

MATLAB

MATLAB 7.x 基础教程

张笑天 杨奋强 编著



西安电子科技大学出版社
<http://www.xduph.com>



XDUP 229100

封面设计:  佳易传播

MATLAB

MATLAB 7.x程序设计语言 (第二版) (普通高等教育“十一五”国家级规划教材)

基于MATLAB 7.x的系统分析与设计——小波分析 (第三版)

基于MATLAB 6.x的系统分析与设计——神经网络 (第二版)

基于MATLAB 7.x的系统分析与设计——信号处理 (第二版)

基于MATLAB 7.x的系统分析与设计——控制系统 (第二版)

基于MATLAB 6.x的系统分析与设计——虚拟现实

基于MATLAB的系统分析与设计——模糊处理

基于MATLAB的系统分析与设计——图像处理

基于MATLAB的系统分析与设计——时频分析

现代通信系统分析与仿真——MATLAB通信工具箱

MATLAB及其在理工课程中的应用指南 (第三版) (普通高等教育“十一五”国家级规划教材)

MATLAB基础与编程入门

MATLAB外部接口编程

控制系统设计与仿真 (MATLAB版)

MATLAB应用图像处理

MATLAB辅助模糊系统设计

MATLAB 6.x图形编程与图像处理

MATLAB辅助现代工程数字信号处理

Simulink建模与仿真

Simulink动态建模与仿真基础

Stateflow逻辑系统建模

MATLAB应用程序集成与发布

MATLAB小波分析高级技术



MATLAB 7.x基础教程

ISBN 978-7-5606-1999-6



9 787560 619996 >

定价: 26.00元

TP317/108

2008

MATLAB 7.x 基础教程

张笑天 杨奋强 编著

西安电子科技大学出版社

2008

内 容 简 介

MATLAB 是美国 MathWorks 公司推出的高效的科学计算软件。本书基于 MATLAB 7.x, 全面地介绍了 MATLAB 的工作环境和基本功能, 包括 MATLAB 的基本操作、数据结构、数据类型、数值计算、程序设计、符号计算、基本绘图以及 Simulink 仿真等。本书内容简明扼要, 实例丰富, 便于读者掌握。

本书适用于理工科学校相关专业的在校大学生, 也可供相关领域的科学研究和工程技术人员学习参考。

图书在版编目(CIP)数据

MATLAB 7.x 基础教程 / 张笑天, 杨奋强编著. —西安: 西安电子科技大学出版社, 2008.4

ISBN 978-7-5606-1999-6

I. M… II. ①张… ②杨… III. 计算机辅助计算-软件包, MATLAB 7-教材 IV. TP391.75

中国版本图书馆 CIP 数据核字(2008)第 020481 号

策 划 臧延新

责任编辑 臧延新

出版发行 西安电子科技大学出版社(西安市太白南路 2 号)

电 话 (029)88242885 88201467 邮 编 710071

<http://www.xduph.com>

E-mail: xdupfxb@pub.xaonline.com

经 销 新华书店

印刷单位 陕西天意印务有限责任公司

版 次 2008 年 4 月第 1 版 2008 年 4 月第 1 次印刷

开 本 787 毫米×1092 毫米 1/16 印 张 17.75

字 数 420 千字

印 数 1~4000 册

定 价 26.00 元

ISBN 978-7-5606-1999-6 / TP · 1038

XDUP 2291001-1

*** 如有印装问题可调换 ***

本社图书封面为激光防伪覆膜, 谨防盗版。

前 言

MATLAB 是美国 MathWorks 公司自 1984 年开始推出的一种使用简便的科学计算软件, 该公司于 2007 年推出了最新版本 MATLAB R2007a。MATLAB 将高性能的数值计算和可视化结合在一起, 并且已深入到各行各业众多学科的应用当中, 如数学计算, 算法开发, 数据采集, 建模、仿真和原型设计, 数据分析、研究和可视化, 科学和工程绘图, 以及应用开发环境等。由于 MATLAB 在矩阵和向量公式的计算方面有着独特的优势, 用户可以通过简洁的表达式实现复杂的计算和公式的推导, 因此 MATLAB 成为高校中高等数学、线性代数、数理统计、自动控制理论、数字信号处理、动态系统仿真等课程的基本教学工具。MATLAB 还附带了一套功能强大、内容丰富的专用工具箱, 适用于各种各样的专业领域。用户不仅可以利用工具箱中提供的算法, 还可以对其中的算法进行修改, 甚至能够开发自己的算法来扩充工具箱。迄今为止, MATLAB 已经成为科学研究和工程计算必不可少的工具之一, 也是各高校相关专业在读本科生、研究生必须掌握的基本技能。

本书的内容框架基于 MATLAB 7.x 系列最新版本软件的使用方法和设计理念。书中首先介绍了该版本的新特点和工作环境, 其次介绍了 MATLAB 计算的基本使用方法, 之后又对公式计算和数据可视化作了详细的介绍, 最后介绍了 Simulink 的系统仿真功能。本书共分为 8 章。第 1 章为 MATLAB 概述, 介绍了 MATLAB R2007a 的新特点和新产品, 以及新的开发环境, 使读者对新版本的 MATLAB 有一个整体的认识。第 2 章为矩阵与数组, 主要介绍矩阵的创建、索引、信息获取、基本操作和运算, 还介绍了维数小于 2 的特殊矩阵以及多维数组, 目的是使读者掌握 MATLAB 的基本数据结构。第 3 章为数据类型, 详细地介绍了 MATLAB 的 15 种基本数据类型和 2 种自定义数据类型, 掌握这些数据类型是掌握数据运算和操作的基础。第 4 章为数学运算基础, 全面地介绍了包括矩阵与线性代数、多项式与插值、快速傅里叶变换、函数的函数、求解微分方程以及稀疏矩阵等数值计算功能, 它们是 MATLAB 最基本的科学计算功能。第 5 章为 M 文件程序设计基础, 分别介绍了 M 文件程序的设计、调试、优化方法, 还介绍了数据文件的输入输出。第 6 章为符号计算功能, 详细介绍了符号对象的创建方法、符号数学计算功能、符号表达式的化简和替换、符号线性代数、符号方程求解、符号绘图函数以及 Maple 函数的调用方法。第 7 章为基本绘图功能, 首先介绍图形窗口, 然后详细地介

绍二维图形、三维图形和特殊图形的绘制函数。第 8 章为 Simulink 仿真环境, 分别介绍了 Simulink 的基础、Simulink 的模块库、子系统及封装技术、仿真运行与分析方法以及 S-函数的设计与应用, 以便为读者今后深入学习和使用打下基础。为使读者领会最新工具的特点, 全书内容始终贯穿 MATLAB 最新版本软件的应用理念和设计思想。本书以简洁通俗的语言阐述基本理论, 并结合大量实例对 MATLAB 的使用方法进行说明, 便于读者快速掌握。全书内容深入浅出、叙述全面, 适合于不同层次多个专业的读者, 不仅可以作为初学读者的入门指导教程, 也可以供中级读者使用参考。

本书的编写得到了西安电子科技大学出版社的大力支持, 臧延新编辑为本书的出版付出了辛勤的劳动, 在此表示衷心的感谢! 由于作者水平有限, 书中的疏漏和不足之处在所难免, 敬请各位专家和读者批评指正。

作 者

2007 年 10 月

目 录

第 1 章 MATLAB 概述	1	2.4 基本操作和运算	25
1.1 MATLAB R2007a 简介	2	2.4.1 矩阵的扩大和缩小	25
1.1.1 MATLAB 的新版本特性	2	2.4.2 改变矩阵的形状	26
1.1.2 MATLAB 的新产品概况	4	2.4.3 矩阵的算术运算	28
1.2 桌面工具与开发环境	5	2.4.4 矩阵的关系运算和逻辑运算	29
1.2.1 主菜单	6	2.5 空矩阵、标量和向量	30
1.2.2 工具栏	6	2.5.1 空矩阵	30
1.2.3 当前路径	6	2.5.2 标量	31
1.2.4 工作区间	7	2.5.3 向量	31
1.2.5 命令窗	7	2.6 多维数组	31
1.2.6 历史命令记录	7	2.6.1 多维数组的创建	32
1.2.7 Strat 菜单	7	2.6.2 多维数组的索引	33
1.3 编辑/调试器	7	2.6.3 改变多维数组的形状	34
1.3.1 M 文件的创建	8	2.6.4 多维数组的运算	35
1.3.2 M 文件的运行和调试	9	第 3 章 数据类型	37
1.3.3 M 文件的结果发布	9	3.1 数值类型	37
1.4 帮助系统	11	3.1.1 整数	37
1.4.1 命令窗查询帮助	11	3.1.2 浮点数	39
1.4.2 帮助浏览器	11	3.1.3 复数	41
第 2 章 矩阵与数组	13	3.1.4 无穷与非数	42
2.1 创建矩阵	13	3.1.5 判断数据类型	43
2.1.1 创建矩阵和数值序列	13	3.1.6 数据显示形式	43
2.1.2 创建特殊矩阵	14	3.2 逻辑类型	44
2.1.3 合并矩阵	16	3.2.1 创建逻辑数组	44
2.2 索引	20	3.2.2 逻辑数组的用途	45
2.2.1 线性索引	20	3.2.3 判断逻辑类型	45
2.2.2 访问单个元素	21	3.3 字符串	45
2.2.3 访问多个元素	21	3.3.1 创建字符数组	46
2.3 获取矩阵信息	22	3.3.2 字符串单元数组	46
2.3.1 矩阵的阶数与维数	22	3.3.3 字符串的操作	47
2.3.2 矩阵元素的数据类型	24	3.3.4 字符串类型与数值类型之间的转化	50
2.3.3 矩阵的数据结构	24	3.4 日期与时间	50

3.4.1 日期的表现形式	50	4.6 稀疏矩阵	100
3.4.2 日期表现形式之间的转化	51	4.6.1 创建稀疏矩阵	100
3.4.3 当前日期与时间	51	4.6.2 稀疏矩阵的查看	103
3.5 结构	52	4.6.3 稀疏矩阵的操作	104
3.5.1 创建结构数组	52	第5章 M 文件程序设计基础	112
3.5.2 结构数组的操作	53	5.1 M 文件介绍	112
3.6 单元数组	54	5.1.1 脚本和函数	112
3.6.1 创建单元数组	54	5.1.2 P 代码文件	114
3.6.2 单元数组的操作	55	5.1.3 变量类型	114
3.7 函数句柄	56	5.1.4 关键字和特殊值	115
3.7.1 创建和调用函数句柄	56	5.1.5 符号参考	116
3.7.2 利用句柄调用函数	57	5.2 程序流程控制	119
3.8 MATLAB 类	57	5.2.1 条件控制语句	119
3.9 Java 类	57	5.2.2 循环控制语句	121
第4章 数学运算基础	58	5.2.3 错误控制语句	123
4.1 矩阵与线性代数	58	5.2.4 程序终止语句	124
4.1.1 矩阵分析	58	5.3 数据输入/输出	124
4.1.2 求解线性方程组	63	5.3.1 打开文件	124
4.1.3 逆矩阵与伪逆矩阵	66	5.3.2 读写操作	126
4.1.4 矩阵的分解	66	5.3.3 关闭文件	132
4.1.5 矩阵的非线性运算	68	5.3.4 更多文件 I/O 函数	133
4.1.6 特征值与特征向量	70	5.4 程序调试与优化	134
4.1.7 奇异值分解	72	5.4.1 程序的调试	134
4.2 多项式与插值	73	5.4.2 程序的优化	139
4.2.1 多项式	73	第6章 符号计算功能	147
4.2.2 插值	78	6.1 符号对象的创建与使用	147
4.3 快速傅里叶变换	83	6.1.1 创建符号变量和表达式	147
4.3.1 快速傅里叶变换的概念	83	6.1.2 创建符号数学函数	150
4.3.2 快速傅里叶变换的应用	84	6.2 数学计算功能	150
4.4 函数的函数	85	6.2.1 符号微积分	150
4.4.1 函数的表示方法	86	6.2.2 函数的极限	152
4.4.2 函数的最小值与零点	87	6.2.3 级数求和	153
4.4.3 数值积分	88	6.2.4 泰勒级数展开	153
4.4.4 嵌套函数与匿名函数	89	6.3 表达式的化简和替换	154
4.5 求解微分方程	90	6.3.1 符号表达式的化简	154
4.5.1 常微分方程初值问题	90	6.3.2 符号表达式的替换	158
4.5.2 延迟微分方程初值问题	93	6.4 线性代数	160
4.5.3 常微分方程边值问题	95	6.4.1 基本代数运算	160
4.5.4 求解偏微分方程	98	6.4.2 线性代数运算	161

6.4.3 特征值	164	7.4.4 离散数据图	213
6.4.4 约当标准型	165	7.4.5 方向和速度向量图	215
6.4.5 奇异值分解	165	7.4.6 等高线图	219
6.4.6 特征值轨迹	166	第 8 章 Simulink 仿真环境	224
6.5 求解符号方程	168	8.1 Simulink 基础	224
6.5.1 求解代数方程	168	8.1.1 Simulink 的启动	224
6.5.2 求解代数方程组	168	8.1.2 Simulink 工作环境	225
6.5.3 求解常微分方程	169	8.1.3 Simulink 模块的基本操作	228
6.6 简易符号绘图函数	171	8.1.4 Simulink 仿真步骤	230
6.6.1 二维基本绘图	171	8.1.5 Simulink 求解算法	233
6.6.2 二维极坐标绘图	172	8.2 Simulink 的模块库	235
6.6.3 三维曲线绘图	173	8.2.1 Commonly Used Blocks 模块库	235
6.6.4 三维网格绘图	174	8.2.2 Continuous 模块库	236
6.6.5 三维表面绘图	175	8.2.3 Discontinuities 模块库	236
6.6.6 等高线绘图	177	8.2.4 Discrete 模块库	236
6.7 调用 Maple 函数	178	8.2.5 Logic and Bit Operations 模块库	237
6.7.1 maple 函数	178	8.2.6 Lookup Tables 模块库	238
6.7.2 mfun 函数	179	8.2.7 Math Operations 模块库	238
6.7.3 sym 函数	179	8.2.8 Model Verification 模块库	239
6.8 积分变换	180	8.2.9 Model-Wide Utilities 模块库	239
6.8.1 傅里叶变换	180	8.2.10 Ports & Subsystems 模块库	240
6.8.2 拉普拉斯变换	181	8.2.11 Signal Attributes 模块库	240
6.8.3 Z 变换	182	8.2.12 Signal Routing 模块库	241
第 7 章 基本绘图功能	184	8.2.13 Sinks 模块库	242
7.1 图形窗口	184	8.2.14 Sources 模块库	242
7.1.1 图形窗口的创建与设置	184	8.2.15 User-Defined Functions 模块库	243
7.1.2 图形窗口的工具栏	186	8.3 子系统及封装技术	243
7.1.3 图形窗口的主菜单	187	8.3.1 创建子系统	243
7.2 绘制二维图形	190	8.3.2 封装子系统	245
7.2.1 基本绘图函数	190	8.3.3 自定义模块库	249
7.2.2 图形处理函数	194	8.4 仿真运行与分析	249
7.3 绘制三维图形	201	8.4.1 仿真的运行控制	249
7.3.1 三维曲线图	201	8.4.2 仿真数据的输入和输出	250
7.3.2 三维网格图	202	8.4.3 错误诊断	254
7.3.3 三维曲面图	203	8.4.4 改善仿真性能和精度	256
7.4 绘制特殊图形	205	8.4.5 使用命令运行仿真	257
7.4.1 条形图与区域图	206	8.4.6 观察输出轨迹	258
7.4.2 饼形图	210	8.4.7 线性化模型	259
7.4.3 直方图	211	8.4.8 寻找稳态工作点	260

8.5 S-函数的设计与应用	262	8.5.4 编写 S-函数	264
8.5.1 S-函数的概念	262	8.5.5 应用实例	272
8.5.2 S-函数的使用	262	参考文献	276
8.5.3 S-函数的工作方式	263		

第 1 章 MATLAB 概述

MATLAB 的名称源自 Matrix Laboratory, 它是一种高效的科学计算语言。MATLAB 将高性能的计算与可视化集成在一起, 并提供了大量的内置函数及工具箱, 从而被广泛地应用于科学计算、控制系统、信息处理等领域的分析、仿真和设计工作中。其典型应用主要有:

- 数学计算;
- 算法开发;
- 数据采集;
- 建模、仿真和原型设计;
- 数据分析、研究和可视化;
- 科学和工程绘图;
- 应用开发环境, 包括创建图形用户界面。

MATLAB 是一个以没有维数限制的数组为基本数据元素的交互式系统, 这样就可以解决很多专业计算上的问题, 特别是包含矩阵和向量公式的计算问题。历经十余年的发展, MATLAB 已经成为大学里初等数学、高等数学及理工学科相关课程的标准教学工具。在工业上, MATLAB 是进行高效率研究、开发及分析的工具。

MATLAB 的特色就在于其附带的一套专用工具箱。对于许多用户来说, 工具箱可以帮助用户学习和运用专业技术。工具箱是用来解决特殊问题的 MATLAB 函数库, 它包含信号处理、控制系统、神经网络、模糊逻辑、小波分析以及系统仿真等功能。

MATLAB 系统主要由以下几个部分构成。

1. 桌面工具与开发环境

桌面工具与开发环境由一系列帮助用户使用 MATLAB 函数和文件的工具和设备组成。大多数工具采用图形用户界面方式, 包含 MATLAB 桌面、命令窗、历史命令记录、编辑调试器、代码分析器和其他报告, 以及用于查看帮助、工作区、文件和搜索路径的浏览器。

2. MATLAB 数学函数库

数学函数库是一个汇集了大量计算算法的库。它不仅包含了最基本的函数, 如求和、正弦、余弦以及复数运算, 还包含了更为复杂的函数, 如矩阵求逆、求特征值、贝塞尔函数以及快速傅里叶变换等。

3. MATLAB 语言

MATLAB 是一种高级矩阵语言。它包含程序流程控制、函数、数据结构、输入/输出以及面向对象编程。用户可以在命令窗中输入语句以实现快速执行命令, 也可以在 M 文件中

编写程序以实现较复杂程序的执行。

4. 图形处理

MATLAB 可以方便地将向量或矩阵用图形显示出来, 并且可以对其进行注释和打印。它不仅包含高级函数, 如二维和三维数据的显示、图像处理、动画以及表示图形, 还包含低级函数, 如完全自定义图形显示以及建立图形用户界面。

5. MATLAB 外部接口/API

该库允许用户编写可以与 MATLAB 进行交互的 C 或 Fortran 程序。它包含从 MATLAB 调用例程的工具, 作为计算引擎调用 MATLAB 以及对 MAT 文件进行读写。

1.1 MATLAB R2007a 简介

MATLAB R2007a 版本对 R2006b 版本的 MATLAB 与 Simulink 进行了更新, 并且还对其他 82 项产品进行了更新和错误修正。R2007a 还增加了对基于 Intel 的 Mac、Windows Vista 及 64 位 Sun Solaris SPARC 平台的支持。

MATLAB 产品系列包括以下新功能:

- 支持多核或多处理器系统, 实现主要 MATLAB 数学函数的多线程计算;
- 通过分布式计算工具箱, 可同时在四个 MATLAB 会话中运行并行算法;
- 在统计工具箱中提供新的分类和数据集数组;
- 定点工具箱使 C 语言的编译速度加快;
- 在控制系统工具箱中实现带延迟控制环路的精确建模与分析;
- 通过系统辨识工具箱生成非线性模型;
- 支持遗传算法和直接查找工具箱中的模拟退火算法。

Simulink 产品系列中的新功能包括:

- Simulink、信号处理模块库、Embedded MATLAB Function Block、视频和图像处理模块库以及 Real-Time Workshop 等支持多维信号;
- 改进了 Real-Time Workshop Embedded Coder 中的代码效率和 MISRA-C 支持;
- 为 Simulink Fixed Point 中定点系统的分析和定标提供了新的图形界面;
- 在 SimEvents 中提供向量和矩阵支持;
- 推出一个新的多畴物理建模产品 Simscape。

1.1.1 MATLAB 的新版本特性

MATLAB R2007a 包含 MATLAB 7.4 和 Simulink 6.6 的更新。本小节将对这些更新做详细的介绍。

1. MATLAB 7.4

1) 开发环境

- 编辑器中增强了分隔符匹配, 包括语言构造如 for、if 和 switch;
- 能够自动整理编辑器中的 M-Lint 警告信息子集;

- 增强 Windows 和 Linux 平台上的桌面工具管理, 包括最大化和隐藏工具的功能;
- 支持数组编辑器中的撤消和重复操作, 以及用于交互式操作的即时更正;
- 在未评估代码时也能够发布 M 代码函数;
- 能够从 Windows Explorer 中将 MATLAB 文件打开到已经正在运行的 MATLAB;
- 当前版本的帮助浏览器搜索结果中也包括了演示。

2) 语言和编程

- 新的 inputParser 类, 使得解析和验证 M 文件函数输入参量更为容易;
- 新的 assert 函数, 如果条件不是真, 则发出错误, 允许代码内建测试;
- 新的 verLessThan 函数, 可检查 MATLAB 的版本, 让用户更容易地编写在多个版本中运行的代码;
- 将参量编号到格式化字符串函数(例如 sprintf), 无需在如转换等应用程序中对参数进行重新排序。

3) 数学

- 新的 bsxfun 函数, 为需要单个扩展的二进制运算实现更简单的代码编写和更高的性能;
- 新的 ilu 函数, 用于执行不完全 LU 因数分解, 以作为稀疏迭代方法的预调节器。

4) 文件 I/O 和外部接口

- textscan 函数新的 CollectOutput 选项, 用于自动搜集单个数组中相同数据类型的值;
- 能够程序化连接到一个 COM Automation 服务器的实例, 使用自定义界面创建 Automation 服务器, 并且充分利用事件界面。

5) 性能和大数据集处理

- 支持多个线性代数和元素方式数值运算的多线程计算, 可以在多核和多处理器系统上提升性能;
- 所有平台上的优化基本线性代数子程序(BLAS)库的版本得到升级;
- 提升 Windows XP 64 位平台的性能;
- 对于 Solaris, 支持 64 位 MATLAB, 允许处理更大的数据集。

2. Simulink 6.6

1) 多维信号支持

- 对创建、使用和记录二维以上的信号的模型提供仿真和代码生成支持;
- 用于处理多维信号 Permute Dimensions 和 Squeeze 模块;
- 增强 Assignment、Selector、Concatenate 以及其他模块的功能以支持二维以上的信号。

2) 大型建模

- 配置集引用可在模型引用层次中的模型之间共享配置集;
- 模型引用现在支持非零仿真起始时间;
- 提供可以删除子系统或模型内容, 以及在子系统和模型之间复制内容的实用函数;
- 提供状态日志和模型线性化命令中的状态名支持;
- 提供识别模型所需的文件以及能将其压缩成 zip 文件的工具;
- 新增模块、模型指导器检查以及实用函数, 用于检测作为向量的总线, 并自动将这

些总线转化为向量;

- 新的警告, 用于指示在 Simulink 中载入模型时, 另一个程序已经改变了磁盘上的模型文件;

- 新的警告, 用于指示在 MATLAB 路径上存在具有相同名称的多个模型或模块库;
- 模块回调, 用于在 Simulink 复制或删除模块之前执行自定义代码。

3) MATLAB 语言的支持功能

- 增强的嵌入式 MATLAB 函数模块, 支持多维信号、帧信号、函数句柄和 31 种额外标准库函数;

- 新增命令行功能以检查现有 M 函数是否符合嵌入式 MATLAB 子集, 从而便于它们作为嵌入式 MATLAB 函数包含在 Simulink 和 Stateflow 中。

4) 嵌入式软件设计和实现

- 改进的 MISRA-C, 支持生成子系统、图和静态库文件;
- 增强 Legacy Code Tool, 支持向量和复数数据类型工作;
- 能够控制模型的阶跃函数原型;
- 更高效的代码, 用于实现子系统和宽信号运算;
- 支持通过封装参数传递混合数据类型对象。

5) 增强的可用性

- 改进模型指导器, 用于导航检查和显示状态;
- 端口名称显示选项, 用于显示子系统模块的信号名称和对应端口模块的端口号;
- 由模型浏览器显示的对象属性的自定义更为方便。

1.1.2 MATLAB 的新产品概况

1. 新增的产品

MATLAB R2007a 新增了 Link for Cadence Incisive 和 Simscape 两个新产品。Link for Cadence Incisive 提供了一个协同仿真接口, 将 MATLAB/Simulink 专用集成电路(ASIC)的设计与现场可编程门阵列(FPGA)的开发的硬件设计流程集成在一起, 在 MATLAB/Simulink 与 Cadence 设计系统的 Incisive 平台仿真器之间建立了一个双向的链接。利用 Link for Cadence Incisive 能够实现在 MATLAB/Simulink 之中验证 HDL 的设计。它提供了 Verilog 语言的协同仿真支持, 并通过 Verilog 模块提供了 VHDL 语言与混和语言的协同仿真支持。

Link for Cadence Incisive 的主要特性如下:

- 支持 Verilog 语言;
- Simulink 模型与一个或多个 Incisive 仿真器相结合;
- MATLAB 测试台功能, 允许使用 MATLAB 代码来对 HDL 代码进行仿真和检查;
- MATLAB 组件功能, 允许使用 MATLAB 代码的仿真来替换那些并不是在 HDL 内编写的代码实体;
- 可选的 MATLAB/Simulink 和 Incisive 之间的通信模式, 为用户提供了共享内存(面向快速性能)和 TCP/IP Sockets(面向多功能);
- 交互式或批处理方式协同仿真、调试、测试以及 MATLAB 中产生的 HDL 代码的验证。

Simscape 是在 Simulink 基础上的扩展工具模块,用于实现多畴物理系统的建模和仿真。Simscape 可以模拟如机械、电气、液压以及其他物理学领域的系统,可广泛应用于航空业、国防、汽车业和工业装备制造制造业。附带的物理建模产品可将 Simscape 扩展到更复杂的液压系统、三维机械系统和一维机械系统的建模。

Simscape 的主要特性如下:

- 使用统一环境实现机械、电气和液压系统的建模和仿真;
- 提供建模所需的模块库和基本数学元素;
- 提供桥接不同建模域的连接模块;
- 能够对由 SimMechanics、SimDriveline 或 SimHydraulics 创建的模型进行编辑和仿真。

2. 终止的产品

MATLAB R2006b 中包含的两个产品在 MATLAB R2007a 中已不再存在,它们分别是:

- Embedded Target for Motorola HC12;
- xPC TargetBox。

1.2 桌面工具与开发环境

桌面工具与开发环境能够帮助用户方便地使用 MATLAB 函数和文件。本节将介绍 MATLAB R2007a 的桌面工具与开发环境。当启动运行 MATLAB 时,最先显示的是它的桌面,桌面主要由主菜单、工具栏、当前路径、工作区间、命令窗、历史命令记录以及 Start 菜单组成,如图 1-1 所示。

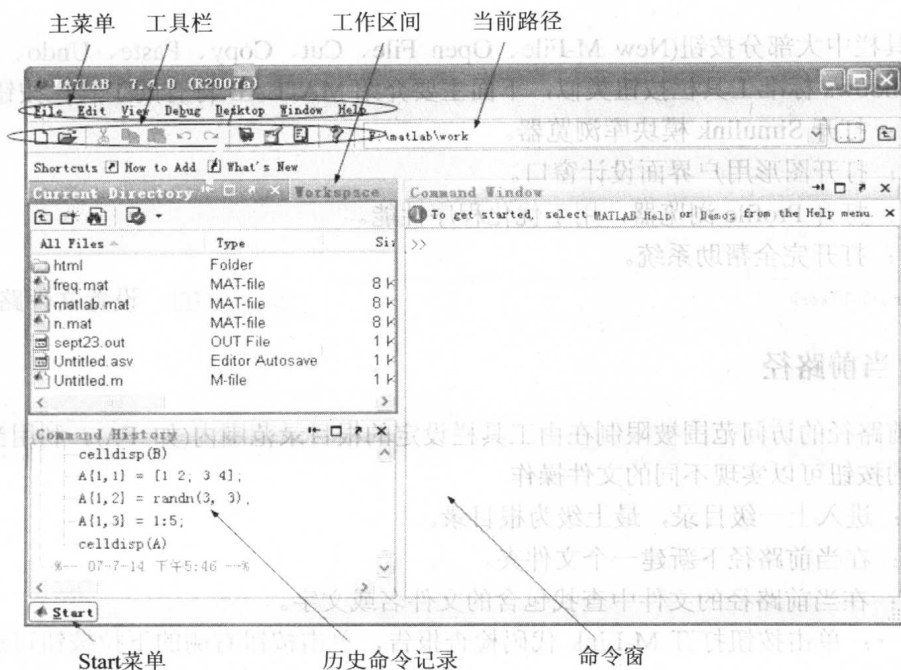


图 1-1 MATLAB R2007a 主界面

1.2.1 主菜单

主菜单中的大部分选项功能与 Windows 标准菜单界面类似, 本小节主要介绍默认情况下的主菜单。

【File 菜单】:

- Import Data: 向工作区间导入数据;
- Save Workspace As: 将工作区间变量存储在一个 MAT 文件中;
- Set Path: 设置搜索路径;
- Preferences: 环境设置。

【Edit 菜单】: 用于复制、粘贴文字或文件, 与 Windows 的 Edit 菜单基本类似。

【Debug 菜单】: 用于设置程序的调试。

【Desktop 菜单】: 用于设置当前窗口的显示形式, 以及打开或关闭某个窗口, 显示或不显示某个工具栏。

【Window 菜单】: 用于激活某个窗口。

【Help 菜单】: 打开全部产品系列帮助文件或打开某个部分的帮助。


【View 菜单】: 当“Current Directory”被激活时, 主菜单上会增加一个 View 菜单, 用于设置当前路径下所要显示的文件类型; 当“Workspace”被激活时, 主菜单上也会增加一个 View 菜单, 用于设置工作区间变量的显示形式。

【Graphics 菜单】: 当“Workspace”被激活时, 主菜单上还会增加一个 Graphics 菜单, 用于打开绘图工具来绘制工作区间的变量。


1.2.2 工具栏

工具栏中大部分按钮(New M-File、Open File、Cut、Copy、Paste、Undo、Redo 等)功能与 Windows 标准工具栏按钮类似, 下面主要介绍 MATLAB 特有的工具栏按钮。

: 打开 Simulink 模块库浏览器。

: 打开图形用户界面设计窗口。

: 打开 Profile 浏览器, 用于优化程序性能。

: 打开完全帮助系统。

F:\matlab\work





: 设置文件路径。


1.2.3 当前路径

当前路径的访问范围被限制在由工具栏设定的根目录范围内(如 F:\)。利用当前路径窗口提供的按钮可以实现不同的文件操作。

: 进入上一级目录, 最上级为根目录。


: 在当前路径下新建一个文件夹。

: 在当前路径的文件中查找包含的文件名或文字。


: 单击按钮打开 M-Link 代码检查报告, 单击按钮右侧的下拉按钮可选择打开不同类型的报告。


1.2.4 工作区间


工作区间窗口下会显示现有内存中的变量以及变量的各种信息，不同类型变量显示的图标也不同。工作区间还提供了很多特殊的按钮来实现对变量的操作。

：新建一个变量，可以打开数组编辑器对其赋值。


：打开数组编辑器对选中变量进行查看或编辑。

：从硬盘向工作区间导入数据。

：保存工作区间的所有数据。

：打印工作区间。

：删除工作区间的变量。

：单击按钮绘制选中变量的图，单击按钮右侧的下拉按钮选择不同的绘制方式。

1.2.5 命令窗


MATLAB 的命令窗是输入数据、运行 MATLAB 函数或 M 文件、显示结果的主要工具，它提供了最快捷的操作方式。在命令窗键入变量及其取值，就可以创建一个变量；在命令窗键入函数及其参数，就可以运行该函数；在命令窗键入 M 文件名或 Simulink 模型文件名，就可以运行该文件。

1.2.6 历史命令记录

历史命令记录窗口中显示的是近期在命令窗下运行的命令。绿色注释为每次启动运行 MATLAB 的日期时间，可以单击其左侧的“+”来显示该部分或“-”来隐藏该部分。如果需要查找某个历史命令，激活任何一个历史命令，然后输入想要查找的历史命令名，当输入第一个字母时，MATLAB 就会给出提示以帮助用户查找。如果需要运行某个历史命令，双击该历史命令即可。用户还可以在命令窗按下“↑”键或“↓”键来选择需要调入的历史命令。

对于一条或多条选中的历史命令，单击右键可以弹出操作菜单，实现历史命令的编辑、运行、创建 M 文件或进行程序性能优化等操作。

1.2.7 Start 菜单

 **Start** 【Start 菜单】：位于 MATLAB 主窗口的左下角，用于直接打开各种 MATLAB 工具。

1.3 编辑/调试器

MATLAB 提供了建立、编辑和调试文件的强大工具。利用 MATLAB 的编辑/调试器不仅能给编程带来很大方便，还能实现许多特殊功能。

1.3.1 M 文件的创建

M 文件的创建方法有很多种，可以通过在主菜单选择 **File > New > M-File** 或在工具栏单击新建按钮来创建，也可以在当前路径窗口下通过右键菜单来创建，还可以通过 **edit** 命令来创建。利用工具栏按钮新建的 M 文件如图 1-2 所示。

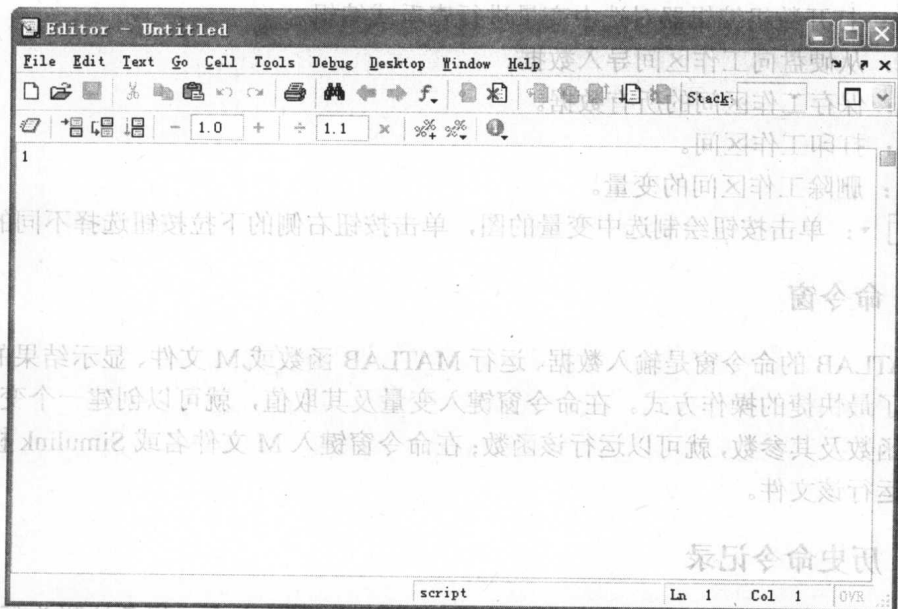


图 1-2 MATLAB 编辑器界面

MATLAB 编辑器标题栏下的第一行为主菜单，第二行为工具栏，第三行为单元工具栏。编辑器主菜单的功能与 MATLAB 主菜单的功能类似，但又有如下几个特有的功能菜单：

【Text 菜单】：

- 执行选中的代码；
- 将选中区域注释掉或删除行前的注释符；
- 设置选中区域的缩进，使程序便于阅读；
- 字母大小写变换。

【Go 菜单】：

- 将光标移动到行首或行尾，或移动到指定行；
- 设置书签或移动光标到书签处。

【Cell 菜单】：

- 设置单元模式有效或无效；
- 运行当前单元或者整个文件；
- 分割单元或插入特殊文本(如单元标题、斜体字和公式)。

【Tool 菜单】：

- 打开 M-Lint 代码检查报告；

- 打开 Profile。

【Debug 菜单】:

- 程序调试;
- 设置断点;
- 清除断点。

工具栏与单元工具栏的基本功能都包含在主菜单中, 这里不再重复介绍。

1.3.2 M 文件的运行和调试

1. M 文件的运行

M 文件的运行方式有很多种, 可以在命令窗输入文件名运行, 也可以在编辑器中通过 Run 来运行, 还可以选中 M 文件中全部或部分代码并执行选中部分。

2. M 文件的调试

M 文件需要通过设置断点进入调试模式。设置断点最简单的方法就是单击该行左边行号右侧的“—”, 也可以通过工具栏或 Debug 菜单来设置。

断点设置完毕后, 单击 Run, 程序指针将指向第一个断点。之后, step、step in、step out 等按钮的状态变为可用状态, 从而可以选择各种方式对程序进行调试, 鼠标所指变量将显示变量的值。

有关 M 文件的具体调试方法将在第 5 章介绍。

1.3.3 M 文件的结果发布

MATLAB 提供了两种途径的结果发布: 一种是在 M 文件编辑器下使用单元, 另一种是在 Microsoft Word 环境下使用 Notebook。本小节将介绍一个通过使用单元将结果发布为 HTML 格式文件的实例。

1. 为发布文档添加标题

- 选择 Cell > Insert Text Markup > Cell Title 就可以插入一个标题, 将默认的标题 TITLE 修改为自定义标题, 如 Plot Sine Wave;

- 在标题下面可以添加文字注释, 如 %Program for plot wave。

2. 添加文档分类目录

- 选择 Cell > Insert Cell Divider 就可以添加一个分类目录, 在双百分号后空一格书写分类名, 如 %% Calculate、%% Plot Figure and Define Title and Label;

- 在各分类下编写代码或插入特殊格式文字、公式。选择 Cell > Insert Text Markup>Bold/Italic/Monospaced Text 可以插入特殊格式的文字; 选择 Cell > Insert Text Markup > TeX Equation 可以插入公式。

3. 保存并发布结果

- 选择 File > Save and Publish To HTML 就可以将结果发布为 HTML 格式的文件;
- 选择 File > Save and Publish To... 可以将结果发布为 XML、LaTeX、Word 文档等格式的文件。

以下为单元模式的 M 文件源代码：

```
%% Plot Sine Wave
% Program for plot wave
%% Calculate
%%
%
%  $0 < x < 4\pi$ 
%
%%
%  $y = \sin(x)$ 
x = 0:0.01:6*pi;
y = sin(x);
%% Plot Figure and Define Title and Label
plot(x,y);
title('Sine Wave');
xlabel('x');
ylabel('y');
```

以 HTML 格式发布该文件，其发布结果如图 1-3 所示。

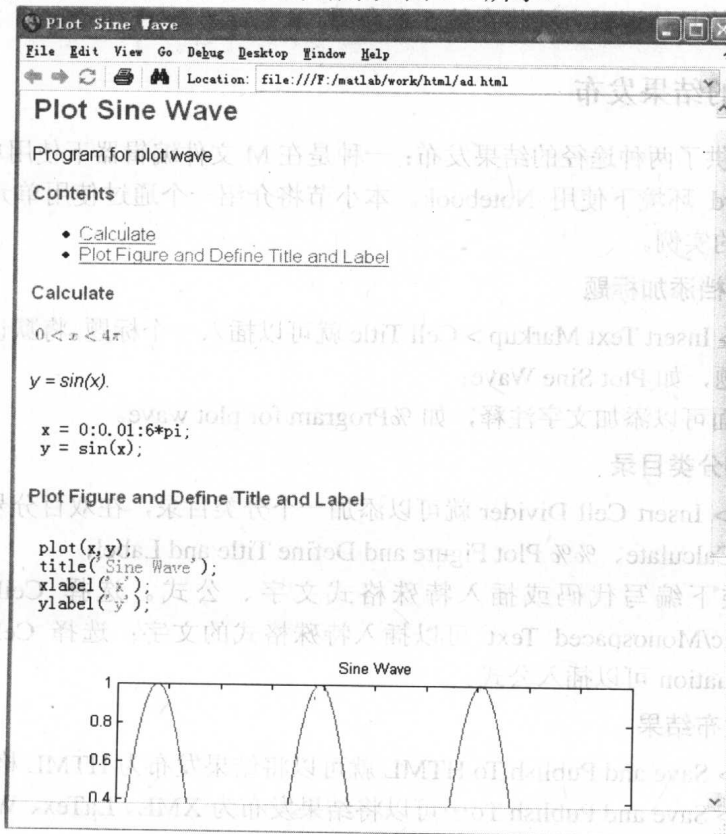


图 1-3 HTML 格式的发布结果

1.4 帮助系统

MATLAB 为用户提供了非常完善的帮助系统,通过命令窗查询帮助或打开帮助浏览器都可以很方便地获得帮助信息,通过帮助信息可以很容易地掌握所要学习的内容。

1.4.1 命令窗查询帮助

在命令窗输入帮助命令可以快速地获取帮助,以下列出的是 MATLAB 的所有帮助函数:

- **builddocsearchdb**: 建立可查找的文档数据库;
- **demo**: 通过帮助浏览器查看演示程序;
- **doc**: 在帮助浏览器中查看帮助;
- **docopt**: UNIX 平台的 Web 浏览器;
- **docsearch**: 在帮助浏览器中搜索并查看帮助;
- **echodemo**: 在命令窗单步运行 M 文件演示;
- **help**: 在命令窗获取 MATLAB 函数帮助;
- **helpbrowser**: 打开帮助浏览器访问在线文档或演示;
- **helpwin**: 打开帮助浏览器查看所有函数的帮助;
- **info**: 显示 MathWorks 的联系信息;
- **lookfor**: 按照关键字查找所有帮助条目;
- **playshow**: 运行 M 文件演示程序;
- **support**: 打开 MathWorks 技术支持网页;
- **web**: 在 Web 浏览器或帮助浏览器中打开网址;
- **whatsnew**: 查询 MathWorks 的 Release Notes。

最常用的命令是 **help**, 如 **help **命令给出所有运算符和特殊符号的信息; **help FUN** 给出函数 **FUN** 的语法和功能; **help TOOL** 给出工具箱 **TOOL** 的内容; **help help** 给出使用 **help** 命令获取帮助的方法。

1.4.2 帮助浏览器

通过主菜单的 **Help** 菜单选项或者工具栏上的帮助按钮打开帮助浏览器,可以使用户使用 MATLAB 功能强大的帮助系统。帮助浏览器的菜单与 MATLAB 菜单功能基本类似,它有更重要的功能,即四个标签页: 分类内容目录(Contents)、索引(Index)、查找结果(Search Results)、演示(Demos), 一个下拉列表框: 查找(Search for)。

- **Contents**: 查看帮助文档的分类内容目录, 默认情况下目录与显示页同步;
- **Index**: 在帮助文档中查找指定的索引条目;
- **Search Results**: 分开显示帮助文档与演示中的 Search for 的查找结果;
- **Demos**: 查看和运行 MathWorks 的演示示例;

- Search for: 在帮助文档和演示中查找指定文字，查找方法有以下几种：
 - 将搜索关键字放在双引号内实现精确匹配，如 “word1 word2”;
 - 使用通配符 * 代替一个或多个字母，如 wo*d1;
 - 使用布尔运算符进行高级查找，如 word1 NOT word2。

第2章 矩阵与数组

矩阵是 MATLAB 最基本的数据结构，其元素可以是数值类型、逻辑类型、字符类型甚至其他 MATLAB 数据类型。无论单一数据还是一组数据，MATLAB 均采用矩阵方式来存储。对于单一数据，MATLAB 用 1×1 的矩阵来表示；对于一组数据，MATLAB 用 $1 \times n$ 的矩阵来表示。MATLAB 也支持多维的数据结构，即多维数组。

2.1 创建矩阵

所有输入 MATLAB 的数据均以矩阵或者多维向量的形式存储，即使最简单的数值变量也会存储为 1×1 的矩阵。本节将介绍如何创建矩阵。

2.1.1 创建矩阵和数值序列

1. 创建矩阵

创建一个矩阵最简单的方法就是用中括号 `[]` 将元素置于其中。将中括号中的元素用空格或者逗号隔开，就可以创建一个行矩阵；将中括号中的元素用分号隔开，就可以创建一个列矩阵。

