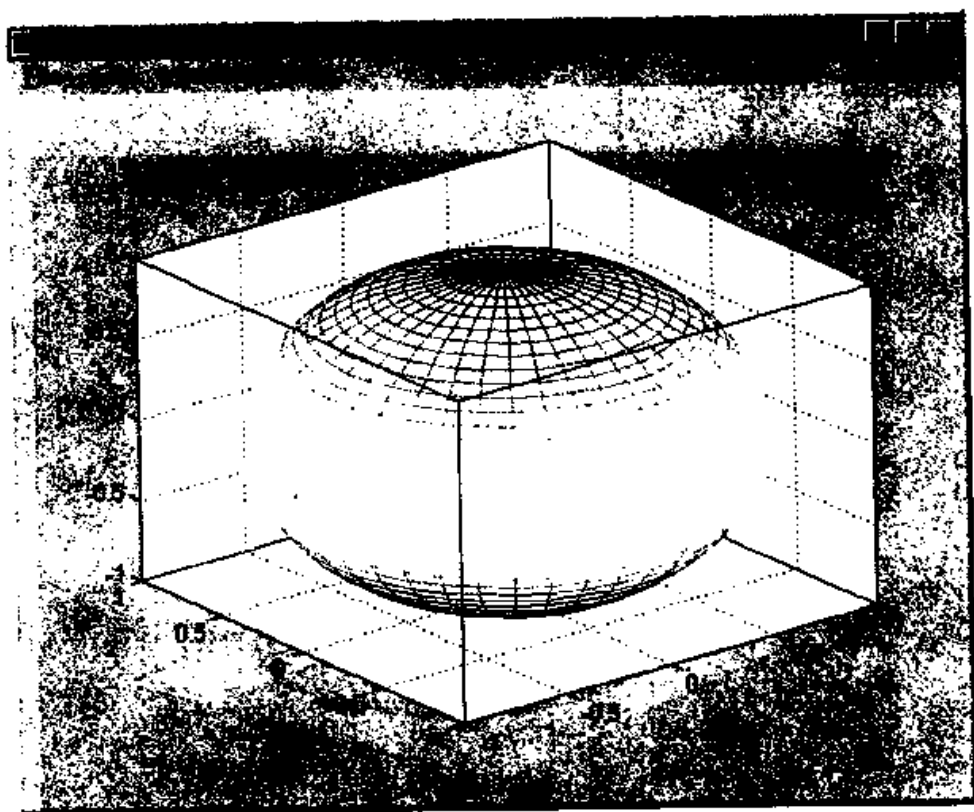


图 6.36

### 6.2.9 设定视角

`view (azimuth,elevation)`用于设定视角，默认值为`(-37.5,30)`。`azimuth`为视线与y轴负向的夹角；`elevation`为视线与xy平面的夹角。如图6.37所示。