【例】 在命令窗创建矩阵。

在命令窗输入：

```
>> A = [13 61 95 -8 12]
```

运行结果：

```
A =  
    13    61    95    -8    12
```

在命令窗输入：

```
>> B = [1,2,3;4,5,6;7,8,9]
```

运行结果：

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

当矩阵的阶数较大时，直接在命令窗口输入矩阵元素较为不便。为解决此问题，可以

先将矩阵按创建原则写入一个 M 文件中, 在 MATLAB 的命令窗口或程序中直接执行该 M 文件, 即将矩阵调入工作区间。

【例】 在 .M 文件中创建矩阵。

在编辑窗输入:

```
C=[1 2 3 4 5 6 7 8 9 10
    11 12 13 14 15 16 17 18 19 20
    21 22 23 24 25 26 27 28 29 30]
```

运行结果:

```
C =
     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30
```

2. 创建数值序列

在矩阵的合并以及索引操作中, 常常要用到数值序列, 因此 MATLAB 提供了一个创建数值序列的特殊运算符“:”。使用“:”运算符可创建一个数值序列, 步进值的大小可以设置, 默认步进值为 1。

利用“:”运算符创建时(默认步进), 数值序列的起始值和结束值可以是小数, 也可以为负, 起始值和结束值之差也可以不是整数, 但是起始值必须小于结束值, 否则会返回一个空矩阵。

【例】 创建步进值为 1 的数值序列。

在命令窗输入:

```
>> A=10:14, B=-2.2:4.3
```

运行结果:

```
A =
    10    11    12    13    14
B =
   -2.2000   -1.2000   -0.2000    0.8000    1.8000    2.8000    3.8000
```

可以在起始值和结束值之间插入步进值的设置。

【例】 创建步进值为-0.4 的数值序列。

在命令窗输入:

```
>> C=1.6:-0.4:-0.8
```

运行结果:

```
C =
    1.6000    1.2000    0.8000    0.4000    0.0000   -0.4000   -0.8000
```

2.1.2 创建特殊矩阵

MATLAB 提供了许多创建特殊矩阵的函数, 表 2-1 列举了一些常用的矩阵创建函数。

表 2-1 矩阵创建函数

函数名	描 述
ones	创建全部元素为 1 的矩阵或向量
zeros	创建全部元素为 0 的矩阵或向量
eye	创建单位矩阵
magic	创建幻方矩阵
diag	创建对角矩阵
rand	创建均匀分布伪随机数组
randn	创建正态分布随机数组
randperm	创建随机置换向量

这些函数的返回值通常为 double 类型的矩阵。也可以使用 ones、zeros、eye 等函数创建任意数据类型的矩阵。

【例】 创建 int8 类型的单位矩阵。

在命令窗输入：

```
>> A=eye(4,4,'int8')
```

或

```
>> A=eye([4,4],'int8')
```

运行结果：

```
A =
    1     0     0     0
    0     1     0     0
    0     0     1     0
    0     0     0     1
```

【例】 创建元素全为 0 的矩阵。

在命令窗输入：

```
>> B=zeros(2,3)
```

运行结果：

```
B =
    0     0     0
    0     0     0
```

【例】 创建幻方矩阵。

在命令窗输入：

```
>> C=magic(3)
```

运行结果：

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

命令 diag(diag(X)) 可从一个已有的矩阵 X 中抽取对角元素，使其成为对角矩阵，也可

以利用 `diag(V)` 将向量 `V` 转化为对角矩阵。

【例】 创建对角矩阵。

在命令窗输入：

```
>> C=magic(3);
```

```
>> D=diag(diag(C))
```

运行结果：

D =

```
8    0    0
0    5    0
0    0    2
```

如果在命令窗输入：

```
>> D=diag([8,5,2])
```

运行结果：

D =

```
8    0    0
0    5    0
0    0    2
```

`rand` 函数用于产生在 0~1 之间均匀分布的伪随机数。

【例】 创建随机矩阵。

在命令窗输入：

```
>> E=rand(4,3)
```

运行结果：

E =

```
0.392227019534168    0.031832846377421    0.823457828327293
0.655477890177557    0.276922984960890    0.694828622975817
0.171186687811562    0.046171390631154    0.317099480060861
0.706046088019609    0.097131781235848    0.950222048838355
```

`randperm(n)` 命令将 1~n 的整数随机排序后产生一个行向量。

【例】 创建随机置换向量。

在命令窗输入：

```
>> F=randperm(6)
```

运行结果：

F =

```
6    3    5    1    2    4
```

2.1.3 合并矩阵

1. 矩阵的合并

MATLAB 可以将相同行数或列数的矩阵合并，也可以将不同阶数的矩阵合并为块状矩

阵。利用中括号 [] 或者矩阵串接函数可以将相同行数或列数的矩阵合并，利用矩阵串接函数还可以合并块状矩阵。

【例】 合并同阶矩阵。

在命令窗输入：

```
>> A=magic(3),B=ones(2,3),C=zeros(3,4),D=[A;B],E=[A C]
```

运行结果：

A =

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

B =

```
1     1     1
1     1     1
```

C =

```
0     0     0     0
0     0     0     0
0     0     0     0
```

D =

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

E =

```
8     1     6     0     0     0     0
3     5     7     0     0     0     0
4     9     2     0     0     0     0
```

MATLAB 中包含如表 2-2 所示的矩阵串接函数。

表 2-2 矩阵串接函数

函 数 名	描 述
cat	按指定维串接矩阵
horzcat	水平串接矩阵
vertcat	垂直串接矩阵
repmat	复制并平铺矩阵
blkdiag	分块对角化串接矩阵

cat 函数与用中括号 [] 合并矩阵的方法作用基本相同，但是它还可以按照指定维串接更高维数的数组。

【例】 合并同阶矩阵。

在命令窗输入：

```
>> A=magic(3),B=ones(3),C=cat(2,A,B),C=horzcat(A,B)
```

运行结果:

A =

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

B =

```
1     1     1
1     1     1
1     1     1
```

C =

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

C =

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

【例】 复制并平铺矩阵。

在命令窗输入:

```
>> A=magic(3),D= repmat(A,1,3)
```

运行结果:

A =

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

D =

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

blkdiag 函数从对角方向合并矩阵,形成的分块对角化矩阵除对角之外的元素均为 0。

【例】 创建分块对角化矩阵。

在命令窗输入:

```
>> A=magic(3),B=[1 2 3;4 5 6],E=blkdiag(A,B)
```

运行结果:

A =

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


B =

1	2	3
4	5	6

E =

8	1	6	0	0	0
3	5	7	0	0	0
4	9	2	0	0	0
0	0	0	1	2	3
0	0	0	4	5	6

2. 不同数据类型矩阵的合并

MATLAB 中的矩阵可以由不同数据类型的元素来合并,其结果为某个特定类型的矩阵, MATLAB 自动将所有元素转化为其中的一种数据类型。表 2-3 给出了 5 种数据类型之间的转化结果。

表 2-3 各数据类型之间转化结果

类 型	character	integer	single	double	logical
character	character	character	character	character	invalid
integer	character	integer	integer	integer	integer
single	character	integer	single	single	single
double	character	integer	single	double	double
logical	invalid	integer	single	double	logical

例如,用 double 类型和 single 类型合成矩阵,最终得到的会是 single 类型的矩阵, MATLAB 自动将 double 类型的元素转化为 single 类型。如果合成矩阵中含有空矩阵,空矩阵将会被忽略。

【例】 合并 double 类型和 single 类型。

在命令窗输入:

```
>> x = [single(-2.8) pi 5.13*10^100],class(x)
```

运行结果:

```
x =
    -2.8000    3.1416   Inf
ans =
single
```

【例】 合并 double 类型和 integer 类型。

在命令窗输入:

```
>> y=[int8(-22) int8(13) pi 45/16],class(y)
```

运行结果:

```
y =
   -22    13     3     3
ans =
int8
```

【例】 合并 character 类型和 double 类型。

在命令窗输入：

```
>> z=['A' 'B' 'C' 68 69],class(z)
```

运行结果：

```
z =  
    ABCDE  
ans =  
    char
```

【例】 合并 character 类型和 double 类型。

在命令窗输入：

```
>> u=[true false pi],class(u)
```

运行结果：

```
u =  
    1.0000         0    3.1416  
ans =  
    double
```

2.2 索 引

本节将介绍如何利用下标和索引对矩阵元素进行访问和赋值。

2.2.1 线性索引

MATLAB 中可以只用一个下标访问二维矩阵中的元素。MATLAB 对数据的存储形式不同于数据在命令窗中显示的形式，而是以列元素为序的方式存储。例如，矩阵 $A=[2\ 6\ 7; 4\ 2\ 8; 3\ 0\ 9]$ 实际上是以如下形式存储的：

```
2, 4, 3, 6, 2, 0, 7, 8, 9
```

如果要访问某个 $n \times m$ 阶矩阵的 row 行 col 列元素，则不但可以使用 $A(\text{row}, \text{col})$ 命令来访问，还可以使用 $A((\text{col} - 1) * n + \text{row})$ 来访问。

【例】 访问 3 阶幻方矩阵 A 中 2 行 3 列的元素。

在命令窗输入：

```
>> A=magic(3),A(8)
```

运行结果：

```
A =  
    8     1     6  
    3     5     7  
    4     9     2  
ans =  
    7
```

2.2.2 访问单个元素

要访问某个矩阵的指定元素，需要用命令 $A(\text{row}, \text{column})$ 指定行号和列号，其中， A 为矩阵变量，行号(row)在前，列号(column)在后。

【例】访问3阶幻方矩阵 A 中2行3列的元素。

在命令窗输入：

```
>> A=magic(3),A(2,3)
```

运行结果：

```
A =  
      8      1      6  
      3      5      7  
      4      9      2  
  
ans =  
      7
```

2.2.3 访问多个元素

MATLAB 可以一次性访问一个矩阵中的多个元素。对于如下 4×4 的幻方矩阵 A ，可以一次性计算它的列元素和，也可以用“:”运算符来减小它的尺寸。

【例】访问4阶幻方矩阵 A 的多个元素。

在命令窗输入：

```
>> A=magic(4),sum(A)
```

运行结果：

```
A =  
     16      2      3     13  
      5     11     10      8  
      9      7      6     12  
      4     14     15      1  
  
ans =  
     34     34     34     34
```

继续在命令窗输入：

```
>> B=A(2:3,:)
```

或输入：

```
>> B=A(2:3,1:4)
```

运行结果：

```
B =  
      5     11     10      8  
      9      7      6     12
```

其中，“2:3”代表从第2行到第3行，“1:4”代表从第1列到第4列，“:”代表所有列。

利用数值序列也可以访问或者修改矩阵中的多个元素。

【例】 修改 4 阶幻方矩阵 A 的多个元素。

在命令窗输入：

```
>> C=A;C(1:3:end)=0
```

运行结果：

```
C =  
    0     2     3     0  
    5    11     0     8  
    9     0     6    12  
    0    14    15     0
```

可以看出，以 3 为步进，从 1 到 16 的所有元素的值均被修改为 0。“end”关键字在此处代表矩阵元素的最后一个索引，即 16。在程序不知道矩阵具体阶数的情况下，使用关键字“end”会为编程带来很大方便。

一个矩阵也可以作为其他矩阵元素的索引。例如，可以用 4 阶幻方矩阵的第 9 个元素组成的矩阵作为索引。

【例】 用 4 阶幻方矩阵 A 的元素组成的矩阵作为索引。

在命令窗输入：

```
>> B=A(9*ones(2,5))
```

运行结果：

```
B =  
     3     3     3     3     3  
     3     3     3     3     3
```

2.3 获取矩阵信息

本节将介绍如何获取矩阵的基本信息。

2.3.1 矩阵的阶数与维数

表 2-4 给出有关获取矩阵维数、阶数的函数。

表 2-4 有关矩阵阶数与维数的函数

函 数 名	描 述
length	获取向量长度和矩阵各维的最大阶数
ndims	获取矩阵维数
numel	获取矩阵的元素数目
size	获取矩阵各维的长度

【例】 删除矩阵的行(降低矩阵阶数)。

在命令窗输入：

```
>> A=magic(4),A(1:2,:)=[]
```

运行结果:

A =

```
16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

A =

```
9     7     6    12
     4    14    15     1
```

【例】 获取向量长度。

在命令窗输入:

```
>> x=1:10,length(x)
```

运行结果:

x =

```
1     2     3     4     5     6     7     8     9    10
```

ans =

10

【例】 获取矩阵维数、阶数与元素数。

在命令窗输入:

```
>> A=magic(3),ndims(A)
```

运行结果:

A =

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

ans =

2

继续在命令窗输入:

```
>> size(A)
```

运行结果:

ans =

```
3     3
```

或输入:

```
>> size(A,2)
```

运行结果:

ans =

3

再在命令窗输入:

```
>> numel(A)
```

运行结果:

```
ans =  
9
```

对于 size 函数, size(A,1)得到 A 的行阶数, size(A,2)得到 A 的列阶数, size(A,n)得到 A 的第 n 维的阶数。

2.3.2 矩阵元素的数据类型

表 2-5 给出了有关判断矩阵元素数据类型的函数。

表 2-5 判断矩阵元素数据类型的函数

函 数 名	描 述
isa	判断输入是否为指定数据类型
iscell	判断输入是否为 cell 数组
iscellstr	判断输入是否为字符串 cell 数组
ischar	判断输入是否为 character 数组
isfloat	判断输入是否为 single 或 double 数据类型
isinteger	判断输入是否为 integer 数据类型
islogical	判断输入是否为 logical 数组类型
isnumeric	判断输入是否为 numeric 数组
isreal	判断输入是否为 real 数组
isstruct	判断输入是否为 structure 数组

【例】 使用 isnumeric 和 isreal 数据类型判断函数。

在命令窗输入:

```
>> A = [5+i 8/3 4.233 9j pi 15-2i];  
for m = 1:numel(A)  
    if isnumeric(A(m)) && isreal(A(m))  
        disp(A(m))  
    end  
end
```

运行结果:

```
2.6667  
4.2330  
3.1416
```

2.3.3 矩阵的数据结构

表 2-6 给出了有关判断矩阵元素数据结构的函数。

表 2-6 判断矩阵元素数据结构的函数

函 数 名	描 述
isempty	判断输入是否为空数组
isscalar	判断输入数组是否为标量
issparse	判断输入是否为稀疏矩阵
isvector	判断输入数组是否为向量

2.4 基本操作和运算

MATLAB 提供了极为丰富的矩阵操作和运算功能，以便用户实现对矩阵的运算分析。用户不但可以扩大或缩小矩阵，改变矩阵的形状，还可以对矩阵进行算术运算、关系运算和逻辑运算。

2.4.1 矩阵的扩大和缩小

1. 矩阵的扩大

如果在矩阵尺寸外存储一个元素，那么这个矩阵将扩大到指定的尺寸并能容纳新添加的元素。

【例】 用元素扩展矩阵。

在命令窗输入：

```
>> A=magic(3),A(4,5)=100
```

运行结果：

A =

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

A =

```
8     1     6     0     0
3     5     7     0     0
4     9     2     0     0
0     0     0     0    100
```

也可以利用数值序列对矩阵进行扩展。

【例】 用数值序列扩展矩阵。

在命令窗输入：

```
>> A=magic(3),A(3:4,3:5)=100
```

运行结果：

A =

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

A =

```
8     1     6     0     0
3     5     7     0     0
4     9    100    100    100
0     0    100    100    100
```

2. 矩阵的缩小

通过对矩阵的行或列赋空值，可以实现对矩阵的行或列的删除。

【例】 删除矩阵的行。

在命令窗输入：

```
>> A=magic(3),A(3,:)=[]
```

运行结果：

A =

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

A =

```
8     1     6
3     5     7
```

但是，不能直接删除矩阵的某个元素。如果需要删除某个元素，可以通过线性索引的方法来删除。

【例】 删除矩阵的元素。

在命令窗输入：

```
>> A=magic(3),A(3)=[]
```

运行结果：

A =

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

A =

```
8     3     1     5     9     6     7     2
```

2.4.2 改变矩阵的形状

表 2-7 给出了有关改变矩阵形状的函数。

表 2-7 改变矩阵形状的函数

函 数 名	描 述
reshape	改变矩阵的形状
rot90	旋转矩阵 90°
fliplr	水平翻转矩阵
flipud	垂直翻转矩阵
flipdim	沿指定维翻转矩阵
transpose	矩阵转置
ctranspose	矩阵复共轭转置

【例】 改变矩阵的形状。

在命令窗输入：

```
>> A=[1 4 7 10;2 5 8 11;3 6 9 12],B=reshape(A,2,6)
```

运行结果：

A =

```
1     4     7    10
2     5     8    11
3     6     9    12
```

B =

```
1     3     5     7     9    11
2     4     6     8    10    12
```

利用 transpose 函数或者转置运算符“.”可以对矩阵进行转置。

【例】 矩阵的转置。

继续在命令窗输入：

```
>> C=A.'
```

运行结果：

C =

```
1     2     3
4     5     6
7     8     9
10    11    12
```

【例】 矩阵的复共轭转置。

在命令窗输入：

```
>> A=[1+9i 2-6i 3+7i;14-6i 5+2i 2-4i],B=A'
```

运行结果：

A =

```
1.0000 + 9.0000i    2.0000 - 6.0000i    3.0000 + 7.0000i
14.0000 - 6.0000i    5.0000 + 2.0000i    2.0000 - 4.0000i
```

B =

```
1.0000 - 9.0000i    14.0000 + 6.0000i
```

2.0000 + 6.0000i 5.0000 - 2.0000i

3.0000 - 7.0000i 2.0000 + 4.0000i

【例】 将矩阵旋转 90° 。

在命令窗输入：

```
>> A=[1 4 7 10; 2 5 8 11; 3 6 9 12], B=rot90(A)
```

运行结果：

A =

```
1     4     7    10
2     5     8    11
3     6     9    12
```

B =

```
10    11    12
7     8     9
4     5     6
1     2     3
```

【例】 水平翻转矩阵。

继续在命令窗输入：

```
>> C=fliplr(A)
```

运行结果：

C =

```
10     7     4     1
11     8     5     2
12     9     6     3
```

2.4.3 矩阵的算术运算

表 2-8 给出了 MATLAB 提供的算术运算符。

表 2-8 算 术 运 算 符

运算符	描 述	实 例
+	矩阵加法运算或数组对应元素相加	A+B
-	矩阵减法运算或数组对应元素相减	A-B
*	数组对应元素相乘	A.*B
./	数组对应元素的右除	A./B
.\	数组对应元素的左除	A.\B
^	数组元素的幂	A.^B
.'	矩阵或向量的转置	A.'
/	矩阵的复共轭转置	A'
*	矩阵相乘	A*B
/	矩阵右除	A/B
\	矩阵左除	A\B
^	矩阵的幂	A^n

2.4.4 矩阵的关系运算和逻辑运算

1. 关系运算

表 2-9 给出了 MATLAB 的矩阵关系运算符。

表 2-9 关系运算符

运 算 符	描 述
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于
~=	不等于

【例】 矩阵的关系运算。

在命令窗输入：

```
>> A = [2 7 6; 9 0 5; 3 0.5 6];
```

```
>> B = [8 7 0; 3 2 5; 4 -1 7];
```

```
>> A==B
```

运行结果：

```
ans =
     0     1     0
     0     0     1
     0     0     0
```

2. 逻辑运算

MATLAB 的逻辑运算符有三种：元素方式逻辑运算符、比特方式逻辑运算符和短路逻辑运算符。

元素方式逻辑运算的操作数为两个长度相同的逻辑向量，如果向量元素为有限非零数值类型，MATLAB 自动将其转化为逻辑“1”。表 2-10 给出了 MATLAB 的元素方式的逻辑运算符。实例中，A = [0 1 1 0 1]; B = [1 1 0 0 1]。

表 2-10 元素方式的逻辑运算符

运算符	描 述	实 例
&	对应元素同时为 1 时运算结果为 1，否则为 0	A & B = 01001
	对应元素同时为 0 时运算结果为 0，否则为 1	A B = 11101
~	输入元素为 1 时运算结果为 0，否则为 1	~A = 10010
xor	对应元素相同时运算结果为 0，否则为 1	xor(A,B) = 10100

比特方式逻辑运算的操作数必须是非负整数类型的标量或数组。表 2-11 给出了比特方式的逻辑运算函数。实例中：

```
A = 28;           % binary 11100
B = 21;           % binary 10101
```

表 2-11 比特方式的逻辑运算函数

函数	描 述	实 例
bitand	二进制操作数按位与	bitand(A,B) = 20 (binary 10100)
bitor	二进制操作数按位或	bitor(A,B) = 29 (binary 11101)
bitcmp	返回 n 位整数的补码	bitcmp(A,5) = 3 (binary 00011)
bitxor	二进制操作数按位异或	bitxor(A,B) = 9 (binary 01001)

短路逻辑运算符的操作数是蕴含标量值的逻辑表达式。表 2-12 给出了短路逻辑运算符。

表 2-12 短路逻辑运算符

运算符	描 述
&&	表达式逻辑与，即两个表达式操作数同为真时，运算结果为 1，否则为 0
	表达式逻辑或，即两个表达式操作数同为假时，运算结果为 0，否则为 1

短路逻辑运算常用于 if 或者 while 语句的条件判断中。另外，短路逻辑运算还可以避免除数为零的错误：

【例】 表达式逻辑与。

在命令窗输入：

```
>> a=1;b=0;
```

```
>> x = (b ~= 0) && (a/b > 18.5)
```

运行结果：

```
x =
```

```
0
```

2.5 空矩阵、标量和向量

尽管矩阵是二维的，但在 MATLAB 里它们并不一定有着矩形的形状，例如一个 1×6 的矩阵却是线形的，本节将要讨论这些特殊的矩阵。

2.5.1 空矩阵

一个矩阵至少应该有一维，而 MATLAB 中可以存在 0×0 的空矩阵，更为复杂的空矩阵有如 5×0 维和 0×3 维的矩阵。事实上，空矩阵常常用于循环语句中，作为第一次矩阵合并或串接的矩阵，这会给编程带来很多方便。

【例】 空矩阵。

在命令窗输入：

```
>> A=[],B=ones(5,0)
```

运行结果：

```
A =
```

```
[]
```

```
B =
```

Empty matrix: 5-by-0

2.5.2 标量

单一的实数或者复数代表 MATLAB 中 1×1 的矩阵，也称做“标量”。

【例】 标量。

在命令窗输入：

```
>> A=4;B=ndims(A),isscalar(A)
```

运行结果：

```
B =  
2  
ans =  
1
```

运行结果表明，A 的维数为 2(1×1 矩阵)，但是 A 为标量。

2.5.3 向量

矩阵的某个维阶数如果等于 1，那么它就是向量。一个向量可以由其他向量组合而成，但是对应维的阶数必须为 1。例如，构成列向量的元素必须是标量或者列向量，用空矩阵合成向量将会被忽略。

【例】 向量。

在命令窗输入：

```
>> A=[pi 32 -3],B=[],C=[2 4],D=[A B C]
```

运行结果：

```
A =  
3.1416 32.0000 -3.0000  
B =  
[]  
C =  
2 4  
D =  
3.1416 32.0000 -3.0000 2.0000 4.0000
```

2.6 多维数组

多维数组是二维常规矩阵的扩展，可以利用下标来访问一个二维矩阵，第一个下标代表行索引，第二个代表列索引。如果是多维矩阵，则需要增加下标来索引。例如图 2-1 中，访问第 2 行第 3 列第 2 页的元素，需要用到下标(2,3,2)。

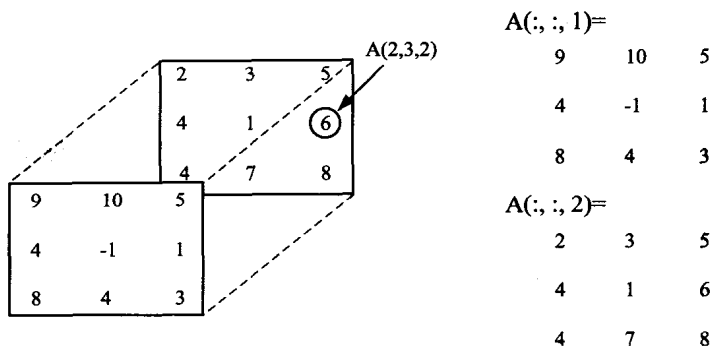


图 2-1 多维数组示意图

当一个数组维数增加时，它的下标也相应的增加。例如一个 4 维数组有 4 个下标，前两个下标代表行和列，后两个下标则分别代表数组的第 3 维和第 4 维。

2.6.1 多维数组的创建

多维数组的创建方法与矩阵的创建类似。另外，MATLAB 还提供了专门的串接函数用于创建多维数组。

最简单的创建方法就是将矩阵扩展为多维数组。

【例】 创建多维数组。

在命令窗输入：

```
>> A = [5 7 8; 0 1 9; 4 3 6]; A(:, :, 2) = [1 0 4; 3 5 6; 9 8 7]
```

运行结果：

$A(:, :, 1) =$

```
5    7    8
0    1    9
4    3    6
```

$A(:, :, 2) =$

```
1    0    4
3    5    6
9    8    7
```

与矩阵的扩展类似，多维数组也可以直接进行扩展。

【例】 扩展多维数组。

继续在命令窗输入：

```
>> A(:, :, 3) = 0
```

运行结果：

$A(:, :, 1) =$

```
5    7    8
0    1    9
4    3    6
```

```
A(:, :, 2) =
```

1	0	4
3	5	6
9	8	7

```
A(:, :, 3) =
```

0	0	0
0	0	0
0	0	0

多维数组的创建同样可以使用 `randn`、`ones`、`zeros` 等函数。

【例】 利用函数创建多维数组。

在命令窗输入：

```
>> B = randn(4,3,2)
```

运行结果：

```
B(:, :, 1) =
```

-0.6918	-1.4410	0.8156
0.8580	0.5711	0.7119
1.2540	-0.3999	1.2902
-1.5937	0.6900	0.6686

```
B(:, :, 2) =
```

1.1908	-1.6041	-0.8051
-1.2025	0.2573	0.5287
-0.0198	-1.0565	0.2193
-0.1567	1.4151	-0.9219

利用串接函数也可以实现多维数组的创建。

【例】 利用串接函数创建多维数组。

在命令窗输入：

```
>> C = cat(3, [2 8; 1 5], [1 3; 4 9])
```

运行结果：

```
C(:, :, 1) =
```

2	8
1	5

```
C(:, :, 2) =
```

1	3
4	9

2.6.2 多维数组的索引

很多运用于矩阵的方法都可以推广到多维数组中。运用下标可以访问多维数组的一个元素，同样可以使用向量作为数组的下标。

【例】 多维数组的下标。

在命令窗输入：

```
>> A=10*rand(4,3,2),b=A(1,1,2),c=A([3,4],1,2)
```

运行结果：

```
A(:,:,1) =
    7.0936    6.5510    9.5974
    7.5469    1.6261    3.4039
    2.7603    1.1900    5.8527
    6.7970    4.9836    2.2381
```

```
A(:,:,2) =
    7.5127    8.9090    1.4929
    2.5510    9.5929    2.5751
    5.0596    5.4722    8.4072
    6.9908    1.3862    2.5428
```

```
b =
    7.5127
```

```
c =
    5.0596
    6.9908
```

矩阵的线性索引法也可以运用到多维数组当中，例如 $[d_1 \ d_2 \ d_3 \ \cdots \ d_n]$ 维的数组下标为 $(s_1 \ s_2 \ s_3 \ \cdots \ s_n)$ ，其索引为

$$(s_n - 1)(d_n - 1)(d_n - 2) \cdots (d_1) + (s_{n-1} - 1)(d_n - 2) \cdots (d_1) + \cdots + (s_2 - 1)(d_1) + s_1$$

在索引中，要避免含糊的语句出现。例如，输入 $A(:,:,2) = 1:3$ 将会出现一个错误，因为它试图用一维向量访问二维目标，但是输入 $A(1,:,2) = 1:3$ ，命令则会顺利地执行。

2.6.3 改变多维数组的形状

如果不修改数组的尺寸和形状，MATLAB 会始终保持数组创建时的维数。运用 reshape 函数可以按照索引对数组重新定形。

【例】 改变多维数组的形状。

在命令窗输入：

```
>> A=cat(3,[2 8 7;4 1 5],[4 1 3;4 3 9]),B=reshape(A,3,4),C=reshape(A,2,2,3)
```

运行结果：

```
A(:,:,1) =
     2     8     7
     4     1     5
```

```
A(:,:,2) =
     4     1     3
     4     3     9
```



```

B =
     2     1     4     3
     4     7     4     3
     8     5     1     9

C(:,:,1) =
     2     8
     4     1

C(:,:,2) =
     7     4
     5     4

C(:,:,3) =
     1     3
     3     9

```

可见用 `reshape` 函数可以任意更改数组的形状，但要保证 `reshape` 函数中的数字乘积等于数组的元素总数。

如果数组某个维的阶数为 1，那么可以用函数 `squeeze` 删除该维。

【例】 多维数组的降维。

在命令窗输入：

```
>> B = repmat(5,[2 1 1 4]);size(B),C = squeeze(B)
```

运行结果：

```

ans =
     2     1     1     4

C =
     5     5     5     5
     5     5     5     5

```

2.6.4 多维数组的运算

大部分 MATLAB 数学运算函数允许将多维数组作为参量，但是函数运行的实际参量是数组的特定维，如元素、向量或矩阵。可以用数组下标或索引形式作为这些函数的输入参量。

例如，`A=cat(3,[1 2 3;9 8 7;4 6 5],[0 3 2;8 8 4;5 3 5],[6 4 7;6 8 5;5 4 3]);eig(A)` 将会返回一个错误，这是因为 `eig` 函数的输入参量必须是矩阵。如果用如下形式的数组作为输入参量，程序将顺利通过。

【例】 求数组的特征值。

在命令窗输入：

```
>> eig(A(:,:,1)),eig(A(:,:,2))
```

运行结果：

ans =

15.3638

-0.6819 + 1.2197i

-0.6819 - 1.2197i

ans =

12.9129

-2.6260

2.7131

第3章 数据类型

MATLAB 提供了许多数据类型供用户使用, 此外用户还可以根据自己的需要创建新的数据类型。利用结构和单元数组这两种数据类型, 还可以将不同类型的数据存储到一个数组当中。

MATLAB 中共有 15 种基本数据类型和 2 种自定义类型(用户类和 Java 类), 它们之间的关系如图 3-1 所示。

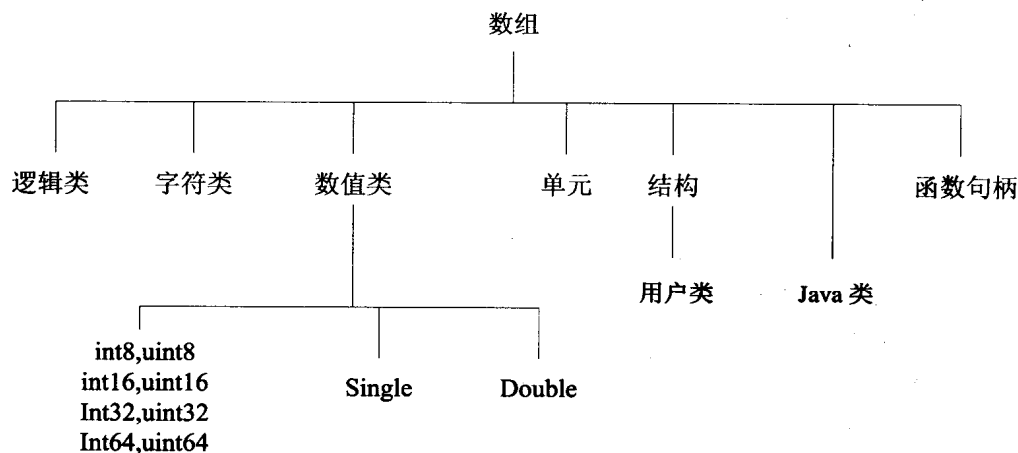


图 3-1 MATLAB 的数据类型及它们之间的关系

3.1 数值类型

MATLAB 的默认数据存储类型为双精度浮点类型(double)。用户可以根据自己的需要将数据或数组数据存储为其他类型。所有的数值类型(Numeric)数据均可以作为数组的索引。

3.1.1 整数

MATLAB 中有 4 种有符号整数类型、4 种无符号整数类型。有符号整数类型数据需要占用 1 位来表示数据的正负, 因此它的取值范围小于无符号整数。表 3-1 给出了这 8 种数据类型的取值范围和类型转化函数。

表 3-1 整数类型的取值范围和类型转化函数

数据类型	取值范围	类型转化函数
有符号 8 位整数	$-2^7 \sim 2^7 - 1$	int8
有符号 16 位整数	$-2^{15} \sim 2^{15} - 1$	int16
有符号 32 位整数	$-2^{31} \sim 2^{31} - 1$	int32
有符号 64 位整数	$-2^{63} \sim 2^{63} - 1$	int64
无符号 8 位整数	$0 \sim 2^8 - 1$	uint8
无符号 16 位整数	$0 \sim 2^{16} - 1$	uint16
无符号 32 位整数	$0 \sim 2^{32} - 1$	uint32
无符号 64 位整数	$0 \sim 2^{64} - 1$	uint64

MATLAB 的默认数据存储类型为 double 类型, 利用类型转化函数可将数据存储为整数类型。如果数据中含有小数部分, MATLAB 将按照四舍五入的方法将其转化为整数。

【例】 整数类型的转化。

在命令窗输入:

```
>> x = 4.5001;
```

```
>> int16(x)
```

运行结果:

```
ans =
```

```
5
```

用户也可以根据需要对数据的小数部分进行舍入。

【例】 舍入后转化为整数类型。

在命令窗输入:

```
>> x = 4.5001;
```

```
>> y = int16(floor(x)), z = int16(ceil(x))
```

运行结果:

```
y =
```

```
4
```

```
z =
```

```
5
```

整数类型转化函数也可将字符串类型转化为整数类型。

【例】 转化 ASCII 码。

在命令窗输入:

```
>> str = '01 Aa', int8(str)
```

运行结果:

```
str =
```

```
01 Aa
```

```
ans =
```

```
48 49 32 65 97
```

某个整数类型的变量只能与该整数类型或者 double 类型的变量进行算术运算，其运算结果仍为该类型的整数。

【例】 整数类型的算术运算。

在命令窗输入：

```
>> x = uint32(250) * pi;
>> class(x)
```

运行结果：

```
ans =
uint32
```

整数的取值范围可以通过函数 intmax 和 intmin 来查看，如果待转化的数据超出了取值范围，则将该数据变为取值范围内的最值。

【例】 整数类型的范围。

在命令窗输入：

```
>> intmax('int8'), x = int8(333)
```

运行结果：

```
ans =
127
x =
127
```

3.1.2 浮点数

MATLAB 的浮点数分为单精度浮点数类型(single)和双精度浮点数类型(double)。double 为默认数据类型，需要 64 位的存储空间，第 63 位代表符号，第 52~62 位存储指数部分且有 1023 的偏移量，第 51~0 位存储小数部分；single 需要 32 位的存储空间，第 31 位代表符号，第 30~23 位存储指数部分且有 127 的偏移量，第 22~0 位存储小数部分。

直接输入变量值就可以创建 double 类型的变量，而创建 single 类型的变量则需要输入类型转化函数。

【例】 创建浮点数。

在命令窗输入：

```
>> x = 132; y = single(132);
>> whos('x','y')
```

运行结果：

Name	Size	Bytes	Class	Attributes
x	1x1	8	double	
y	1x1	4	single	

浮点类型数据可以与 char、double、single、logical、int* 以及 uint* 类型的数据进行算术运算，其运算结果类型如表 3-2 所示。

表 3-2 浮点类型与各种类型数据运算结果

类 型	single	double	(u)int*	char	logical
double	single	double	(u)int*	double	double
single	single	single	—	single	single

要查看浮点类型的取值范围,可使用 `realmax('double')`、`realmax('single')`、`realmin('double')` 以及 `realmin('single')` 等函数来实现。

如果浮点数运算结果的精度不能令人满意,很可能是由于计算机硬件的局限导致的。硬件没有足够的位数来表示结果,就会截短数据。在每一个浮点数与最接近它的更大的浮点数之间存在一个差值,这个差值的大小不仅取决于浮点数的类型,还取决于这个浮点数的大小。

【例】 浮点相对精度。

在命令窗输入:

```
>> format long
```

```
>> eps(7),eps(3)
```

运行结果:

```
ans =
```

```
8.881784197001252e-016
```

```
ans =
```

```
4.440892098500626e-016
```

在命令窗输入:

```
>> eps(single(7)),eps(single(3))
```

运行结果:

```
ans =
```

```
4.7683716e-007
```

```
ans =
```

```
2.3841858e-007
```

对于浮点数的运算操作要特别小心,以免带来不必要的麻烦。下面给出一些常见错误的例子作为参考。

【例】 舍入误差。

在命令窗输入:

```
>> x = 1 - 3*(4/3 - 1)
```

运行结果:

```
x =
```

```
2.220446049250313e-016
```

在命令窗输入:

```
>> a = 0.0;
```

```
for i = 1:10
```

```
    a = a + 0.1;
```

```
end
```

```
>> a == 1
```

运行结果:

```
ans =
```

```
0
```

在命令窗输入:

```
>> b = 1e-13 + 1 - 1e-13; c = 1e-13 - 1e-13 + 1;
```

```
>> b-c
```

运行结果:

```
ans =
```

```
-1.110223024625157e-016
```

在命令窗输入:

```
>> sin(2*pi)
```

运行结果:

```
ans =
```

```
-2.449293598294706e-016
```

【例】 灾难对消。

在命令窗输入:

```
>> sqrt(1e-18 + 1) - 1
```

运行结果:

```
ans =
```

```
0
```

【例】 解线性方程组。

在命令窗输入:

```
>> A = [2 eps -eps; eps 1 1; -eps 1 1]; b = [2; eps + 2; -eps + 2];
```

```
>> x = A\b
```

运行结果:

```
Warning: Matrix is close to singular or badly scaled.
```

```
Results may be inaccurate. RCOND = 2.465190e-032.
```

```
x =
```

```
1.0e+015 *
```

```
0.0000000000000001
```

```
2.251799813685249
```

```
-2.251799813685247
```

3.1.3 复数

MATLAB 的复数分为实部和虚部两个部分, 虚部可以用字母 i 或 j 来表示。直接将复数的数学表达式作为命令输入, 就可以创建一个复数。另外, 运用命令 `complex(a,b)` 也可以

创建复数。complex 函数的第一个参量是实部数组，第二个参量是虚部数组。

【例】 创建复数。

在命令窗输入：

```
>> format short
```

```
>> 1 - 2j
```

运行结果：

```
ans =  
1.0000 - 2.0000i
```

或在命令窗输入：

```
>> complex(1,-2)
```

运行结果：

```
ans =  
1.0000 - 2.0000i
```

利用函数 real(z)和 imag(z)可以获取复数的实部和虚部。

【例】 获取复数的实部和虚部。

在命令窗输入：

```
>> A = [ 1 2; 3 4]; B = [ 5 6; 7 8];
```

```
>> Z=complex(A,B)
```

```
Zr = real(Z)
```

```
Zi = imag(Z)
```

运行结果：

```
Z =  
1.0000 + 5.0000i    2.0000 + 6.0000i  
3.0000 + 7.0000i    4.0000 + 8.0000i  
Zr =  
1      2  
3      4  
Zi =  
5      6  
7      8
```

3.1.4 无穷与非数

MATLAB 中分别用 inf 和 -inf 表示正无穷和负无穷，用 NAN 表示非数。

【例】 无穷。

在命令窗输入：

```
>> 1/0
```

运行结果：

```
ans =  
Inf
```


【例】 非数。

在命令窗输入：

```
>> 0/0
```

运行结果：

```
ans =  
NaN
```

3.1.5 判断数据类型

判断数据类型的各种命令如表 3-3 所示。

表 3-3 数据类型的判断命令

命 令	操 作
whos x	显示数据类型
xType = class(x)	将 x 的数据类型赋给另一个变量
isnumeric(x)	判断 x 是否为数值类型
isa(x, 'integer') isa(x, 'uint64') isa(x, 'float') isa(x, 'double') isa(x, 'single')	判断 x 是否为引号中指定的数据类型(列举出的类型包括但不限于这些数据类型)
isreal(x)	判断 x 是否为实数
isnan(x)	判断 x 是否为非数
isinf(x)	判断 x 是否为无穷
isfinite(x)	判断 x 是否为有限数

3.1.6 数据显示形式

MATLAB 默认的数据显示形式是 5 位长度的，但是用户可以通过参数选择对话框(File > Preferences)或者利用 format 函数设置数据的显示形式。如果要保存该设置，建议使用参数选择对话框进行设置。

【例】 数据显示形式。

在命令窗输入：

```
>> x = [ 2.2 1e-13]
```

运行结果：

```
x =  
2.2000 0.0000
```

继续在命令窗输入：

```
>> format long
```

```
>> x
```

```
x =  
2.2000000000000000 0.0000000000000100
```

【例】 改变进位制显示形式。

在命令窗输入：

```
>> format hex
```

```
>> x = uint8(15)
```

运行结果：

```
x =  
0f
```

3.2 逻辑类型

逻辑类型(logical)的变量有两种取值，即逻辑真或逻辑假，分别用“1”和“0”来表示。用特定的 MATLAB 函数或者运算符可以判断某个逻辑条件是否为满足，输入的逻辑数据可以是标量，也可以是数组。

3.2.1 创建逻辑数组

通过输入“true”或“false”就可以直接创建一个逻辑数组，也可以通过对数组进行逻辑运算来创建，还可以通过 MATLAB 函数来产生。

【例】 直接创建逻辑数组。

在命令窗输入：

```
>> x = [true, true, false]
```

运行结果：

```
x =  
1    1    0
```

【例】 利用逻辑运算创建逻辑数组。

在命令窗输入：

```
>> y = [2, 3, 5] < 4
```

运行结果：

```
y =  
1    1    0
```

【例】 利用函数创建逻辑数组。

在命令窗输入：

```
>> z = isfinite([2, 4.3, inf])
```

运行结果：

```
z =  
1    1    0
```

运算结果为逻辑值的函数如表 3-4 所示。

表 3-4 运算产生逻辑结果的函数

函 数	操 作
true、false	设值为真或假
logical	数值类型转化为 logical 类型
& (and)、 (or)、~(not)、xor、any、all	逻辑运算
&&、	与、或
== (eq)、~= (ne)、< (lt)、> (gt)、<= (le)、>= (ge)	关系运算
is*、cellfun	测试运算
strcmp、strncmp、strcmpi、strncmpi	字符串比较

3.2.2 逻辑数组的用途

逻辑数组主要用在条件语句和数组索引中。数组的逻辑索引在许多地方有着特殊的用途。

【例】 数组的逻辑索引。

在命令窗输入：

```
>> A = rand(3), A(A < 0.5) = 0
```

运行结果：

```
A =
    0.0971    0.3171    0.4387
    0.8235    0.9502    0.3816
    0.6948    0.0344    0.7655

A =
     0         0         0
    0.8235    0.9502         0
    0.6948         0    0.7655
```

3.2.3 判断逻辑类型

表 3-5 给出了用于判断 x 是否为逻辑数组的命令。

表 3-5 判断 x 是否为逻辑数组的命令

命 令	操 作
whos x	显示尺寸和类型
islogical(x)	判断是否为 logical 类型
isa(x, 'logical')	判断是否为 logical 类型
class(x)	显示数组类型
cellfun('islogical', x)	判断每个单元是否为 logical 类型

3.3 字 符 串

MATLAB 中的字符串由 Unicode 编码的字符构成。每个字符都有一个数值与之对应，

字符串实际上是由字符的数字代码组成的向量。

3.3.1 创建字符数组

创建字符数据最简单的方法就是在单引号内输入字符,利用 `strcat` 函数也可以将多个字符串串接到一起。

【例】 创建字符数据。

在命令窗输入:

```
>> char1 = 'G.Bush';
```

```
>> whos char1
```

运行结果:

Name	Size	Bytes	Class	Attributes
char1	1x6	12	char	

【例】 串接字符串。

在命令窗输入:

```
>> a = 'G';
```

```
>> b = 'Bush';
```

```
>> char2 = strcat(a, b)
```

运行结果:

```
char2 =
```

```
G.Bush
```

如果要创建二维字符数组,则要保证每行有相同的长度。如果长度不同,则需插入空格以达到要求。下面给出的例子是两个长度为 13 的字符串组成的数组。

【例】 创建字符串数组。

在命令窗输入:

```
>> names = ['Thomas R. Lee'; 'Sr. Developer']
```

运行结果:

```
names =
```

```
Thomas R. Lee
```

```
Sr. Developer
```

3.3.2 字符串单元数组

由于创建字符数组要求字符串有相同的长度,在许多情况下会给编程带来很多不便。为此, MATLAB 提供了一种便捷的方法,即通过创建字符串单元数组来存储长度可变的字符串。利用 `cellstr` 函数可将已有的字符数组转化为字符串单元数组。

【例】 字符数组转化字符串单元数组。

在命令窗输入:

```
>> data = ['Allison Jones'; 'Development'; 'Phoenix'];
```

```
>> celldata = cellstr(data)
```

运行结果:

```
celldata =
    'Allison Jones'
    'Development'
    'Phoenix'
```

利用 char 函数可以将字符串单元数组转化为字符数组, MATLAB 会在长度不够的字符串末尾插入空格。

【例】 字符串单元数组转化字符数组。

继续在命令窗输入:

```
>> strings = char(celldata)
```

运行结果:

```
strings =
Allison Jones
Development
Phoenix
```

表 3-6、表 3-7 给出了有关字符串单元数组元素和集合的函数。

表 3-6 字符串单元数组的元素操作函数

函 数	描 述
cellstr	将字符数组转化为字符串单元数组
char	将字符串单元数组转化为字符数组
deblank	移除字符串尾部的空格
iscellstr	判断是否为字符串单元数组
sort	元素排序
strcat	串接字符串
strcmp	比较字符串
strmatch	查找匹配字符串

表 3-7 字符串单元数组的集合操作函数

函 数	描 述
intersect	求两个向量的交集
ismember	判断是否为序列成员
setdiff	求两个向量的差集
setxor	求两个向量的异或
union	求两个向量的并集
unique	找出向量中的元素(不重复)

3.3.3 字符串的操作

1. 字符串的比较

通过 strcmp、strcmpi、strncmp、strncmpi 等字符串比较函数可以实现字符串的比较。参加比较的可以是字符数组, 也可以是字符串单元数组。这些函数不仅可以比较整个字符

串，还可以比较字符串的一部分。

`strcmp` 函数可以判断两个字符串是否相等，`strncmp` 函数可以判断两个字符串的前 n 个字符是否相等；`strcmpi`、`strncmpi` 函数的功能基本与前两者相同，只是后两者不区分字符大小写。

【例】 字符串比较。

在命令窗输入：

```
>> str1 = 'hello';  
>> str2 = 'help';  
>> strcmp(str1,str2)
```

运行结果：

```
ans =  
0
```

继续在命令窗输入：

```
>> strncmp(str1, str2, 3)
```

运行结果：

```
ans =  
1
```

运行结果表明两个字符不等，但它们的前 3 个字符相等。

字符串比较函数还可以对字符串单元数组进行比较。

【例】 字符串单元数组比较。

在命令窗输入：

```
>> A = {'pizza'; 'chips'; 'candy'};  
>> B = {'pizza'; 'chocolate'; 'pretzels'};  
>> strncmp(A,B,1)
```

运行结果：

```
ans =  
1  
1  
0
```

字符串的比较还可以通过 `>`、`>=`、`<`、`<=`、`==`、`~=` 等关系逻辑运算符来实现。

【例】 用运算符比较字符串。

在命令窗输入：

```
>> A = 'fat';  
>> B = 'fit';  
>> A == B
```

运行结果：

```
ans =  
1    0    1
```

利用 `isletter`、`isspace`、`isstrprop` 等函数可以对字符串中的字符进行分类。`isletter` 函数用

于判断某个字符是否为字母；`isspace` 函数用于判断某个字符是否为空白；`isstrprop` 函数用于判断某个字符是否为指定字符类型。

【例】 判断字符类型。

在命令窗输入：

```
>> str = 'Dec 29';
```

```
>> isletter(str)
```

运行结果：

```
ans =  
      1      1      1      0      0      0
```

2. 字符串的查找和替换

MATLAB 提供了一些能够对字符串中的字符进行查找和替换的函数，如 `strrep`、`findstr`、`strtok`、`strmatch` 等函数。其中，`strrep` 函数用于替换字符串中的某个字符；`findstr` 函数用于查找字符串中的一段字符；`strtok` 函数用于查找字符串中第一个定界字符的前一个字符；`strmatch` 函数用于确定字符数组或字符串单元数组的行字符中与给定字符匹配的行索引。

【例】 替换字符。

在命令窗输入：

```
>> label = 'gOOdbey';
```

```
>> newlabel = strrep(label, 'O', 'o')
```

运行结果：

```
newlabel =  
goodbye
```

【例】 查找字符。

继续在命令窗输入：

```
>> position = findstr('O', label)
```

运行结果：

```
position =  
      2      3
```

【例】 查找匹配字符。

在命令窗输入：

```
>> str = strvcat('max', 'minimax', 'maximum', 'min'), strmatch('max', str)
```

运行结果：

```
str =  
max  
minimax  
maximum  
min  
ans =  
      1  
      3
```

3.3.4 字符串类型与数值类型之间的转化

1. 将数值类型转化为字符串类型

表 3-8 给出了将数值数据转化为字符串的函数。

表 3-8 数值类型转化为字符串类型的函数

函数	描 述	实 例
char	将正整数转化为对应字符	[72 105] → 'Hi'
int2str	将整数转化为字符类型	[72 105] → '72 105'
num2str	将数值类型转化为指定精度和形式的字符类型	[72 105] → '72/105'
mat2str	将数值类型转化为指定精度的可估值字符类型	[72 105] → '[72 105]'
dec2hex	将正整数转化为十六进制表示的字符	[72 105] → '48 69'
dec2bin	将正整数转化为二进制表示的字符	[72 105] → '10010001101001'
dec2base	将正整数转化为任意进制(2~36)表示的字符	[72 105] → '110 151'

2. 将字符串类型转化为数值类型

表 3-9 给出了将字符串转化为数值数据的函数。

表 3-9 字符串类型转化为数值类型的函数

函数	描 述	实 例
uint*	将字符转换为对应的整数代码	'Hi' → 72 105
str2num	将字符类型转化为整数类型	'72 105' → [72 105]
str2double	将字符类型转化为 Double 类型	{'72' '105'} → [72 105]
hex2num	将十六进制字符串转化为 Double 精度的数	'A' → '-1.4917e-154'
hex2dec	将十六进制字符串转化为十进制整数	'A' → 10
bin2dec	将二进制字符串转化为十进制整数	'1010' → 10
base2dec	将任意进制(2~36)字符串转化为十进制整数	'12' → 10

3.4 日期与时间

MATLAB 中的日期与时间有三种表现形式：日期字符串、日期值序列与日期向量。利用转化函数可以实现任意形式之间的转化。

3.4.1 日期的表现形式

表示日期与时间信息的字符串可有许多不同的形式，例如 2007 年 10 月 31 日下午 3 点 45 分 17 秒可表示为以下几种形式：

31-Oct-2007 15:45:17

10/31/07

15:45:17

03:45:17 PM

通过 date 函数可以得到当前日期的字符串形式。

MATLAB 还用日期值序列来表示当前日期时间。以日期值序列 1 表示 0000 年 1 月 1 日 0:00, 其他日期时间在此基础上按天数累加, 不足 1 天的部分用小数来表示。例如, 2007 年 7 月 6 日中午 12 点整表示为 733229.5。通过 now 函数可以得到当前日期时间的数值序列形式。

日期向量是以向量形式来表示当前日期时间的, 其元素为[year month day hour minute second]。通过 clock 函数可以得到当前日期时间的向量形式。

3.4.2 日期表现形式之间的转化

各种日期形式的转化函数如表 3-10 所示, 其调用格式为 date*(x), 其中 x 为输入参量, 可以是三种形式中的任意一种。

表 3-10 日期形式的转化函数

函 数	描 述
datenum	转化为数值序列形式
datestr	转化为字符串形式
datevec	转化为向量形式

datenum 函数在日期计算方面有很重要的作用, 它可以将任意形式的日期字符串(如常用的 'dd-mmm-yyyy', 'mm/dd/yyyy' 或 'dd-mmm-yyyy, hh:mm:ss.ss')转化为数值序列形式。

datestr 函数可以将日期转化为 19 种日期字符串的输出形式, 可显示日期、时间或者两者的结合, 调用格式为 datestr(D,dateform)。“D”为输入日期变量; dateform 的取值为 1~19, 分别代表 19 种字符串形式。

3.4.3 当前日期与时间

当前的日期字符串可以通过 date 函数获得。

【例】 获取当前日期字符串。

在命令窗输入:

```
>> date
```

运行结果:

```
ans =  
06-Jul-2007
```

当前的日期时间数值序列可以通过 now 函数获得。

【例】 获取当前日期时间数值序列。

在命令窗输入:

```
>> now
```

运行结果:

```
ans =  
7.3323e+005
```

3.5 结 构

结构是根据字段组合起来的不同类型的数据集合，结构数组的字段可以包含任意类型的数据。例如，一个字段可以作为用于表示学生姓名的文本字符串，另外的字段可以作为表示课程号及相关成绩的数值数据。

3.5.1 创建结构数组

创建结构数组的方法有两种：一种是通过赋值语句来创建；另一种是利用结构函数来创建。

1. 利用赋值语句创建结构

直接对不同的字段进行赋值，就可以创建一个简单的 1×1 的结构数组。可以在结构名后加下标对创建好的 1×1 的结构进行扩展。

【例】 利用创建赋值语句创建结构。

在命令窗输入：

```
>> student.name = 'John Doe';  
>> student.course = [10105 10083];  
>> student.score = [77 89]
```

运行结果：

```
student =  
    name: 'John Doe'  
  course: [10105 10083]  
    score: [77 89]
```

【例】 利用创建赋值语句进行扩展。

继续在命令窗输入：

```
>> student(2).name = 'Ann Lane';  
>> student(2).course = [10105 10083];  
>> student(2).score = [74 68]
```

运行结果：

```
student =  
1x2 struct array with fields:  
    name  
  course  
    score
```

2. 利用结构函数创建结构

利用 struct 函数创建结构数组，其调用格式为

```
strArray = struct('field1',val1,'field2',val2, ...)
```

其中，自变量为字段名及其对应值。字段值可以是单一值或单元数组，但是必须保证它们具有相同的尺寸。

利用函数 `struct` 创建一个字段名为 `temp` 和 `rainfall` 的 `1×3` 结构数组 `weather` 的不同方法如表 3-11 所示。

表 3-11 创建结构数组的不同方法

方 法	语 法	初始状态
<code>struct</code>	<code>weather(3) = struct('temp', 72, 'rainfall', 0.0);</code>	<code>weather(3)</code> 的字段初值如命令中所示， <code>weather(1)</code> 与 <code>weather(2)</code> 的初值为空
<code>struct</code> 函数与 <code>repmat</code> 函数	<code>weather=repmat(struct('temp', 72,'rainfall', 0.0),1, 3);</code>	所有结构字段初值均为命令中的值
<code>struct</code> 函数与单元数组	<code>weather = struct('temp', {68, 80, 72}, 'rainfall', {0.2, ..., 0.4, 0.0});</code>	各个结构字段初值与命令中单元数组的值一一对应

3.5.2 结构数组的操作

1. 访问结构数组中的数据

利用结构数组索引可以对结构数组的字段值或字段元素值进行访问和赋值。另外，还可以利用逗号分隔的序列来访问。

【例】 访问结构数组的子阵。

在命令窗输入：

```
>> patient.name = 'John Doe';
>> patient.billing = 127.00;
>> patient.test = [79 75 73; 180 178 177.5; 220 210 205];
>> patient(2).name = 'Ann Lane';
>> patient(2).billing = 28.50;
>> patient(2).test = [68 70 68; 118 118 119; 172 170 169];
>> mypatients = patient(1:2)
```

运行结果：

```
mypatients =
1x2 struct array with fields:
    name
    billing
    test
```

【例】 访问某个结构的字段或字段元素。

继续在命令窗输入：

```
>> str = patient(2).name, test2b = patient(2).test(2,2)
```

运行结果：

```
str =
```

```
Ann Lane
```

```
test2b =
```

```
118
```

利用类似的访问方法可以对结构的字段元素进行赋值，例如语句 `patient(2).test(2,2) = 7` 会将 `patient(2)` 结构 `test` 字段的第 2 行第 2 列元素 118 修改为 7。

2. 添加和删除字段

利用赋值语句对一个结构所要添加的字段进行赋值，就可以添加一个字段，这与创建结构时添加字段的方法基本相同。另外，还可以利用 `setfield` 函数添加字段或修改字段的值。

【例】 添加字段。

在命令窗输入：

```
>> mystr(1,1).name = 'alice';  
>> mystr(1,1).ID = 0;  
>> mystr(2,1).name = 'gertrude';  
>> mystr(2,1).ID = 1;  
>> mystr = setfield(mystr, {3,1}, 'order', 3)
```

运行结果：

```
mystr =  
3x1 struct array with fields:  
    name  
    ID  
    order
```

3.6 单元数组

单元数组可以将各种不同类型或尺寸的数据存储到同一个数组当中。例如，可以将一个 1×20 的字符类型数组、一个 3×11 的 `double` 类型数组和一个 1×1 的 `uint32` 类型数组存储到同一个单元数组中。

访问单元数组数据的方法与矩阵索引方法基本相同，不同的是在单元数组索引时，需要用大括号 `{ }` 将下标置于其中。

3.6.1 创建单元数组

单元数组的创建方法与矩阵的创建方法基本相同，不同的是创建矩阵需要使用中括号 `[]`，而创建单元数组需要使用大括号 `{ }`。

【例】 创建单元数组。

在命令窗输入：

```
>> A = {[1 4 3; 0 5 8; 7 2 9], 'Anne Smith'; 3+7i, -pi:pi/4:pi}
```

运行结果：

A =

```
[3x3 double]    'Anne Smith'
[3.0000 + 7.0000i] [1x9 double]
```

也可以通过单元数组的元素一一赋值进行创建, 在命令窗输入:

```
>> B(1,1) = {[1 4 3; 0 5 8; 7 2 9]};
>> B(1,2) = {'Anne Smith'};
>> B(2,2) = {-pi:pi/4:pi};
>> B(2,1) = {3+7i}
```

运行结果:

B =

```
[3x3 double]    'Anne Smith'
[3.0000 + 7.0000i] [1x9 double]
```

利用大括号{}可将几个单元数组合并为一个单元数组。

【例】合并单元数组。

继续在命令窗输入:

```
>> {A B}
```

运行结果:

ans =

```
{2x2 cell}    {2x2 cell}
```

利用中括号[]可将几个单元数组的内容合并为一个数组。

【例】合并单元数组的内容。

继续在命令窗输入:

```
>> [A B]
```

运行结果:

ans =

```
[3x3 double]    'Anne Smith'    [3x3 double]    'Anne Smith'
[3.0000 + 7.0000i] [1x9 double] [3.0000 + 7.0000i] [1x9 double]
```

3.6.2 单元数组的操作

1. 单元数组的删除

与矩阵的删除方法类似, 对单元数组向量下标赋空值就可以删除单元数组的行或列。本小节将以上一小节创建的单元数组为例进行删除操作。

【例】删除单元数组的行。

继续在命令窗输入:

```
>> A(1,:)=[]
```

运行结果:

A =

```
[3.0000 + 7.0000i] [1x9 double]
```

2. 运用函数或运算符操作单元数组

利用索引可以对单元数组进行函数或运算符操作。

【例】 单元求和。

在命令窗输入：

```
>> A{1,1} = [1 2; 3 4];  
>> A{1,2} = randn(3, 3);  
>> A{1,3} = 1:5;  
>> B = sum(A{1,1})
```

运行结果：

```
B =  
4      6
```

3. 单元数组的显示

直接在命令窗输入单元数组名，就可以显示单元数组的构成单元。如果存在数组单元，那么这种方法只能显示数组的尺寸。如果需要显示更详细的内容，可以利用 `celldisp` 函数来实现。

【例】 显示单元数组。

继续在命令窗输入：

```
>> celldisp(A)
```

运行结果：

```
A{1} =  
1      2  
3      4  
  
A{2} =  
-0.0956  -1.3362  -0.6918  
-0.8323   0.7143   0.8580  
0.2944   1.6236   1.2540  
  
A{3} =  
1      2      3      4      5
```

3.7 函数句柄

函数句柄是提供间接调用函数方法的 MATLAB 值或数据类型。用户可以根据需要在调用其他函数时传递函数句柄，也可以将函数句柄储存在数据结构中以便之后使用。

3.7.1 创建和调用函数句柄

在已有函数名前加符号@，就可以创建一个函数句柄。例如，存在一个已有函数，名为 `functionname`，创建一个句柄 `fhandle`：

```
fhandle = @functionname
```

此外，还可以通过匿名函数创建一个函数句柄，例如创建一个计算输入变量 x 的平方的函数：

```
sqr = @(x) x.^2;
```

3.7.2 利用句柄调用函数

如果要执行一个与函数句柄相关的函数，使用如下语法格式将函数句柄 `fhandle` 看做为一个函数名：

```
fhandle(arg1, arg2, ..., arg n)
```

如果被调用的为无输入参数的函数句柄，则使用如下语句：

```
fhandle()
```

3.8 MATLAB 类

所有 MATLAB 数据类型都以面向对象的类执行。通过创建附加的类，可以在 MATLAB 的开发环境中添加自定义的数据类型。这些用户自定义的类定义了新数据类型的结构、M 文件函数或方法，即由用户为每个类编写，用于定义该数据类型的行为。

这些方法也可以定义将多种不同 MATLAB 运算符，包括算术运算符、下标引用以及串接等应用于新数据类型的途径。例如，多项式类可以重新定义加法运算符(+)，以使它能够正确地完成多项式的加法运算。

通过利用 MATLAB 的类可以实现：

- 创建方法来推翻 MATLAB 现存的功能性；
- 限制一个类的对象中已经允许的运算；
- 在由同一个父类继承的这些相关类中增强公共性；
- 增强用户代码的重复利用率。

3.9 Java 类

MATLAB 提供了一个 Java 程序语言的接口，通过该接口可以从 Java 类中创建一个对象并且在这些对象中调用 Java 方法。Java 类是 MATLAB 的一种数据类型。通过 MATLAB 接口可以获得原有的类和第三方的类，用户也可以为自己的 Java 类创建定义并将其引入 MATLAB。

通过 MATLAB Java 接口可以实现：

- 访问 Java API(应用程序接口)类的数据包，可以支持一些基本活动，如 I/O 和网络；
- 访问第三方 Java 类；
- 在 MATLAB 中方便地建立 Java 类；
- 用 MATLAB 或 Java 语法调用 Java 对象的方法；
- 在 MATLAB 变量和 Java 对象中传递数据。

第 4 章 数学运算基础

MATLAB 提供了丰富的数学运算和数据分析的函数,本章将详细介绍这些函数的相关概念及其用法。4.1 节将介绍矩阵与线性代数的运算函数,包括矩阵分析、求解线性方程组、求逆矩阵与行列式、矩阵的分解、矩阵的非线性运算以及特征值和特征向量。4.2 节将介绍多项式与插值的函数。4.3 节将介绍 MATLAB 的离散傅里叶变换算法。4.4 节将介绍函数的函数。4.5 节将介绍常微分方程的初值问题、边值问题以及偏微分方程的求解。4.6 节将介绍稀疏矩阵的创建、查看及运算函数。

4.1 矩阵与线性代数

矩阵对于线性代数有着重要的作用,它是由实数或复数构成的二维数组。线性代数包括矩阵运算、线性方程组、特征值、奇异值和矩阵分解。

4.1.1 矩阵分析

MATLAB 提供的矩阵分析函数如表 4-1 所示。

表 4-1 矩阵分析函数

函 数	描 述
norm	向量或矩阵的范数
normest	估计矩阵的 2-范数
rank	矩阵的秩
det	行列式
trace	矩阵的对角元素之和
null	化零矩阵
orth	正交化
rref	矩阵的约化行阶梯形
subspace	两个子空间的夹角

1. 向量和矩阵的范数

对于一个 n 维向量,可以用一个数 $\|x\|$ 来度量该 n 维向量的大小。如果 $\|x\|$ 满足以下三个性质,则称 $\|x\|$ 为向量 x 的范数。

- 非负性：对于一切 x 都有

$$\|x\| > 0, \text{ 且 } \|x\| = 0 \text{ 的充要条件是 } x=0 \quad (4-1)$$

- 正齐性：对任何实数 α 和向量 x ，有

$$\|\alpha x\| = |\alpha| \|x\| \quad (4-2)$$

- 三角不等式：对任何向量 x 和 y ，有

$$\|x+y\| \leq \|x\| + \|y\| \quad (4-3)$$

显然，满足以上三个条件的函数有很多种，如果定义：

$$\varphi_1(x) = |x_1| + |x_2| + \cdots + |x_n| \quad (4-4)$$

$$\varphi_2(x) = (|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2)^{1/2} \quad (4-5)$$

$$\varphi_\infty(x) = \max_{1 \leq i \leq n} \{|x_i|\} \quad (4-6)$$

由于式(4-4)、(4-5)和(4-6)都满足范数的三个性质，因此它们都是 n 维向量 x 的范数。将 $\varphi_1(x)$ 、 $\varphi_2(x)$ 和 $\varphi_\infty(x)$ 分别称为向量 x 的 1-范数、2-范数和 ∞ -范数，并分别用记号 $\|x\|_1$ 、 $\|x\|_2$ 和 $\|x\|_\infty$ 来表示。另外，记号 $\|x\|$ 泛指向量的范数。

对于一个矩阵 A ，相应的矩阵范数 $\|A\|$ 应当对一切 n 阶矩阵 A 和一切 n 维向量 x 满足

$$\|Ax\| \leq \|A\| \|x\| \quad (4-7)$$

满足式(4-7)的矩阵范数称为与向量范数相容的范数。如果以 $\|A\|_1$ 、 $\|A\|_2$ 和 $\|A\|_\infty$ 分别表示与向量的 1-范数、2-范数和 ∞ -范数相容的矩阵范数，那么对于 n 阶矩阵 $A=(a_{ij})$ ，有

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}| \quad (4-8)$$

$$\|A\|_2 = (A^T A \text{ 的最大特征值})^{1/2} \quad (4-9)$$

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}| \quad (4-10)$$

MATLAB 中，常用 `norm` 函数计算向量或矩阵的 1-范数、2-范数和 ∞ -范数，其调用格式为 `norm(x,p)` 或 `norm(A,p)`，作用是计算向量 x 或矩阵 A 的 p -范数。其中， p 为 1、2 或 `inf`。命令 `norm(x)` 或 `norm(A)` 相当于命令 `norm(x,2)` 或 `norm(A,2)`，而 `normest(A)` 实现矩阵 A 的 2-范数的快速估算。

【例】 求向量的 1-范数、2-范数和 ∞ -范数。

在命令窗输入：

```
>> x = [1 2 3];
>> [norm(x,1),norm(x,2),norm(x,inf)]
```

运行结果：

```
ans =
    6.0000    3.7417    3.0000
```

【例】 求矩阵的 1-范数、2-范数和 ∞ -范数。

在命令窗输入：

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

>> [norm(A,1), norm(A,2), norm(A,inf)]
```

运行结果：

```
ans =
    18.0000    16.8481    24.0000
```

2. 矩阵的秩

对于一个矩阵 A，如果 $A=0$ ，则 A 的秩为零；如果 $A \neq 0$ ，则称 A 中非零子式的最高阶数为 A 的秩。MATLAB 中常用命令 `rank(A)` 来以默认允许误差计算矩阵 A 的秩，而用命令 `rank(A,tol)` 来以给定容许误差 tol 计算矩阵的秩。

【例】 求矩阵的秩。

在命令窗输入：

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

>> rank(A)
```

运行结果：

```
ans =
     2
```

3. 矩阵的行列式

当矩阵 A 为方阵时，可以利用命令 `det(A)` 来计算 A 的行列式。MATLAB 中行列式的计算是通过高斯消去法先实现矩阵的 LU 分解，然后再计算下三角矩阵 L 和上三角矩阵 U 的行列式之积(对角线元素之积)，即 A 的行列式。

【例】 求矩阵的行列式。

在命令窗输入：

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

>> det(A)
```

运行结果：

```
ans =
   -39
```

4. 矩阵的迹

矩阵的迹定义为主对角线的元素之和。无论矩阵是否为方阵，MATLAB 中都可以用命令 `trace(A)` 来计算矩阵 A 的迹。

【例】 求矩阵的迹。

在命令窗输入：

```
>> A = [1    3    2
        6    5    4];

>> trace(A)
```

运行结果：

```
ans =

    6
```

5. 化零矩阵

矩阵 A 的化零矩阵 Z 满足 $A*Z$ 的元素近似为零，并且 $Z'*Z=I$ 。MATLAB 中可以用命令 $Z = \text{null}(A)$ 求矩阵 A 的化零矩阵，用命令 $Z = \text{null}(A, 'r')$ 求矩阵 A 的有理形式的化零矩阵，有理形式的化零矩阵 $Z'*Z \neq I$ 。

【例】 求矩阵的化零矩阵。

在命令窗输入：

```
>> A = [1    2    3
        3    2    1
        1    2    3];

>> Z = null(A);

>> A*Z
```

运行结果：

```
ans =

    1.0e-015 *
         0
    -0.6661
         0
```

【例】 求矩阵的有理形式的化零矩阵。

在命令窗输入：

```
>> A = [1    2    3
        3    2    1
        1    2    3];

>> z = null(A, 'r');

>> A*z
```

运行结果：

```
ans =

    0
    0
    0
```

6. 正交化

矩阵 A 的标准正交基 B 的列向量构成的线性空间与矩阵 A 的列向量构成的线性空间相

同, 并且标准正交基 B 满足 $B'B = \text{eye}(\text{rank}(A))$ 。MATLAB 提供了命令 `orth(A)` 来计算矩阵 A 的标准正交基。

【例】 求矩阵的标准正交基。

在命令窗输入:

```
>> A = [1    2    3
        3    2    1
        1    2    3];
>> B = orth(A)
```

运行结果:

```
B =
   -0.6008    0.3730
   -0.5274   -0.8496
   -0.6008    0.3730
```

继续在命令窗输入:

```
>> B'*B
```

运行结果:

```
ans =
    1.0000    0.0000
    0.0000    1.0000
```

7. 矩阵的约化行阶梯形

MATLAB 提供了通过使用不完全主元高斯-约当消去法计算矩阵 A 的约化行阶梯形 R 的命令 `R = rref(A)`。其默认允许误差为 $\max(\text{size}(A)) * \text{eps} * \text{norm}(A, \text{inf})$ 。

【例】 求矩阵的约化行阶梯形。

在命令窗输入:

```
>> A = [1    2    3
        3    2    1
        1    2    3];
>> R = rref(A)
```

运行结果:

```
R =
    1    0   -1
    0    1    2
    0    0    0
```

8. 两个子空间的夹角

MATLAB 提供了计算矩阵 A 和 B 的列子空间夹角的命令 `subspace(A,B)`。如果子空间夹角很小, 则表明两个子空间几乎是线性相关的。

【例】 求矩阵的子空间夹角。

在命令窗输入:

```
>> A = [1    2    3
        3    2    1
        1    2    3];
>> B = [1    2    5
        3    2    2
        1    2    4];
>> subspace(A,B)
运行结果:
ans =
    1.5708
```

4.1.2 求解线性方程组

通常遇到的线性方程组都具有如 $Ax = b$ 的形式。本小节将对恰定系统、超定系统、欠定系统的情况分别进行讨论。

线性代数中, 求解形如 $Ax = b$ 的线性方程组需要求出所有可能存在的解, 即通解。求线性方程组通解分为三个步骤:

(1) 利用命令 `null(A)` 求相应齐次系统 $Ax = 0$ 的通解, 得到方程组 $Ax = 0$ 的基础解向量, 基础解向量的任意线性组合均为方程组 $Ax = b$ 的解。

(2) 寻找非齐次系统 $Ax = b$ 的特解。

(3) 将步骤(1)与步骤(2)的结果相加, 即为非齐次系统 $Ax = b$ 的通解。

下面将分别讨论恰定系统、超定系统、欠定系统特解的求法。

1. 恰定系统

恰定系统 $Ax = b$ 的矩阵 A 为方阵, 但是 A 可能是奇异矩阵或非奇异矩阵。当 A 为非奇异矩阵时, 可直接利用命令 `x=A\b` 或 `x=inv(A)*b` 来求系统的特解。

【例】 求非奇异矩阵恰定系统的特解。

在命令窗输入:

```
>> A = [1    2    4
        3    2    1
        1    2    3];
>> b = [1; 2; 3];
>> x = A\b
```

运行结果:

```
x =
   -2.5000
    5.7500
   -2.0000
```

当 A 为奇异矩阵时, $Ax = b$ 的特解不存在或者不唯一。这种情况下用求非奇异矩阵恰定系统特解的命令不再适用。对于这种系统, 可以先通过命令 `rref([A b])` 将增广矩阵 $[A \ b]$

约化为行阶梯形式并查看结果。如果 A 的行秩小于增广矩阵的行秩，则方程组无解；反之，可以利用命令 $x = \text{pinv}(A)*b$ 来求解。

【例】 求奇异矩阵恰定系统的特解。

在命令窗输入：

```
>> A = [ 1      3      7
        -1      4      4
          1     10     18 ];
>> b = [5; 2; 12];
>> rref([A b]), x = pinv(A)*b
```

运行结果：

```
ans =
    1.0000         0    2.2857    2.0000
         0    1.0000    1.5714    1.0000
         0         0         0         0

x =
    0.3850
   -0.1103
    0.7066
```

这种情况下，A 的行秩等于增广矩阵的行秩，求得的解即精确解。如果 A 不变， $b = [5; 2; 11]$ ，在命令窗输入：

```
>> rref([A b])
```

运行结果：

```
ans =
    1.0000         0    2.2857         0
         0    1.0000    1.5714         0
         0         0         0    1.0000
```

显然，A 的行秩小于增广矩阵的行秩，第三行表示方程“ $0=1$ ”，因而无解。

2. 超定系统

超定系统的未知量个数少于方程组的个数。超定系统常用于实验数据的曲线拟合问题。

【例】 采集一个频率为 1、直流分量为 0 的正弦信号，横轴(时间轴)与纵轴上的点分别为

```
t = 0:0.32:6.4
y = [-0.4794   -0.1790    0.1395    0.4439    0.7033    0.8912    0.9887
      0.9857    0.8827    0.6901    0.4274    0.1213   -0.1971   -0.4955
     -0.7436   -0.9162   -0.9957   -0.9742   -0.8538   -0.6467   -0.3739]'
```

那么基函数应该选取 $\sin(t)$ 和 $\cos(t)$ ，问题等价于求解满足 $a \sin(t) + b \cos(t) = y$ 的 a 和 b。在 M 文件中输入：

```
t = [0:0.32:6.4]';
```

```

y = [-0.4094    -0.1890    0.1095    0.4239    0.7033    0.8712    0.9687...
      0.9657    0.8627    0.6901    0.4374    0.1213   -0.1871   -0.4755...
      -0.7436   -0.9362   -0.9757   -0.9742   -0.8438   -0.6467   -0.3539];

```

```

A = [sin(t) cos(t)];
ab = A\y;
a = ab(1);
b = ab(2);
Y = a*sin(t)+b*cos(t);
plot(t,Y,'-',t,y,'p');

```

运行结果如图 4-1 所示。

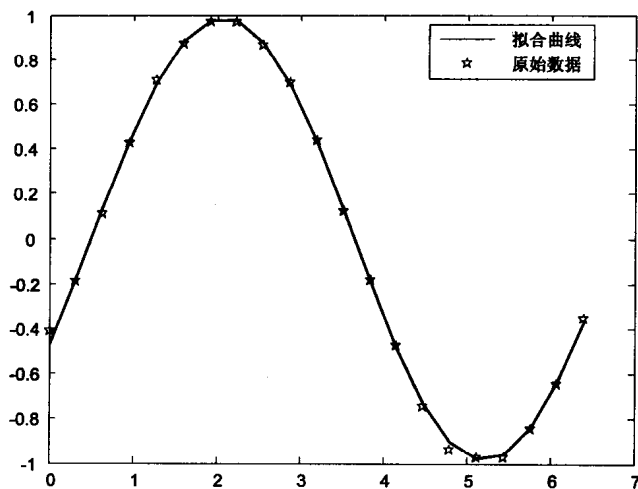


图 4-1 原始数据与拟合曲线

3. 欠定系统

欠定系统的未知量个数多于方程组的个数。如果 A 的行秩不小于其增广矩阵的行秩，其解存在无穷多个。可以用命令 $x=A \setminus b$ 求其特解。

【例】求欠定系统的特解。

在命令窗输入：

```

>> A = [6 8 7 3
        6 8 7 4];
>> b = [2; 2];
>> x = R\b

```

运行结果：

```

x =
    0
 0.2500
    0
    0

```

4.1.3 逆矩阵与伪逆矩阵

对于一个非奇异方阵 A ，其行列式不为零，因而存在逆矩阵。使用 MATLAB 命令 `inv(A)` 可以求出非奇异方阵 A 的逆矩阵。如果 A 的行列式等于零，或者 A 不是方阵，那么 A 的逆矩阵将不存在。这时，只有通过计算 A 的伪逆矩阵来满足某些特殊的需求，通过命令 `B=pinv(A)` 可以求得 A 的伪逆矩阵 B ，并且 A 与 B 满足 $ABA=A$ 、 $BAB=B$ 。

4.1.4 矩阵的分解

MATLAB 中的线性方程组求解都是基于以下三种矩阵分解，对应函数分别为 `chol`、`lu` 以及 `qr`：

- Cholesky 分解：分解为对称正定上下三角矩阵。
- LU 分解：分解为下三角矩阵和上三角矩阵。
- QR 分解：分解为正交矩阵和非奇异上三角矩阵。

1. Cholesky 分解

实际应用中，有一类常见且极重要的矩阵，即对称正定矩阵。如果对于任何 n 维非零向量 x ，对称矩阵 A 都满足 $x^T A x > 0$ ，那么 A 是对称正定矩阵。可以证明，对称正定矩阵的各阶顺序主子行列式都大于零，并且存在单位上三角矩阵 G ，使得 $A=G^T G$ 。这种分解称为 Cholesky 分解。

MATLAB 提供的 `chol` 函数可以直接实现对称正定矩阵的 Cholesky 分解，调用格式为 `G=chol(A)`。

【例】 Cholesky 分解。

在命令窗输入：

```
>> A = pascal(3), G = chol(A)
```

运行结果：

A =

```
1    1    1
1    2    3
1    3    6
```

G =

```
1    1    1
0    1    2
0    0    1
```

2. LU 分解

MATLAB 中的许多矩阵运算都以 LU 分解为基础，如方阵的求逆操作、求行列式等。LU 分解将方阵 A 分解为一个下三角矩阵 L 和一个上三角矩阵 U 的乘积，常用的调用格式有两种：

- `[L,U]=lu(A)`：得到一个上三角矩阵 U 和一个准下三角矩阵 L ，使得 $A=LU$ 。准下三角矩阵 L 实际上是下三角矩阵的置换形式。

• $[L,U,P]=lu(A)$: 得到一个单位下三角矩阵 L 、上三角矩阵 U 和一个由 0、1 组成的置换矩阵 P , 使得 $PA=LU$ 。

【例】 LU 分解。

在命令窗输入:

```
>> A = [10 20 30
        20 45 80
        30 80 171];

>> [L1 U1] = lu(A), [L2 U2 P2] = lu(A)
```

运行结果:

```
L1 =
    0.3333    0.8000    1.0000
    0.6667    1.0000         0
    1.0000         0         0

U1 =
   30.0000   80.0000  171.0000
         0   -8.3333  -34.0000
         0         0    0.2000

L2 =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.8000    1.0000

U2 =
   30.0000   80.0000  171.0000
         0   -8.3333  -34.0000
         0         0    0.2000

P2 =
     0     0     1
     0     1     0
     1     0     0
```

LU 分解可用于求矩阵 A 的逆和行列式:

$$\det(A) = \det(L) \cdot \det(U)$$

$$\text{inv}(A) = \text{inv}(U) \cdot \text{inv}(L)$$

3. QR 分解

矩阵的正交分解是将矩阵分解为一个正交矩阵 Q 和一个上三角矩阵 R 的乘积。此种分解适用于任意矩阵, 是非常重要的分解形式。其调用格式如下:

• $[Q,R]=qr(A)$: 生成一个与 A 同阶的上三角矩阵 R 和一个正交矩阵 Q , 使得 $A=QR$ 。

• $[Q,R,P]=qr(A)$: 得到一个置换矩阵 P 、上三角矩阵 R 和正交矩阵 Q , 使得 $AP=QR$, 选择置换矩阵 P 使得 $\text{abs}(\text{diag}(R))$ 按降序排列。

• $[Q,R]=qr(A,0)$: 生成一种“经济”的分解。如果矩阵 A 是 $m \times n$ 阶, 并且 $m > n$, 则只计算 Q 的前 n 列。

• $[Q,R,P]=qr(A,0)$: 生成一种“经济”的分解。其中 P 是一个置换向量, 使得 $Q \cdot R = A(:,P)$, 选择置换列向量使得 $\text{abs}(\text{diag}(R))$ 按降序排列。

【例】 QR 分解。

在命令窗输入:

```
>> A = [1  1  1
        2  2  2
        3  3  3];
```

```
>> [Q,R] = qr(A)
```

运行结果:

```
Q =
   -0.2673    0.9562    0.1195
   -0.5345   -0.0439   -0.8440
   -0.8018   -0.2895    0.5228

R =
   -3.7417   -3.7417   -3.7417
         0     0.0000    0.0000
         0         0   -0.0000
```

4.1.5 矩阵的非线性运算

MATLAB 支持矩阵的几种非线性运算, 分别为求矩阵的正整数次幂、负幂、分数次幂、矩阵元素的幂以及指数矩阵。

1. 矩阵的正整数幂

利用命令 A^p 可以求矩阵 A 的 p 次幂, 其中 A 为方阵, p 为正整数。

【例】 矩阵的正整数次幂。

在命令窗输入:

```
>> A = [1 1 1; 1 2 3; 1 3 6];
>> X = A^2
```

运行结果:

```
X =
     3     6    10
     6    14    25
    10    25    46
```

2. 矩阵负幂、分数次幂

如果方阵 A 为非奇异矩阵, 则 A 的逆存在, 可以用 $A^{(-p)}$ 求 A 的逆矩阵 A^{-1} 的 p 次幂。当 p 为分数时, 则可以利用 A^p 求 A 的分数次幂。

【例】 矩阵的负幂和分数次幂。

继续在命令窗输入：

```
>> Y = A^(-2)
```

运行结果：

```
Y =  
    19.0000   -26.0000    10.0000  
   -26.0000    38.0000   -15.0000  
    10.0000   -15.0000     6.0000
```

再输入：

```
>> Z = A^(1/2)
```

运行结果：

```
Z =  
    0.8775    0.4387    0.1937  
    0.4387    1.0099    0.8874  
    0.1937    0.8874    2.2749
```

3. 矩阵元素的幂

利用.^运算符可以求由矩阵每个元素的幂组成的矩阵。

【例】 矩阵元素的幂。

继续在命令窗输入：

```
>> A.^2
```

运行结果：

```
ans =  
     1     1     1  
     1     4     9  
     1     9    36
```

4. 指数矩阵

对于一个线性系统，其状态方程通常可以写作：

$$\frac{dx}{dt} = Ax$$

其中， $x=x(t)$ 为关于 t 的函数向量， A 为与 t 无关的矩阵，则系统的解可以表示为

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

【例】 非时变线性系统的解。

在命令窗输入：

```
>> A = [0   -5   -1  
        5    1   -9  
       -4    8   -6];  
>> x0 = [1; 0; 1];
```

```

>> X = [];
for t = 0:0.01:1
    X = [X expm(t*A)*x0];
end
>> plot3(X(1,:),X(2,:),X(3,:),'-')

```

运行结果如图 4-2 所示。

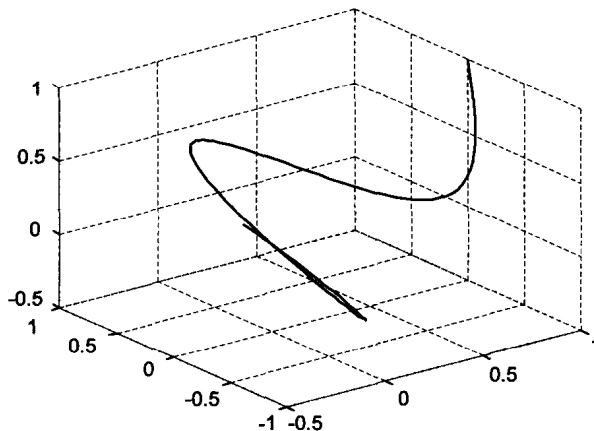


图 4-2 非时变线性系统的解曲线

4.1.6 特征值与特征向量

对于一个方阵 A ，它的特征值 λ 和特征向量 x 满足 $Ax = \lambda x$ 。如果将所有特征值 λ 组成对角矩阵 Λ ，将所有特征向量 x 组成矩阵 V ，则有 $AV = V\Lambda$ 。根据 V 是否奇异，可将 A 化为对角阵或约当阵。

1. 化为对角阵

当 V 为非奇异矩阵时， A 的每个特征值都对应一个特征向量，可以写成 $A = V\Lambda V^{-1}$ 的形式。MATLAB 中，通常使用命令 $[V,D] = \text{eig}(A)$ 求出矩阵 A 的所有特征值组成的对角矩阵 D 和 A 的所有特征向量组成的矩阵 V 。

【例】化对角阵。

在命令窗输入：

```

>> A = [ 6   11   16
        -9  -10  -13
         5    7   12];

```

```

>> [V,D] = eig(A)

```

运行结果：

```

V =
   -0.3897 - 0.3871i   -0.3897 + 0.3871i    0.5564
         0.7760         0.7760        -0.6338
   -0.2997 + 0.0793i   -0.2997 - 0.0793i    0.5373

```

```
D =
    -0.4598 + 3.1608i      0      0
         0    -0.4598 - 3.1608i      0
         0         0      8.9196
```

可以验证, $V \cdot D \cdot \text{inv}(V)$ 的值与 A 的值相等。

2. 化为约当阵

当 V 为奇异矩阵时, A 的独立特征向量数小于 A 的阶数, 因而不可以写成 $A = V \Lambda V^{-1}$ 的形式对其进行对角化, 但是可以将 A 化为与对角型相似的约当型。MATLAB 中, 通常使用命令 $[X, J] = \text{jordan}(A)$ 求出矩阵 A 的约当矩阵 J 和广义特征向量组 X , 使得 $A = XJX^{-1}$ 。

【例】 化对角阵。

在命令窗输入:

```
>> A = [ 6   12   19
        -9  -20  -33
         4    9   15];
```

```
>> [X, J] = jordan(A)
```

运行结果:

```
X =
   -1.7500    1.5000    2.7500
    3.0000   -3.0000   -3.0000
   -1.2500    1.5000    1.2500
```

```
J =
   -1     0     0
    0     1     1
    0     0     1
```

可以验证, $X \cdot J \cdot \text{inv}(X)$ 的值与 A 的值相等。

3. Schur 分解

MATLAB 中还提供了无需特征值分解的 Schur 分解, 即将 A 分解为正交矩阵 U 和分块上三角矩阵 S (由 1×1 、 2×2 块组成), 且满足 $A = USU^T$ 。调用格式为 $[U, S] = \text{schur}(A)$ 。

【例】 Schur 分解。

在命令窗输入:

```
>> A = [ 6   12   19
        -9  -20  -33
         4    9   15];
```

```
>> [U, S] = schur(A)
```

运行结果:

```
U =
   -0.4741    0.6648    0.5774
```

```

    0.8127    0.0782    0.5774
   -0.3386   -0.7430    0.5774
S =
   -1.0000   20.7846  -44.6948
         0    1.0000   -0.6096
         0         0    1.0000