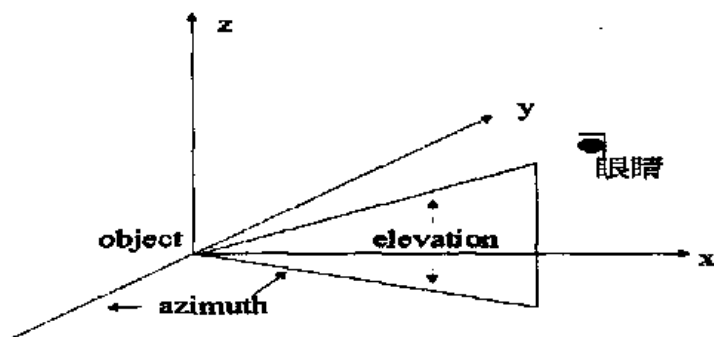


图 6.37 视角定义图

不过在实际应用中，建议先在命令窗口中执行：

```
rotate3d
```

然后就可以用鼠标直接随意调整改变视角。

可以用`rotate3d`改变范例6.28的图形视角，如图6.38所示。

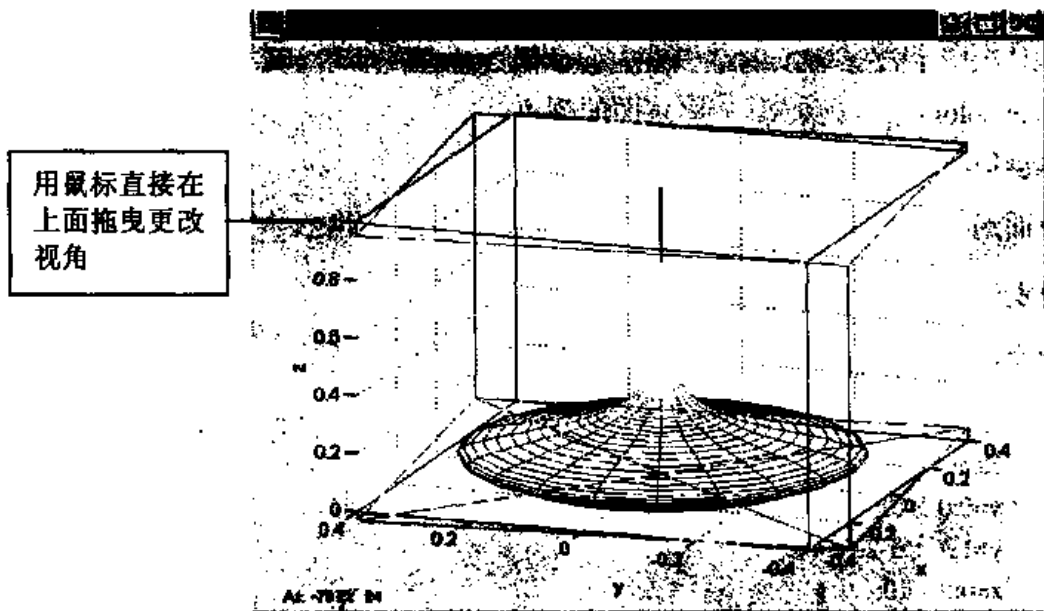


图 6.38 直接旋转视角

### 6.2.10 light的指令格式

```
light('position', [a b c], 'color', [r g b], 'style', 'infinite')
```

表示在坐标位置 `[a b c]` 上摆设光源，使绘出的图形能够反光，感觉上更为亮丽。这

三个参数的含义如下:

**position** 设定光源的坐标。

**color** 设定光线的颜色, 由  $[0\ 0\ 0]$  到  $[1\ 1\ 1]$ 。

**style** 光源的距离, 可以设定为 *local* 或 *infinite*。

除此之外还可以利用 *set* 指令, 对 *surf* 与 *patch* 完成的图形做更多的处理, 比如:

```
h=surf(x,y,z)
set(h,'property1','value1','property2','value2',.....)
```

就可以将对象 *h* 中的种种特效做一些处理, 可供设定的属性参见下面的说明:

<b>AmbientLightColor</b>	设定光源的背景颜色。其值 $[0\ 0\ 0]$ 到 $[1\ 1\ 1]$ 。
<b>AmbientStrength</b>	设定光源背景颜色的反射强度。其值为 0 至 1。
<b>DiffuseStrength</b>	设定光源背景颜色的扩散强度。其值为 0 至 1。
<b>SpecularStrength</b>	设定光源颜色的反射强度。其值为 0 至 2。
<b>SpecularExponent</b>	设定光源的反射面积。其值为 1 至 500。
<b>SpecularColorReflectance</b>	设定光源是否完全反射。其值为 0 至 1。
<b>FaceLighting</b>	面反射效果。可设定为 <i>none</i> , <i>flat</i> , <i>gouraud</i> , <i>phong</i> 。
<b>EdgeLighting</b>	与 <i>FaceLighting</i> 的参数设定相同, 只不过是边缘效果设定。
<b>BackfaceLighting</b>	处理对象的内部表面与外部表面的光线效果。可设定为 <i>reverselit</i> , <i>unlit</i> 或 <i>lit</i> 。
<b>FaceColor</b>	表面的颜色。
<b>EdgeColor</b>	边缘的颜色。

下面通过对范例 6.28 的图形进行效果设定来说明其用法。

### 范例 6.30

程序: ex5624.m

```
%3D sphere plot light effect
clear
x=0:0.1:10;
y=1./(x.^3-2.*x+4);
z=0:0.1:10;
[u,v,w]=sphere(40);
h=surf(u,v,w);
set(h,'facelighting','phong','facecolor','flat',...
    'edgecolor',[.4 .4 .4],'specularexponent',5)
light('position',[0 0 1];
xlabel('x');
ylabel('y');
zlabel('z');
```

读者也可以更改其设定, 看看效果的差别

```
grid on
box on
```

执行结果如图6.39所示。

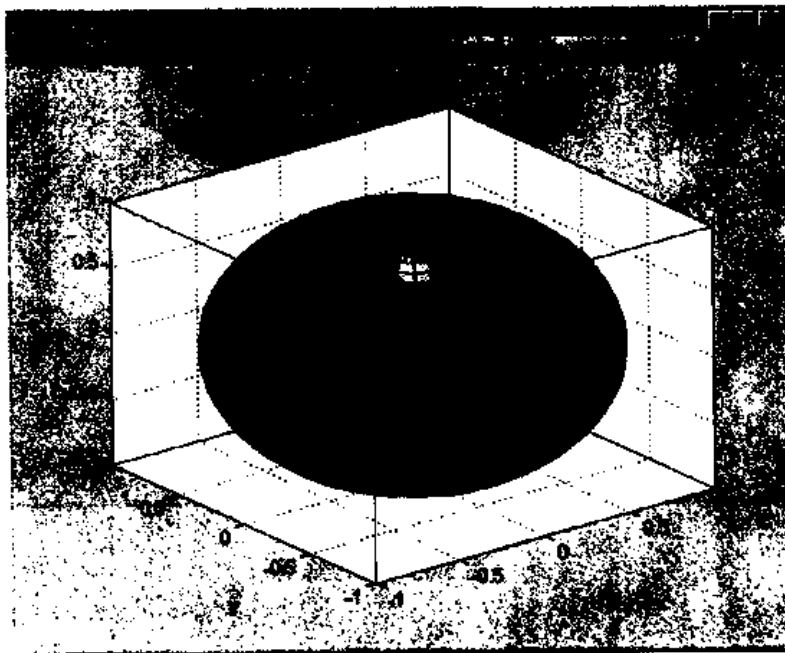


图 6.39 设定图形反光

### 范例6.31

程序: ex5624\_1.m

```
%3D cylinder plot and light effect
clear
x=1:0.1:2*pi;
y=cos(x+1);
z=0:0.1:10;
[u,v,w]=cylinder(y);
h=surf(u,v,w);
set(h,'facelighting','phong','facecolor','interp','edgecolor',[.2 .2 .2],
'specularexponent',5,'backfacelighting','reverselit')
light('position',[1 -1 1],'style','local');
xlabel('x');
ylabel('y');
zlabel('z');
grid on
box on
```