```

可以验证, $U \cdot S \cdot U'$ 的值与 A 的值相等。

4.1.7 奇异值分解

奇异值分解在矩阵分析中占有极为重要的位置。对于一个矩阵 A , 它的一个奇异值 σ 和对应的奇异向量 u 、 v 满足 $Av = \sigma u$, $A^T u = \sigma v$ 。如果将所有奇异值组成对角矩阵 S , 将奇异列向量 u 、 v 组成正交矩阵 U 、 V , 则 $AV = US$, $A^T U = VS$, $A = USV^T$ 。MATLAB 中提供的奇异值分解函数 `svd` 的调用格式如下:

- $[U, S, V] = \text{svd}(A)$: 得到一个与 A 同阶的奇异对角矩阵 S , 两个酉矩阵 U 和 V 。若 A 为 $m \times n$ 阵, 则 U 为 $m \times m$ 阵, V 为 $n \times n$ 阵。 S 中奇异值非负且按降序排列。

- $[U, S, V] = \text{svd}(A, 0)$: 得到一种“经济”的分解, 只计算出矩阵 U 的前 n 列, 矩阵 S 的大小为 $n \times n$ 。

【例】 奇异值分解。

在命令窗输入:

```

>> A = [ 9    4
         6    8
         2    7];
>> [U, S, V] = svd(A)

```

运行结果:

```

U =
   -0.6105    0.7174    0.3355
   -0.6646   -0.2336   -0.7098
   -0.4308   -0.6563    0.6194
S =
   14.9359         0
         0    5.1883
         0         0
V =
   -0.6925    0.7214
   -0.7214   -0.6925

```

继续在命令窗输入:

```

>> [U, S, V] = svd(A, 0)

```

运行结果:

```

U =
    -0.6105    0.7174
    -0.6646   -0.2336
    -0.4308   -0.6563

S =
    14.9359         0
         0     5.1883

V =
    -0.6925    0.7214
    -0.7214   -0.6925

```

可以验证, $U \cdot S \cdot V'$ 的值与 A 的值相等。

4.2 多项式与插值

本节将介绍有关多项式的运算函数、多项式拟合、部分分式的展开, 还将介绍在信号或图像处理中常用的插值方法。

4.2.1 多项式

MATLAB 提供了丰富的多项式运算函数, 包括计算多项式的根、矩阵特征多项式的系数、估计多项式的值、计算多项式的乘除法、多项式的求导、多项式拟合以及多项式之比与部分分式展开式之间的相互转化。表 4-2 给出了有关多项式运算的函数。

表 4-2 多项式运算函数

函 数	描 述
conv	多项式乘法
deconv	多项式除法
poly	由根求多项式
polyder	多项式求导
polyfit	多项式曲线拟合
polyval	多项式估值
polyvalm	矩阵多项式估值
residue	部分分式展开
roots	求多项式的根

1. 多项式的表示方法

MATLAB 中的多项式可表示为按降幂排列的多项式系数组成的行向量。例如, 多项式 $p(x)=x^3-2x+3$ 可表示为如下向量形式:

```
p = [1 0 -2 3];
```

2. 多项式的根

利用 roots 函数可以计算多项式的根。例如求方程 $x^3-1=0$ 的根。

【例】 求多项式的根。

在命令窗输入：

```
>> p = [1 0 0 -1];
```

```
>> r = roots(p)
```

运行结果：

```
r =  
-0.5000 + 0.8660i  
-0.5000 - 0.8660i  
1.0000
```

由多项式的根可以反推出多项式，仍以方程 $x^3 - 1 = 0$ 的根为例。

【例】 由多项式的根反推多项式。

继续在命令窗输入：

```
>> poly(r)
```

运行结果：

```
ans =  
1.0000 0.0000 0.0000 -1.0000
```

3. 特征多项式

利用 `poly` 函数也可以计算矩阵特征多项式的系数，且系数按降幂排列。

【例】 求矩阵的特征多项式。

在命令窗输入：

```
>> A = [ 1 3 -1  
        5 2 6  
        9 0 1];
```

```
>> poly(A)
```

运行结果：

```
ans =  
1.0000 -4.0000 -1.0000 -167.0000
```

4. 多项式估值

对于一个多项式，可以利用 `polyval` 函数估计指定自变量时的函数值。例如估计多项式 $f(x) = x^2 - 2$ 在 $x = 3$ 处的值。

【例】 多项式的单点估值。

在命令窗输入：

```
>> f = [1 0 -2];
```

```
>> polyval(f,3)
```

运行结果：

```
ans =  
7
```


用户也可以在自变量 x 取多个值时对多项式进行估值, 此时 `polyval` 函数的第二个输入参量可以是向量或矩阵。

【例】 多项式的矩阵估值。

继续在命令窗输入:

```
>> A = [ 1  2
        3  4];
>> polyval(f,A)
```

运行结果:

```
ans =
    -1     2
     7    14
```

但是很多情况下需要估计自变量为矩阵的多项式的值, 如 $f(X)=X^2-2I$ 的估值。这时需要用到矩阵多项式估值函数 `polyvalm`。

【例】 自变量为矩阵的多项式估值。

继续在命令窗输入:

```
>> X = [ 1  2
        3  4];
>> polyvalm(f,X)
```

运行结果:

```
ans =
     5    10
    15    20
```

5. 多项式的乘除法

利用卷积函数 `conv` 和反卷积函数 `deconv` 可以分别实现多项式的乘法和除法。例如对多项式 $a(x) = x^2 + 2x + 3$ 和多项式 $b(x) = 3x^2 + 2x + 1$ 进行乘法和除法运算。

【例】 多项式的乘法。

在命令窗输入:

```
>> a = [1 2 3];
>> b = [3 2 1];
>> c = conv(a,b)
```

运行结果:

```
c =
     3     8    14     8     3
```

【例】 多项式的除法。

继续在命令窗输入:

```
>> [q,r] = deconv(c,a)
```

运行结果:

```
q =  
    3    2    1  
  
r =  
    0    0    0    0    0
```

其中, q 代表商式的系数, r 代表余式的系数。

6. 多项式求导

利用 `polyder` 函数可以对多项式求导, 还可以对两个多项式相乘或相除之后得到的多项式求导。

【例】 多项式求导。

在命令窗输入:

```
>> p = [2 3 7];  
>> q = polyder(p)
```

运行结果:

```
q =  
    4    3
```

当 `polyder` 函数有两个输入参量, 一个输出参量时, 实现输入多项式相乘后求导。

【例】 多项式相乘后求导。

在命令窗输入:

```
>> a = [1 3 5];  
>> b = [2 4 6];  
>> c = polyder(a,b)
```

运行结果:

```
c =  
    8    30    56    38
```

当 `polyder` 函数有两个输入参量, 两个输出参量时, 实现输入多项式相除后求导。第一个输入参量为被除数, 第二个输入参量为除数; 第一个输出参量为分子, 第二个输出参量为分母。

【例】 多项式相除后求导。

继续在命令窗输入:

```
>> [q,d] = polyder(a,b)
```

运行结果:

```
q =  
   -2   -8   -2  
  
d =  
    4   16   40   48   36
```

7. 多项式拟合

利用 `polyfit` 函数可以寻找最小二乘法拟合的多项式系数。调用格式为 `p = polyfit(x,y,n)`。

其中, x 、 y 为需要拟合的原始数据向量, n 为多项式的次数。

【例】多项式拟合。

在命令窗输入:

```
>> x = [1 2 3 4 5];
>> y = [5.5 44 123.2 291 467];
>> p = polyfit(x,y,4);
>> x2 = 1:.1:5;
>> y2 = polyval(p,x2);
>> plot(x,y,'*',x2,y2)
```

运行结果如图 4-3 所示。

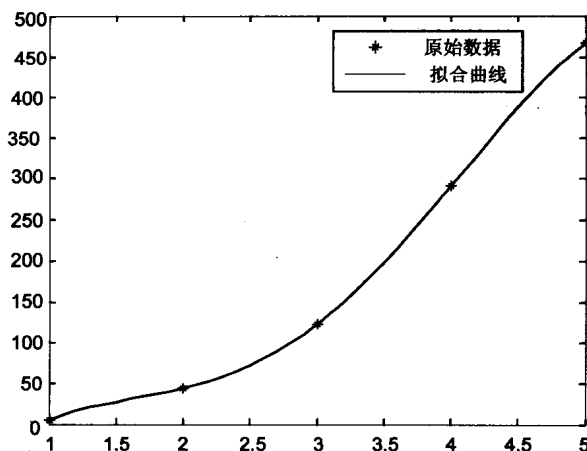


图 4-3 原始数据与拟合曲线

8. 多项式的部分分式展开

`residue` 函数用于实现两个多项式之比与部分分式展开式之间的相互转化。对于多项式 $b(s)$ 和 $a(s)$, 有

$$\frac{b(s)}{a(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \cdots + \frac{r_n}{s-p_n} + k_s$$

命令 `[r,p,k] = residue(b,a)` 将多项式之比转化为部分分式展开式; 命令 `[b,a] = residue(r,p,k)` 将部分分式展开式转化为多项式之比。其中, r 为由 r_n 组成的向量, p 为由 p_n 组成的向量, k 为 k_s , b 为 $b(s)$, a 为 $a(s)$ 。下面以传递函数

$$\frac{2s-1}{s^2+2s-3} = \frac{1.75}{s+3} + \frac{0.25}{s-1}$$

为例进行转换。

【例】多项式之比转化为部分分式展开式。

在命令窗输入:

```
>> b = [2 -1];
>> a = [1 2 -3];
```

```
>> [r,p,k] = residue(b,a)
```

运行结果:

```
r =
    1.7500
    0.2500

p =
   -3.0000
    1.0000

k =
    []
```

【例】 部分分式展开式转化为多项式之比。

继续在命令窗输入:

```
>> [b1,a1] = residue(r,p,k)
```

运行结果:

```
b1 =
     2     -1

a1 =
    1.0000    2.0000   -3.0000
```

4.2.2 插值

插值是利用已知数据的规律, 计算未测的中间点值或预测数据所在范围以外的值的过程。MATLAB 提供了丰富的插值方法, 使用户可以在光滑性、执行速度与占用存储空间之间进行选择。

MATLAB 的 polyfun 路径下的插值函数如表 4-3 所示。

表 4-3 polyfun 路径下的插值函数

函 数	描 述
griddata	数据格栅化与表面拟合
griddata3	三维数据格栅化与超曲面拟合
griddatan	数据格栅化与超曲面拟合(维数>3)
interp1	一维插值
interp2	二维插值
interp3	三维插值
interpft	FFT 法一维插值
interpnn	N 维插值
mkpp	构造分段光滑多项式
pchip	分段光滑三次 Hermite 插值多项式
ppval	分段光滑多项式估值
spline	三次样条插值
unmkpp	分段光滑多项式的详细信息

1. 一维插值

MATLAB 中的一维插值包括多项式插值和基于 FFT 的插值。函数 `interp1` 常用于实现一维多项式插值，这在数据分析和曲线拟合中非常有用。该函数由已知数据 (x_i, y_i) 构造一个多项式 $P(x)$ ，使得在 x_i 点处的函数值 $P(x_i)$ 等于 y_i ，其余 x 点处的值由 $P(x)$ 决定。

函数 `interp1` 的调用格式为 `y = interp1(xi, yi, x, method)`。其中， x_i 、 y_i 为已知数据点组成的向量； x 为插值点组成的向量； y 为插值点向量处的函数值向量；`method` 为可选字符串，用于指定插值方法：

- `method = 'nearest'`: 最邻近插值。该方法将插值点 x 处的函数值 y 置为与之最接近的已知点 x_i 处的值，即 y_i 。

- `method = 'linear'`: 线性插值。该方法用直线连接相邻的已知数据点 (x_i, y_i) ，插值点 x 处的函数值 y 由该直线确定。

- `method = 'spline'`: 三次样条插值。用三次函数连接相邻的已知数据点 (x_i, y_i) ，插值函数在已知点处连续，并且它的一阶导数和二阶导数也连续。

- `method = 'pchip'` OR `'cubic'`: 三次插值。这两种方法等价，用于实现分段光滑 Hermite 插值，并保留数据的单调性和形状。

选择使用插值方法时，还应该考虑到各种方法的光滑性、执行速度与占用存储空间：

- 最邻近插值：速度最快，但是光滑性最差。

- 线性插值：比最邻近插值占用的内存更多，执行速度稍慢，插值结果是连续的，但是端点处斜率不连续。

- 三次样条插值：最长的相对运行时间，比三次插值占用的内存要少。产生最光滑的曲线。但是如果输入数据不均匀或者间距太小，则会得到不希望得到的结果。

- 三次插值：比最邻近插值法或线性插值法需要占用更多的内存和执行时间，但是得到的函数及其一阶导数都是连续的。

如果插值点 x 超出了已知点 x_i 的取值范围，则默认情况下返回函数值为 NaN。下面的例子将对以上几种插值方法进行比较：

【例】 多项式插值。

创建一个 M 文件，输入如下代码：

```
% Original Data
x = 1:10;
y = [ 0.0701   -0.0689    0.0505   -0.0326    0.0196 ...
      -0.0113    0.0062   -0.0033    0.0018   -0.0009];

% Interpolation
xi = 1:0.1:10;
yi1 = interp1(x,y,xi,'nearest');
yi2 = interp1(x,y,xi,'linear');
yi3 = interp1(x,y,xi,'spline');
yi4 = interp1(x,y,xi,'pchip');
figure;
subplot(221);
```

```

plot(x,y,'*',xi,yi1);
title('最邻近插值');
subplot(222);
plot(x,y,'*',xi,yi2);
title('线性插值');
subplot(223);
plot(x,y,'*',xi,yi3);
title('三次样条插值');
subplot(224);
plot(x,y,'*',xi,yi4);
title('三次插值');

```

程序运行结果如图 4-4 所示。

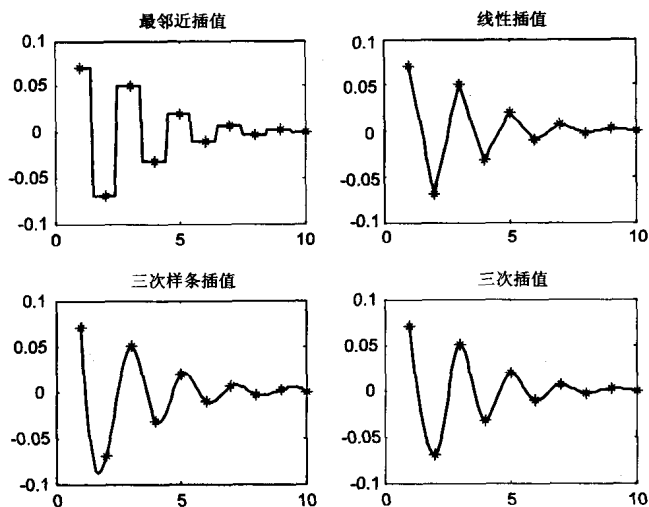


图 4-4 各种插值方法的对比

通过函数 `interpft` 可以实现基于 FFT 的一维插值。该方法首先计算周期函数值向量的傅里叶变换，再计算更多点数的傅里叶反变换，调用格式为 `y = interpft(x,n)`。其中，`x` 为等间隔周期函数采样值，`n` 为需要返回的等间隔点数。

【例】多项式插值。

在命令窗输入：

```

>> y = sin(0:9);
>> N = length(y);
>> L = 5;
>> M = N*L;
>> x = 0:L:L*N-1;
>> xi = 0:M-1;
>> yi = interpft(y,M);
>> plot(x,y,'o',xi,yi);

```

```
>> legend('原始数据','插值函数')
```

运行结果如图 4-5 所示。

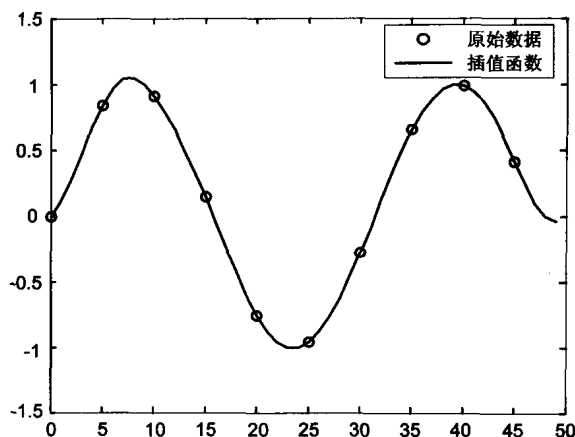


图 4-5 原始数据与插值函数

2. 二维插值

二维插值在图像处理和数据可视化中非常重要，可以利用函数 `interp2` 来实现。常用调用格式为 `ZI = interp2(X,Y,Z,XI,YI,method)`，其中，`X`、`Y` 为已知数据点，`Z` 为已知点处的函数值，`XI`、`YI` 为插值点，`ZI` 为求得的插值点的函数值。`method` 为可选字符串，用于指定插值方法：

- `method = 'nearest'`：最邻近插值。该方法通过已知数据生成分段光滑的常数曲面，插值点处的函数值即最邻近点处的值。

- `method = 'linear'`：双线性插值。该方法由已知点生成一个双线性曲面，插值点处的函数值由最近的 4 个已知点确定。

- `method = 'cubic'`：双三次插值。该方法由已知点生成一个双三次曲面，插值点处的函数值由最近的 16 个已知点确定。这种方法产生的曲面最光滑，因而在图像处理中有重要的应用。在要求插值函数及其导数连续时，这种方法非常必要。

下面的例子将对以上几种二维插值方法进行比较。

【例】 二维插值。

创建一个 M 文件，输入如下代码：

```
z = peaks(x,y);
[xi,yi] = meshgrid(-3:0.25:3);
zi1 = interp2(x,y,z,xi,yi,'nearest');
zi2 = interp2(x,y,z,xi,yi,'bilinear');
zi3 = interp2(x,y,z,xi,yi,'bicubic');
figure;
subplot(221);
surf(x,y,z);
title('原始数据');
```

```
subplot(222);
surf(xi,yi,zi1);
title('最邻近插值');
subplot(223);
surf(xi,yi,zi2);
title('双线性插值');
subplot(224);
surf(xi,yi,zi3);
title('双三次插值');
figure;
subplot(221);
contour(x,y,z);
title('原始数据');
subplot(222);
contour(xi,yi,zi1);
title('最邻近插值');
subplot(223);
contour(xi,yi,zi2);
title('双线性插值');
subplot(224);
contour(xi,yi,zi3);
title('双三次插值');
```

运行这段程序，得到插值曲面图和等高线图如图 4-6 和图 4-7 所示。

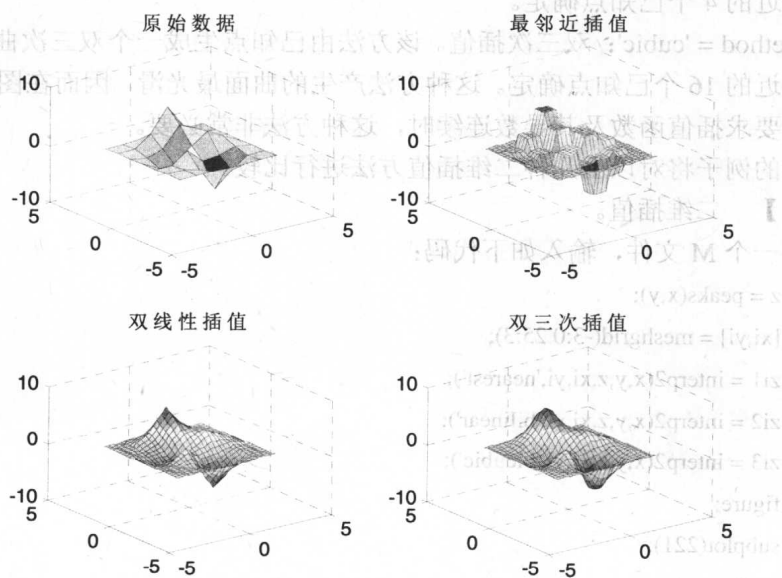


图 4-6 插值曲面图

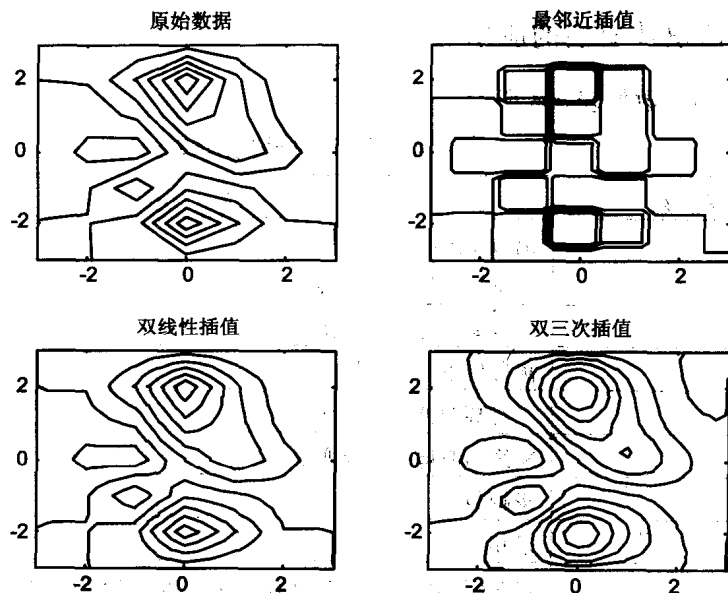


图 4-7 等高线图

4.3 快速傅里叶变换

快速傅里叶变换(FFT)是计算序列的离散傅里叶变换(DFT)的有效算法,在信号和图像处理中有着很重要的作用。利用FFT可以进行卷积、滤波、频谱分析以及功率谱估计。本节主要介绍快速傅里叶变换的概念及其应用。

4.3.1 快速傅里叶变换的概念

序列 $x(n)$ 的离散傅里叶变换(DFT)及离散傅里叶反变换(IDFT)定义如下式所示:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk}, \quad 0 \leq k \leq N-1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}nk}, \quad 0 \leq n \leq N-1$$

通常FFT算法采取的是按时间抽取(DIT)或按频率抽取(DIF)基2算法,即把一个 $N(N=2^m)$ 点序列DFT分裂为2个 $N/2$ 点序列的DFT,再把这2个 $N/2$ 的DFT分裂为4个 $N/4$ 点序列的DFT,如此不断分裂(共分裂成 $\lg N$ 级),直至分为 $N/2$ 个2点序列的DFT。不难看出, N 点序列的DFT计算量为复乘 N^2 次,复加 $N(N-1)$ 次;而基2FFT的计算量为复乘 $N \lg N$ 次,复加 $(N-1) \lg N$ 次,大大节省了运算量。

4.3.2 快速傅里叶变换的应用

有关 FFT 的函数如表 4-4 所示。

表 4-4 FFT 的相关函数

函 数	描 述
fft	离散傅里叶变换
fft2	二维离散傅里叶变换
fftn	N 维离散傅里叶变换
ifft	离散傅里叶反变换
ifft2	二维离散傅里叶反变换
ifftn	N 维离散傅里叶反变换
abs	求幅值
angle	求相角
unwrap	以 2π 为周期移动序列相位实现相角的平滑跳变
fftshift	将零频率部分移至谱中央
cplxpair	将复数按复共轭对排序
nextpow2	求以 2 为底的对数并向正无穷舍入

通常用 `fft(x)` 或 `fft(x,n)` 命令来求序列的 DFT。若序列 x 的长度为 N , `fft(x)` 则实现计算 x 的 N 点 DFT 的功能; 而 `fft(x,n)` 则实现计算 x 的 n 点 DFT, $n > N$ 时对序列补零, $n < N$ 时对序列进行截断。建议使用 2^m 点序列的 DFT, 这样能够实现 FFT 快速算法。

利用 `fft` 函数分析序列的频率分量非常有效。下面给出一个计算周期正弦信号谐波的实例, 设 N 点序列为 $x = 2 \sin\left(\frac{2\pi}{N}n\right) + 0.5 \sin\left(3 \cdot \frac{2\pi}{N}n\right)$ 。

【例】 周期信号的频谱分析。

创建一个 M 文件, 输入如下代码:

```
% Sine Wave
N=50;
n = 0:N-1;
w1 = 2*pi/N;
w2 = 3*2*pi/N;
x = 2*sin(w1*n) + 0.5*sin(w2*n);
plot(n,x);
xlabel('n');
ylabel('x');

% Frequency Spectrum
H = abs(fft(x,N))/N*2;
figure;
```

```
stem(0:N/2-1,H(1:N/2),'markersize',1);  
xlabel('k');  
ylabel('Magnitude');
```

运行这段程序，得到序列 x 的波形图和幅度频率图分别如图 4-8 和图 4-9 所示。

可以看出，直流分量($k=0$)以及其他频率分量的幅值为 0，基波($k=1$)幅值为 2，3 次谐波($k=3$)幅值为 0.5，符合构成序列 x 的各频率分量幅值。

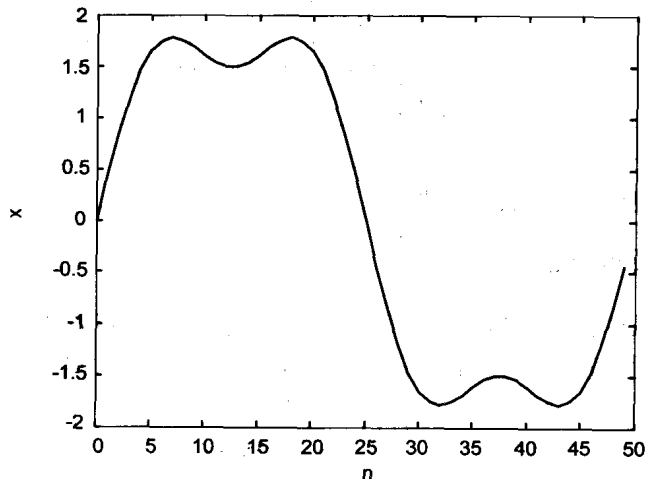


图 4-8 x 的波形图

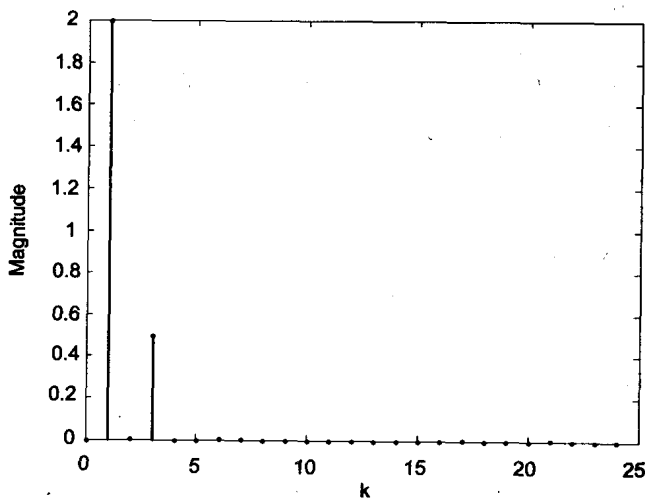


图 4-9 x 的幅度频率图

4.4 函数的函数

函数的函数是指一个函数以其他函数作为输入参量。例如函数的画图函数 `fplot`，其调用格式为 `fplot(@fun, limits)`。其中，`@fun` 是需要绘制的函数句柄，`limits` 为指定的绘制范围。表 4-5 给出了本节所要讨论的函数。

表 4-5 函数的函数

函 数	描 述
fplot	绘图函数
fminbnd	指定区间一元函数的最小值
fminsearch	多元函数的最小值
fzero	找出一元函数的零点
quad	数值估计积分, 自适应 Simpson 积分
quadl	数值估计积分, 自适应 Lobatto 积分
quadv	向量化积分
dblquad	二重积分
triplequad	三重积分

4.4.1 函数的表示方法

MATLAB 的数学函数可以有两种表示方法, 一种方法是将表达式保存在 M 文件内作为 MATLAB 函数, 另一种方法是直接作为匿名函数而表示。例如对于函数

$$f(x) = \frac{2x - 3}{x^2 - 5x - 6}$$

创建一个名为 ff 的 M 文件, 输入如下代码并保存为 ff.m:

```
function y = ff(x)
y = (2*x-3)/(x.^2 - 5*x - 6);
```

这样, 可以创建一个 ff 函数的句柄 fh, 可以按照直接调用函数 ff 的方法调用句柄 fh。

【例】 调用 M 文件函数的句柄。

在命令窗输入:

```
>> fh = @ff;
>> y1 = ff(3), y2 = fh(3)
```

运行结果:

```
y1 =
    -0.2500
y2 =
    -0.2500
```

表达数学函数的另一种方法是创建匿名函数, 仍以前面的函数为例进行说明。

【例】 创建匿名函数。

在命令窗输入:

```
>> fh = @(x)(2*x-3)/(x.^2 - 5*x - 6);
>> y = fh(3)
```

运行结果:

```
y =
    -0.2500
```

MATLAB 同样支持多元函数的匿名函数。

【例】 创建多元匿名函数。

在命令窗输入：

```
>> fh = @(x,y) x.^2 + 2*y.^2;
```

```
>> y = fh(3,1)
```

运行结果：

```
y =
```

```
11
```

利用 fplot 函数可以绘制已知函数的图像。

【例】 绘制数学函数图像。

在命令窗输入：

```
>> fplot(@(x)2*sin(3*x),[-1 1]);
```

运行结果如图 4-10 所示。

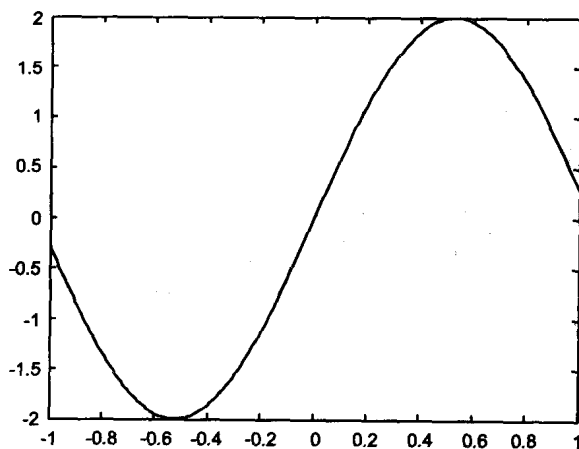


图 4-10 利用 fplot 函数绘图

4.4.2 函数的最小值与零点

函数的最小值与零点问题对于最优化问题的求解非常重要。

1. 一元函数的最小值

利用 fminbnd 函数可以求出一元函数在指定区间内的最小值。fminbnd 函数的常用调用格式为 $x = \text{fminbnd}(\text{fun}, x1, x2)$ ，其中，fun 为一元函数的句柄，x1、x2 分别为指定区间的下限和上限。

【例】 计算一元函数的最小值。

在命令窗输入：

```
>> x = fminbnd(@(x)-sin(x),0,2*pi)
```

运行结果：

```
x =
```

```
1.5708
```

2. 多元函数的最小值

利用 `fminsearch` 函数可以求出多元函数在指定向量附近的最小值。`fminsearch` 函数的常用调用格式为 `x = fminsearch(fun,x0)`，其中，`fun` 为多元函数的句柄，`x0` 为指定初始向量。

【例】 计算多元函数的最小值。

创建一个 M 文件函数，输入如下代码并保存为 `val3.m`：

```
function f = val3(v)
x = v(1);
y = v(2);
z = v(3);
f = abs(x) + y.^2 + z.^4;
```

在命令窗输入：

```
>> x = fminsearch(@val3,[1,1,1])
```

运行结果：

```
x =
    1.0e-004 *
   -0.0000    0.0004   -0.3072
```

3. 一元函数的零点

利用 `fzero` 函数可以求出一元函数在指定值附近或者指定区间内的最小值。`fzero` 函数的常用调用格式为 `x = fzero(fun,x0)`，其中，`fun` 为一元函数句柄，当 `x0` 为标量时，求得与 `x0` 最近的零点；当 `x0` 为二值向量，且 `fun(x0(1))` 与 `fun(x0(2))` 异号时(否则会产生出错信息)，求得该区间内的零点。

【例】 计算一元函数的零点。

在命令窗输入：

```
>> y1 = fzero(@(x)cos(x),0)
```

运行结果：

```
y1 =
   -1.5708
```

或输入：

```
>> y2 = fzero(@(x)cos(x),[0 pi])
```

运行结果：

```
y2 =
    1.5708
```

4.4.3 数值积分

在大多数情况下很难求得一个函数的原函数，或者就没有有限形式的原函数解析表达式，因而计算函数的定积分往往采用数值积分的方法。例如通过 `quad(@humps,0,1)` 命令用 Simpson 法计算函数 `humps` 在 0~1 区间上的积分。

实际中往往遇到参数方程的曲线积分, MATLAB 中通常运用 quad 函数或 quadl 函数计算曲线的长度。例如计算曲线 $x(t)=\sin(t)$, $y(t)=\cos(t)$, $z(t)=t$ 在 $t \in [0, 3\pi]$ 上的长度, 其表达式为

$$\int_0^{3\pi} \sqrt{\cos^2(t) + \sin^2(t) + 1} dt$$

【例】 计算曲线长度。

在命令窗输入:

```
>> leng = quad(@(t)sqrt(cos(t).^2 + sin(t).^2 + 1),0,3*pi)
```

运行结果:

```
leng =  
13.3286
```

利用 dblquad 函数可以计算形如

$$\int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} f(x, y) dx dy$$

的二重积分, dblquad 函数的调用格式为 `result = dblquad(fun2,xmin,xmax,ymin,ymax)`, 其中, fun2 代表二元函数的句柄, xmax、xmin、ymax、ymin 分别代表自变量 x、y 的上下限。

【例】 计算 $f(x, y) = y \sin(x) + x \cos(y)$ 的二重积分。

在命令窗输入:

```
>> result = dblquad(@(x,y)(y*sin(x) + x*cos(y)),pi,2*pi,0,pi)
```

运行结果:

```
result =  
-9.8696
```

4.4.4 嵌套函数与匿名函数

在某些情况下不仅需要调用函数, 而且还需要与该函数相关的参数。例如, 如果要使用 fzero 函数寻找三次多项式 $x^3 + bx + c$ 在不同的系数 b、c 情况下的零点, 则需要一个函数能够在系数 b、c 变化时计算该多项式, 即能够在调用 fzero 函数的同时提供多项式函数中的 b、c 取值。通常的做法有两种: 使用嵌套函数或使用匿名函数。

1. 嵌套函数

以三次多项式 $x^3 + bx + c$ 为例, 对于不同的 b、c 取值, 可以使用嵌套函数实现零点求解。

【例】 利用嵌套函数求解零点。

创建一个 M 文件, 输入如下代码并保存为 findzero.m:

```
function y = findzero(b, c, x0)  
options = optimset('Display', 'off'); % Turn off Display  
y = fzero(@poly, x0, options);
```

```
function y = poly(x) % Compute the polynomial.
```

```
y = x^3 + b*x + c;
```

```
end
```

```
end
```

如果要寻找 $x^3 + x + 1$ 在 $x = 0$ 附近的零点, 则在命令窗输入:

```
>> findzero(1, 1, 0)
```

运行结果:

```
ans =
```

```
-0.6823
```

2. 匿名函数

仍以三次多项式 $x^3 + bx + c$ 为例, 对于不同的 b 、 c 取值, 可以使用匿名函数实现零点求解。

【例】 利用匿名函数求解零点。

创建一个 M 文件, 输入如下代码并保存为 poly.m:

```
function y = poly(x, b, c) % Compute the polynomial.
```

```
y = x^3 + b*x + c;
```

如果要寻找 $x^3 + x + 1$ 在 $x = 0$ 附近的零点, 则在命令窗输入:

```
>> b = 1;
```

```
>> c = 1;
```

```
>> fzero(@(x) poly(x, b, c), 0)
```

运行结果:

```
ans =
```

```
-0.6823
```

4.5 求解微分方程

本节将介绍如何用 MATLAB 函数求解常微分方程初值问题、延迟微分方程初值问题、常微分方程边值问题以及偏微分方程初值边值问题。

4.5.1 常微分方程初值问题

本小节主要介绍常微分方程的初值问题的数值解法。MATLAB 中的 ode 求解器主要针对以下几种类型的常微分方程初值问题:

- 显式常微分方程形如 $y' = f(t, y)$ 。
- 线性隐式常微分方程形如 $M(t, y) \cdot y' = f(t, y)$, $M(t, y)$ 为矩阵。
- 完全隐式常微分方程形如 $f(t, y, y') = 0$ 。

常微分方程的初值问题的 ode 求解器如表 4-6 所示。

表 4-6 ode 求解器

求解器	求解对象	计算方法
ode45	非刚性微分方程	基于显式 Runge-Kutta(4,5)公式
ode23	非刚性微分方程	基于显式 Runge-Kutta(2,3)公式
ode113	非刚性微分方程	变阶 Adams-Bashforth-Moulton PECE 求解器
ode15s	刚性微分方程、微分代数方程	基于数值微分公式(反向微分公式)
ode23s	刚性微分方程	基于 2 阶 Rosenbrock 公式
ode23t	适度刚性微分方程、微分代数方程	梯形法
ode23tb	刚性微分方程	梯形法(2 阶反向微分公式)
ode15i	完全隐式微分方程	反向微分公式

1. 求解显式常微分方程和线性隐式常微分方程

这类问题的求解方法的实质是从给定初值开始, 计算每一个步长内的微分方程的数值积分。对于非刚性问题, 常采用 ode45、ode23、ode113 求解器求解; 而对于刚性问题, 常采用 ode15s、ode23s、ode23t、ode23tb 等求解器求解。这 7 种 ode 求解器的调用格式是相同的, 如果需要改变求解的计算方法, 只需要改变求解器的函数名即可。最简单常用的求解器调用格式为

$[t,y] = \text{solver}(\text{odefun}, \text{tspan}, y_0, \text{options})$

其中, solver 为使用的 ode 求解器函数; odefun 为常微分方程组的函数句柄, 形如 $\text{dydt} = f(t,y)$ 或 $M(t,y) * \text{dydt} = f(t,y)$, t 为时间标量, y 为列向量; tspan 为指定求解区间, $\text{tspan} = [t_0 \text{ tf}]$ 则计算 $[t_0 \text{ tf}]$ 区间的解, $\text{tspan} = [t_0, t_1, t_2, \dots, \text{tf}]$ 则计算每个时间点 $t_0, t_1, t_2, \dots, \text{tf}$ (保证时间点单调) 上的解; y_0 为初值列向量; options 为改变默认积分属性的可选参数。

对于非刚性问题, 常用 ode45 来求解。下面以一个 LC 振荡电路为例, 说明求解非刚性问题时 ode 的用法。

【例】 求解振荡电路(非刚性问题)。

设 $L=1$, $C=1$, 初始电压 $u_0=1$, 初始电流 $i_0=0$ 。电路如图 4-11 所示。

则电压、电流满足如下微分方程:

$$\begin{cases} \frac{di}{dt} = \frac{1}{L} u \\ \frac{du}{dt} = -\frac{1}{C} i \end{cases}$$

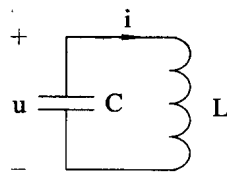


图 4-11 电路连接图

要求解这个电路方程, 创建一个 M 文件, 输入如下代码并保存为 LC.m:

```
function dydt = LC(t,y)
dydt = [ y(2)
        -y(1)];
```

在命令窗输入:

```

>> [t,y] = ode45(@LC,[0 2*pi],[0 ; 1]);
>> plot(t,y(:,1),'b-');
>> hold on;
>> plot(t,y(:,2),'r-');
>> hold off;
>> axis([0 2*pi -1 1]);
>> legend('电流','电压')

```

运行结果如图 4-12 所示。

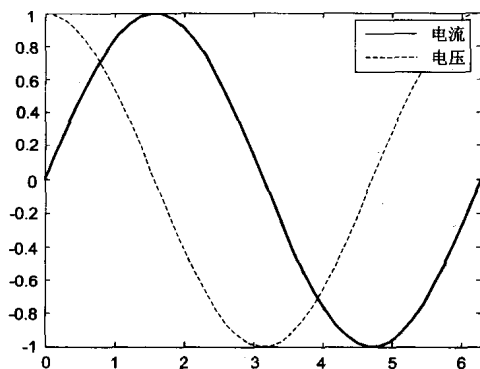


图 4-12 电路的电压和电流波形

对于刚性问题，它的解在很小的时间区间内可能有大的跳变。这时不再适合用 ode45 求解，而通常采用 ode15s。下面以 van der Pol(范德坡)方程为例，说明求解刚性问题时 ode 的用法。刚性 van der Pol 方程的函数名为 vdp1000，方程式为

$$\begin{cases} y_1' = y_2 \\ y_2' = 1000(1 - y_1^2)y_2 - y_1 \end{cases}$$

在命令窗输入：

```

>> [t,y] = ode15s(@vdp1000,[0 3000],[2; 0]);
>> plot(t,y(:,1),'-');

```

运行结果如图 4-13 所示。

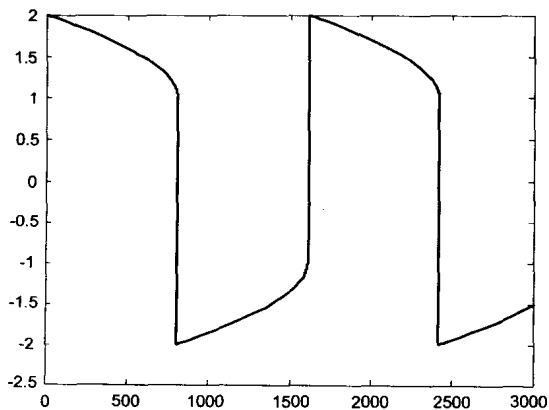


图 4-13 van der Pol 方程的波形结果

2. 求解完全隐式常微分方程

完全隐式常微分方程通常采用 ode15i 求解器来求解。该方法采用变阶反向微分公式，其基本调用格式为

`[t,y] = ode15i(odefun,tspan,y0,yp0,options)`

其中，odefun 为完全隐式常微分方程组的函数句柄，形如 $f(t,y,y')=0$ ， t 为时间标量， y 为列向量；tspan 为指定求解区间， $tspan=[t0\ tf]$ 则计算 $[t0\ tf]$ 区间的解， $tspan=[t0,t1,t2,\dots,tf]$ 则计算每个时间点 $0,t1,t2,\dots,tf$ (保证时间点单调) 上的解； $y0$ 、 $yp0$ 为代初值条件 $y(t_0)$ 、 $y'(t_0)$ 的列向量，并且必须满足 $f(t0,y0,yp0)=0$ ；options 为改变默认积分属性的可选参数。

下面以 Weissinger 方程 $ty^2(y')^3 - y^3(y')^2 + t(t^2+1)y' - t^2y=0$ 为例，说明 ode15i 的用法。该方程解析解为

$$y(t) = \sqrt{t^2 + 0.5}$$

在命令窗输入：

```
>> t0 = 1;
>> y0 = sqrt(3/2);
>> yp0 = 0;
>> [t,y] = ode15i(@weissinger,[1 10],y0,yp0);
>> ytrue = sqrt(t.^2 + 0.5);
>> plot(t,y,t,ytrue,'*');
>> legend('数值解','解析解');
```

运行结果如图 4-14 所示。

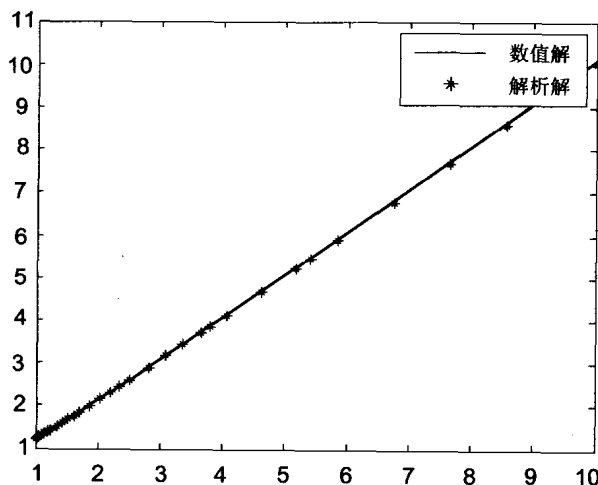


图 4-14 Weissinger 方程的数值解和解析解

4.5.2 延迟微分方程初值问题

MATLAB 提供了两种求解延迟微分方程的求解器，表 4-7 给出了这两种求解器的描述。

表 4-7 延迟微分方程求解器

求解器	描述
dde23	求解常数延迟微分方程初值问题
ddesd	求解一般延迟微分方程初值问题

对于形如 $y'(t) = f(t, y(t), y(t - \tau_1), \dots, y(t - \tau_k))$ 的延迟常微分方程，其中 τ_1, \dots, τ_k 为正常数，实际中常采用 dde23 求解器来求解。调用格式为

`sol = dde23(ddefun,lags,history,tspan,options)`

其中，`ddefun` 为延迟微分方程的函数句柄；`lags` 为常向量，代表 τ_1, \dots, τ_k ；`history` 用来指定 $t < t_0$ 时的 y 值，可以是常向量、关于 t 的函数或者由前一个时刻的解 `sol` 来确定；`tspan` 为指定积分区间的行向量 $[t_0, t_f]$ 且 $t_0 < t_f$ ；`options` 为可选的积分参量。返回值 `sol` 为结构类型，`sol.x` 为时间点向量，`sol.y` 为时间点上对应的解 y ，`sol.y'` 为时间点上对应的解 y' ，`sol.solver` 为求解器名“dde23”。

下面以延迟方程组为例来说明 dde23 的具体用法。

$$\begin{cases} y_1'(t) = y_1(t-1) + y_3(t) \\ y_2'(t) = y_1(t-1) + y_2^2(t-0.2) \\ y_3'(t) = y_3(t) \end{cases}$$

$$\text{history: } \begin{cases} y_1(t) = 1 \\ y_2(t) = 1 \\ y_3(t) = 1 \end{cases}$$

创建一个 M 文件，输入如下代码并保存为 `dde_fun.m`：

```
function dydt = dde_fun(t,y,Z)
    ylag1 = Z(:,1);
    ylag2 = Z(:,2);
    dydt = [ylag1(1)+y(3)
            ylag1(1)+ylag2(2).^2
            y(3)];
```

在命令窗输入：

```
>> lags = [1 0.2];
>> sol = dde23(@dde_fun,lags,ones(3,1),[0,1]);
>> plot(sol.x,sol.y(1,:));
>> hold on;
>> plot(sol.x,sol.y(2,:), 'r');
>> plot(sol.x,sol.y(3,:), 'k-');
>> legend('y1','y2','y3');
```

运行结果如图 4-15 所示。

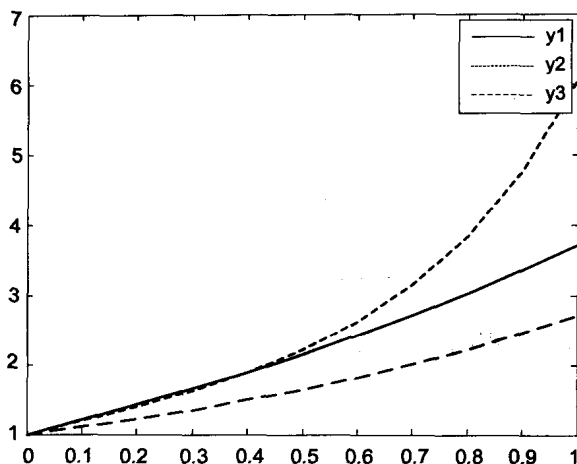


图 4-15 延迟微分方程的求解曲线

4.5.3 常微分方程边值问题

本小节介绍常微分方程边值问题的求解方法，只针对如下形式的常微分方程系统：

$$\begin{cases} y' = f(x, y) \\ g(y(a), y(b)) = 0 \end{cases}$$

即求解区间 $[a, b]$ 内第一类边值问题的 y 值。

与初值问题不同，常微分方程边值问题有可能无解、有无穷多解或有限多解。因此，在求解常微分方程边值问题时，需要对解进行猜测，这个过程是十分必要的。

在求解常微分方程边值问题时，可能会遇到如无限区间或奇异系数等问题，还有可能遇到出现未知参数的问题，如

$$\begin{cases} y' = f(x, y, p) \\ g(y(a), y(b), p) = 0 \end{cases}$$

MATLAB 提供的求解器 `bvp4c` 用于求解常微分方程第一类边值问题，`bvp4c` 的常用调用格式为

```
sol = bvp4c(odefun,bcfun,solinit,options)
```

其中，`odefun` 表示微分方程组的函数句柄，可以是 `dydx = odefun(x,y)`，也可以是 `dydx = odefun(x,y,parameters)`；`bcfun` 为计算边值条件残值的函数句柄，可以是 `res = bcfun(ya,yb)`，也可以是 `res = bcfun(ya,yb,parameters)`；`solinit` 是一个包含对解的初始猜测的结构，有 `x`、`y`、`parameters` 三个字段，满足 `a = solinit.x(1)` 且 `b = solinit.x(end)`，`solinit.y(:,i)` 是结点 `solinit.x(i)` 的解的初始猜测，`parameters` 为未知参数初始猜测的可选向量；`options` 为可选积分参量。

边值问题的解可能有很多个，通常通过初始猜测来获得所期望的解。下面以一个二阶微分方程为例来说明。设该方程及其边值条件为

$$\begin{cases} y'' + |y| = 0 \\ y(0) = 0, y(4) = 1 \end{cases}$$

先将其转化为一阶常微分方程:

$$\begin{cases} y_1' = y_2 \\ y_2' = -|y_1| \\ y_1(0) = 0, y_1(4) = 1 \end{cases}$$

创建一个关于常微分方程的 M 文件, 输入如下代码并保存为 twoode.m:

```
function dydx = twoode(x,y)
    dydx = [      y(2)
             -abs(y(1))];
```

创建一个边值问题的 M 文件, 输入如下代码并保存为 twobc.m:

```
function res = twobc(ya,yb)
    res = [      ya(1)
            yb(1) + 1];
```

如果猜测初始值为 $y_1(x) = 1$, $y_2(x) = 0$, 在命令窗输入:

```
>> solinit = bvpinit(linspace(0,4,5),[1 0]);
>> sol = bvp4c(@twoode,@twobc,solinit);
>> x = linspace(0,4);
>> y = deval(sol,x);
>> plot(x,y(1,:));
```

运行结果如图 4-16 所示。

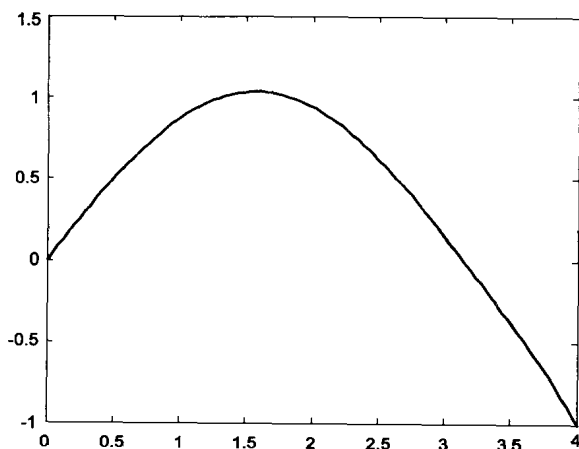


图 4-16 猜测初始值为 $y_1(x)=1$, $y_2(x)=0$ 时的求解曲线

如果猜测初始值为 $y_1(x)=-1$, $y_2(x)=0$, 在命令窗输入:

```
>> solinit = bvpinit(linspace(0,4,5),[-1 0]);
>> sol = bvp4c(@twoode,@twobc,solinit);
>> x = linspace(0,4);
```

```
>> y = deval(sol,x);
>> plot(x,y(1,:));
>> axis([0 4 -1 1.5]);
```

运行结果如图 4-17 所示。

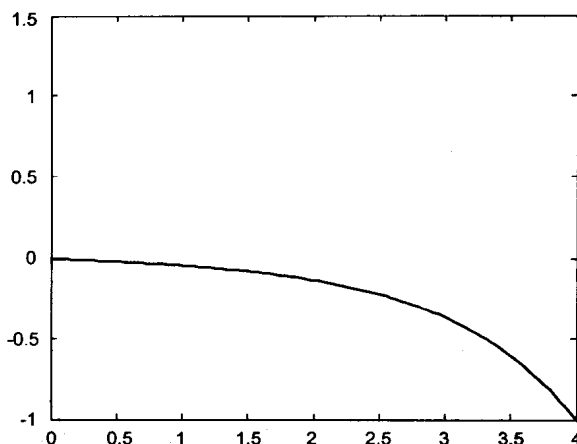


图 4-17 猜测初始值为 $y_1(x)=-1$, $y_2(x)=0$ 时的求解曲线

对于含有未知参数的边值问题, 如 Mathieu 方程:

$$y'' + (\lambda - 2q \cos 2x)y = 0$$

由于含有未知参数 λ , 故需要 3 个边值条件: $y'(0)=0$, $y'(\pi)=0$, $y(0)=1$ 。先将其转化为一阶常微分方程:

$$\begin{cases} y_1' = y_2 \\ y_2' = -(\lambda - 2q \cos 2x)y_1 \\ y_1(0)=1, y_2(\pi)=0, y_2(0)=0 \end{cases}$$

创建一个关于常微分方程的 M 文件, 输入如下代码并保存为 **mat4ode.m**:

```
function dydx = mat4ode(x,y,lambda)
dydx = [ y(2)
        -(lambda - 2*5*cos(2*x))*y(1) ];
```

创建一个关于边值问题的 M 文件, 输入如下代码并保存为 **mat4bc.m**:

```
function res = mat4bc(ya,yb,lambda)
res = [ ya(2)
        yb(2)
        ya(1)-1 ];
```

创建一个关于初值猜测的 M 文件, 输入如下代码并保存为 **mat4init.m**:

```
function yinit = mat4init(x)
yinit = [ cos(4*x)
        -4*sin(4*x) ];
```

在命令窗输入:

```

>> lambda = 15;
>> solinit = bvpinit(linspace(0,pi,10),@mat4init,lambda);
>> sol = bvp4c(@mat4ode,@mat4bc,solinit);
>> xint = linspace(0,pi);
>> sxint = deval(sol,xint);
>> plot(xint,Sxint(1,:))
>> axis([0 pi -1 1.1])
>> title('Eigenfunction of Mathieu 方程的特征函数')
>> xlabel('x')
>> ylabel('y')
>> title('Mathieu 方程的特征函数')

```

运行结果如图 4-18 所示。

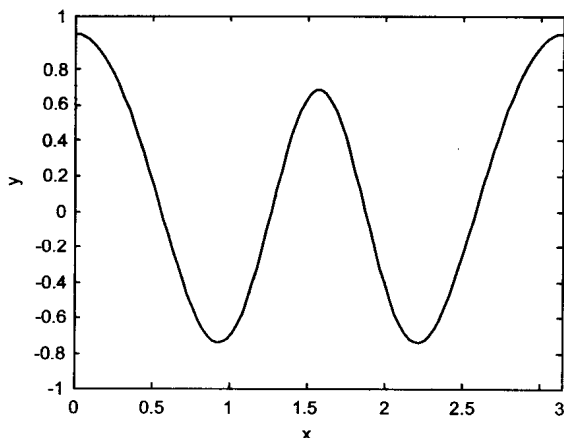


图 4-18 Mathieu 方程的特征函数

4.5.4 求解偏微分方程

本小节介绍如何根据初值和边值条件求解偏微分方程。MATLAB 提供的 `pdepe` 求解器专用于求解关于空间变量 x 和时间 t 的抛物线形或椭圆形偏微分方程，形如：

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right) \quad (4-11)$$

其中， $t_0 \leq t \leq t_f$ ， $a \leq x \leq b$ ，且时间区间 $[a, b]$ 必须为有限。 m 可以为 1、2 或 3，分别对应平板形、圆柱形和球形对称。如果 $m > 0$ ，则应有 $a \geq 0$ 。

式(4-11)中， $f(x, t, u, \partial u / \partial x)$ 为流项且必须依赖于 $\partial u / \partial x$ ， $s(x, t, u, \partial u / \partial x)$ 为源项。规定对角矩阵 $c(x, t, u, \partial u / \partial x)$ 应与 $\partial u / \partial x$ 相乘。

在初始时刻 t_0 ，对于一切 x ，应满足形如式(4-12)的初值条件：

$$u(x, t_0) = u_0(x) \quad (4-12)$$

在边界 $x=a$ ， $x=b$ 处，对于一切 t ，应满足形如式(4-13)的边值条件

$$p(x, t, u) + q(x, t)f(x, t, u, \partial u / \partial x) = 0 \quad (4-13)$$

MATLAB 中求解偏微分方程的 pdepe 求解器的基本调用格式为

$$\text{sol} = \text{pdepe}(m, \text{pdefun}, \text{icfun}, \text{bcfun}, \text{xmesh}, \text{tspan})$$

其中, m 为对应式(4-11)中的 m , 取值为 0、1、2; pdefun 为定义偏微分方程组成部分的函数, 以格式 $[c, f, s] = \text{pdefun}(x, t, u, \text{dudx})$ 计算式(4-11)的 c 项、 f 项和 s 项; icfun 为估计初值的函数; bcfun 的调用格式为 $[pl, ql, pr, qr] = \text{bcfun}(xl, ul, xr, ur, t)$, 是用于估计边值条件中 p 项和 q 项的函数, $*l$ 代表左边界, $*r$ 代表右边界; tspan 是从 t_0 到 t_f 指定需要求解处的时间向量 $[t_0, t_1, \dots, t_f]$; xmesh 为对应任意 tspan 中元素所需要求解的从 a 到 b 的值向量 $[x_0, x_1, \dots, x_n]$ 。

输出参量 sol 为三维数组, 如 $\text{sol}(:, :, k)$ 为解 u 的第 k 个分量, $\text{sol}(i, :, k)$ 为时间点 $\text{tspan}(i)$ 和空间点 $\text{xmesh}(:)$ 处的解的第 k 个分量, $\text{sol}(i, j, k)$ 为时间点 $\text{tspan}(i)$ 和空间点 $\text{xmesh}(j)$ 处的解的第 k 个分量。

下面以一个最简单的偏微分方程为例来介绍求解器 pdepe 的用法。设系统微分方程为

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

该方程中, $t \geq 0$, $0 \leq x \leq 1$, 且初值条件为

$$u(x, 0) = \sin \pi x$$

边值条件为

$$\begin{cases} u(0, t) = 0 \\ \pi e^{-t} + \frac{\partial u}{\partial t}(1, t) = 0 \end{cases}$$

要求解这个偏微分方程, 先将原方程写成标准型

$$\pi^2 \frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x} \left(x^0 \frac{\partial u}{\partial x} \right) + 0$$

则有

$$c \left(x, t, u, \frac{\partial u}{\partial x} \right) = \pi^2, \quad f \left(x, t, u, \frac{\partial u}{\partial x} \right) = \frac{\partial u}{\partial x}, \quad s \left(x, t, u, \frac{\partial u}{\partial x} \right) = 0$$

创建一个关于偏微分方程的 M 文件, 输入如下代码并保存为 `pdex1pde.m`:

```
function [c,f,s] = pdex1pde(x,t,u,DuDx)
c = pi^2;
f = DuDx;
s = 0;
```

创建一个关于初值问题的 M 文件, 输入如下代码并保存为 `pdex1ic.m`:

```
function u0 = pdex1ic(x)
u0 = sin(pi*x);
```

创建一个关于边值问题的 M 文件, 输入如下代码并保存为 `pdex1bc.m`:

```
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
```

```
(4-13) pr = pi * exp(-t);
```

```
qr = 1;
```

在命令窗输入:

```
>> x = linspace(0,1,20);
```

```
>> t = linspace(0,2,5);
```

```
>> m = 0;
```

```
>> sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
```

```
>> u = sol(:,1);
```

```
>> surf(x,t,u)
```

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

```
>> ylabel('t');
```

```
>> title('20 网格点偏微分方程数值解');
```

运行结果如图 4-19 所示。

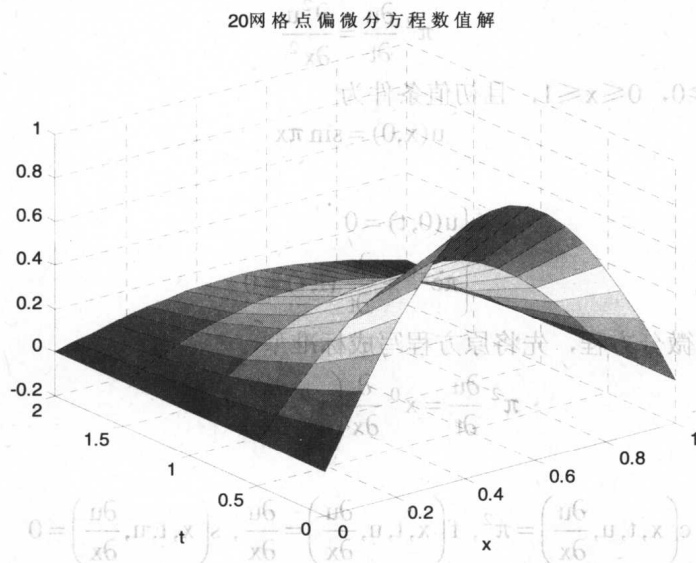


图 4-19 20 网格点偏微分方程数值解

4.6 稀疏矩阵

如果一个大矩阵中有很多零元素，MATLAB 若将这些零值与其他元素值一起存储，将会耗费很大的存储空间，并且可能需要更多的计算时间。稀疏矩阵提供了存储这类矩阵的有效方法，它只存储非零元素以及它们的行索引，从而节省了大量的存储空间。

4.6.1 创建稀疏矩阵

用户可以根据需要建立不同数据类型的稀疏矩阵。所有的 MATLAB 内置算法、逻辑、

索引都适用于稀疏矩阵，甚至稀疏矩阵与满矩阵的混合。如果运算对象是稀疏矩阵，那么返回值也将是稀疏矩阵；如果是满矩阵，那么返回值则是满矩阵。

表 4-8 给出了一些常用的稀疏矩阵操作函数。

表 4-8 常用的稀疏矩阵操作函数

函 数 名	描 述
full	将稀疏矩阵转化为满矩阵
issparse	判断是否为稀疏矩阵
nnz	矩阵的非零元素数目
nonzeros	矩阵的非零元素
nzmax	为矩阵非零元素分配的存储空间
spalloc	为稀疏矩阵分配存储空间
sparse	创建稀疏矩阵
speye	创建稀疏单位矩阵
sprand	创建均匀分布的随机稀疏矩阵

创建稀疏矩阵的方法很多，这里将介绍直接创建法、满阵转化法以及对角元素创建法。

1. 直接创建法

直接创建法是创建稀疏矩阵最直接的方法。用户可以利用函数 `sparse` 进行创建，其调用格式为

`S = sparse(i,j,s,m,n)`

其中，`i`、`j` 分别代表行和列索引向量；`s` 代表对应指定索引向量的非零元素值向量；`m`、`n` 分别代表新建稀疏矩阵的行阶和列阶。

【例】 直接创建稀疏矩阵。

在命令窗输入：

```
>> A = sparse([3 4 2],[2 3 4],[10 30 40],5,5)
```

运行结果：

```
A =
      (3,2)      10
      (4,3)      30
      (2,4)      40
```

结果的左侧为索引下标，右侧为该处元素的取值。

2. 满阵转化法

满阵转化法是创建稀疏矩阵的间接方法。用户依然可以利用函数 `sparse` 进行创建，其调用格式为

`S = sparse(A)`

`A` 为待转化的满矩阵。

【例】 利用满矩阵创建稀疏矩阵。

在命令窗输入：

```
>> A = [ 13    0   13    0
         0   23    0   24
         0    0   33    0
        21    0    0   43];
```

```
>> B = sparse(A)
```

运行结果:

```
B =
(1,1)      13
(4,1)      21
(2,2)      23
(1,3)      13
(3,3)      33
(2,4)      24
(4,4)      43
```

3. 对角元素创建法

这是创建稀疏矩阵的一种特殊方法。用户可以利用函数 `spdiags` 进行创建，其调用格式为

```
S = spdiags(B,d,m,n)
```

其中， B 为满矩阵； d 为整数向量，用来指定 B 的对应列向量在新建稀疏矩阵中的对角位置(如 0 代表主对角线)； m 、 n 分别代表新建稀疏矩阵的行阶和列阶。

【例】 利用满矩阵创建对角稀疏矩阵。

在命令窗输入:

```
>> B = [ 21   13    0
        31   23    0
        41   33   13
        51   43   24 ];
```

```
>> d = [-3 0];
```

```
>> A = spdiags(B,d,6,4)
```

运行结果:

```
A =
(1,1)      13
(4,1)      21
(2,2)      23
(5,2)      31
(3,3)      33
(6,3)      41
(4,4)      43
```

为了便于理解该函数的用法，可在命令窗输入 `full(A)` 得到直观的满阵形式:

```
ans =
    13     0     0     0
     0    23     0     0
     0     0    33     0
    21     0     0    43
     0    31     0     0
     0     0    41     0
```

4.6.2 稀疏矩阵的查看

MATLAB 支持对稀疏矩阵数量形式和图形形式信息的查看,包括获取非零元素的信息、查看图形表示的稀疏矩阵以及获取所有非零元素的索引和取值。

1. 获取非零元素信息

有关获取非零元素信息的高级命令有以下几种:

- nnz: 获取稀疏矩阵非零元素数目。
- nonzeros: 获取由稀疏矩阵所有非零元素构成的列向量。
- nzmax: 获取稀疏矩阵占用的存储空间大小。

west0479 是 MATLAB 提供的稀疏矩阵,下面将以该矩阵为例进行说明。

【例】 获取稀疏矩阵信息。

在命令窗输入:

```
>> load west0479
```

```
>> whos
```

运行结果:

Name	Size	Bytes	Class	Attributes
west0479	479x479	24564	double	sparse

继续在命令窗输入:

```
>> nnz(west0479)
```

运行结果:

```
ans =
    1887
```

继续在命令窗输入:

```
>> nonzeros(west0479)
```

运行结果:

```
ans =
    1.0000e+000
   -3.7648e-002
   -3.4424e-001
    1.0000e+000
   -2.4523e-002
```

```
-3.7371e-001
```

```
:
```

继续在命令窗输入:

```
>> nzmax(west0479)
```

运行结果:

```
ans =
```

```
1887
```

2. 查看图形表示的稀疏矩阵

通过图形表示稀疏矩阵可以直观地查看非零元素的分布。利用 MATLAB 的 `spy` 函数可以查看稀疏矩阵的结构。

【例】 查看图形表示的稀疏矩阵。

在命令窗输入:

```
>> spy(west0479)
```

运行结果如图 4-20 所示。

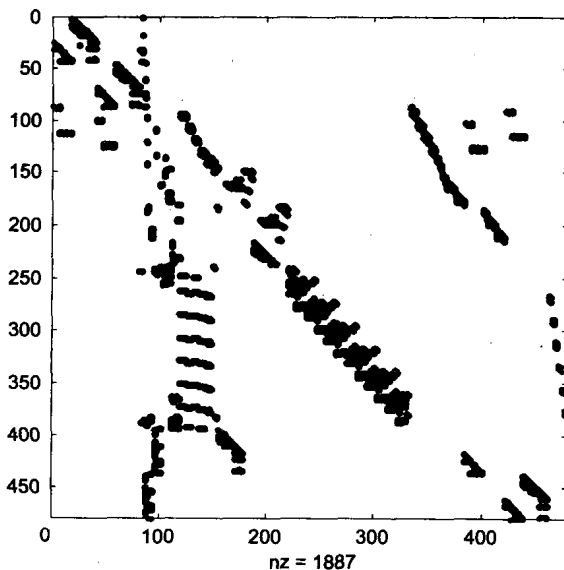


图 4-20 图形表示的稀疏矩阵

3. 获取非零元素索引和取值

利用 `find` 函数可以获取非零元素的索引和取值, 其调用格式为

```
[i,j,s] = find(S)
```

其中, `i`、`j` 分别代表行索引、列索引; `s` 为该索引处的值; `S` 为稀疏矩阵。

4.6.3 稀疏矩阵的操作

大多数 MATLAB 标准数学函数都适用于稀疏矩阵。此外, MATLAB 还提供了一系列专门用于稀疏矩阵操作的函数。稀疏矩阵的操作遵循以下几条规则:

● 返回值为标量或者尺寸固定的向量的函数作用于稀疏矩阵时,其结果为满存储形式。例如,无论输入参量是否为稀疏矩阵, `size` 函数的返回值都是满向量。

● 输入参量为标量或者向量,返回值为矩阵的函数(如 `zeros`、`rand` 以及 `eye` 函数),通常返回值为满矩阵。这是为了避免得到不希望得到的稀疏矩阵。但是,如果需要得到这类特殊的稀疏矩阵,可以采用相应的返回值为稀疏矩阵的函数(如 `sparse`、`sprand` 以及 `speye`)来实现。

● 输入参量为矩阵、返回值为矩阵或者向量的一元函数,其结果将保留为操作数的类型。例如,若操作数 S 为稀疏矩阵,则 `chol(S)` 也是稀疏矩阵,并且 `diag(S)` 为稀疏向量。列操作函数如 `max(S)`、`sum(S)` 将返回稀疏向量,尽管这些向量有可能不包含零元素。注意 `sparse` 和 `full` 函数例外。

● 如果操作数均为稀疏矩阵,那么二元运算的结果为稀疏矩阵;如果操作数均为满矩阵,那么二元运算的结果则为满矩阵。对于混合操作数,如果运算不保留稀疏性,则结果为满矩阵。例如, S 为稀疏矩阵, F 为满矩阵,则 $S+F$ 、 $S \cdot F$ 、 $F \cdot S$ 的运算结果为满矩阵,而 $S \cdot F$ 与 $S \& F$ 的运算结果为稀疏矩阵。个别情况下,尽管矩阵只有很少的零元素,运算结果仍为稀疏矩阵。

● 利用 `cat` 函数或中括号 `[]` 串接混合操作数将得到稀疏矩阵。

● 子阵索引位于赋值语句的右侧时,操作将保留操作数的存储形式,除非操作结果为标量。例如当 i 、 j 中有一个为向量时, $T = S(i,j)$ 将返回一个稀疏矩阵; i 、 j 均为标量时,则返回一个满标量。子阵索引位于赋值语句左侧时,如 $T(i,j) = S$,不会改变左侧矩阵的存储形式。

● 乘除法仅仅作用于稀疏矩阵的非零元素。稀疏矩阵 S 除以零,其结果为稀疏矩阵且在 S 的非零元素位置取值为 `Inf`。稀疏矩阵 S 乘以 `Inf` 或 `NaN`,其结果在 S 的非零元素位置的取值分别为 `Inf` 或 `NaN`,但是零元素位置取 `NaN`。

1. 稀疏矩阵的置换与重新排序

稀疏矩阵的行列置换可用如下两种方法表示:

- 置换矩阵 P ,以 $P \cdot S$ 的方式作用于行,或以 $S \cdot P$ 的方式作用于列。
- 置换向量 p ,由 $1:n$ 置换而成的满向量,以 $S(p,:)$ 的方式作用于行,或以 $S(:,p)$ 的方式作用于列。

【例】 创建置换向量和置换矩阵。

在命令窗输入:

```
>> p = [1 4 3 5 2]
I = eye(5,5);
P = I(p,:);
e = ones(4,1);
S = diag(10:10:50) + diag(e,1) + diag(e,-1)
```

运行结果:

```
p =
    1     4     3     5     2
```

P =

1	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	0	0	1
0	1	0	0	0

S =

10	1	0	0	0
1	20	1	0	0
0	1	30	1	0
0	0	1	40	1
0	0	0	1	50

接下来, 可以利用置换向量 **p** 或者置换矩阵 **P** 对 **S** 进行置换。

【例】 行置换。

继续在命令窗输入:

```
>> S(p,:)
```

运行结果:

ans =

10	1	0	0	0
0	0	1	40	1
0	1	30	1	0
0	0	0	1	50
1	20	1	0	0

或输入:

```
>> P*S
```

运行结果:

ans =

10	1	0	0	0
0	0	1	40	1
0	1	30	1	0
0	0	0	1	50
1	20	1	0	0

【例】 列置换。

继续在命令窗输入:

```
>> S(:,p)
```

运行结果:

ans =

10	0	0	0	1
1	0	1	0	20
0	1	30	0	1


```

0    40    1    1    0
0     1    0   50    0

```

或输入:

```
>> S*P'
```

运行结果:

```

ans =
10     0     0     0     1
 1     0     1     0    20
 0     1    30     0     1
 0    40     1     1     0
 0     1     0    50     0

```

如果需要稀疏矩阵表达式, 则应创建稀疏形式的单位矩阵 I 。对比上例, 应有 $I = \text{speye}(5)$, 且 $P = I(p,:)$, $p = (1:5)*P'$, $p = (P*(1:5))'$ 。

置换矩阵 P 的逆可以由 P' 来计算, 置换向量 p 的逆则由 $r(p)=1:5$ 来计算。

【例】 置换向量的逆。

继续在命令窗输入:

```
>> r(p) = 1:5
```

运行结果:

```

r =
 1     5     3     2     4

```

对矩阵的列进行排序能使 LU 分解和 QR 分解具有更优的稀疏性, 而对矩阵的行和列进行排序能使 Cholesky 分解具有更优的稀疏性。通过命令 $p = \text{colperm}(S)$ 可以得到稀疏矩阵 S 的列置换向量 p , 列按非零元素升序排列。对列进行排序, 有利于 LU 分解, 如 $\text{lu}(S(:,p))$ 。如果对行和列进行排序, 有利于 Cholesky 分解, 如 $\text{chol}(S(p,p))$ 。

【例】 稀疏排序。

继续在命令窗输入:

```
>> p = colperm(S)
```

运行结果:

```

p =
 1     5     2     3     4

```

MATLAB 还提供了稀疏反向 Cuthill-McKee 排序的函数 symrcm , 其调用格式为 $r = \text{symrcm}(S)$ 。该命令返回 S 的对称反向 Cuthill-McKee 排序 r , 使 S 的非零元素集中在主对角线附近。

实现稀疏对称最小度排序的函数为 symmmd , 其调用格式为 $p = \text{symmmd}(S)$ 。该命令将返回 S 的对称最小度排列向量 p , S 为对称正定矩阵。

【例】 稀疏对称最小度排序。

```
>> B = bucky+4*speye(60);
```

```
r = symrcm(B);
```

```
p = symmmd(B);
```

```

R = B(r,r);
S = B(p,p);
subplot(2,2,1), spy(R), title('B(r,r)')
subplot(2,2,2), spy(S), title('B(s,s)')
subplot(2,2,3), spy(chol(R)), title('chol(B(r,r))')
subplot(2,2,4), spy(chol(S)), title('chol(B(s,s))')

```

运行结果如图 4-21 所示。

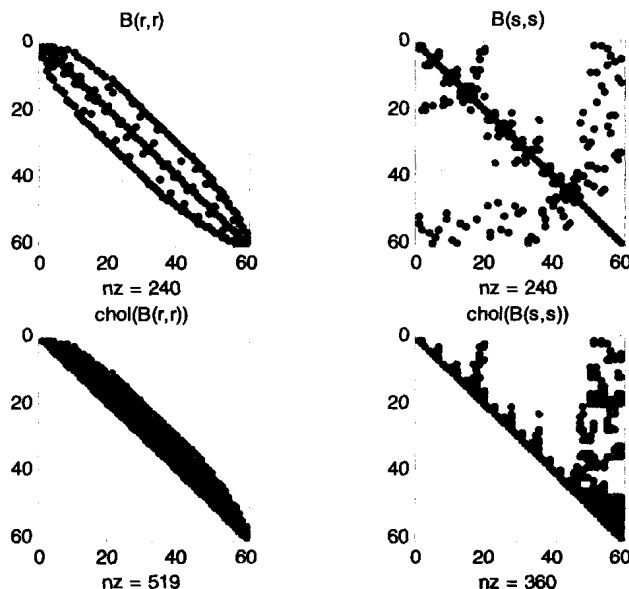


图 4-21 稀疏对称最小度排序

2. 稀疏矩阵的分解

稀疏矩阵的不完全 LU 分解通常使用函数 `luinc`，其调用格式如下：

- `[L,U] = luinc(X,'0')`：稀疏矩阵不完全 LU 分解，L 为下三角矩阵的置换形式，U 为上三角矩阵，'0' 是一种分解标准。

- `[L,U,P] = luinc(X,'0')`：L 是主对角线元素为 1 的下三角阵，U 为上三角阵，P 为单位矩阵的置换形式。

- `[L,U] = luinc(X,options)`：options 为结构，其字段 `droptol` 表示指定的舍入误差；字段 `milu` 如果为 1，表示改变分解以便于从上三角分解因子中抽取被去掉的列元素，默认值为 0；字段 `ugiag` 如果为 1，表示用 `droptol` 值代替上三角因子中的对角线上的零元素，默认值为 0；字段 `thresh` 表示中心临界值。

【例】稀疏矩阵的 LU 分解。

在命令窗输入：

```

>> S = [11  0 13  0
        41  0  0 44
         0 22  0 24

```

```
0 0 63 0];
```

```
>> S=sparse(S);
```

```
>> [L1,U1] = luinc(S,'0'), [L2,U2,p2] = luinc(S,'0')
```

运行结果:

L1 =

(1,1)	0.2683
(2,1)	1.0000
(3,2)	1.0000
(1,3)	0.2063
(4,3)	1.0000
(1,4)	1.0000

U1 =

(1,1)	41
(2,2)	22
(3,3)	63
(1,4)	44
(2,4)	24

L2 =

(1,1)	1.0000
(4,1)	0.2683
(2,2)	1.0000
(3,3)	1.0000
(4,3)	0.2063
(4,4)	1.0000

U2 =

(1,1)	41
(2,2)	22
(3,3)	63
(1,4)	44
(2,4)	24

p2 =

(4,1)	1
(1,2)	1
(2,3)	1
(3,4)	1

稀疏矩阵的不完全 Cholesky 分解通常使用函数 cholinc, 其调用格式如下:

- $R = (X, \text{droptol})$: 稀疏矩阵 X 的不完全 Cholesky 分解, droptol 为指定误差。
- $R = \text{cholinc}(X, \text{options})$: options 为结构, 其字段 droptol 表示舍入误差; 字段 michol 如果为 1, 则从对角线上抽取出被去掉的元素; 字段 rdiag 如果为 1, 表示用 droptol 值代替

上三角分解因子中的对角线上的零元素。

- $R = \text{cholinc}(X, '0')$: 正定对称实稀疏矩阵的不完全分解。如果分解不存在, 则会产生出错信息。

- $[R, p] = \text{cholinc}(X, '0')$: 不产生任何出错信息。如果 R 存在, 则 $p = 0$; 如果 R 不存在, 则 p 为正整数。

- $R = \text{cholinc}(X, 'inf')$: 进行 Cholesky 无穷分解。

稀疏矩阵的完全 QR 分解可以通过命令 $[Q, R] = \text{qr}(S)$ 来实现, 但通常情况下这种做法是不切实际的, 因为正交矩阵 Q 的零元素所占的比例不高。实用的命令是:

- $R = \text{qr}(S)$: 只产生一个上三角阵 R , 满足 $R' * R = S' * S$, 这种方法在计算 $S' * S$ 时减少了内在数字信息的损耗。

- $[C, R] = \text{qr}(A, b)$: 用于稀疏最小二乘问题 $\min \|Ax - b\|$ 的两步解, $[C, R] = \text{qr}(A, b)$, $x = R \setminus c$ 。

- $R = \text{qr}(A, 0)$: 得到稀疏矩阵“经济”的分解。

- $[C, R] = \text{qr}(A, b, 0)$: 得到稀疏最小二乘问题“经济”的分解。

3. 稀疏矩阵的线性方程组求解

求解线性方程组的方法有两类:

- 直接法: 基于高斯消去法, 如通过 LU 分解或 Cholesky 分解求解。运用运算符 \setminus 或 $/$ 直接执行。

- 迭代法: 求得限定步长范围内的近似解, 通常只应用于稀疏矩阵。

【例】直接法求解线性方程组: $Ax = b$ 。

在命令窗输入:

```
>> A = [ 11    0   13    0
         41    0    0   44
          0   22    0   24
          0    0   63    0];
```

```
p = colperm(A);
```

```
b = [1; 2; 3; 5];
```

```
x = A(:,p) \ b;
```

```
x(p) = x;
```

验证运行结果:

```
>> A*x-b
```

```
ans =
```

```
0
```

```
0
```

```
0
```

```
0
```

【例】迭代法求解线性方程组: $Ax = b$ 。

继续在命令窗输入:

```
>> [x,flag,relres,iter,resvec] = bicg(A,b)
```

运行结果:

```
x =  
    -0.0029  
     0.0838  
     0.0794  
     0.0481  
  
flag =  
      0  
  
relres =  
    7.8166e-015  
  
iter =  
      4  
  
resvec =  
     6.2450  
     3.3483  
     3.7538  
     0.2061  
     0.0000
```

验证运行结果:

```
>> A*x-b  
ans =  
    1.0e-013 *  
    -0.0555  
    -0.1354  
     0.3819  
    -0.2665
```

可见残向量不为零,因此迭代法的结果存在一定误差。

4. 稀疏矩阵的特征值与奇异值

稀疏矩阵的特征值可以通过 `eigs` 函数来求解。利用命令 `[V,lambda] = eigs(A,k,sigma)` 可以求得稀疏矩阵 A 的 k 个特征值 λ 和对应特征向量 V 。其中, σ 取 'lm', 表示求最大模值的特征值, 取 'sm', 表示求最小模值特征值; 对实对称问题, σ 取 'la', 表示求最大特征值, 取 'sa', 表示求最小特征值, 取 'be' 表示求最大和最小的特征值; 对非对称和复数问题, σ 取 'lr', 表示求最大实部特征值, 取 'sr', 表示求最小实部特征值, 取 'li', 表示求最大虚部特征值, 取 'si', 表示求最小虚部特征值。

稀疏矩阵的奇异值可以通过 `svds` 函数来求解。利用命令 `[U,S,V] = svds(A,k)` 可以求得稀疏矩阵 A 的 k 个最大奇异值; 利用命令 `[U,S,V] = svds(A,k,0)` 可以求得稀疏矩阵 A 的 k 个最小奇异值。

第 5 章 M 文件程序设计基础

MATLAB 是一种基于矩阵数据结构的高级程序语言。用户可以通过编写 M 文件实现高级的程序设计, 如开发和扩充自己的函数库、创建和运行脚本命令文件、输入或输出各种类型的数据文件、面向对象编程等。M 文件的扩展名为 .m, 包括脚本文件和函数文件。通常使用脚本文件实现复杂的运算或者系统数值仿真, 使用函数文件编写用户自定义的函数。M 文件的语法风格与 C 语言类似, 因而通俗易懂学。

本章将介绍 M 文件的编写方法、编程技巧及相关注意事项。

5.1 M 文件介绍

M 文件分为脚本文件和函数文件, 扩展名都是 .m。M 文件的创建方法有很多种, 可以通过在主菜单选择 File > New > M-File 或在工具栏单击新建按钮来创建, 也可以在当前路径窗口下通过右键菜单来创建, 还可以通过 edit 命令来创建。本节将介绍有关 M 文件的基本分类和组成。

5.1.1 脚本和函数

1. 脚本文件

脚本文件中没有输入或输出参量, 因而它是最简单的 M 文件类型。当运行一个脚本文件时, MATLAB 将逐行执行文件内的每条指令。它对工作区间的数据进行操作, 或者创建新的数据并保存到工作区间, 而且在运行结束后数据仍然存在。需要注意的是, 脚本的运行有可能覆盖工作区间原本需要保留的数据。下面给出一个脚本文件创建和运行的例子。

创建一个 M 文件, 输入如下代码并保存为 plotstems.m:

```
% An M-file script                                % Comment lines
n = 0:20;                                           % Computations
stems1(1,:) = sin(2*pi/21*n);
stems1(2,:) = cos(2*pi/21*n);
stems1(3,:) = exp(-0.2*n);
for k = 1:3
    stem(n, stems1(k,:), 'full')    % Graphics output
    pause
```

end

在命令窗输入:

```
>> plotstems
```

运行结果如图 5-1 所示。

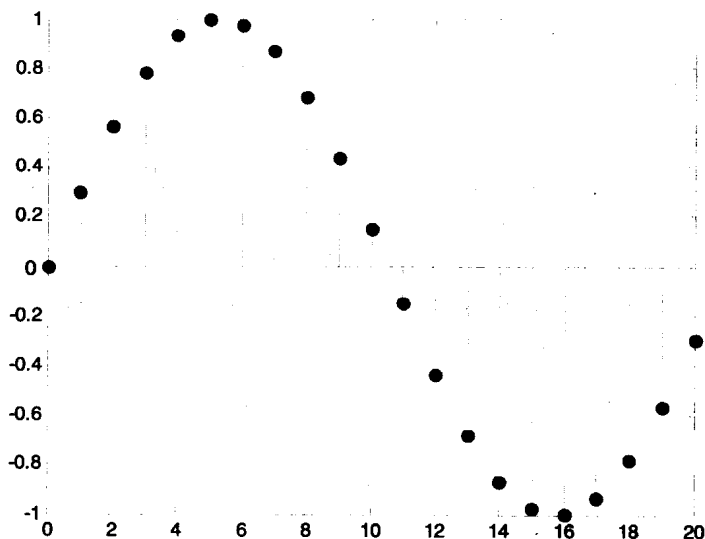


图 5-1 使用脚本文件绘图结果

运行完毕后,可以按回车键显示下一幅图。文件中的变量 `k`、`n`、`stems1` 等将保留于工作区间之内。

2. 函数文件

函数文件是含有输入和输出参量的 M 文件,它在自己的工作区间(函数工作区间)内对变量进行操作。函数工作区间与 MATLAB 工作空间不同,它只在函数内部传递变量而且不会互相覆盖。

MATLAB 的函数文件通常由以下部分组成:

- 函数定义行;
- H1 行: 帮助文本的第一行,以“%”开始;
- 帮助文本;
- 注释;
- 函数体。

例如,对于一个名为 `fact.m` 的计算 n 阶乘的函数文件,其各组成部分描述如下:

<code>function f = fact(n)</code>	函数定义行
<code>% Compute a factorial value.</code>	H1行
<code>% FACT(N) returns the factorial of N,</code>	帮助文本
<code>% usually denoted by N!</code>	
<code>% Put simply, FACT(N) is PROD(1:N).</code>	注释
<code>f = prod(1:n);</code>	函数体

如果要计算 $4!$ ，在命令窗输入：

```
>> fact(4)
```

运行结果：

```
ans =
```

```
24
```

5.1.2 P 代码文件

对于一个函数或者脚本 M 文件，可以对其进行预解析，即由 M 文件创建对应的 P 代码文件。例如，使用命令 `pcode fact` 对 M 文件 `fact.m` 进行解析，生成伪代码并保存为 `fact.p` 文件。如果同时存在 `fact.m` 和 `fact.p` 两个文件，MATLAB 则优先运行 `fact.p` 文件。由于 P 代码文件是 M 文件经过解析的结果，因此它的运行速度要比 M 文件快。在运行运算量巨大的程序时这种差距尤为显著，例如大型的图形用户界面(GUI)应用。另外，P 代码文件还能够隐藏程序的算法，这是 M 文件所不能实现的。

5.1.3 变量类型

编写 M 文件时，所要用到的变量不需要预先定义。变量名必须以字母开头，其后可以是字母、数字或下划线。变量名的大小写是有区别的，并且不能与函数名或者 MATLAB 中的关键字重名。

MATLAB 中有三种变量类型：局部变量、全局变量和持续变量。

1. 局部变量

局部变量存在于 MATLAB 函数之中，只在它所在的函数内有效(嵌套函数除外)，并且与 MATLAB 工作区间相互独立。除了全局变量或持续变量，一个函数内定义的变量在调用另一个函数时不会在内存中保留，它也不会覆盖 MATLAB 工作区间内的变量。

2. 全局变量

全局变量是由 `global` 关键字定义在 MATLAB 函数之中，并可以与其他函数或命令共享的变量。例如，在一个名为 `add3.m` 的 M 文件函数内，存在两个全局变量 `a`、`b`：

```
function y = add3(x)
```

```
global a b
```

```
y = a + b + x;
```

运行该函数之前，需要对全局变量进行声明，在命令窗输入：

```
>> global a b
```

```
>> a = 1;
```

```
>> b = 2;
```

```
>> add3(3)
```

运行结果：

```
ans =
```

```
6
```


3. 持续变量

持续变量只能在 M 文件函数内声明和使用, 由关键字 `persistent` 来定义。只有声明了该变量的函数才可以对其进行访问。如果函数存在, 即使 MATLAB 再调用其他函数, 该函数内的持续变量仍将保留于内存中。例如, 在一个名为 `findSum.m` 的 M 文件函数内, 存在一个持续变量 `SUM_X`:

```
function findSum(inputvalue)
persistent SUM_X
if isempty(SUM_X)
    SUM_X = 0;
end
SUM_X = SUM_X + inputvalue
```

第一次调用 `findSum` 函数时(如 `findSum(3)`), 计算得到的 `SUM_X` 的值等于函数的输入参量与原始 `SUM_X`(`SUM_X = 0`)相加之和。

```
>> findSum(3)
SUM_X =
    3
```

再次调用 `findSum` 函数时, 由于 `SUM_X` 是持续变量, 第一次的计算结果(`SUM_X = 3`)将仍旧存在, 故计算得到的 `SUM_X` 的值等于函数的输入参量与现在的 `SUM_X`(`SUM_X = 3`)相加之和。

```
>> findSum(3)
SUM_X =
    6
```

如此往复, `SUM_X` 的值将反复与输入参量进行叠加。

```
>> findSum(3)
SUM_X =
    9
...
```

5.1.4 关键字和特殊值

1. 关键字

MATLAB 中定义了一些关键字, 在定义变量名时不能与之重名。通过 `iskeyword` 命令可以查看 MATLAB 中定义的所有关键字。

```
>> iskeyword
ans =
    'break'
    'case'
    'catch'
    'classdef'
```

```

'continue'
'else'
'elseif'
'end'
'for'
'function'
'global'
'if'
'otherwise'
'parfor'
'persistent'
'return'
'switch'
'try'
'while'

```

2. 特殊值

MATLAB 中定义了一些具有特殊值的变量，如表 5-1 所示。与关键字不同，定义与这些特殊值重名的变量相当于修改这些特殊值。

表 5-1 特 殊 值

变 量	含 义
ans	最近结果，没有输出变量赋值表达式时，MATLAB 自动将输出变量存储于 ans
eps	浮点相对精度，MATLAB 计算容许误差
intmax	计算机能表示的最大 8、16、32 或 64 位整数
intmin	计算机能表示的最小 8、16、32 或 64 位整数
realmax	计算机能表示的最大浮点数
realmin	计算机能表示的最小正浮点数
pi	圆周率
i,j	虚数单位
inf	无穷
NaN	非数
computer	计算机类型信息字符串
version	MATLAB 版本信息字符串

5.1.5 符号参考

在 M 文件编程中，经常要用到一些特殊的符号。本小节将对所有特殊符号(不包括算术、关系和逻辑运算符)进行归纳，作为编程的参考。

1. 星号 —— *

星号通常用在文件操作中, 作为文件名或路径名的通配符, 可以代表一个或多个字符。例如, 命令 `dir('Untitled*.m')` 将罗列所有当前路径下以 `Untitled` 开头且扩展名为 `.m` 的文件。

2. “猴头” —— @

“猴头”可以作为构成一般函数或匿名函数句柄的运算符, 也可以作为 MATLAB 类路径的标志符:

- 构成函数句柄, 如 `fhandle = @plot`、`fhandle = @(x) 2*x^2`。
- 类路径标志符, 如 `\@myclass\get.m`。

3. 冒号 —— :

冒号是产生数值序列的运算符, 常用于数组的创建和索引, 在第 2 章中已作过阐述。另外, 利用冒号可以将一个数组转化为列向量。

【例】 数组转化列向量。

在命令窗输入:

```
>> A = magic(3), B = A(:)
```

运行结果:

```
A =  
    8    1    6  
    3    5    7  
    4    9    2  
  
B =  
    8  
    3  
    4  
    1  
    5  
    9  
    6  
    7  
    2
```

利用冒号还可以在不改变已有数组形状的前提下对数组进行赋值。

【例】 保留数组形状并赋值。

在命令窗输入:

```
>> A = magic(3), A(:) = 1:9
```

运行结果:

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

A =

1	4	7
2	5	8
3	6	9

4. 逗号 —— ,

逗号用于分隔以下几种类型的元素:

- 分隔矩阵的行元素, 如 `A = [3.25, 8.32, 6.78]`。
- 分隔数组的索引, 如 `X = A(2, 1, 3)`。
- 分隔函数输入或输出参量, 如 `function[data, text] = xlsread(file, sheet, range, mode)`。
- 分隔一行内的命令或语句并显示运行结果, 如 `for k = 1:3, sum(A(k)), end`。

5. 大括号 —— { }

大括号用于单元数组的创建和索引:

- 创建单元数组, 如 `C = {[3 4 1 9], magic(4), 'Cstr'}`。
- 索引单元数组, 如 `A = C{1,1,2}`。

6. 单点号 —— .