设定里面能够反光

执行结果如图6.40所示。

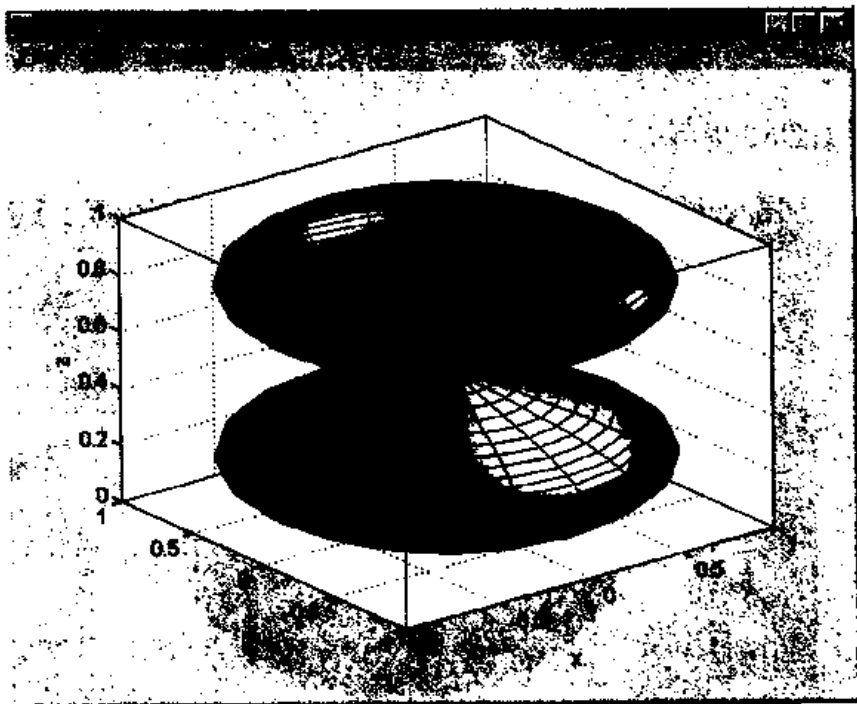


图 6.40 执行结果

### 6.3 设定图形属性

使用set指令, 可以对已经命名的对象进行属性(Property)设定。对象包括图形(Figure)、坐标轴(Axes)和画在坐标系内的线(line)、平面(Surface)和图像(Image)等等。

本节将分别对这些对象属性的设定方法进行说明。

#### 6.3.1 设定双坐标轴

如果一个图形文件中存在两种图形, 但是其坐标刻度不同, 就可以将它们在坐标轴的上下左右4个方向上分开表示, 从而不致发生混淆。所用的指令有set及axes, 下面分别进行说明:

set指令的格式:

```
set(axes名称, '属性名称1', '属性值1', '属性名称2', '属性值2', ...)
```

axes指令的格式:

```
axes('属性名称1', '属性值1', '属性名称2', '属性值2', ...)
```

下面用一个范例来说明其用法。

#### 范例6.32

程序: ex5625.m

```
%DOUBLE AXIS  
clear
```



```

x=0:0.1:10;
for i=1:length(x);
    y(i)=x(i);
    ycos(i)=cos(y(i));
end
ha=line(x,y,'color','b','linestyle',':');hold on
ap=gca
xlabel('x')
ylabel('y')
bp=axes('position',get(ap,'position'),'YAxisLocation','right',...
        'XAxisLocation','top','color','none')
hb=line(x,ycos,'color','red','parent',bp);hold on
ylabel('cos y')

```

产生一条直线，而另一个为余弦图形

将x坐标设定在上方，y坐标设定在右方

把第二个图形放到指定为bp的原图形坐标系中

执行结果如图6.41所示。

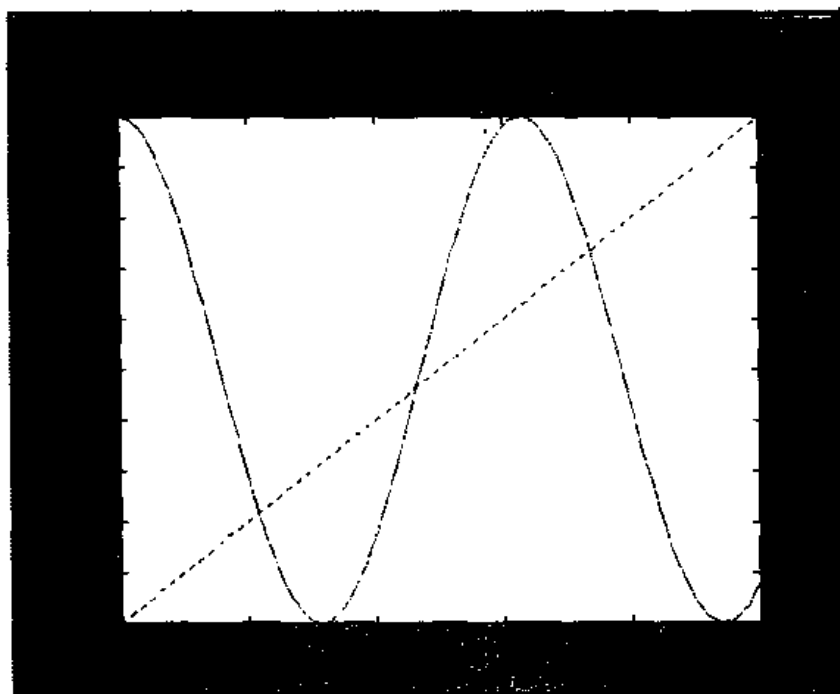


图 6.41

### 6.3.2 设定坐标轴颜色

可以通过改变下列坐标轴颜色的属性，从而达到设计者要求的图形效果：

Color	坐标轴背景颜色。
XColor,YColor,ZColor	坐标轴线颜色。

用法如下所示：

```
set(gca,'color','w')
```

表示把背景设定为白色。

```
set(get(gca, 'title'), 'color', 'k')
```

表示把目前图形上标题的字体颜色设定为黑色。

### 6.3.3 设定坐标轴其他属性

Axes的其他属性如下，读者也可进入GUI查看修改。修改的方法在后面介绍。

CameraPosition	设定观察点的位置。
CameraTarget	目标的位置设定。
CameraUpVector	在进行2-D及3-D观察时，向上的方向调整量。
CameraViewAngle	目标的投影大小及角度。
CLim	数据颜色的映射值。
DataAspectRatio	沿着x, y, z轴延伸的数据相对单位值。
PlotBoxAspectRatio	沿着x, y, z轴延伸的绘图区域相对单位值。
TickDir Mode	坐标轴标签的方向模式。
XLim	x, y, z轴坐标的范围设定。
Ylim	
Zlim	
XlimMode	自动或者手动设定x, y, z轴的坐标值范围。
YLimMode	
ZLimMode	
XTick	设定标签在坐标轴上的位置。
YTick	
Ztick	
XTickMode	自动或者手动设定x, y, z轴上的标签位置。
YTickMode	
ZTickMode	
XTickLabel	x, y, z轴的标签文字标示。
YTickLabel	
ZTickLabel	
XTickLabelMode	自动或者手动设定x, y, z轴上的文字标签位置。
YTickLabelMode	
ZTickLabelMode	