单点号可用于结构的字段定义或对象方法的分类符:

- 定义结构字段, 如 `funds(2,3).bondtype = 'Corporate'`。
- 指定对象属性, 如 `val = asset.current_value`。

7. 双点号 —— ..

双点号用于表示上一级目录, 如当前路径为 `F:\matlab\work`, 命令 `cd ../bin` 则设置当前路径为 `F:\matlab\bin`。

8. 省略号 —— ...

省略号由三个点号构成, 是 MATLAB 中的续行符。当一行语句过长时, 可以在其后加省略号并在下一行续行书写。

9. 点圆扩号 —— .()

点圆括号常用于指定动态结构的字段名。如 `type = funds(3,2).(fundtype)` 指定的 `fundtype` 就是动态的字段名。

10. 叹号 —— !

叹号用于执行需要在 MATLAB 中执行的系统命令, 如命令 `!regedit` 将打开系统注册表。

11. 圆扩号 —— ()

圆括号常用于数组的索引(见第 2 章), 或用于指定函数的输入参数, 如 `function dydt = odefun(t,y)`。

12. 百分号 —— %

百分号主要用于注释一程序, 有时也用于分类符的转换, 如 `sprintf('%s = %d', name, value)`。

13. 百分扩号 —— %{ %}

百分括号用于注释一段程序，被注释的段程序以 %{ 开头，以 %} 结束。

14. 分号 —— ;

分号用于创建数组、抑制 MATLAB 输出或分隔一行内的命令或语句：

- 分隔矩阵的行，如 $A = [3, 2; 3, 1]$ 。
- 抑制输出，即语句末尾存在分号时将不显示该语句的运行结果。
- 分隔一行内的命令或语句，如 $A = 1; B = 4, C = 3$ ；将只显示 B 的结果。

15. 单引号 —— ''

单引号用于创建字符数组，如 $Str = 'Word'$ 。

16. 空格符 ——

空格符用于分隔以下几种元素：

- 分隔矩阵的行元素，如 $A = [5 \ 8 \ 3]$ 。
- 分隔函数的输入参量，如 $function [data \ text] = xlsread(file, sheet, range, mode)$ 。

17. 斜杠与反斜杠 —— / \

斜杠或反斜杠用于分隔表示路径的字符。在 Windows 系统的路径表示中，斜杠与反斜杠的作用是相同的；在 UNIX 系统中，只能用斜杠来表示路径。

18. 中扩号 —— []

中括号有以下几种用途：

- 创建数组，如 $A = [3 \ 4 \ 2; 2 \ 4 \ 5]$ 。
- 串接数组，如 $A = [eye(6), magic(6)]$ 。
- 函数的声明和调用，将返回值包括在中括号内，如 $[data, text] = xlsread(file, sheet, range, mode)$ 。

5.2 程序流程控制

通常情况下程序是顺序执行的，而利用程序控制语句可以实现特殊的执行流程控制。MATLAB 的程序流程控制可分为条件控制语句、循环控制语句、错误控制语句和程序终止语句。

5.2.1 条件控制语句

条件控制语句用于实现根据条件选择需要执行的程序。如果基于一个条件的真或假来选择执行某段程序，则使用 if 语句；如果要根据多种可能存在的条件来选择执行某段程序，则使用 switch 语句。

1. if 语句

if 语句的一般调用格式为

if 条件表达式 1
语句体 1

```
elseif 条件表达式 2
    语句体 2
elseif 条件表达式 3
    语句体 3
...
else
    语句体 n
end
```

如果 if 或者某个 elseif 之后的条件表达式为真, 则执行该 if 或者 elseif 之下的语句体; 如果 if 以及所有 elseif 之后的条件表达式为假, 则执行 else 之下的语句体。if 语句中, 可以没有 elseif 条件或者 else 条件, 但是必须有 if 条件。

【例】 检测输入值是否为 -1、0、1。

在命令窗输入:

```
>> input = 1;
>> if input == -1
    disp('negative one');
elseif input == 0
    disp('zero');
elseif input == 1
    disp('positive one');
else
    disp('other value');
end
```

运行结果:

positive one

2. switch 语句

switch 语句的一般调用格式为

```
switch 表达式
case 表达式取值 1
    语句体 1
case 表达式取值 2
    语句体 2
...
otherwise
    语句体 n
end
```

当 switch 后的表达式取值为某个 case 之后的取值时, 程序将执行该 case 之下的语句体。如果找到第一个满足表达式取值的 case, 其他的 case 语句将不再执行; 如果没有满足表达式取值的 case, 则执行 otherwise 之下的语句体。

【例】 检测输入值是否为 -1、0、1。

在命令窗输入：

```
>> input = 1;
>> switch input
    case -1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
        disp('other value');
end
```

运行结果：

```
positive one
```

5.2.2 循环控制语句

循环控制语句用于重复执行某段程序代码，可用通过循环条件来控制循环的次数。

MATLAB 的循环控制语句有 for 语句和 while 语句。

1. for 语句

for 语句按照预先指定好的循环次数执行一条语句或语句体，其常用调用格式为

```
for index = start1:increment1:end1
    语句体
end
```

其中，start1 为起始值，increment1 为步进值，end1 为结束值，循环次数为 $1 + [(end1 - start1) / increment1]$ (中括号 [] 为向零取整运算)。

【例】 计算 4!。

在命令窗输入：

```
>> x = 1;
>> for i = 1:1:4
    x = x * i;
end
>> x
```

运行结果：

```
x =
    24
```

此外还有一种 for 语句格式，以矩阵 A 的列阶数作为循环次数。

```
for k = A
    语句体
end
```

2. while 语句

如果 while 语句之后的表达式为真，while 语句将始终执行，其常用的调用格式为

```
while 表达式
    语句体
end
```

【例】 计算 4!。

在命令窗输入：

```
>> i = 0;
>> x = 1;
>> while i < 4
    i = i + 1;
    x = i * x;
end
x
```

运行结果：

```
x =
    24
```

while 语句也可以是关于矩阵或数组的表达式，但在这种情况下当所有元素全部为真时表达式才为真。

3. continue 语句

continue 语句的作用是结束本次 for 或 while 循环(即跳过循环体内该语句之后的所有语句)，直接进入下一个循环的执行判断。

【例】 显示 magic.m 文件行数。

在命令窗输入：

```
>> fid = fopen('magic.m', 'r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) || strcmp(line, '%', 1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines', count));
```


运行结果:

25 lines

4. break 语句

break 语句的作用是终止 for 或 while 循环, 即只跳出本层循环, 而不跳出外层循环。

【例】 显示 magic.m 文件帮助。

在命令窗输入:

```
>> fid = fopen('magic.m', 'r');
s = "";
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
        break
    end
    s = strvcats(s, line);
end
disp(s)
```

运行结果:

```
function M = magic(n)
%MAGIC Magic square.
% MAGIC(N) is an N-by-N matrix constructed from the integers
% 1 through N^2 with equal row, column, and diagonal sums.
% Produces valid magic squares for all N > 0 except N = 2.
```

5.2.3 错误控制语句

错误控制语句 try-catch-end 的作用是在 try 之下的一个语句出现错误时跳出该语句体并执行 catch 语句体, 其基本调用格式为

```
try
    语句体 1
catch
    语句体 2
end
```

使用 lasterr 函数可以查询最后的错误信息, 查询结果为空字符串时表示语句体 1 成功执行。

【例】 错误控制判断。

在命令窗输入:

```
>> n = 4;
A = eye(3);
try
```

```
an = A(n,:),  
catch  
an = A(end,:),  
end  
lasterr
```

运行结果:

```
an =  
    0    0    1  
ans =  
Index exceeds matrix dimensions.
```

5.2.4 程序终止语句

程序终止语句 `return` 用于终止当前的命令序列并返回正在被调用的函数，也可以用于终止 `keyboard` 方式。例如计算行列式的函数 `det.m` 中，利用程序终止语句 `return` 对于输入矩阵为空的情况做了如下处理(即令空矩阵的行列式值为 1 并返回该函数):

```
function d = det(A)  
%DET det(A) is the determinant of A.  
if isempty(A)  
    d = 1;  
    return  
else  
    ...  
end
```

5.3 数据输入/输出

MATLAB 提供了一系列用于文件输入/输出(I/O)的底层函数，这些函数是基于 ANSI 标准 C 语言 I/O 函数的。对文件进行底层 I/O 操作时，一般有以下三个步骤:

- (1) 打开文件;
- (2) 读写数据;
- (3) 关闭文件。

本节将主要介绍 MATLAB 的文件操作的三个步骤及需要用到的函数。

5.3.1 打开文件

对文件进行读写操作以前必须先打开文件。MATLAB 的 `fopen` 函数用于打开文件，其调用格式如下:

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

其中参数 `filename` 为要打开的文件名,也可以在这个参数中指定文件的路径信息。`permission` 指定要对文件进行操作的方式,可以为以下任一字符串。

- 'r': 只读方式。
- 'w': 只写方式。该方式将覆写原有文件内容,若文件不存在,则生成新文件。
- 'a': 追加方式。保留原有文件内容,在文件尾追加写入数据。
- 'r+': 可读可写方式。
- 'w+': 清除文件内容或创建文件用于文件读写。
- 'a+': 可读、追加方式。如果文件不存在,则创建该文件。
- 'W': 写文件,写的过程中不自动刷新文件内容,当关闭文件时保存所写数据。
- 'A': 以追加方式写文件,过程中不自动刷新文件内容,当关闭文件时保存所写数据。

通常情况下 `fopen` 默认以二进制方式打开文件,可以在 `permission` 后加 'b' 来显式说明。若要以文本的方式打开文件,则给 `permission` 后加 't'。如 'rb' 表示以二进制只读方式打开文件, 'w+t' 表示以文本可读可写方式打开文件。在 UNIX 系统上,文本和二进制方式并无区别,因此不用考虑加 'b' 加 't' 的问题。

`fid` 为 `fopen` 返回的一个文件标识符(file identifier)。`fid` 为 1 表示标准输出,为 2 表示标准错误。若文件打开失败,则 `fid` 为 -1,并且可以返回一个错误消息。下面给出一个打开文件的例子。

【例】 打开文件。

在命令窗输入:

```
>> fid=0;
while fid < 1
    filename=input('Open file: ','s');
    [fid,message] = fopen(filename, 'r');
    if fid == -1
        disp(message)
    end
end
```

命令窗中将显示:

Open file:

如果输入一个不存在的文件 `nofile.txt`:

Open file: nofile.txt

运行结果:

No such file or directory

如果不输入任何文件名:

Open file:

运行结果:

Cannot open file. Existence? Permissions? Memory? ...

如果输入正确的文件名 existfile.txt(假设文件 existfile.txt 存在于当前路径):

Open file: existfile.txt

则命令窗内不出现任何消息。

5.3.2 读写操作

1. 读写二进制文件

fread 函数用于读取二进制文件, 其基本调用格式为

$A = \text{fread}(\text{fid})$

其中 A 为读取数据返回的矩阵。

【例】 读取二进制文件。

创建一个二进制文件 char.bin, 其内容如下:

TEST

R5Y

在命令窗输入:

```
>> fid = fopen('char.bin', 'r');
```

```
A = fread(fid)
```

```
fclose(fid);
```

运行结果:

A =

84 69 83 84 13 10 82 53 89

继续在命令窗输入:

```
>> disp(char(A));
```

运行结果:

TEST

R5Y

$A = \text{fread}(\text{fid}, \text{SIZE})$ 表示读取 fid 所标识的文件, 从当前文件指针所指位置开始的前 SIZE 个字节。打开文件后, 文件指针的位置最初位于文件开始处。其中 SIZE 可以为以下任意一种格式:

- N: 读取前 N 个字节, 返回一个列向量。
- inf: 读至文件尾部。
- [M, N]: 至多读取 M×N 个数据到一个 M×N 的矩阵 A 中。按列顺序将数据读取到矩阵 A 中, 不足的部分补零。N 可以为 inf, M 不可以为 inf。

$A = \text{fread}(\text{fid}, \text{SIZE}, \text{PRECISIOSIZE})$ 表示读取 SIZE 个 PRECISIOSIZE 字长的数据到矩阵 A 中, 其中 SIZE 为可选参数, PRECISIOSIZE 一般为一个表示数据类型的字符串, 默认情况下为 'uint8'。表 5-2 列出了可以作为 PRECISIOSIZE 参数的字符串, 其中不论是 MATLAB 还是相应的 C 或 Fortran 语言的数据类型, 都可以作为 PRECISIOSIZE 参数。

表 5-2 通用精度类型

MATLAB	C 或 Fortran	描述
'schar'	'signed char'	有符号整数, 8 位
'uchar'	'unsigned char'	无符号整数, 8 位
'int8'	'integer*1'	整数, 8 位
'int16'	'integer*2'	整数, 16 位
'int32'	'integer*4'	整数, 32 位
'int64'	'integer*8'	整数, 64 位
'uint8'	'integer*1'	无符号整数, 8 位
'uint16'	'integer*2'	无符号整数, 16 位
'uint32'	'integer*4'	无符号整数, 32 位
'uint64'	'integer*8'	无符号整数, 64 位
'single'	'real*4'	浮点数, 32 位
'float32'	'real*4'	浮点数, 32 位
'float64'	'real*8'	浮点数, 64 位
'double'	'real*8'	浮点数, 64 位
'bitSIZE'	—	有符号整数, SIZE 位($1 \leq \text{SIZE} \leq 64$)
'ubitSIZE'	—	无符号整数, SIZE 位($1 \leq \text{SIZE} \leq 64$)

表 5-3 列出的字符串同样可用, 但是在不同平台上它们的大小会有差异。

表 5-3 与平台有关的精度类型

MATLAB	C 或 Fortran	描述
'char'	'char*1'	字符
'short'	'short'	整数, 16 位
'int'	'int'	整数, 32 位
'long'	'long'	整数, 32 或 64 位
'ushort'	'unsigned short'	无符号整数, 16 位
'uint'	'unsigned int'	无符号整数, 32 位
'ulong'	'unsigned long'	无符号整数, 32 或 64 位
'float'	'float'	浮点数, 32 位

默认情况下 `fread` 返回的是一个 `double` 类型的矩阵, 在 `PRECISIONSIZE` 参数中可以指定返回矩阵的数据类型:

`'source1'=>'source2'`

该命令表示读入的源数据格式为 `source1`, 返回的目的数据格式为 `source2`。如果 `source1` 与 `source2` 相同, 则该命令可以简写为

`*source1`

例如:

`'uint8'=>'uint8'`: 以 8 位整数读入数据, 并将其保存在一个 8 位整数的矩阵中。

`*'uint8'`: 上面格式的简写形式, 等价于 `'uint8'=>'uint8'`。

'bit4=>int8': 以 4 位整数读入数据, 并将其保存在一个 8 位有符号整数矩阵中。每 4 位整数变成一个 8 位整数。

'double=>real*4': 以 double 型读入数据, 并将其保存在一个 32 位浮点数矩阵中。

【例】 以特定数据类型读取二进制文件。

仍以文件 char.txt 为例, 在命令窗输入:

```
>> fid = fopen('char.bin', 'r');  
A = fread(fid, 'uint8=>char')  
fclose(fid);
```

运行结果:

```
A =  
  
TEST  
R5Y
```

fread 函数还可以指定在读取过程中跳跃的字节数, 其调用格式为

```
A = fread(fid, SIZE, PRECISIOSIZE, SKIP)
```

其中 SIZE 为可选参数, SKIP 为跳跃的字节数。

【例】 字节跳跃读取二进制文件。

在命令窗输入:

```
>> fid = fopen('char.bin', 'r');  
A = fread(fid, 'uint8=>char', 1)  
fclose(fid);
```

运行结果:

```
A =  
  
TS  
RY
```

使用 fwrite 函数可将二进制文件写到文件中, 其调用格式如下:

- count = fwrite(fid, A): 将矩阵 A 中的元素写入到 fid 所标识的文件中去。count 表示已经成功写入元素的数目。

- count = fwrite(fid, A, PRECISION): 以指定的数据精度 PRECISION, 将矩阵 A 中的元素写入 fid 所标识的文件中。PRECISION 如表 5-2 所示。

- count = fwrite(fid, A, PRECISION, SKIP): 以指定的数据精度 PRECISION, 指定的跳跃数 SKIP, 将矩阵 A 中的元素写入到 fid 所标识的文件中。当 PRECISION 为 'bitN' 或 'ubitN' 时, SKIP 为跳跃的位数, 其他情况为跳跃的字节数。

2. 读写文本文件

fgetl 函数和 fgets 函数常用于文本文件的读取。

fgetl 函数用于返回下一行文本的字符串且不保留换行符, 其调用格式为

```
tline=fgetl(fid)
```

其中, fid 为文件标识符, 当成功读取文件的下一行时, fgetl 返回该行字符串到 tline; 当 fgetl 读到文件末尾时, 返回 -1; 若读取错误, 则返回一个空字符串, 可以用函数 ferror 获取错

误信息。

【例】 利用 fgetl 函数读取文件 fgetl.m 的内容。

在命令窗输入：

```
>> fid=fopen('fgetl.m');  
while 1  
    tline = fgetl(fid);  
    if ~ischar(tline), break, end  
    disp(tline)  
end  
fclose(fid);
```

运行结果：

```
function tline = fgetl(fid)  
%FGETL Read line from file, discard newline character.  
%   TLINE = FGETL(FID) returns the next line of a file associated with file  
%   identifier FID as a MATLAB string. The line terminator is NOT  
%   included. Use FGETS to get the next line with the line terminator  
%   INCLUDED. If just an end-of-file is encountered, -1 is returned.  
%  
%   If an error occurs while reading from the file, FGETL returns an empty  
%   string. Use FERROR to determine the nature of the error.  
%  
%   MATLAB reads characters using the encoding scheme associated with the  
%   file. See FOPEN for more information.  
%  
%   FGETL is intended for use with files that contain newline characters.  
%   Given a file with no newline characters, FGETL may take a long time to  
%   execute.  
%  
%   Example  
%       fid=fopen('fgetl.m');  
%       while 1  
%           tline = fgetl(fid);  
%           if ~ischar(tline), break, end  
%           disp(tline)  
%       end  
%       fclose(fid);  
%  
%   See also FGETS, FOPEN, FERROR.
```

```
% Copyright 1984-2005 The MathWorks, Inc.
% $Revision: 5.15.4.5 $ $Date: 2005/12/12 23:25:45 $
%

try
    [tline,lt] = fgets(fid);
    tline = tline(1:end-length(lt));
    if isempty(tline)
        tline = "";
    end

catch
    if nargin ~= 1
        error(nargchk(1,1,nargin,'struct'))
    end
    rethrow(lasterror)
end
```

fgets 函数的功能和用法与 fgetl 函数基本相同,不同的是 fgets 函数保留换行符。

3. 读写格式化文本数据

fscanf 函数和 fprintf 函数分别用于数据的格式化读和写。fscanf 的调用格式为

```
[A,count] = fscanf(fid, FORMAT, SIZE)
```

该命令用于将 fid 所标识文件以 FORMAT 格式读入到矩阵 A 中。参数 SIZE 决定读取多少数据。SIZE 可以为 N、inf、[M, N]任一种格式,具体的同 fread 函数中的 SIZE 参数。count 是一个可选参数,表示成功读取的数据个数。FORMAT 的常用格式如下:

- %c: 字符格式;
- %s: 字符串格式;
- %d: 有符号十进制整数;
- %o: 有符号八进制数;
- %x: 有符号十六进制数;
- %u: 无符号十进制整数;
- %e,%f,%g: 浮点数。

关于格式符的更多用法可以参见 C 语言参考手册。

函数 fprintf 可以将数据格式化写入到文件中去,其调用格式为

```
count = fprintf(fid, FORMAT, A, ...)
```

该函数将数据矩阵 A 以 FORMAT 格式写入到 fid 所标识的文件中去。count 表示已经成功写入的字节数。FORMAT 是格式符,这里的用法基本同 fscanf 中的 FORMAT 参数,不过可以在 FORMAT 中加入字符串或者转义字符。常见的转义字符如表 5-4 所示。

表 5-4 转义字符

转义字符	描 述
\n	换行
\t	横向跳到下一制表位置
\b	退格
\r	回车
\f	走纸换页
\\	反斜杠 "\"
\"	单引号
%%	百分号

下面通过一个例子来说明 fprintf 函数的用法。

【例】 fprintf 函数的用法。

创建一个脚本文件，输入如下代码并运行：

```
x = 0:.2:1;
y = [x; exp(x)];
fid = fopen('exp.txt', 'wt');
fprintf(fid, 'x:%6.2f \t y=%12.8f\n', y);
fclose(fid)
```

在命令窗输入 type exp.txt 命令查看结果：

```
>> type exp.txt
x: 0.00   y= 1.00000000
x: 0.20   y= 1.22140276
x: 0.40   y= 1.49182470
x: 0.60   y= 1.82211880
x: 0.80   y= 2.22554093
x: 1.00   y= 2.71828183
```

4. 文件定位

前面提到过文件指针的概念，这里将详细介绍文件指针是如何在文件内部定位的。当打开文件时，操作系统通过一个文件指针来指示当前文件的位置。这个指针将决定对文件进行操作时，从文件的何处开始进行读或写操作。MATLAB 提供了函数 fseek、ftell，用于实现对文件指针的操作。

fseek 函数用于设定文件指针的位置，其调用格式为

```
status = fseek(fid, OFFSET, ORIGIN)
```

该函数将 fid 所标识的文件的文件指针，移到以 ORIGIN 为基点，以 OFFSET 为偏移量的位置。若操作成功，则 status 为 0，否则 status 为 -1。

参数 ORIGIN 为文件指针移动的基点，可以为以下字符串：

- 'bof': -1, 文件起始位置;
- 'cof': 0, 指针当前位置;

- 'eof': 1, 文件末尾。

OFFSET 为文件的偏移地址, 说明如下:

- OFFSET > 0: 向文件末尾方向移动文件指针;
- OFFSET = 0: 偏移地址为 0, 将文件指针移动到 ORIGIN 所指位置;
- OFFSET < 0: 向文件起始位置移动文件指针。

例如, `fseek(fid, 8, 'bof')` 表示从文件的起始位置 'bof' 处向文件末尾方向移动 8 个字符, 结果如图 5-2 所示。`fseek(fid, 0, 'eof')` 表示将文件指针移动到文件末端。

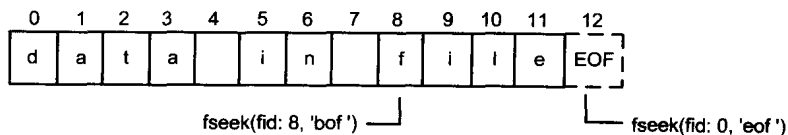


图 5-2 文件定位示例

`ftell` 函数用于获取当前文件指针的位置, 其调用格式为

`position = ftell(fid)`

如果 `position` 为 -1, 则表示查询当前文件指针位置不成功, 可以用 `ferror` 获取错误信息。

5.3.3 关闭文件

`fclose` 函数用于关闭文件, 这是执行文件操作后必须执行的操作, 与 `fopen` 函数配套使用。前面的很多例子中已经使用过 `fclose` 函数, 这里主要介绍 `fclose` 函数的其他调用格式:

- `status = fclose(fid)`: 当函数返回 0 时表示成功关闭文件, 返回 -1 时则表示关闭过程中发生错误。如果 `fid` 不标识一个已经打开的文件, 或者为 0(标准输入)、1(标准输出)、2(标准错误), 则 `fclose` 抛出一个异常。

- `status = fclose('all')`: 关闭除 `fid` 为 0、1、2 以外的所有文件。成功时返回 0, 否则返回 -1。

【例】 打开文件、文件定位、读写二进制文件、关闭文件。

创建文件 `test1.dat`, 其内容为

```
Pine
apples and tea.
Orangutans and monkeys,
Dragonflies or fleas.
```

创建文件 `test2.dat`, 其内容为

```
Oranges and lemons,
```

要将 `test1.dat` 中第二行以后的数据写入到 `test2.dat` 文件的尾端, 在命令窗输入:

```
>> fid1 = fopen('test1.dat', 'r');
fseek(fid1, 4, 'bof');
A = fread(fid1, inf, 'uint32');
fid2 = fopen('test2.dat', 'r+');
fseek(fid2, 0, 'eof');
```

```
fwrite(fid2, A, 'uint32');
```

```
fclose('all');
```

要显示改动后 test2.dat 文件的内容, 在命令窗输入:

```
>> type test2.dat
```

运行结果:

```
Oranges and lemons,
```

```
apples and tea.
```

```
Orangutans and monkeys,
```

```
Dragonflies or fleas.
```

5.3.4 更多文件 I/O 函数

除以上小节介绍的 MATLAB 底层文件 I/O 操作函数之外, 在此基础上, MATLAB 针对不同的文件格式, 提供了更为丰富的文件 I/O 函数。这些函数功能更为强大, 使用也更方便。表 5-5 列出了 MATLAB 支持的可读写数据文件格式及相关函数。这些函数的具体使用方法较为简单, 此处不再赘述。

表 5-5 MATLAB 支持的可读写数据文件格式

文件格式	文件内容	扩展名	读写函数
MATLAB 格式	保存在 MATLAB 工作区间	.mat	load、save
文本	文本	任意	textscan
	文本	任意	textread
	纯文本	任意	dlmread、dlmwrite
	逗号分隔的数字	.csv	csvread、csvwrite
可扩充标记语言	XML 格式文件	.xml	xmlread、xmlwrite
音频	NeXT/SUN 声音	.au	auread、auwrite
	Microsoft WAVE 声音	.wav	wavread、wavwrite
音频视频交错格式	音频/视频	.avi	aviread
科学数据	常用数据格式数据	.cdf	cdfread、cdfwrite
	灵活图像传输系统数据	.fits	fitsread
	层次数据格式数据	.hdf	hdfread
电子数据表	Excel 工作表	.xls	xlsread、xlswrite
	Lotus 123 工作表	.wk1	wk1read、wk1write
图形	TIFF 图像	.tiff	imread、imwrite
	PNG 图像	.png	imread、imwrite
	HDF 图像	.hdf	imread、imwrite
	BMP 图像	.bmp	imread、imwrite
	JPEG 图像	.jpeg	imread、imwrite
	GIF 图像	.gif	imread、imwrite
	PCX 图像	.pcx	imread、imwrite
	XWD 图像	.xwd	imread、imwrite
	Cursor 图像	.cur	imread、imwrite
	Icon 图像	.ico	imread、imwrite

5.4 程序调试与优化

对于规模较大的程序，调试和优化是必不可少的环节。本节将结合实例介绍程序调试和优化的概念和方法。

5.4.1 程序的调试

编写程序的过程中不免出现错误，调试错误的过程与编写程序的过程同样重要，因此掌握一定的调试技巧，对于提高编程效率是大有益处的。

MATLAB 程序的错误主要有两类：一类是语法错误，比如函数名拼写错误、调用参数不匹配、标点符号丢失、关键字不完整等，一般这类错误在编译期间即可被发现。MATLAB 会给出错误警告的内容及相应行。根据错误信息就可以很容易地排除和修正这类错误。另一类错误是运行时错误。这类错误往往是由于逻辑错误而引起的，表现在程序运行结果与预想结果不一致，但编译器却无法给出错误所在。较语法错误来说，运行时错误更具隐蔽性，调试难度也较大。本小节主要针对运行时错误来介绍几种调试的方法。

1. 以命令行模式进行 M 文件调试

用命令行模式调试 M 文件，不仅快速便捷、功能强大，而且具有较好的通用性，适合于各种不同的平台。表 5-6 列出了 M 文件的常用调试函数。

表 5-6 常用调试函数

命 令	功 能 描 述
dbstop	设置断点
dbclear	清除断点
dbcont	继续执行调试
dbmex	调试 mex 文件(UNIX 系统下)
dbstack	列出函数调用关系
dbstatus	列出所有断点情况
dbstep	步进执行
dbtype	列出带有行号的 M 文件
dbquit	退出调试模式
dbup	改变当前工作区间
dbdown	改变当前工作区间(同 dbup 相反的过程)

下面对几个常用的调试函数进行具体的介绍。

(1) dbstop 函数用于在 M 文件中设置断点，其调用格式如下：

- dbstop in MFILE at LINENO: 在 M 文件 MFILE 的第 LINENO 行处设置断点。这里的 LINENO 处若为注释，则断点设为该段注释结束后的下一行程序段。

- dbstop in MFILE at LINENO@N: 设置程序在执行到 MFILE 的第 LINENO 行，第 N

个匿名函数处停止。

- **dbstop in MFILE at LINENO@N if EXPRESSION:** 如果 EXPRESSION 为真, 则设置 dbstop in MFILE at LINENO@N。

- **dbstop if error:** 当执行 M 文件遇到错误时, 停止运行程序到产生错误的行, 并使程序进入调试状态。这里的错误不包括 try...catch 语句中检测到的错误, 用户不能再运行产生错误行以后的程序。

- **dbstop if catch error:** 基本同上, 不同的是这里包括 try...catch 语句中检测到的错误。

- **dbstop if error identifier:** 当程序遇到信息为 identifier 的错误时, 程序停止在产生错误的行, 其他同命令 dbstop if error。

- **dbstop if warning:** 当执行 M 文件遇到警告时, 程序暂停在产生错误的行并进入调试状态。执行此命令后可以用 dbcont 或 dbstep 恢复执行程序。

- **dbstop if naninf 或 dbstop if infnan:** 当执行 M 文件遇到无穷值或者非数值时, 程序暂停并进入调试状态。

(2) **dbclear** 函数用于清除 M 文件中的断点, 其调用格式如下:

- **dbclear all:** 清除所有 M 文件中的断点。

- **dbclear in MFILE:** 清除 MFILE 文件中的所有断点。

- **dbclear in MFILE at LINENO:** 清除 MFILE 文件中第 LINENO 行的断点。

- **dbclear in MFILE at LINENO@N:** 清除 MFILE 文件第 LINENO 行, 第 N 个匿名函数中设置的断点。

- **dbclear in MFILE at SUBFUN:** 清除在 MFILE 文件子函数 SUBFUN 中的所有断点。

- **dbclear if error:** 清除由命令 dbstop if error 或 dbstop if error identifier 设置的断点。

- **dbclear if caught error:** 清除由命令 dbstop if caught error 或 dbstop if caught error identifier 设置的断点。

- **dbclear if warning:** 清除由命令 dbstop if warning 或 dbstop if warning identifier 设置的断点。

- **dbclear if naninf 或 dbclear if infnan:** 清除由命令 dbstop if naninf 或 dbstop if infnan 设置的断点。

(3) **dbcont** 函数用于从断点处恢复继续执行程序, 直到遇到下一个断点或者错误, 或者程序执行完毕返回基本工作区间, 其调用格式为 dbcont。

(4) **dbstep** 函数用于从当前断点处恢复, 步进地执行程序, 其调用格式如下:

- **dbstep:** 执行当前断点处的下一行程序。dbstep 会忽略当前行所调用函数中的断点。

- **dbstep NLINEs:** 该命令将执行到当前断点所在行的下 NLINEs 行程序处, 若中间程序行有断点, 则执行到断点处。

- **dbstep in:** 执行下一行程序, 若此行有函数的调用, 则进入该函数内部, 否则同 dbstep。

- **dbstep out:** 该命令运行完函数的剩余部分, 程序停留在离开函数处。

(5) **dbstatus** 函数用于查看所有断点的情况, 其调用格式如下:

- **dbstatus:** 列出所有文件中的断点。

- **dbstatus MFILE:** 列出 MFILE 文件中所有的断点信息。

- `s = dbstatus`: 该表达式以一个 $M \times 1$ 的结构形式返回所有断点信息, 各字段信息如下:
 - `name`: 文件名。
 - `file`: 包含有断点的文件的路径。
 - `line`: 断点行号向量。
 - `anonymous`: 在 `line` 字段中的行号向量对应的匿名函数序列向量。如某断点行有第 2 个匿名函数, 则其对应的匿名函数序列为 2。
 - `expression`: 与 `line` 中行对应的断点条件表达式字符串向量。
 - `cond`: 条件字符串, 如 'error', 'caught error', 'warning' 或 'naninf'。
 - `identifier`: 当 `cond` 是 'error', 'caught error' 或者 'warning' 时, 该字段为信息类别字符串向量。

(6) `dbtype` 函数用于显示带有行号的 M 文件内容, 其调用格式如下:

- `dbtype MFILE`: 显示指定文件 `MFILE` 的内容并带有行号信息。
- `dbtype MFILE start:end`: 显示指定文件 `MFILE` 从 `start` 到 `end` 行之间的内容并带有行号信息。

(7) `dbstack` 函数用于显示当前断点所在的 M 文件名和产生断点的行号, 并且以执行顺序列出, 其调用格式如下:

- `dbstack`: 显示函数调用堆栈中的文件名及行号信息。
- `dbstack(N)`: 省略堆栈前 `N` 个函数的调用信息。
- `dbstack('-completenames')`: 显示堆栈中函数文件的路径、文件名与行号。
- `[ST, I] = dbstack`: 执行此命令后, 其中 `ST` 以一个 $M \times 1$ 的结构体形式返回堆栈信息。

各字段信息如下:

- `file`: 出现函数的文件名, 如果没有文件, 则这个字段为空。
- `name`: 文件中的函数名。
- `line`: 函数行号。

`I` 为返回的当前工作区间索引。

【例】 `dbstack` 的使用。

在调式模式下, 如果输入:

```
K>> dbstack
```

运行结果:

```
In collatzplot at 6
```

```
> In test at 1
```

如果输入:

```
K>> dbstack(1)
```

运行结果:

```
> In test at 1
```

如果输入:

```
K>> dbstack('-completenames')
```

运行结果:

```
In F:\Matlab\work\collatzplot.m>collatzplot at 6
```

```
> In F:\Matlab\work\test.m>test at 1
```

如果输入:

```
K>> [ST,I]=dbstack
```

运行结果:

```
ST =
```

```
file: 'collatz.m'
```

```
name: 'collatz'
```

```
line: 6
```

```
I = 1
```

(8) dbquit 函数用于退出调试模式, 其调用格式如下:

- dbquit: 退出调试模式, 所有断点仍有效。如果调试文件 1 并跳转到文件 2, 执行该命令将终止文件 1 和文件 2 的调试状态。如果在调试文件 1 的同时也在调试文件 2, 则运行一次 dbquit 只能将其中之一退出调试模式。

- dbquit('all')或 dbquit all: 一次性终止所有文件的调试状态。

2. 以 GUI 方式调试 M 文件

用户可以选择用 GUI 方式进行 M 文件的调试, 这种方式更加直观和方便。新建一个 M 文件, 在编译器主菜单 Debug 选项的下拉菜单中可以看到如图 5-3 所示的各种调试命令。

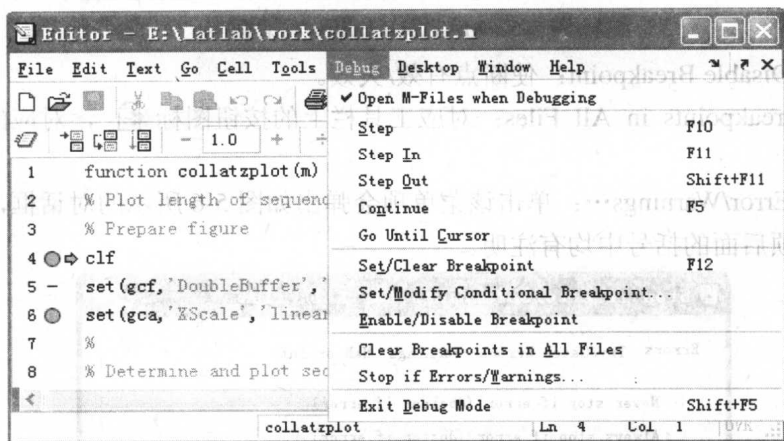



图 5-3 GUI 方式调试


主菜单的部分调试命令在调试工具栏中也有相对应的图标, 如图 5-4 所示。






图 5-4 调试工具栏

下面将对这些命令作简要的介绍。

- Step: 单步执行。对应工具栏上的按钮图标 , 快捷键为 F10, 对应的调试命令为 dbstep。

- Step in: 单步执行, 遇到函数则跳转到函数内部。对应工具栏上的按钮图标 , 快捷键为 F11, 对应的调试命令为 dbstep in。

- **Step out:** 跳出函数内部。对应工具栏上的按钮图标，快捷键为 Shift+F11，对应的调试命令为 `dbstep out`。
- **Continue:** 继续执行。对应工具栏上的按钮图标，快捷键为 F5，对应的调试命令为 `dbcont`。
- **Go Until Cursor:** 运行程序直到光标所在的行。
- **Set/Clear Breakpoint:** 设置或清除断点。对应工具栏上的按钮图标，快捷键为 F12，对应的调试命令为 `dbstop/dbclear`。
- **Set/Modify Conditional Breakpoint...**: 设置或修改当前光标所在行的条件断点。单击此命令会弹出如图 5-5 所示的对话框，在文字框中可输入条件表达式。同调试命令 `dbstop in MFILE at LINENO if EXPRESSION`，其中 LINENO 即光标所在行，EXPRESSION 即条件表达式。

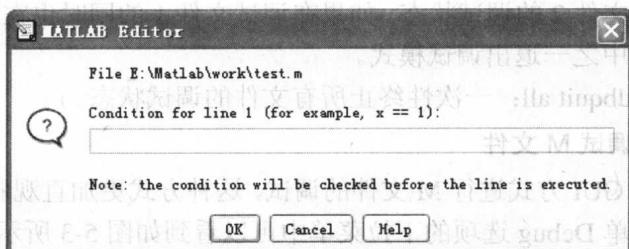



图 5-5 MATLAB Editor 对话框

- **Enable/Disable Breakpoint:** 使断点有效/失效。
- **Clear Breakpoints in All Files:** 对应工具栏上的按钮图标，对应的调试命令为 `dbclear all`。
- **Stop if Error/Warnings...**: 单击该菜单项会弹出如图 5-6 所示的对话框，相应的调试命令在相应选项后面的括号中均有注明。

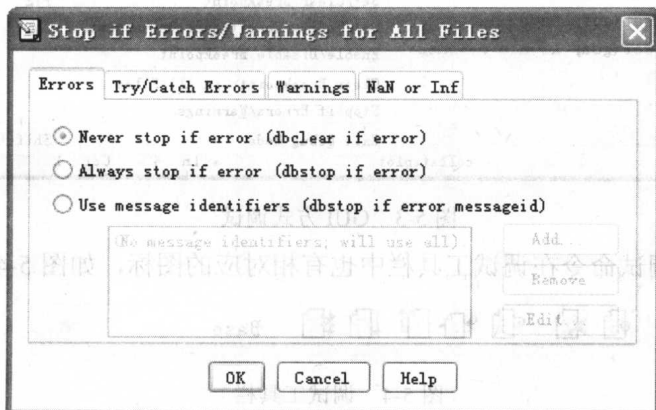



图 5-6 Stop if Error/Warnings for All Files 对话框

- **Exit Debug Mode:** 对应工具栏上的按钮图标，快捷键为 Shift+F5，对应的调试命令为 `dbquit`。
- 在工具栏的左端还有一个下拉列表框 `Stack: Base`。当程序处于调试状态时该框被激活。这个下拉列表框中显示了函数的调用关系，对应于调试命令中的 `dbstack`。

3. 调试辅助函数

前面介绍了通过命令调试程序以及通过 GUI 调试程序的方法。另外 MATLAB 还提供了一些调试辅助函数，可以帮助用户更快地发现问题的所在。调试辅助函数的功能描述如表 5-7 所示。

表 5-7 调试辅助函数


函 数	功 能
echo	当执行 M 文件时，在命令窗中显示其源代码
disp	显示变量值或文本
sprintf, fprintf	显示不同类型的格式化数据
whos	列出工作区间中的所有变量
size	显示矩阵的维数
keyboard	中断程序执行，允许用户键盘输入
return	从一个 keyboard 中断中恢复出来，继续执行程序
warning	显示指定的警告信息
lasterror	返回最近一个错误的信息及其相关信息
lastwarn	返回最近一个警告信息

5.4.2 程序的优化

本小节将介绍如何利用优化工具分析和改进程序的性能，以及如何编写性能较高的 MATLAB 程序语言。

1. 使用 M-Lint Code Check Report 工具

M-Lint Code Check Report(代码检查报告)工具可用于检查代码中可能存在的错误和问题，并且会向用户提出改进代码的建议。

在当前路径窗口中单击按钮，将打开 M-Lint Code Check Report 窗口，窗口中显示的是当前路径下所有 M 文件的代码分析结果。单击文件名超链接可以打开文件，单击文件行号超链接可以打开文件并定位到检查存在问题的代码行，如图 5-7 所示。

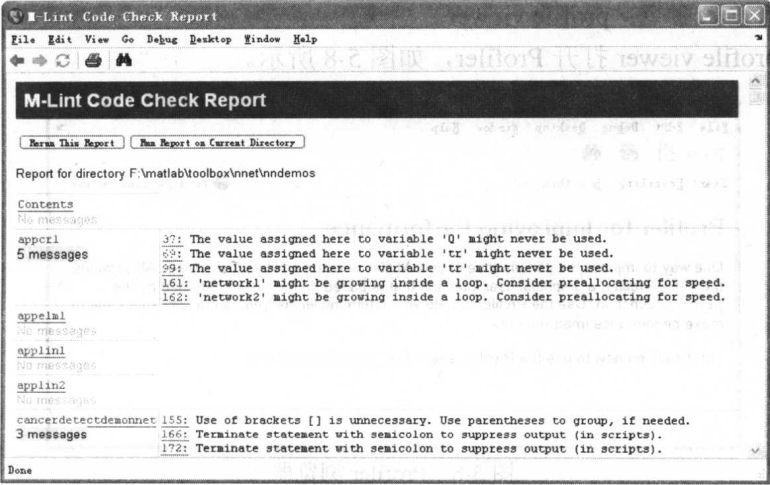


图 5-7 M-Lint Code Check Report 窗口

如果对文件进行了改动,保存改动后单击 **Rerun This Report**,可以更新显示改动后的分析报告。

下面给出了获取 M-Lint 信息的其他方法,通过这些方法同样可以得到 M-Lint 信息,只是信息的形式不同:

- 在 M 文件编辑/调试器界面的主菜单中选择 **Tools > M-Lint > Show M-Lint Report** 来访问 M-Lint 代码分析报告。

- 使用 `mlint` 函数分析指定文件并在命令窗内显示信息,或使用 `mlintrpt` 运行 `mlint` 并以 Web 浏览器方式显示 M-Lint 代码分析报告。

- 使用自动 M-Lint 代码分析校正,可以在编写代码的过程中不停地对代码进行检查。

2. 使用 Profiler 进行优化

MATLAB 提供的 M 文件 Profiler 可以用于实现程序性能的优化。Profiler 是一个基于 profiler 函数返回结果的 GUI。Profiler 可以帮助用户决定如何修改代码以提高程序的性能。

Profiling 是一种测量程序耗费时间的方法。使用 MATLAB Profiler 可以确定代码中最耗费时间的函数,从而决定是否要调用这些函数或者尽可能地减少调用。这在决定一个特定函数调用次数是否合理时十分有用。由于程序通常有很多层,而代码不能够明确地调用最耗时间的函数。程序代码中的函数可能调用了其他耗费时间的函数,因此决定调用哪个函数非常重要。

通过 profiling 可以分析程序性能,并且可以对程序进行以下优化:


- 避免因疏忽带来的不必要的计算。

- 改变算法以避免耗费大量时间的函数。

- 存储结果,避免下次调用时重复计算。

用户可以通过以下方法中的任意一种来打开 Profiler:

- 在 MATLAB 桌面主菜单上选择 **Desktop > Profiler**。

- 在 MATLAB 桌面工具栏上单击 Profiler 按钮 。

- 对于一个 MATLAB 编辑/调试器中的文件,选择 **Tools > Open Profiler**。

- 选中历史命令窗口中的一条或多条语句,在右键菜单上选择 **Profile Code**。

- 在命令窗输入命令: `profile viewer`。

使用命令 `profile viewer` 打开 Profiler,如图 5-8 所示。

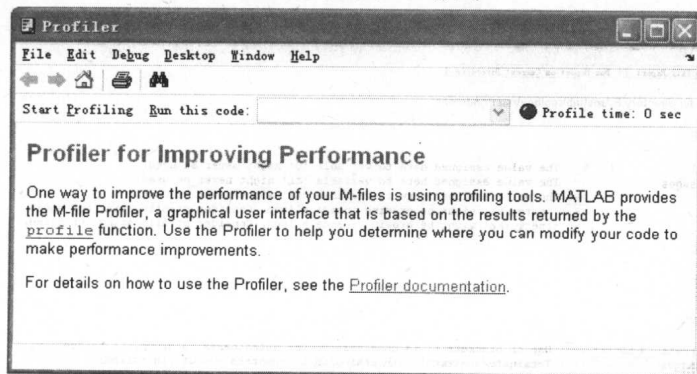


图 5-8 Profiler 浏览器

接下来可以按照以下步骤运行 profiling:

- (1) 在 Profiler 的 Run this code 字段输入将要运行的语句或文件名。
- (2) 单击 Start Profiling 或者在语句输入完毕时按回车键, 启动 profiling。

【例】 使用 Profiler。

打开 Profiler, 在 Run this code 字段输入如下语句:

```
[t,y] = ode23('lotka',[0 100],[20;20])
```

单击 Start Profiling, 运行结果如图 5-9 所示。

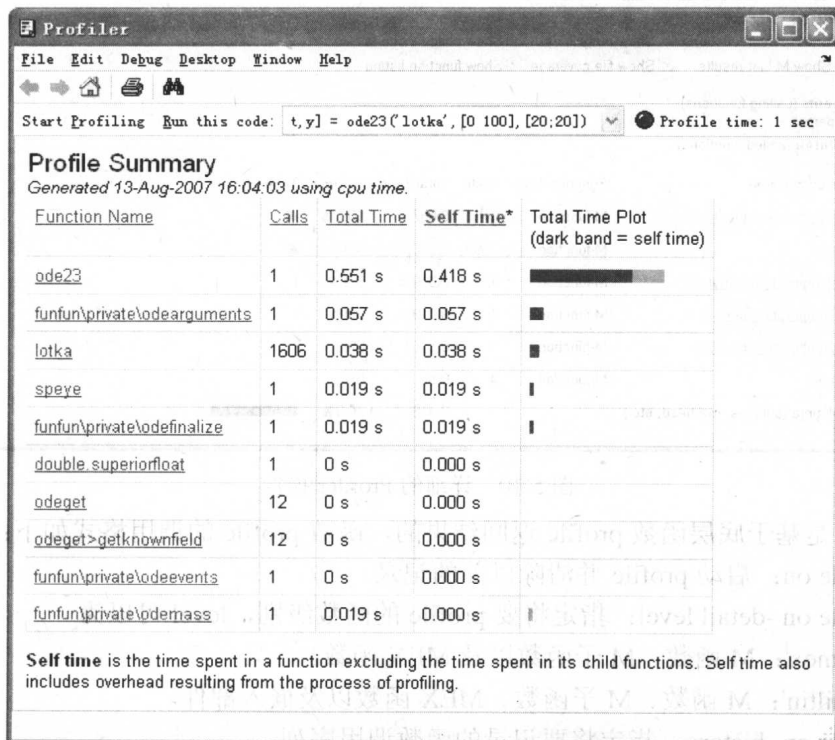


图 5-9 使用 Profiler 分析代码

图 5-9 所示的 Profile Summary 报告中显示了所有函数执行统计以及每个被调用函数的统计:

- **Function Name:** 调用到的函数或子函数列表。第一次显示时, 这些函数将按它们耗费时间的多少从大到小排列。单击 Function Name 超链接后, 可以将函数按字母表顺序排列。
- **Calls:** 函数被调用的次数。单击 Calls 超链接后则按它们被调用次数的多少从大到小排列。
- **Total Time:** 每个函数总共耗费的时间, 包含所有被调用的子函数。单击 Total Time 超链接后可以按它们耗费时间的多少从大到小排列。
- **Self Time:** 每个函数总共耗费的时间, 不包含所有被调用的子函数。单击 Self Time 超链接后可以按它们耗费时间的多少从大到小排列。
- **Total Time Plot:** 图形显示 Self Time 和 Total Time。

单击 Function Name 列下的函数名超链接可以显示更为详细的 Profiler 报告，如图 5-10 所示。

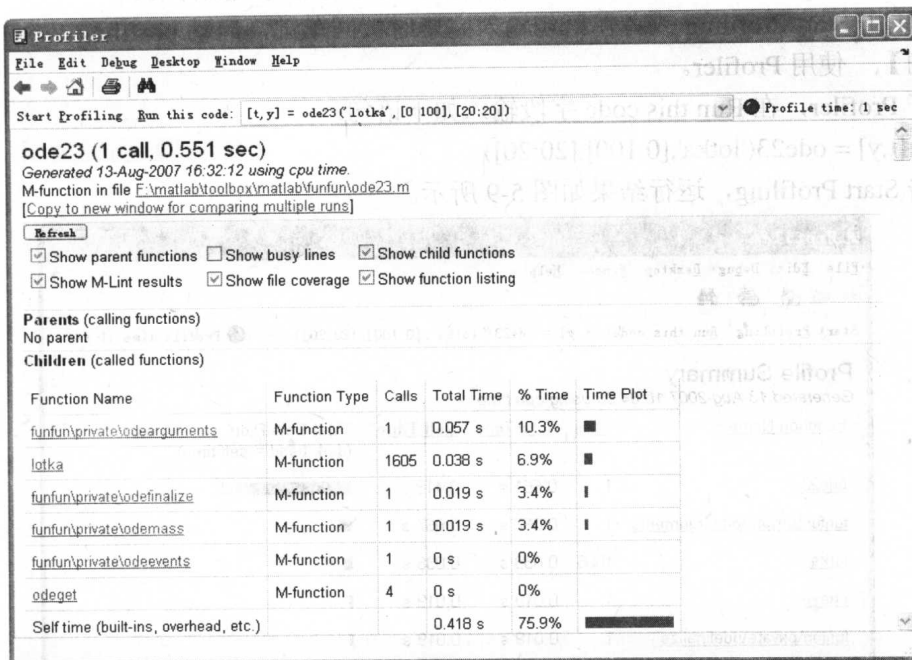


图 5-10 详细的 Profiler 报告

Profiler 是基于底层函数 `profile` 返回结果的，函数 `profile` 的调用格式如下：

- `profile on`: 启动 `profile` 并清除以前的记录。
- `profile on -detail level`: 指定将要 `profile` 的函数级别，`level` 可以为
 - `'mmex'`: M 函数、M 子函数以及 MEX 函数。
 - `'builtin'`: M 函数、M 子函数、MEX 函数以及嵌入部件。
- `profile on -history`: 指定将要记录的函数调用序列。
- `profile off`: 终止 `profile`。
- `profile resume`: 重新启动 `profile`，不清除以前的记录。
- `profile clear`: 清除记录。
- `profile viewer`: 打开 Profiler。
- `s = profile('status')`: 显示包含当前 `profile` 状态的结构。
- `stats = profile('info')`: 终止 `profile` 并显示包含 `profile` 结构的结果。

【例】使用 `profile` 函数。

在命令窗输入：

```
>> profile on
[t,y] = ode23('lotka',[0 100],[20;20]);
profile viewer
profile resume
profile off
```

运行结果如图 5-11 所示。

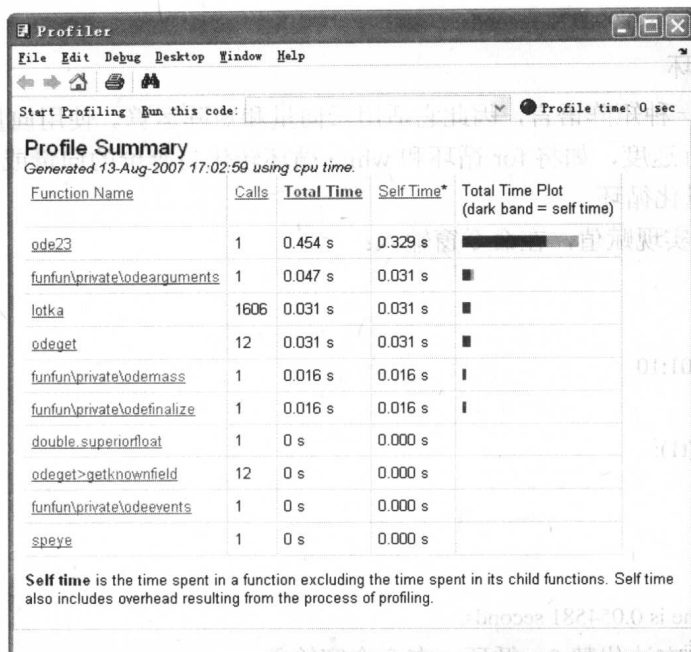


图 5-11 使用 profile 函数的分析结果

继续在命令窗输入：

```
>> p = profile('info')
```

运行结果：

```
p =
```

```
FunctionTable: [10x1 struct]
```

```
FunctionHistory: [2x0 double]
```

```
ClockPrecision: 1.0000e-007
```

```
ClockSpeed: 1.6500e+009
```

```
Name: 'MATLAB'
```

另外，利用 tic 和 toc 函数也可以查看一段程序耗费的时间，其调用格式如下：

```
tic
```

```
程序语句
```

```
toc
```

【例】 使用 tic 函数和 toc 函数。

在命令窗输入：

```
>> tic
```

```
for k = 1:100
```

```
a{k} = magic(k);
```

```
end
```

```
toc
```

运行结果:

Elapsed time is 0.098772 seconds.

3. 向量化循环

MATLAB 是一种矩阵语言, 因此它适用于向量和矩阵运算。使用向量化算法可以加速 M 文件代码的执行速度, 如将 for 循环和 while 循环转化为等价的向量或矩阵运算。

【例】 向量化循环。

使用 for 循环实现赋值, 在命令窗输入:

```
>> tic
i = 0;
for t1 = 0:.01:10
    i = i + 1;
    y1(i) = sin(t1);
end
toc
```

运行结果:

Elapsed time is 0.054581 seconds.

而使用向量化方法代替 for 循环, 在命令窗输入:

```
>> tic
t2 = 0:.01:10;
y2 = sin(t2);
toc
```

运行结果:

Elapsed time is 0.027448 seconds.

通过这个例子可以看出, 向量化方法的运行速度会大大提高。

repmat 函数对于实现向量化非常有用, 该函数的用法已在第 2 章做过介绍。利用 repmat 函数复制并平铺矩阵可以大大提高运算速度。

【例】 使用 repmat 函数。

```
>> A = [1 2 3; 4 5 6];
>> B = repmat(A,2,4)
```

运行结果:

```
B =
     1     2     3     1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6     4     5     6
     1     2     3     1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6     4     5     6
```

向量化方法的常用函数如表 5-8 所示。

表 5-8 向量化方法的常用函数

函 数	描 述
all	检测所有元素是否非零
any	检测所有非零元素
cumsum	求累积和
diff	求微分和近似导数
find	寻找非零元素的索引
ind2sub	将线性索引转化为下标
ipermute	permute 的反变换
logical	将数值转化为逻辑类型
ndgrid	生成用于多维函数和插值的数组
permute	重整多维数组的维数
prod	求数组元素的乘积
repmat	复制并平铺数组
reshape	改变数组的形状
shiftdim	数组维移位
sort	按照升序或降序对数组元素排序
squeeze	移除数组的多余维
sub2ind	将下标转化为线性索引
sum	求数组元素的和

4. 预分配数组

for 循环和 while 循环在每次循环时将改变数据的结构，这会严重影响程序的性能和内存使用。重复地调整数组的结构会使 MATLAB 花费额外的时间，因此在循环之前预分配数组可以实现程序性能的优化。预分配数组的函数有 zeros、cell 等，它们的调用格式基本相同。

【例】 预分配数组。

不进行预分配，在命令窗输入：

```
>> clear all
>> tic
for k = 1:50000
    a1(k) = k;
end
toc
```

运行结果：

Elapsed time is 11.453043 seconds.

进行预分配，在命令窗输入：

```
>> clear all
>> a2 = zeros(1,50000);
```

```
tic
for k = 1:50000
    a2(k) = k;
end
toc
```

运行结果:

Elapsed time is 0.042149 seconds.

可见, 预分配数组之后程序运行速度要高很多。

5. 使用实数函数

当运行数据为实数时, 尽量使用以下专用于实数运算的函数:

- `reallog`: 实数对数运算。
- `realpow`: 实数幂运算。
- `realsqrt`: 实数开方运算。

6. 使用适当的逻辑运算符

在进行逻辑与、逻辑或运算时, 可以使用以下逻辑运算符中的任意一种:

- `&`、`|`: 数组元素和元素之间的逻辑与、逻辑或。
- `&&`、`||`: 标量之间的短路逻辑与、逻辑或。

在 `if` 和 `while` 语句中, 建议使用短路运算符 `&&` 实现逻辑与, 使用 `||` 实现逻辑或。这是因为这些运算符通常只执行一部分判断, 如果满足条件, 则不需要执行整个逻辑表达式。例如, 对于条件语句

```
if (a >= 2) && (a <= 5) && (a ~= 3)
```

当 `a` 小于 2 时, MATLAB 只根据表达式的第一个参数就能判断条件为假。

第6章 符号计算功能

MATLAB 提供了功能强大的符号数学工具箱，使用户能够在 MATLAB 数值环境下进行符号运算。MATLAB 的符号数学工具箱是建立在 Maple 基础上的，当进行 MATLAB 符号运算时，由 Maple 软件计算并将结果返回给 MATLAB。

MATLAB 的符号数学工具箱的主要功能包括符号表达式的创建、符号矩阵的运算、符号表达式的化简和替换、符号线性代数、变精度算法、符号代数方程、符号微分方程、符号积分变换、符号特殊函数、符号函数绘图等。

6.1 符号对象的创建与使用

符号数学工具箱定义了一个称为符号对象的新数据类型。符号对象是一个用字符串来代表符号的存储数据结构。符号数学工具箱利用符号对象来表示变量、表达式和矩阵。事实上，符号对象的计算主要是由 Maple 软件来执行的。

本节将介绍如何利用符号数学工具箱来创建符号变量、符号表达式以及符号数学函数，此外还将介绍符号对象的基本运算操作。

6.1.1 创建符号变量和表达式

MATLAB 的符号数学工具箱提供了用于创建符号变量和表达式的两个基本函数，即 `sym` 和 `syms`。

`sym` 函数的常用调用格式为 `S = sym(A)`，其作用是创建一个由 `A` 表示的“`sym`”类的对象 `S`。如果输入参量是字符串，则生成一个符号类型的数值或变量；如果输入参量是数值标量或数值矩阵，则生成一个符号类型的数值。

【例】 创建符号类型数值。

在命令窗输入：

```
>> a = 3;  
>> b = '3';  
>> s1 = sym(a);  
>> s2 = sym(b);  
>> s1 == s2  
class_ = class(s1)
```

运行结果:

```
ans =  
1  
class_ =  
sym
```

【例】 创建符号类型变量。

在命令窗输入:

```
>> x = sym('x')  
a = sym('alpha')
```

运行结果:

```
x =  
x  
a =  
alpha
```

【例】 创建符号类型矩阵。

在命令窗输入:

```
>> A = magic(3);  
>> sym(A)
```

运行结果:

```
ans =  
[ 8, 1, 6]  
[ 3, 5, 7]  
[ 4, 9, 2]
```

`syms` 函数用于创建符号表达式时的变量声明,这也是最常用的创建符号表达式的方法。

该函数的常用调用格式为 `syms a1 a2...`。

【例】 创建符号表达式。

在命令窗输入:

```
>> syms a b  
>> f = a^2 + 2*b
```

运行结果:

```
f =  
a^2+2*b
```

其中,表达式的创建语句定义了 `f` 为符号变量,无需再定义 `f`。否则,如果继续在命令窗输入:

```
>> syms f  
>> f
```

运行结果会将原本创建好的表达式覆盖:

```
f =  
f
```

另外, 符号表达式也可以通过 `sym` 函数直接创建。

【例】 创建符号表达式。

在命令窗输入:

```
>> f1 = sym(a^2 + 2*b)
```

运行结果:

```
f1 =  
a^2+2*b
```

通过这种方法创建的符号表达式与上例中通过 `syms` 创建的符号表达式的结果是相同的。但是, 如果采用以字符串作为输入参量的方法, 得到的将是另一个结果。

【例】 创建符号表达式。

在命令窗输入:

```
>> f2 = sym('a^2 + 2*b')
```

运行结果:

```
f2 =  
a^2 + 2*b
```

应当注意, `f2` 表达式的“+”号前后各存在一个空字符, 与前面两种方法创建的表达式略有差别。

对于一个已有的表达式, 利用 `findsym` 函数可以寻找该表达式中的符号变量, 例如:

【例】 寻找符号变量。

在命令窗输入:

```
>> syms a b  
>> f = a^2 + 2*b;  
>> findsym(f)
```

运行结果:

```
ans =  
a, b
```

使用 `pretty` 函数可以使符号表达式的显示形式更为美观。

【例】 按照书写习惯显示表达式。

在命令窗输入:

```
>> syms x y  
>> y = 3*x^3 - 2*x^2 + x + 1  
pretty(y)
```

运行结果:

```
y =  
3*x^3-2*x^2+x+1  
  
3      2  
3 x  - 2 x  + x + 1
```

6.1.2 创建符号数学函数

符号数学函数的输入参量中含有符号变量，创建符号数学函数的方法与创建一般函数的方法类似。符号数学函数在一些场合有着特殊的用途。

下面以一个 sinc 符号数学函数为例来说明其创建方法。创建一个 M 文件，输入如下代码并保存为 sincsym.m:

```
function z = sincsym(x)
if isequal(x,sym(0))
    z = sym(1);
else
    z = sin(x)/x;
end
```

在命令窗输入:

```
>> x = sym(2);
y = sincsym(x)
x1 = sym(0);
y1 = sincsym(x1)
z = class(y1)
```

运行结果:

```
y =
    1/2*sin(2)
y1 =
    1
z =
    sym
```

6.2 数学计算功能

MATLAB 的符号数学工具箱提供了重要的数学计算功能。本节将具体介绍如何利用符号数学工具箱实现符号微积分、函数求极限、级数求和以及泰勒级数展开，这些功能在高等数学中占有重要的地位。

6.2.1 符号微积分

微积分运算是整个高等数学的基础，本小节主要介绍如何利用符号数学工具箱对符号表达式进行微积分运算。

1. 微分

在符号数学工具箱中，表达式的微分由函数 diff 实现，其调用格式如下:

- `diff(S)`: 求符号表达式 S 对于默认自变量的微分。
- `diff(S,y)`: 求符号表达式 S 对于自变量 y 的微分。
- `diff(S,n)`: 求符号表达式 S 对于默认自变量的 n 次微分。
- `diff(S,y,n)` 或 `diff(S,n,y)`: 求符号表达式 S 对于自变量 y 的 n 次微分。

符号微分的默认自变量为 x , 如果表达式中不存在 x , 则以字母表中最接近 x 的字母作为微分的默认自变量。不论一个符号变量是否与表达式有关, 符号微分都可以求表达式对于该符号变量的微分。

【例】 符号微分。

在命令窗输入:

```
>> syms x y t
>> z = x^2 + 2*y^3;
>> dzdx = diff(z)
dzdt = diff(z,t)
d2zdx2 = diff(z,2,x)
d3zdy3 = diff(z,y,3)
```

运行结果:

```
dzdx =
      2*x
dzdt =
      0
d2zdx2 =
      2
d3zdy3 =
     12
```

2. 积分

在符号数学工具箱中, 表达式的积分由函数 `int` 实现, 其调用格式如下:

- `int(S)`: 求符号表达式 S 对于默认自变量的不定积分。
- `int(S,v)`: 求符号表达式 S 对于自变量 v 的不定积分。
- `diff(S,a,b)`: 求符号表达式 S 对于默认自变量从 a 到 b 的定积分。

符号积分的默认自变量选取规则与符号微分中的规则类似。

【例】 符号积分。

在命令窗输入:

```
>> syms x y
>> ix = int(z)
iy = int(z,y)
ix1_2 = int(z,1,2)
iy0_2pi = int(z,y,0,pi/2)
```

运行结果:

```

ix =
    log(x)+2*sin(y)*cos(y)*x
iy =
    1/x*y-cos(y)^2
ix1_2 =
    2*sin(y)*cos(y)+log(2)
iy0_2pi =
    1/2*(2*x+pi)/x

```

6.2.2 函数的极限

求极限就是当函数的自变量无限接近某个确定值时，求该函数的值的过程。譬如微分的计算就是利用极限定义的(假如该极限存在)：

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

函数的极限在很多场合有重要的作用，本小节将介绍如何使用符号数学工具箱求函数的极限。

通常使用 `limit` 函数求函数的极限，其调用格式如下：

- `limit(F,x,a)`：计算符号表达式 F 在 x 趋向于 a 条件下的极限。
- `limit(F,a)`：计算符号表达式 F 中默认自变量趋向于 a 条件下的极限。
- `limit(F)`：计算符号表达式 F 在默认自变量趋向于 0 时的极限。
- `limit(F,x,a,'right')` 和 `limit(F,x,a,'left')`：计算符号表达式 F 在 x 趋向于 a 条件下的右极限和左极限。

例如，函数 $x/|x|$ 在 $x \rightarrow 0^+$ ， $x \rightarrow 0^-$ 时存在极限，而在 $x \rightarrow 0$ 时不存在极限。下面将利用 `limit` 函数来求解。

【例】 函数的极限。

在命令窗输入：

```

>> sym x;
>> F = x/abs(x);
F_2_NEGATIVE = limit(F,-2)
F_0 = limit(F,x,0)
F_0_RIGTH = limit(F,x,0,'right')
F_0_LEFT = limit(F,x,0,'left')

```

运行结果：

```

F_2_NEGATIVE =
    -1
F_0 =
    NaN
F_0_RIGTH =

```

```

1
F_0_LEFT =
-1

```

6.2.3 级数求和

利用符号数学工具箱中的 `symsum` 函数可以实现级数的求和，其调用格式如下：

- `symsum(S)`: 求符号表达式 S 对于默认自变量 x 从 0 到 $x-1$ 的和。
- `symsum(S,v)`: 求符号表达式 S 对于自变量 v 从 0 到 $v-1$ 的和。
- `symsum(S,a,b)`: 求符号表达式 S 对于默认自变量 x 从 a 到 b 的和。
- `symsum(S,v,a,b)`: 求符号表达式 S 对于自变量 v 从 a 到 b 的和。

级数求和的默认自变量选取规则与符号微分中的规则类似。

【例】 级数求和。

在命令窗输入：

```

>> syms x v
>> S = x^2 + v;
>> S1 = symsum(S)
S2 = symsum(S,v)
S3 = symsum(S,0,4)
S4 = symsum(S,v,0,4)

```

运行结果：

```

S1 =
1/3*x^3-1/2*x^2+1/6*x+v*x
S2 =
x^2*v+1/2*v^2-1/2*v
S3 =
5*v+30
S4 =
5*x^2+10

```

6.2.4 泰勒级数展开

利用符号数学工具中 `taylor` 函数可以求得符号表达式的泰勒级数展开式，其调用格式如下：

- `taylor(f)`: 计算符号表达式 f 在默认自变量等于 0 处的 5 阶泰勒级数展开式。
- `taylor(f,n,v)`: 计算符号表达式 f 在自变量 $v=0$ 处的 $n-1$ 阶泰勒级数展开式。
- `taylor(f,n,v,a)`: 计算符号表达式 f 在自变量 $v=a$ 处的 $n-1$ 阶泰勒级数展开式。

当符号表达式中有多个自变量时，利用 `taylor` 函数可以求得表达式对于一个自变量的泰勒级数展开，默认自变量选取规则与符号微分中的规则类似。

【例】 泰勒级数展开。

在命令窗输入:

```
>> syms x z
>> f = sin(x) + exp(z);
>> T1 = taylor(f)
T2 = taylor(f,7,x)
T3 = taylor(f,4,z,1)
```

运行结果:

```
T1 =
exp(z)+x-1/6*x^3+1/120*x^5
T2 =
exp(z)+x-1/6*x^3+1/120*x^5
T3 =
sin(x)+exp(1)+exp(1)*(z-1)+1/2*exp(1)*(z-1)^2+1/6*exp(1)*(z-1)^3
```

6.3 表达式的化简和替换

为了得到符号表达式的某种特殊形式或者使符号计算结果更为直观, 可以通过表达式的恒等变换来对表达式进行化简或替换。MATLAB 的符号数学工具箱提供了符号表达式的因式分解、展开、合并、化简、通分、嵌套等运算功能。

6.3.1 符号表达式的化简

MATLAB 的符号数学工具箱提供了丰富的表达式代数恒等变换和三角恒等变换的函数, 本小节将介绍这类变换函数, 如 collect、expand、horner、factor、numden、simplify 和 simple。

1. 因式分解

因式分解函数 factor 的调用格式为 factor(p), 返回值为 p 的因式分解形式。其中, p 为有理系数符号多项式或符号类型的整数, 如果 p 不可分解, 则返回值为 p 本身。

【例】 因式分解。

在命令窗输入:

```
>> syms x;
>> n = (1:4)';
>> p = x.^n - 1;
>> f = factor(p);
>> [p, f]
```

运行结果:

```
ans =
[          x-1,          x-1]
```



```
[          x^2-1,      (x-1)*(x+1)]
[          x^3-1,      (x-1)*(x^2+x+1)]
[          x^4-1,      (x-1)*(x+1)*(x^2+1)]
```

【例】 因数分解。

在命令窗输入：

```
>> N = sym(3);
>> for k = 2:4
    N(k) = 10*N(k-1);
end
>> [N' factor(N')]
```

运行结果：

```
ans =
[          3,          (3)]
[          30,      (2)*(3)*(5)]
[          300,      (2)^2*(3)*(5)^2]
[          3000,      (2)^3*(3)*(5)^3]
```

2. 表达式展开

符号表达式的展开函数 `expand` 的调用格式为 `expand(S)`，返回值为符号表达式 `S` 的展开式。利用 `expand` 函数可以实现多项式、三角函数和指数函数的展开。

【例】 表达式展开。

在命令窗输入：

```
>> syms a b x y
>> f1 = a*(x + y)^2;
>> f2 = x*(x*(x-a)+b);
>> f3 = exp(a+b);
>> f4 = cos(a*x + b*y);
>> p = expand([f1; f2; f3; f4])
```

运行结果：

```
p =
a*x^2+2*a*x*y+a*y^2
x^3-a*x^2+x*b
exp(a)*exp(b)
cos(a*x)*cos(b*y)-sin(a*x)*sin(b*y)
```

3. 合并同类项

符号表达式的同类项合并函数 `collect` 的调用格式如下：

- `collect(S)`：按符号表达式 `S` 中默认自变量的同次幂项的系数合并，默认自变量为 `x`。
- `collect(S,v)`：按符号表达式 `S` 中自变量 `v` 的同次幂项的系数合并。

【例】 合并同类项。

在命令窗输入:

```
>> syms x y;  
>> f = (sin(x)*x)*(x+y)*y;  
>> R1 = collect(f)  
R2 = collect(f,y)  
R3 = collect([(x+1)*(y+1),x+y])
```

运行结果:

```
R1 =  
sin(x)*y*x^2+sin(x)*y^2*x  
R2 =  
sin(x)*y*x^2+sin(x)*y^2*x  
R3 =  
[ (y+1)*x+y+1,      x+y]
```

4. 表达式化简

符号表达式的化简可以通过函数 `simple` 和 `simplify` 来实现, 其调用格式如下:

- `r = simple(S)`: 使用多种算数法则对符号表达式 `S` 进行化简, 得到所有比 `S` 更简短的表达式, 返回值 `r` 为其中最简洁的形式。如果 `S` 为符号表达式矩阵, 则结果为整个矩阵的最简表达式形式, 而不是矩阵的每个元素最简。如果不给出返回值, 那么命令 `simple(S)` 的结果将显示所有可能的表达式并且返回值为其中的最简形式。

- `[r,how] = simple(S)`: 返回最简形式及其使用的化简方法, 并且不显示化简的中间过程。其中, `r` 为化简后的符号表达式, `how` 为代表化简方法的字符串。

- `R = simplify(S)`: 利用 Maple 化简规则化简符号矩阵 `S` 中的每个元素, 返回值为 `R`。

【例】 显示表达式化简的所有形式。

在命令窗输入:

```
>> syms x  
>> S = cos(x)^2+sin(x)^2;  
>> simple(S)
```

运行结果:

```
simplify:  
1  
radsimp:  
cos(x)^2+sin(x)^2  
combine(trig):  
1  
factor:  
cos(x)^2+sin(x)^2  
expand:  
cos(x)^2+sin(x)^2
```

```

combine:
1
convert(exp):
(1/2*exp(i*x)+1/2/exp(i*x))^2-1/4*(exp(i*x)-1/exp(i*x))^2
convert(sincos):
cos(x)^2+sin(x)^2
convert(tan):
(1-tan(1/2*x)^2)^2/(1+tan(1/2*x)^2)^2+4*tan(1/2*x)^2/(1+tan(1/2*x)^2)^2
collect(x):
cos(x)^2+sin(x)^2
mwcos2sin:
1
ans =
1

```

【例】 显示表达式化简的最简形式及其使用方法。

在命令窗输入：

```

>> syms x
>> S = cos(x)^2+sin(x)^2;
>> [r,how] = simple(S)

```

运行结果：

```

r =
1
how =
simplify

```

【例】 利用 Maple 化简规则化简。

在命令窗输入：

```

>> syms x
>> S = cos(x)^2+sin(x)^2;
>> R = simplify(S)

```

运行结果：

```

R =
1

```

5. 表达式通分

符号表达式的通分可以通过函数 `numden` 来实现，其调用格式为 `[N,D] = numden(A)`，表示将 `A` 转化为分子分母都是整系数的最佳形式。其中，`A` 为符号矩阵，`N` 为分子符号矩阵，`D` 为分母符号矩阵。

【例】 表达式通分。

在命令窗输入：

```
>> syms x y
>> [n,d] = numden([x/y + y/x, sym(1-1/2.4)])
```

运行结果:

```
n =
    [ x^2+y^2,      7]
d =
    [ y*x,  12]
```

6. 表达式的嵌套形式

函数 `horner` 可以将符号表达式转换成嵌套形式, 其调用格式为 `F = horner(P)`。其中, `P` 为符号多项式矩阵, 该函数将 `P` 的每一个元素转化为嵌套形式。

【例】 表达式的嵌套形式。

在命令窗输入:

```
>> syms x
>> P = [x^2+x; y^3-2*y];
>> F = horner(P)
```

运行结果:

```
F =
    x*(x+1)
    (-2+y^2)*y
```

6.3.2 符号表达式的替换

MATLAB 的符号数学工具箱提供了两个符号表达式替换的函数 `subexpr` 和 `subs`。利用这两个函数可以将表达式的输出形式简化, 从而得到一个简单的表达式。

`subexpr` 函数的调用格式如下:

- `[Y,SIGMA] = subexpr(X,SIGMA)`: 将符号表达式 `X` 中重复出现的字符串用 `SIGMA` 来代替, 返回值 `Y` 为替换后的符号表达式。

- `[Y,SIGMA] = subexpr(X,'SIGMA')`: 将符号表达式 `X` 中重复出现的字符串用 'SIGMA' 来代替, 返回值 `Y` 为替换后的符号表达式。

`subexpr` 函数可以使非常复杂的表达式的形式得到一定程度的化简。

【例】 表达式重复字符串的替换。

在命令窗输入:

```
>> t = solve('a*x^3+b*x^2+c*x+d = 0');
>> [r,s] = subexpr(t,'s')
```

运行结果:

```
r =
1/6/a*s^(1/3)-2/3*(3*c*a-b^2)/a/s^(1/3)-1/3*b/a
-1/12/a*s^(1/3)+1/3*(3*c*a-b^2)/a/s^(1/3)-1/3*b/a+1/2*i*3^(1/2)*(1/6/a*s^(1/3)
+2/3*(3*c*a-b^2)/a/s^(1/3))
```

$$-1/12/a*s^{1/3}+1/3*(3*c*a-b^2)/a/s^{1/3}-1/3*b/a-1/2*i*3^{1/2}*(1/6/a*s^{1/3})$$

$$+2/3*(3*c*a-b^2)/a/s^{1/3})$$

s =

$$36*c*b*a-108*d*a^2-8*b^3+12*3^{1/2}*(4*c^3*a-c^2*b^2-18*c*b*a*d+27*d^2*a^2$$

$$+4*d*b^3)^{1/2}*a$$

subs 函数的调用格式如下:

- R = subs(S): 用调用函数或工作区间的变量值替换符号表达式 S 中所有出现的对应变量。

- R = subs(S, new): 用变量 new 替换 S 中的默认变量。

- R = subs(S, old, new): 用变量 new 替换 S 中的变量 old。old 为符号变量或代表某个变量名字符串, new 为符号变量、数值变量或表达式。

下面举例说明 subs 函数的用法。

【例】 用工作区间变量替换。

在命令窗输入:

```
>> a = 10;
>> C1 = 2;
>> y = dsolve('Dy = -a*y')
yt = subs(y)
```

运行结果:

```
y =
C1*exp(-a*t)
yt =
2*exp(-10*t)
```

【例】 用输入参量替换。

在命令窗输入:

```
>> syms a C1 t
>> y = C1*exp(-a*t);
>> yt = subs(y, {a, C1}, {10, 2})
```

运行结果:

```
yt =
2*exp(-10*t)
```

【例】 用矩阵替换标量。

在命令窗输入:

```
>> subs(exp(a*t), 'a', magic(2))
```

运行结果:

```
ans =
[ exp(t), exp(3*t)]
[ exp(4*t), exp(2*t)]
```

【例】 相乘标量的替换。

在命令窗输入：

```
>> syms x y
```

```
>> subs(x*y,{x,y},{[0 1;-1 0],[1 1;1 1]})
```

运行结果：

```
ans =
```

```
0      1
```

```
-1     0
```

6.4 线性代数

符号线性代数的运算规则与数值线性代数的运算规则基本相同，因此操作起来非常方便。本节将介绍符号线性代数运算功能，包括基本代数运算、线性代数运算、特征值、约当标准型、奇异值分解以及特征值轨迹等。

6.4.1 基本代数运算

符号对象的基本代数运算与 double 类的 MATLAB 对象运算方法相同。

【例】 符号矩阵的幂。

在命令窗输入：

```
>> syms t;
```

```
>> G = [cos(t) sin(t); -sin(t) cos(t)];
```

```
>> A = G^2
```

```
A = simple(A)
```

运行结果：

```
A =
```

```
[ cos(t)^2-sin(t)^2,  2*cos(t)*sin(t)]
```

```
[ -2*cos(t)*sin(t), cos(t)^2-sin(t)^2]
```

```
A =
```

```
[ cos(2*t), sin(2*t)]
```

```
[ -sin(2*t), cos(2*t)]
```

【例】 符号正交矩阵。

在命令窗输入：

```
>> syms t;
```

```
>> G = [cos(t) sin(t); -sin(t) cos(t)];
```

```
>> I = G'*G
```

```
I = simple(I)
```

运行结果：

```

I =
    [ cos(t)^2+sin(t)^2,      0]
    [      0, cos(t)^2+sin(t)^2]

I =
    [ 1, 0]
    [ 0, 1]

```

6.4.2 线性代数运算

符号对象的线性代数运算与 double 类的 MATLAB 对象的线性代数运算相同。本小节将以希尔伯特矩阵为例来介绍符号数学工具箱中的几种线性代数运算。

【例】 符号线性代数运算。

创建一个数值类型的希尔伯特矩阵，在命令窗输入：

```
>> H = hilb(3)
```

运行结果：

```

H =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

```

将 H 转化为符号矩阵，在命令窗输入：

```
>> H = sym(H)
```

运行结果：

```

H =
    [ 1, 1/2, 1/3]
    [ 1/2, 1/3, 1/4]
    [ 1/3, 1/4, 1/5]

```

对这个无限精度的希尔伯特矩阵求逆及行列式，在命令窗输入：

```
>> inv(H)
```

```
det(H)
```

运行结果：

```

ans =
    [ 9, -36, 30]
    [ -36, 192, -180]
    [ 30, -180, 180]

ans =
    1/2160

```

利用反斜杠求希尔伯特矩阵构成的线性方程组，在命令窗输入：

```
>> b = [1 1 1]';
```

```
>> x = H\b
```

```
x =
```

```

3
-24
30

```

以上求逆、求行列式、求解线性方程组的结果都是对应无限精度有理希尔伯特矩阵的精确解。另一方面，如果以使用 16 位精度的线性代数运算所得结果与上例进行对比，可以看出符号线性代数运算与数值线性代数运算存在着本质上的区别。

【例】 16 位精度数值线性代数运算。

创建一个 16 位精度的希尔伯特矩阵，在命令窗输入：

```

>> digits(16)
>> V = vpa(hilb(3))

```

运行结果：

```

V =
[ 1., .5000000000000000, .3333333333333333]
[ .5000000000000000, .3333333333333333, .2500000000000000]
[ .3333333333333333, .2500000000000000, .2000000000000000]

```

上面的矩阵元素的小数点表明使用的是变精度算法，即将结果四舍五入至 16 位阿拉伯数字。当求矩阵的逆时，由于矩阵条件数很大(3 阶希尔伯特矩阵的条件数约为 500)，因此误差将被放大。在命令窗输入：

```

>> inv(V)

```

运行结果：

```

ans =
[ 9.000000000000179, -36.00000000000080, 30.00000000000067]
[ -36.00000000000080, 192.0000000000042, -180.0000000000040]
[ 30.00000000000067, -180.0000000000040, 180.0000000000038]

```

再看行列式的值，在命令窗输入：

```

>> det(V)

```

运行结果：

```

ans =
.462962962962953e-3

```

求解线性方程组，在命令窗输入：

```

>> V\b

```

运行结果：

```

ans =
3.000000000000041
-24.00000000000021
30.00000000000019

```

【例】 奇异矩阵的运算。

由于 H 非奇异，求其化零矩阵，在命令窗输入：

```

>> null(H)

```


运行结果:

```
ans =  
[ empty sym ]
```

求 H 的列空间, 在命令窗输入:

```
>> colspace(H)
```

运行结果:

```
ans =  
[ 1, 0, 0]  
[ 0, 0, 1]  
[ 0, 1, 0]
```

即得到了一个单位矩阵的置换形式。下面介绍如何通过修改 $H(1,1)$ 的值成为 s , 使 H 奇异。在命令窗输入:

```
>> syms s  
H(1,1) = s  
Z = det(H)  
sol = solve(Z)
```

运行结果:

```
H =  
[ s, 1/2, 1/3]  
[ 1/2, 1/3, 1/4]  
[ 1/3, 1/4, 1/5]  
Z =  
1/240*s-1/270  
sol =  
8/9
```

接着利用 $H = \text{subs}(H,s,\text{sol})$ 将 s 用 sol 替换, 在命令窗输入:

```
>> H = subs(H,s,sol)
```

运行结果:

```
H =  
[ 8/9, 1/2, 1/3]  
[ 1/2, 1/3, 1/4]  
[ 1/3, 1/4, 1/5]
```

此时, H 的行列式将为零, H 的逆矩阵将不存在。 $Z = \text{null}(H)$ 以及 $C = \text{colspace}(H)$ 将返回非平凡的结果。在命令窗输入:

```
>> Z = null(H)  
C = colspace(H)
```

运行结果:

```
Z =
```

```

-4
10/3
C =
[ 1, 0]
[ 0, 1]
[ -3/10, 6/5]

```

需要指出的是, 即使 H 奇异, $\text{vpa}(H)$ 非奇异。行列式不为零, 逆矩阵存在。在命令窗输入:

```

>> digits(16);
det(vpa(H))
inv(vpa(H))

```

运行结果:

```

ans =
-9e-17
ans =
[-462962962962962.2, 1851851851851853., -1543209876543211.]
[ 1851851851851853., -7407407407407411., 6172839506172844.]
[-1543209876543211., 6172839506172844., -5144032921810700.]