XDir  
YDir  
ZDir

} 设定x, y, z轴数值的方向是正常还是反向。

或许上面的介绍还不太完整, 建议通过GUIDE来帮助了解上述参数的意义。  
首先画出3D图形。在命令窗口中输入

```
peaks(30)
```

执行后结果如图6.42所示。接着在命令窗口中输入

```
guide
```

进入GUI, 在Property Editor中选择

Axes

举例来说, 选择DataAspectRatio属性, 此时的值为:

```
[1 1 2.4084687763548]
```

如图6.43所示。将其值改为:

```
[1 1 8]
```

其结果如图6.44所示, 可以发现图形被压扁了。所以读者可以尝试更改x, y轴的刻度, 看看改变后的效果如何。也可以通过尝试更改其他属性体会各个参数的意义。

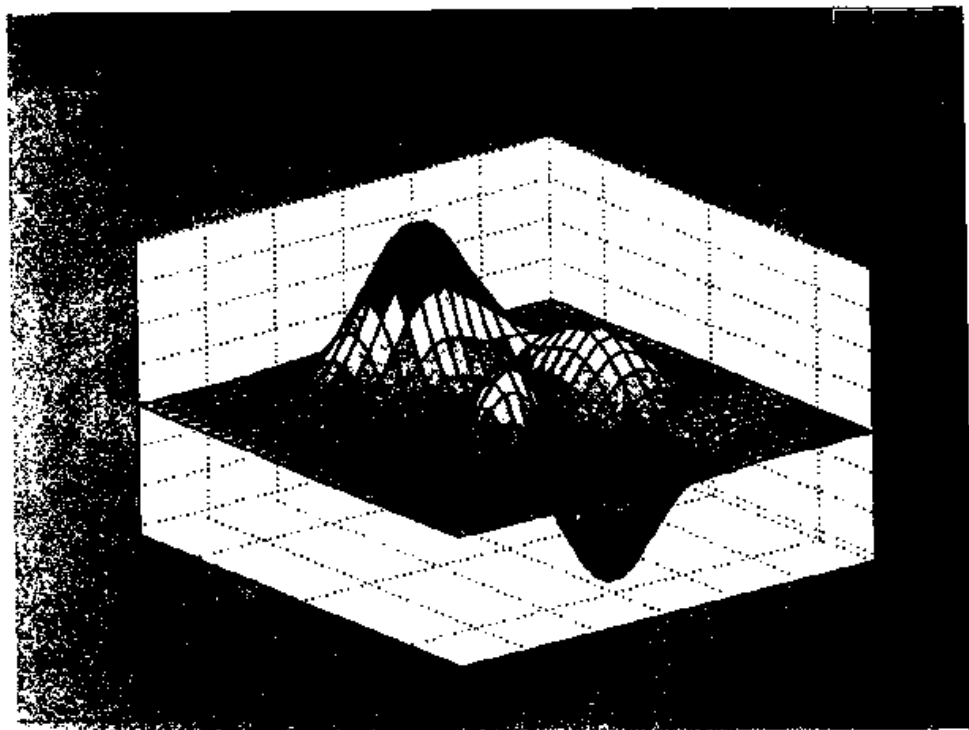


图 6.42

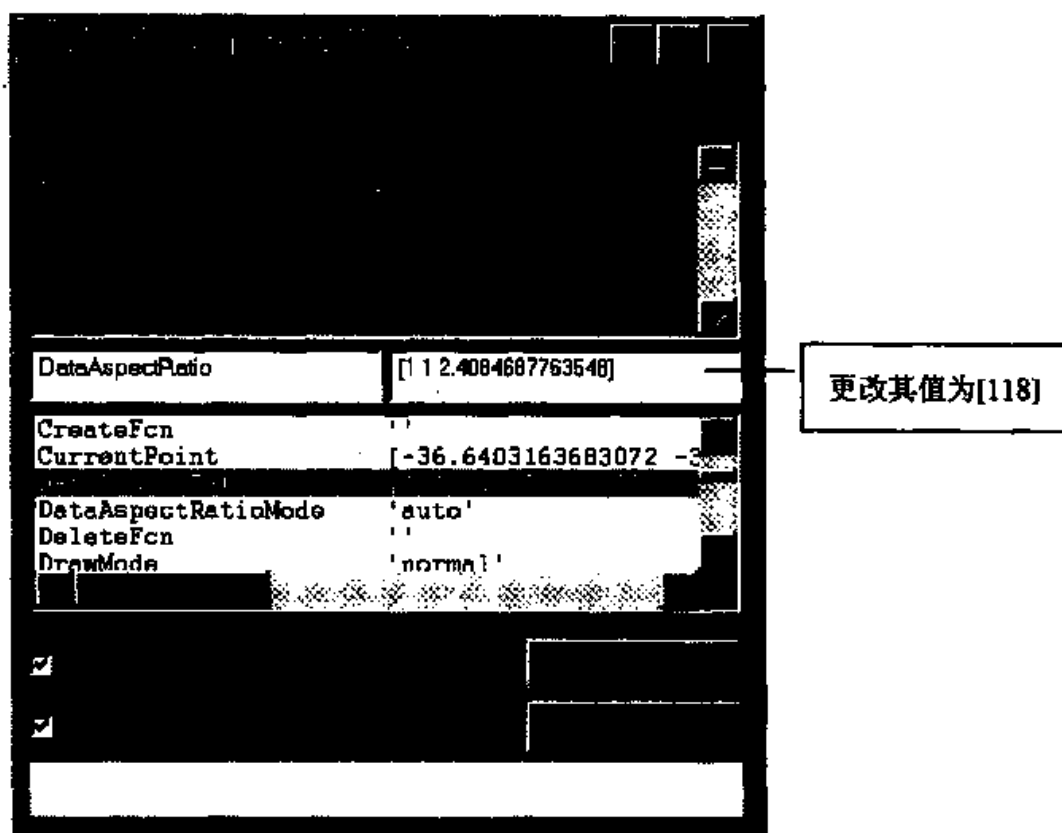


图 6.43 修改属性值

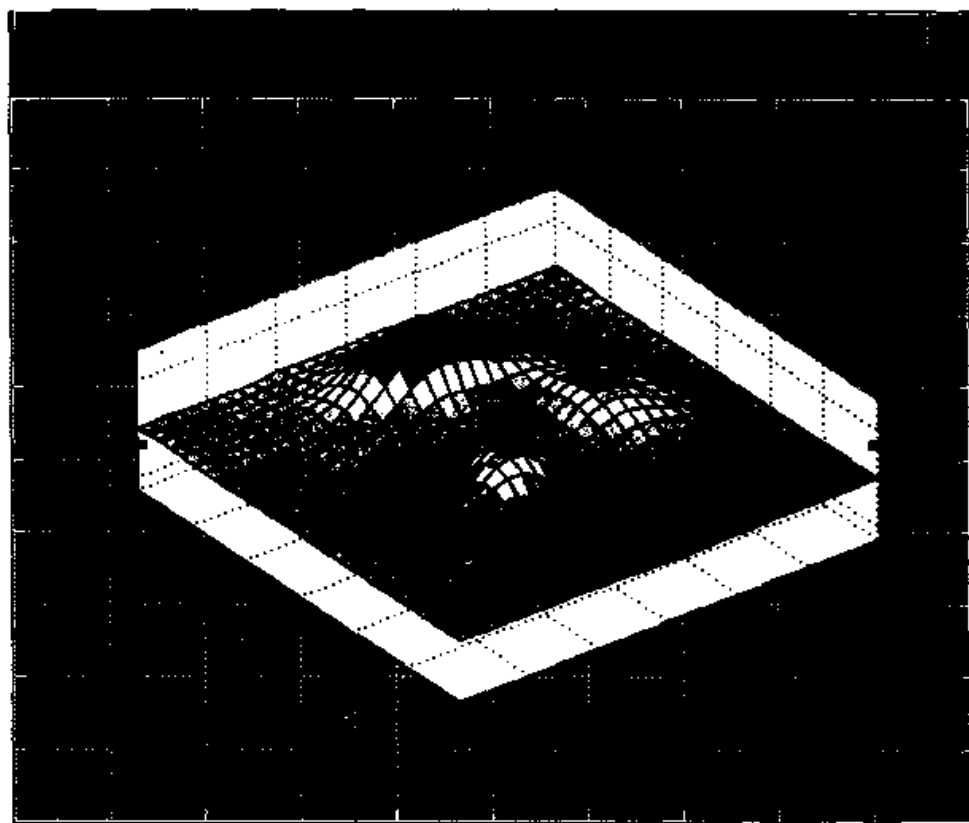


图 6.44 执行结果

### 6.3.4 对象中的x, y, z数据的处理

画在坐标轴内的图形，不管是线还是平面，都有一个很重要的属性值，即：

XData

YData

ZData

这个属性值在GUIDE内看不到，必须通过get指令才能获得。

下面说明如何处理XData, YData, ZData, 找出极大值与极小值，最后显示出属性列表。

#### 范例6.33

程序：ex5626.m

```
%FIND THE MAX AND MIN VALUES
clear
z=peaks(30);
h1=mesh(z); 定义所画的平面为h1
x=get(h1,'xdata');
y=get(h1,'ydata'); 通过get指令获得图形中的数据
z=get(h1,'zdata');
min-z=min(min(z)); Z为二维矩阵，所以要执行两次min
max-z=max(max(z));
[i,j]=find(min-z==z); Find指令找出满足条件的x、y坐标
text(x(i),y(j),z(i,j), ['min value= ',num2str(min-z)] ,...
    'VerticalAlignment','top','HorizontalAlignment','right');
[i,j]=find(max-z==z);
text(x(i),y(j),z(i,j), ['max value= ',num2str(max-z)] ,...
    'VerticalAlignment','bottom','HorizontalAlignment','left');
set(gca,'Xdir','reverse','Ydir','reverse') 将坐标轴反向
xlabel('x');
ylabel('y');
zlabel('z');
view(-40,14)
```

执行结果如图6.45所示。

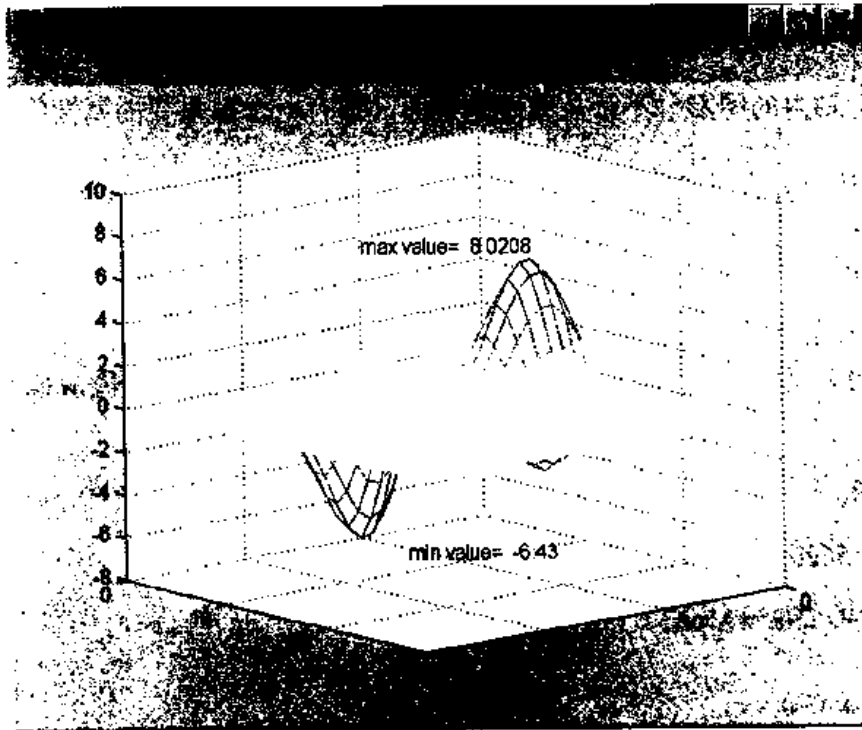


图 6.45

接着执行：

```
get(h1)
```

得到如下属性值列表：

```

CData = [(30 by 30) double array]
CDataMapping = scaled
EdgeColor = flat
EraseMode = normal
FaceColor = [1 1 1]
LineStyle = -
LineWidth = [0.5]
Marker = none
MarkerEdgeColor = auto
MarkerFaceColor = none
MarkerSize = [6]
MeshStyle = both
XData = [(1 by 30) double array]
YData = [(30 by 1) double array]
ZData = [(30 by 30) double array]
FaceLighting = none
EdgeLighting = flat
BackFaceLighting = reverselit
AmbientStrength = [0.3]
DiffuseStrength = [0.6]

```

```
SpecularStrength = [0.9]
SpecularExponent = [10]
SpecularColorReflectance = [1]
VertexNormals = [(30 by 30 by 3) double array]
NormalMode = auto
ButtonDownFcn = ctfpanel SelectMoveResize
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
Interruptible = off
Parent = [1.00232]
Selected = off
SelectionHighlight = on
Tag = Axes1Surface1
Type = surface
UserData = []
Visible = on
```