```

6.4.3 特征值

符号线性代数中, 常用 `eig` 函数来求符号矩阵的特征值或特征向量, 其调用格式如下:

- `E = eig(A)`: 求符号矩阵 A 的特征值。
- `[V,E] = eig(A)`: 求符号矩阵 A 的特征值 E 和特征向量 V 。

下面以上一小节的奇异矩阵为例, 给出符号矩阵特征值和特征向量的求法。

【例】 符号矩阵的特征值与特征向量。

在命令窗输入:

```

>> H = sym(hilb(3));
>> H(1,1) = sym(8/9);
>> [T,E] = eig(H)
T =
[ 1, 28/153+2/153*12589^(1/2), 28/153-2/153*12589^(1/2)]
[ -4, 1, 1]
[ 10/3, 292/255-1/255*12589^(1/2), 292/255+1/255*12589^(1/2)]
E =
[ 0, 0, 0]
[ 0, 32/45+1/180*12589^(1/2), 0]
[ 0, 32/45-1/180*12589^(1/2), 0]

```

其中, T 为特征向量, E 为特征值构成的对角矩阵。为使这个结果更为明了, 在命令窗输入:

```
>> Td = double(T)
Ed = double(E)
运行结果:
Td =
    1.0000    1.6497   -1.2837
   -4.0000    1.0000    1.0000
    3.3333    0.7051    1.5851
Ed =
     0     0     0
     0    1.3344     0
     0     0    0.0878
```

6.4.4 约当标准型

约当标准型的结果类似于矩阵的对角化,即对于一个矩阵 A , 寻找一个非奇异矩阵 V , 使得矩阵 $J = V^{-1}A \cdot V$ 最接近对角矩阵。对于绝大部分矩阵, 约当标准型为特征值组成的对角矩阵 J , 其变换矩阵 V 的列向量则为对应的特征向量。特征值不相等的对称矩阵往往具有这个特点, 但是对于有重特征值的非对称矩阵, 有可能不能对其进行对角化。约当型的主对角线上为矩阵的特征值, 但是超对角线上的某些元素为 1 而不是 0。

计算符号矩阵约当标准型的函数 `jordan` 的常用调用格式为 `[V,J] = jordan(A)`, 其中, J 为约当标准型, V 为变换矩阵, 其列向量代表广义特征向量。

【例】 符号矩阵的约当标准型。

```
>> A = sym([12,32,66,116;-25,-76,-164,-294;
            21,66,143,256;-6,-19,-41,-73]);
>> [V,J] = jordan(A)
V =
     4     -2     4     3
    -6     8    -11    -8
     4     -7    10     7
    -1     2     -3    -2
J =
     1     1     0     0
     0     1     0     0
     0     0     2     1
     0     0     0     2
```

6.4.5 奇异值分解

符号数学工具箱中只支持变精度完全奇异向量分解的数值计算, 其中的一个原因就是符号计算所得的公式过长且过于复杂。如果 A 是一个浮点或变精度数值的符号矩阵, 则 `svd` 函数计算 A 的奇异分解。`svd` 的常用调用格式为 `[U,S,V] = svd(A)`, 其中 U 、 V 为正交矩阵,

S 为对角矩阵, 且满足 $A = U \cdot S \cdot V'$ 。

【例】 符号矩阵的奇异值分解。

在命令窗输入:

```
>> [J,I] = meshgrid(1:5);
```

```
>> digits(8)
```

```
A = sym(1./(I - J+1/2))
```

```
[U,S,V] = svd(vpa(A))
```

运行结果:

A =

```
[ 2, -2, -2/3, -2/5, -2/7]
[ 2/3, 2, -2, -2/3, -2/5]
[ 2/5, 2/3, 2, -2, -2/3]
[ 2/7, 2/5, 2/3, 2, -2]
[ 2/9, 2/7, 2/5, 2/3, 2]
```

U =

```
[ .12424472, -.35258040, .60671803, .64348009, -.27942247]
[ -.56925169, .63744468, -.96205058e-1, .43193383, -.27165153]
[ .74582672, .27450328, -.44025133, .29153926, -.29928930]
[ -.32185699, -.61914288, -.59364816, .11603781, -.38364992]
[ .25813766e-1, .10325402, .27640268, -.54854957, -.78189972]
```

S =

```
[ 3.1415892, 0, 0, 0, 0 ]
[ 0, 3.1412722, 0, 0, 0 ]
[ 0, 0, 3.1295206, 0, 0 ]
[ 0, 0, 0, 2.9218608, 0 ]
[ 0, 0, 0, 0, 1.4618645]
```

V =

```
[ .25813766e-1, -.10325402, .27640268, .54854957, -.78189972]
[ -.32185699, .61914288, -.59364816, -.11603781, -.38364992]
[ .74582672, -.27450328, -.44025133, -.29153926, -.29928930]
[ -.56925169, -.63744468, -.96205058e-1, -.43193383, -.27165153]
[ .12424472, .35258040, .60671803, -.64348009, -.27942247]
```

6.4.6 特征值轨迹

本小节给出一个在矩阵参数变化时, 矩阵特征值的变化情况。

【例】 特征值轨迹。

在命令窗输入:

```
>> A = gallery(3)
```

```
E = [130,-390,0;43,-129,0;133,-399,0];
```

```

syms x t
A = A+t*E
p = poly(A)
x = .8:.01:3.2;
for k = 0:2
    c = sym2poly(subs(p,t,k*0.5e-6));
    y = polyval(c,x);
    lambda = eig(double(subs(A,t,k*0.5e-6)));
    subplot(3,1,3-k)
    plot(x,y,'-',x,0*x,'.',lambda,0*lambda,'o')
    axis([.8 3.2 -.5 .5])
    text(2.25,.35,['t = ' num2str( k*0.5e-6 )]);
end

```

运行结果如下。图 6-1 所示自上至下分别为 $t = 1.0e-6$ 、 $t = 0.5e-6$ 、 $t = 0$ 时的特征值轨迹。

A =

```

-149   -50  -154
 537   180   546
 -27    -9   -25

```

A =

```

[ -149+130*t,  -50-390*t,   -154]
[  537+43*t,   180-129*t,    546]
[  -27+133*t,  -9-399*t,    -25]

```

p =

```

x^3-6*x^2+11*x-t*x^2+492512*t*x-6-1221271*t

```

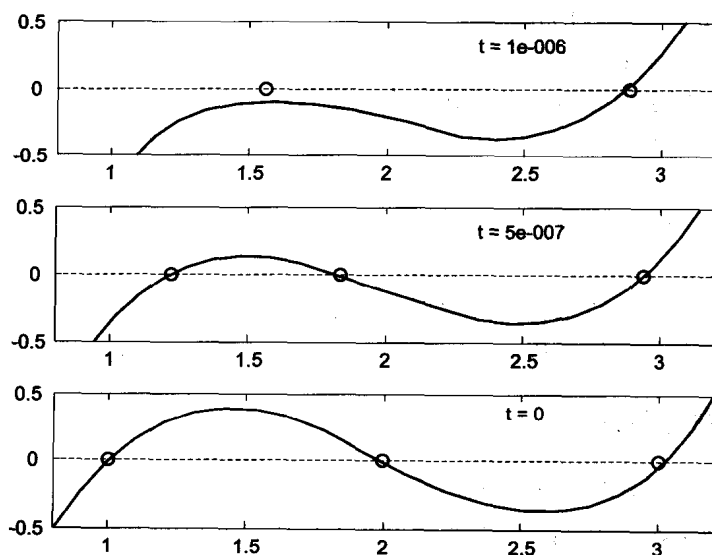


图 6-1 特征值轨迹

6.5 求解符号方程

符号方程包括符号代数方程和符号微分方程。符号工具箱能够求解的代数方程包括线性代数方程(组)和非线性代数方程(组)，能够求解的微分方程只有常微分方程(组)。本节将介绍如何利用符号数学工具箱实现代数方程、代数方程组、常微分方程、常微分方程组的求解。

6.5.1 求解代数方程

符号数学工具箱提供了求解符号代数方程的函数 `solve`，其调用格式如下：

- `g = solve(eq)`：求解关于默认自变量的方程式 `eq` 或 `eq = 0`。其中，`eq` 可以是符号表达式或字符串，返回值 `g` 是所有解构成的列向量。如果 `eq` 是不带等号的符号表达式或字符串，则求解关于默认自变量的方程式 `eq = 0`。

- `g = solve(eq,var)`：求解关于自变量 `var` 的方程式 `eq` 或 `eq = 0`。

通常方程的默认自变量为 `x`。

【例】 求解代数方程。

在命令窗输入：

```
>> syms a b c x
```

```
S = a*x^2 + b*x + c;
```

```
solve(S)
```

运行结果：

```
ans =
```

```
-1/2*(b-(b^2-4*a*c)^(1/2))/a
```

```
-1/2*(b+(b^2-4*a*c)^(1/2))/a
```

当然，也可以指定自变量，继续在命令窗输入：

```
>> b = solve(S,b)
```

运行结果：

```
b =
```

```
-(a*x^2+c)/x
```

需要注意的是，如果待求解的方程形式不是 $f(x) = 0$ ，而是形如 $g(x) = f(x)$ ，则 `solve` 函数的输入参量必须是引号字符串。

6.5.2 求解代数方程组

符号代数方程组的求解也要使用函数 `solve`，其调用格式如下：

- `g = solve(eq1,eq2,...,eqn)`：求解关于 n 个默认自变量的方程组 `eq1`、`eq2`、 \cdots 、`eqn`。其中，`eq1`、`eq2`、 \cdots 、`eqn` 可以是符号表达式或字符串。默认自变量由函数 `findsym` 作用于方程系统决定。

• $g = \text{solve}(eq1, eq2, \dots, eqn, var1, var2, \dots, varn)$: 求解关于 n 个自变量 $var1$ 、 $var2$ 、 \dots 、 $varn$ 的方程组 $eq1$ 、 $eq2$ 、 \dots 、 eqn 。

对于方程数等于输出数的方程组，其解按照字母表排序；对于单输出的方程组，则返回值为所有解组成的结构。

【例】 求解代数方程组(单输出)。

在命令窗输入：

```
>> S = solve('u^2-v^2 = a^2','u + v = 1','a^2-2*a = 3')
```

```
S.a
```

运行结果：

```
S =
```

```
  a: [2x1 sym]
```

```
  u: [2x1 sym]
```

```
  v: [2x1 sym]
```

```
ans =
```

```
  3
```

```
 -1
```

【例】 求解代数方程组(输出数等于方程数)。

在命令窗输入：

```
>> syms x y alpha
```

```
>> [x,y] = solve(x^2*y^2, x-y/2-alpha)
```

运行结果：

```
x =
```

```
  0
```

```
  0
```

```
alpha
```

```
alpha
```

```
y =
```

```
-2*alpha
```

```
-2*alpha
```

```
  0
```

```
  0
```

需要注意的是，由于该方程组第一个方程的解为 $x=\pm 0$ ， $y=\pm 0$ ，故最终的解向量中含有冗余项。

6.5.3 求解常微分方程

符号数学工具箱提供了求解符号常微分方程(组)的函数 `dsolve`。微分方程中，在变量字母前加 D 代表对该变量的微分。符号 $D2$ 、 $D3$ 、 \dots 、 DN 分别对应二阶、三阶、 \dots 、 N 阶微分。例如 $D2y$ 表示的是 d^2y/dt^2 。因变量是前面加 D 的符号变量，并且默认对自变量 t 求导。

应当注意, 符号变量名中不能含有 D。

dsolve 函数的调用格式如下:

● $r = \text{dsolve}(\text{'eq1,eq2,...'}, \text{'cond1,cond2,...'}, \text{'v'})$: 求解由 eq1、eq2... 构成的常微分方程(组), 且对自变量 v 求导, 由 cond1、cond2... 指定边值或初值条件。

● $r = \text{dsolve}(\text{'eq1','eq2'}, \dots, \text{'cond1','cond2'}, \dots, \text{'v'})$: 同上, 但输入参量最多不超过 12。

● $\text{dsolve}(\text{'eq1,eq2,...'}, \text{'cond1,cond2,...'}, \text{'v'})$: 同上。

如果初值条件数少于因变量数, 求解的结果中将含有积分常数 C1、C2 等。dsolve 函数的输出形式有以下三种:

● 对于只有一个方程和一个输出的情况, dsolve 函数的返回值为一个非线性方程多个解构成的列向量。

● 对于多个方程和方程数等于输出数的情况, dsolve 函数将解按照词典顺序排序并赋给输出。

● 对于多个方程和一个输出的情况, dsolve 函数的返回值为所有解组成的结构。

下面给出求解常微分方程的具体例子。

【例】 求解自治常微分方程。

在命令窗输入:

```
>> dsolve('Dx = -a*x')
```

运行结果:

```
ans =
```

```
C1*exp(-a*t)
```

【例】 求解非自治常微分方程。

在命令窗输入:

```
>> dsolve('Df = f + sin(t)')
```

运行结果:

```
ans =
```

```
-1/2*cos(t)-1/2*sin(t)+exp(t)*C1
```

【例】 求解二阶常微分方程。

在命令窗输入:

```
>> dsolve('D2y = -a^2*y', 'y(0) = 1', 'Dy(pi/a) = 0')
```

运行结果:

```
ans =
```

```
cos(a*t)
```

【例】 求解常微分方程组。

在命令窗输入:

```
>> S = dsolve('Dx = y', 'Dy = -x')
```

```
S.x
```

```
S.y
```

运行结果:


```

S =
      x: [1x1 sym]
      y: [1x1 sym]
ans =
      -C1*cos(t)+C2*sin(t)
ans =
      C1*sin(t)+C2*cos(t)

```

6.6 简易符号绘图函数

MATLAB 的符号数学工具箱提供了一些符号绘图函数, 利用这些函数可以方便快捷地绘制出函数的简易图。本节将对这类绘图函数作简要的介绍。

6.6.1 二维基本绘图

函数 `ezplot` 是 MATLAB 最基本最常用的二维绘图函数, 其调用格式如下:

- `ezplot(f)`: 在默认区域 $-2\pi < x < 2\pi$ 内绘制函数表达式 $f = f(x)$ 。
- `ezplot(f,[xmin,xmax])`: 在指定的区域 $[xmin, xmax]$ 内绘制函数表达式 $f = f(x)$ 。如果没有图形窗口存在, 则该命令生成标题为 Figure No.1 的新窗口并在该窗口中作图; 如果已有图形窗口存在, 则在标号最高的图形窗口中作图。
- `ezplot(f,[xmin xmax],fign)`: 打开并在指定标号 `fign` 的窗口中绘制 $[xmin, xmax]$ 区域内函数 $f = f(x)$ 的图像。
- `ezplot(f)`: 在默认区域 $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$ 内绘制函数表达式 $f(x,y) = 0$ 的图像。
- `ezplot(f,[xmin,xmax,ymin,ymax])`: 在 $xmin < x < xmax$ 和 $ymin < y < ymax$ 区域内绘制函数 $f(x,y) = 0$ 的图像。
- `ezplot(f,[min,max])`: 在 $min < x < max$ 和 $min < y < max$ 区域内绘制函数 $f(x,y) = 0$ 的图像。
- `ezplot(x,y)`: 在默认范围 $0 < t < 2\pi$ 内绘制参数方程 $x = x(t)$ 、 $y = y(t)$ 的图像。
- `ezplot(x,y,[tmin,tmax])`: 在指定的范围 $tmin < t < tmax$ 内绘制参数方程 $x = x(t)$ 、 $y = y(t)$ 的图像。
- `ezplot(...,figure)`: 在由 `figure` 句柄标识的图形窗口中绘制指定函数的图像。

【例】 绘制二维图像 $x^2 - y^4 = 0$ 。

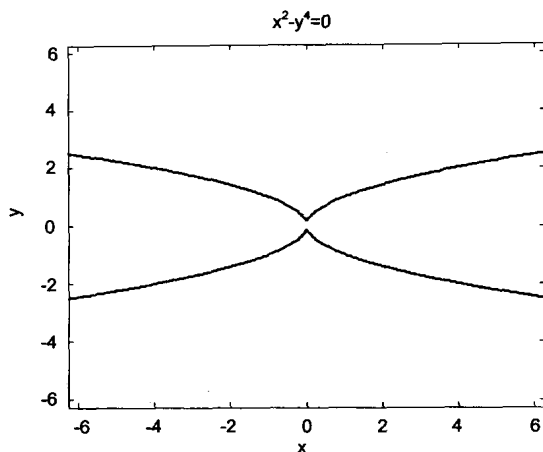
在命令窗输入:

```

>> syms x y
>> ezplot(x^2-y^4)

```

运行结果如图 6-2 所示。

图 6-2 $x^2 - y^4 = 0$ 的图像

【例】 绘制二维图像 $f(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ 。

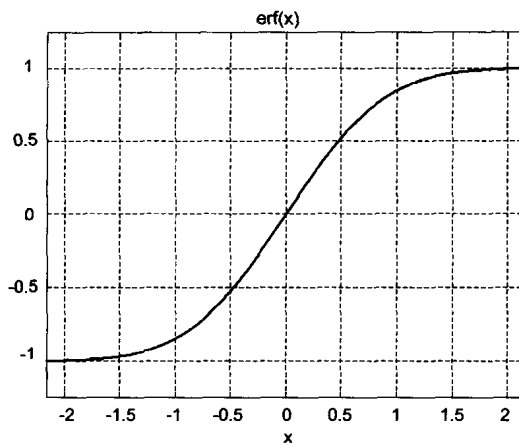
在命令窗输入：

```
>> syms x
```

```
>> ezplot(erf(x))
```

```
grid
```

运行结果如图 6-3 所示。

图 6-3 $\text{erf}(x)$ 的图像

6.6.2 二维极坐标绘图

函数 `ezpolar` 用于二维极坐标函数的绘图，其调用格式如下：

- `ezpolar(f)`: 在默认区域 $0 < \theta < 2\pi$ 内绘制极坐标函数 $\rho = f(\theta)$ 的图像。
- `ezpolar(f,[a,b])`: 在指定区域 $a < \theta < b$ 内绘制极坐标函数 f 的图像。

【例】 二维极坐标绘图。

在命令窗输入：

```
>> syms t
```

```
>> ezpolar(1+cos(4*t))
```

运行结果如图 6-4 所示。

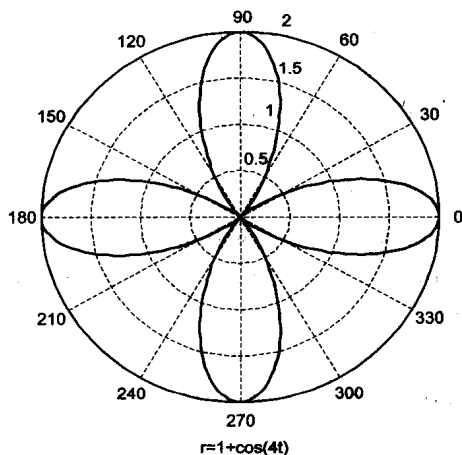


图 6-4 绘制极坐标图

6.6.3 三维曲线绘图

函数 `ezplot3` 用于三维曲线的简易绘图，其调用格式如下：

- `ezplot3(x,y,z)`: 在默认区域 $0 < t < 2\pi$ 内绘制由参数方程 $x = x(t)$ 、 $y = y(t)$ 与 $z = z(t)$ 定义的曲线。
- `ezplot3(x,y,z,[tmin,tmax])`: 在区域 $t_{\min} < t < t_{\max}$ 内绘制由参数方程 $x = x(t)$ 、 $y = y(t)$ 与 $z = z(t)$ 定义的曲线。
- `ezplot3(...,'animate')`: 绘制参数方程的动态轨迹。

【例】 绘制三维参数曲线。

在命令窗输入：

```
>> syms t;
```

```
>> ezplot3(sin(t), cos(t), t,[0,6*pi])
```

运行结果如图 6-5 所示。

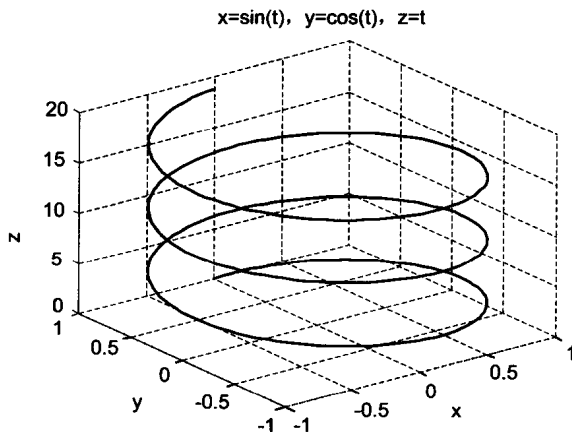


图 6-5 绘制三维参数曲线图

6.6.4 三维网格绘图

函数 `ezmesh` 用于绘制三维网格图，其调用格式如下：

- `ezmesh(f)`: 绘制二元函数 $f(x,y)$ 的三维网格图， f 为函数 f 的符号表达式，默认区域 $-2\pi < x < 2\pi$ ， $-2\pi < y < 2\pi$ 。MATLAB 将根据函数变动的程度选择相应的计算栅格。如果函数 f 在某些栅格点上没有定义，则这些点将不显示。

- `ezmesh(f,domain)`: 在指定区域 domain 内绘制二元函数 f 的三维网格图， domain 可以是四维向量 $[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ 或二维向量 $[\min, \max]$ 。

- `ezmesh(x,y,z)`: 在默认区域 $-2\pi < s < 2\pi$ 、 $-2\pi < t < 2\pi$ 内绘制由参数方程 $x = x(s,t)$ 、 $y = y(s,t)$ 、 $z = z(s,t)$ 定义的网格图。

- `ezmesh(x,y,z,[smin,smax,tmin,tmax])`: 在指定区域内绘制由参数方程定义的网格图。

- `ezmesh(x,y,z,[min,max])`: 在指定区域内绘制由参数方程定义的网格图。

- `ezmesh(...,n)`: 用指定的 $n \times n$ 个栅格点在默认区域内绘制函数 f 的网格图。 n 的默认值为 60。

- `ezmesh(...,'circ')`: 在一圆形区域内画出函数 f 的网格图。

函数 `ezmeshc` 用于同时绘制三维网格图与等高线图，其调用格式与函数 `ezmesh` 类似。例如绘制如下函数表达式的网格图。

$$f(x,y) = \frac{y}{1+x^2+y^2}$$

【例】绘制三维网格图。

在命令窗输入：

```
>> syms x y
```

```
>> ezmesh(y/(1+x^2+y^2),[-5,5,-2*pi,2*pi])
```

运行结果如图 6-6 所示。

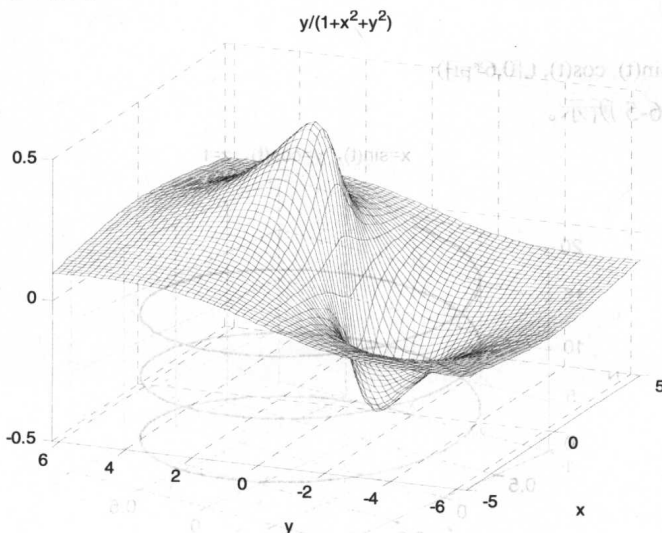


图 6-6 三维网格图

【例】 同时绘制三维网格图与等高线图。

在命令窗输入：

```
>> syms x y
```

```
>> ezmeshc(y/(1+x^2+y^2),[-5,5,-2*pi,2*pi])
```

运行结果如图 6-7 所示。

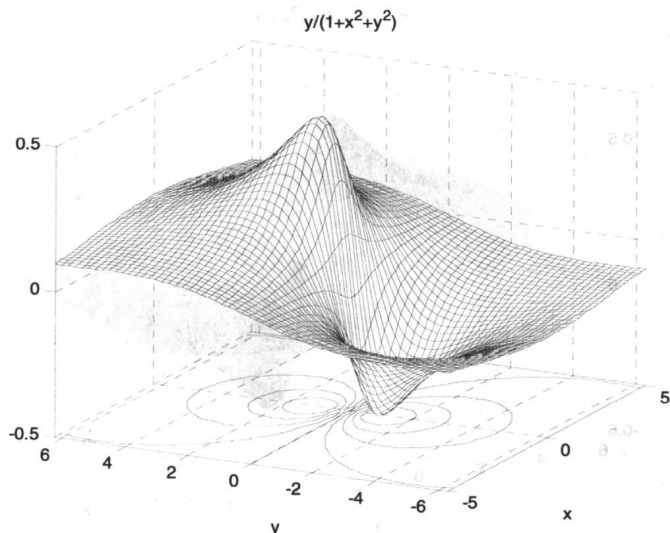


图 6-7 三维网格图与等高线图

6.6.5 三维表面绘图

ezsurf 函数用于绘制三维表面图，其调用格式如下：

- ezsurf(f): 绘制二元函数 $f(x,y)$ 的三维表面图， f 为函数 f 的符号表达式，默认区域 $-2\pi < x < 2\pi$ ， $-2\pi < y < 2\pi$ 。MATLAB 将根据函数变动的程度选择相应的计算栅格。如果函数 f 在某些栅格点上没有定义，则这些点将不显示。

- ezsurf(f,domain): 在指定区域 domain 内绘制二元函数 f 的三维表面图，domain 可以是四维向量 $[xmin,xmax,ymin,ymax]$ 或二维向量 $[min,max]$ 。

- ezsurf(x, y, z): 在默认区域 $-2\pi < s < 2\pi$ 、 $-2\pi < t < 2\pi$ 内绘制由参数方程 $x=x(s, t)$ 、 $y=y(s, t)$ 、 $z=z(s, t)$ 定义的表面图。

- ezsurf(x,y,z,[smin,smax,tmin,tmax]): 在指定区域内绘制由参数方程定义的表面图。

- ezsurf(x,y,z,[min,max]): 在指定区域内绘制由参数方程定义的表面图。

- ezsurf(...,n): 用指定 $n \times n$ 个栅格点在默认区域内绘制函数 f 的表面图。 n 的默认值为 60。

- ezsurf(...,'circ'): 在一圆形区域内画出函数 f 的表面图。

函数 ezsurfc 用于同时绘制三维表面图与等高线图，其调用格式与函数 ezsurf 类似。例如绘制如下函数表达式的表面图。

$$f(x, y) = \frac{y}{1+x^2+y^2}$$

【例】 绘制三维表面图。

在命令窗输入：

```
>> syms x y
```

```
>> ezsurf(y/(1+x^2+y^2),[-5,5,-2*pi,2*pi],35)
```

运行结果如图 6-8 所示。

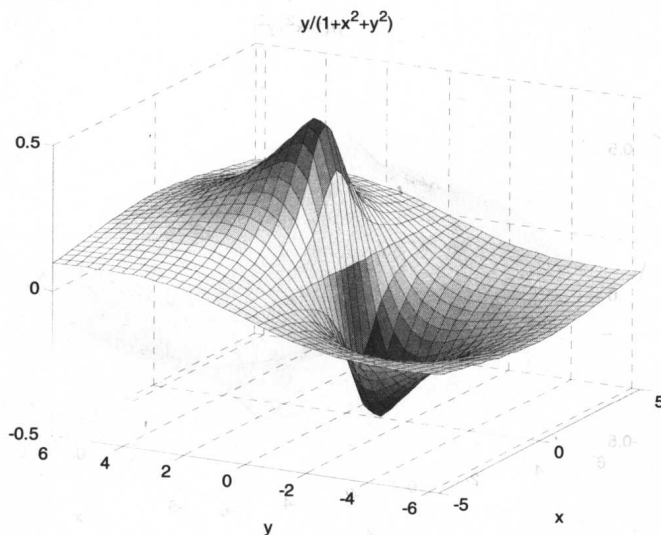


图 6-8 三维表面图

【例】 同时绘制三维表面图与等高线图。

在命令窗输入：

```
>> syms x y
```

```
>> ezsurf(y/(1+x^2+y^2),[-5,5,-2*pi,2*pi],35)
```

运行结果如图 6-9 所示。

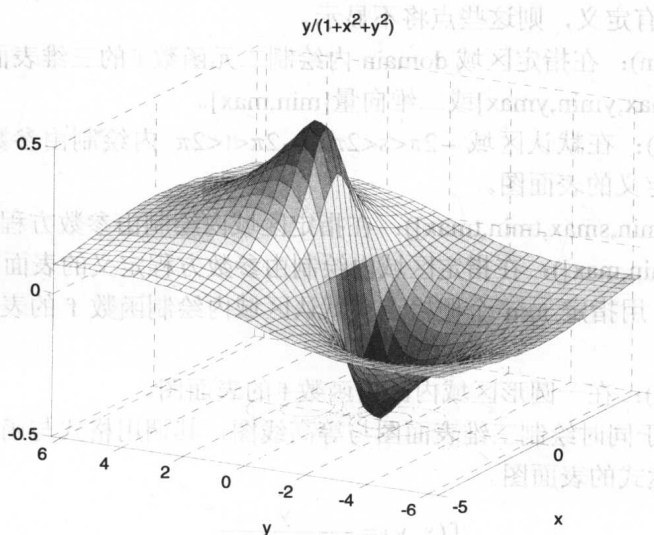


图 6-9 三维表面图与等高线图

6.6.6 等高线绘图

函数 `ezcontour` 用于绘制函数的等高线，其调用格式如下：

- `ezcontour(f)`: 绘制二元函数 $f(x,y)$ 的等高线图， f 为函数 f 的符号表达式，默认区域 $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$ 。MATLAB 将根据函数变动的程度选择相应的计算栅格。如果函数 f 在某些栅格点上没有定义，则这些点将不显示。

- `ezcontour(f, domain)`: 在指定区域 domain 内绘制二元函数 f 的等高线图， domain 可以是四维向量 $[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ 或二维向量 $[\min, \max]$ 。

- `ezcontour(..., n)`: 用指定 $n \times n$ 个栅格点在默认区域内绘制函数 f 的等高线图。 n 的默认值为 60。

函数 `ezcontourf` 用于绘制函数的等高线并以不同颜色填充，其调用格式与函数 `ezcontour` 类似。例如绘制如下函数表达式的等高线图。

$$f(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3} e^{-(x+1)^2-y^2}$$

【例】 绘制等高线图。

在命令窗输入：

```
>> syms x y
>> f = 3*(1-x)^2*exp(-(x^2)-(y+1)^2) ...
    - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) ...
    - 1/3*exp(-(x+1)^2 - y^2);
>> ezcontour(f, [-3,3], 49)
```

运行结果如图 6-10 所示。

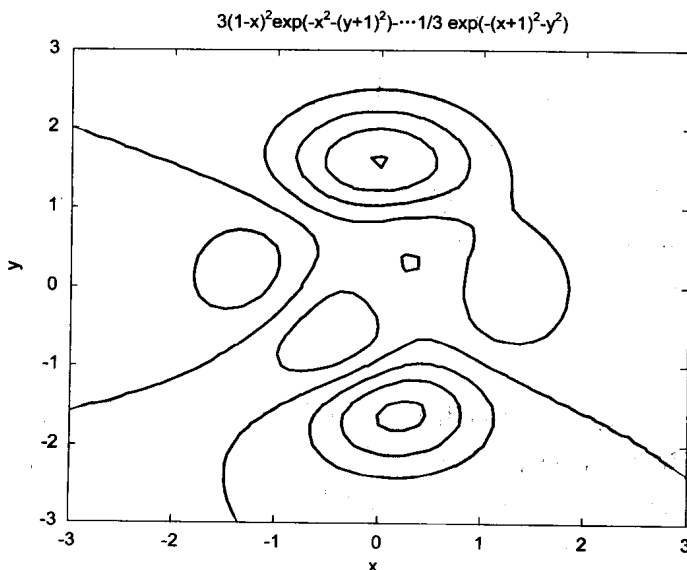


图 6-10 等高线图

【例】 绘制等高线图并以不同颜色填充。

在命令窗输入：

```
>> syms x y
>> f = 3*(1-x)^2*exp(-(x^2)-(y+1)^2) ...
    - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) ...
    - 1/3*exp(-(x+1)^2 - y^2);
```

运行结果如图 6-11 所示。

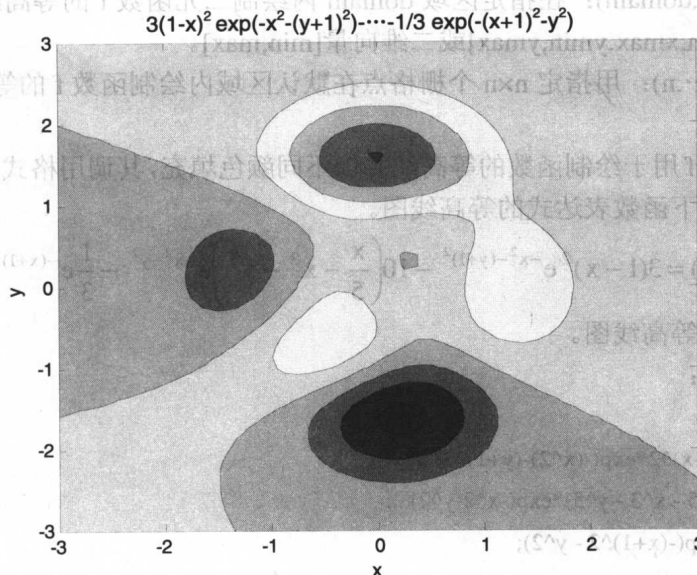


图 6-11 有填充色的等高线图

6.7 调用 Maple 函数

由于 MATLAB 有高性能的数值计算功能，而 Maple 具有强大的符号计算功能，因此通过调用 Maple 函数可以弥补 MATLAB 在符号计算功能方面的不足。本节将介绍几个调用 Maple 函数的相关函数。

6.7.1 maple 函数

maple 函数为用户提供了直接调用 Maple 函数的途径。该函数的输入参量为 sym 对象、字符串或 double 类型的数，返回值为对应于输入类型的 sym 对象、字符串或 double 类型数。利用 maple 函数也可以对自行开发的符号数学程序进行调试。

maple 函数的调用格式如下：

- `r = maple('statement')`: 将 statement 传递给 Maple 内核，且返回计算结果。必要时可以在 statement 后面加分号(;)。
- `maple('function',arg1,arg2,...)`: 接受任何带引号的与输入参量相关的 Maple 函数名

'function', 必要时将输入参量转换为符号表达式并且函数通过给定参量调用。如果输入参量为 syms, 则 maple 返回一个 sym 类型的结果, 否则返回一个 char 类型的结果。

• `[r,status] = maple(...)`: 有条件地返回警告/错误信息。如果语句执行成功, 则 `r` 为计算结果且 `status` 为 0; 如果语句执行失败, 则 `r` 为相应的警告/错误信息, 而 `status` 为一正整数。

• `maple('traceon')` 或 `maple trace on`: 显示所有后续 Maple 语句及其相应结果。

• `maple('traceoff')` 或 `maple trace off`: 关闭上面的操作特性。

【例】 计算最小公因数和最小公因式。

在命令窗输入:

```
>> num1 = maple('gcd(14, 21)')
xyl = maple('gcd(x^2-y^2,x^3-y^3)')
```

运行结果:

```
num1 =
      7
xyl =
      -y+x
```

6.7.2 mfun 函数

`mfun` 函数的调用格式为 `mfun('function',par1,par2,par3,par4)`。计算 Maple 中的已知数学函数 `function`, 返回值为数值量。每个参量 `par` 为对应 `function` 的数值量。用户可以输入的参量最多不超过 4 个。最后指定的参量可以是矩阵, 通常对应于 `X`。其他参量的维数取决于 `function` 的 Maple 规范。用户可以通过以下命令获得 Maple 函数的相关参数信息:

- `help mfunlist`;
- `mhelp function`。

Maple 用 16 位精度计算 `function`, 其结果的每个元素为 MATLAB 数值量。`function` 中的任何奇异值将返回 `NaN`。

【例】 使用 `mfun` 函数。

在命令窗输入:

```
>> mfun('FresnelC',0:5)
mfun('Chi',[3*i 0])
```

运行结果:

```
ans =
      0      0.7799      0.4883      0.6057      0.4984      0.5636
ans =
      0.1196 + 1.5708i      NaN +      NaNi
```

6.7.3 sym 函数

本小节将给出利用 Maple 阶乘函数计算阶乘的例子。计算阶乘时首先要通过 `sym` 函数

调用 Maple 阶乘函数 k!。

【例】 计算 6!和 n!。

在命令窗输入：

```
>> kfac = sym('k!');
>> syms k n
>> subs(kfac,k,6), subs(kfac,k,n)
```

运行结果：

```
ans =
    720
ans =
    n!
```

6.8 积分变换

符号数学工具箱提供了对符号表达式进行积分变换的函数，包括傅里叶变换和傅里叶反变换、拉普拉斯变换和拉普拉斯反变换、Z 变换和逆 Z 变换。以上变换在科学和工程领域中有着极为重要的应用，本节将分别讨论这些变换的实现方法。

6.8.1 傅里叶变换

傅里叶变换和傅里叶反变换的定义式如下：

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega)e^{j\omega t} d\omega$$

傅里叶变换的求解通常使用函数 `fourier` 来实现，其调用格式如下：

- `F = fourier(f)`：对关于默认自变量 `x` (由 `findsym` 确定) 的符号标量函数 `f` 进行傅里叶变换，默认输出是关于符号变量 `w` 的函数 `F`。如果 `f = f(w)`，则 `fourier(f)` 返回一个关于 `t` 的函数 `F(t)`。

- `F = fourier(f,v)`：对关于默认自变量 `x` 的函数 `f` 进行傅里叶变换，得到的输出是关于符号变量 `v` 的函数 `F`。

- `F = fourier(f,u,v)`：对关于自变量 `u` 的函数 `f` 进行傅里叶变换，得到的输出是关于符号变量 `v` 的函数 `F`。

傅里叶反变换的求解通常使用函数 `ifourier` 来实现，其调用格式如下：

- `f = ifourier(F)`：对关于默认自变量 `w` 的符号标量函数 `F` 进行傅里叶反变换，默认输出是关于符号变量 `x` 的函数 `f`。如果 `F = F(w)`，则 `ifourier(F)` 返回一个关于 `t` 的函数 `f(t)`。

- `f = ifourier(F,u)`：对关于默认自变量 `x` 的函数 `F` 进行傅里叶反变换，得到的输出是关于符号变量 `u` 的函数 `f`。

● $f = \text{ifourier}(F,v,u)$: 对关于自变量 v 的函数 F 进行傅里叶反变换, 得到的输出是关于符号变量 u 的函数 f 。

【例】求傅里叶变换对: $f(x) = e^{-x^2} \leftrightarrow F(w) = \sqrt{\pi} e^{-w^2/4}$ 。

在命令窗输入:

```
>> syms x
>> f = exp(-x^2);
>> F = fourier(f)
ifourier(F)
```

运行结果:

```
F =
exp(-1/4*w^2)*pi^(1/2)
ans =
exp(-x^2)
```

6.8.2 拉普拉斯变换

拉普拉斯变换和拉普拉斯反变换的定义式如下:

$$X(s) = \int_{-\infty}^{\infty} x(t) e^{-st} dt$$

$$x(t) = \frac{1}{2\pi j} \int_{c-j\infty}^{c+j\infty} X(s) e^{js} ds$$

拉普拉斯变换的求解通常使用函数 `laplace` 来实现, 其调用格式如下:

● $L = \text{laplace}(F)$: 对关于默认自变量 t (由 `findsym` 确定) 的符号标量函数 F 进行拉普拉斯变换, 默认输出是关于符号变量 s 的函数 L 。如果 $F = F(s)$, 则 `laplace(F)` 返回一个关于 t 的函数 $L(t)$ 。

● $L = \text{laplace}(F,t)$: 对关于默认自变量 t 的函数 F 进行拉普拉斯变换, 得到的输出是关于符号变量 t 的函数 L 。

● $L = \text{laplace}(F,w,z)$: 对关于自变量 w 的函数 F 进行拉普拉斯变换, 得到的输出是关于符号变量 z 的函数 L 。

拉普拉斯反变换的求解通常使用函数 `ilaplace` 来实现, 其调用格式如下:

● $F = \text{ilaplace}(L)$: 对关于默认自变量 s 的符号标量函数 L 进行拉普拉斯反变换, 默认输出是关于符号变量 t 的函数 F 。如果 $L = L(t)$, 则 `ilaplace(L)` 返回一个关于 x 的函数 $F(x)$ 。选择 c 使所有 $L(s)$ 的奇点位于直线 $s = c$ 左侧。

● $F = \text{ilaplace}(L,y)$: 对关于默认自变量 s 的函数 L 进行拉普拉斯反变换, 得到的输出是关于符号变量 y 的函数 F 。

● $F = \text{ilaplace}(L,y,x)$: 对关于自变量 y 的函数 L 进行拉普拉斯反变换, 得到的输出是关于符号变量 x 的函数 F 。

【例】 求拉普拉斯变换对: $f(t) = e^{-at} \leftrightarrow F(s) = \frac{1}{s+a}$ 。

在命令窗输入:

```
>> syms a t x
>> f = exp(-a*t);
>> F = laplace(f,x)
ilaplace(F)
```

运行结果:

```
F =
1/(x+a)
ans =
exp(-a*t)
```

6.8.3 Z 变换

Z 变换和逆 Z 变换的定义式如下:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

$$x(n) = \frac{1}{2\pi j} \oint_{|z|=R} X(z)z^{n-1} dz$$

Z 变换的求解通常使用函数 `ztrans` 来实现, 其调用格式如下:

- $F = \text{ztrans}(f)$: 对关于默认自变量 n (由 `findsym` 确定) 的符号标量函数 f 进行 Z 变换, 默认输出是关于符号变量 z 的函数 F 。如果 $f = f(z)$, 则 `ztrans(f)` 返回一个关于 w 的函数 $F(w)$ 。
- $F = \text{ztrans}(f,w)$: 对关于默认自变量 n 的函数 f 进行 Z 变换, 得到的输出是关于符号变量 w 的函数 F 。
- $F = \text{ztrans}(f,k,w)$: 对关于自变量 k 的函数 f 进行 Z 变换, 得到的输出是关于符号变量 w 的函数 F 。

逆 Z 变换的求解通常使用函数 `iztrans` 来实现, 其调用格式如下:

- $f = \text{iztrans}(F)$: 对关于默认自变量 z (由 `findsym` 确定) 的符号标量函数 F 进行逆 Z 变换, 默认输出是关于符号变量 n 的函数 f 。如果 $F = F(n)$, 则 `iztrans(F)` 返回一个关于 k 的函数 $f(k)$ 。其中, R 为正实数, 使得函数 $F(z)$ 在圆 $|z| = R$ 之外解析。
- $f = \text{iztrans}(F,k)$: 对关于默认自变量 z 的函数 F 进行逆 Z 变换, 得到的输出是关于符号变量 k 的函数 f 。
- $f = \text{iztrans}(F,w,k)$: 对关于自变量 w 的函数 F 进行逆 Z 变换, 得到的输出是关于符号变量 k 的函数 f 。

【例】 求 Z 变换对: $f(n) = a^n \leftrightarrow F(z) = \frac{z/a}{z/a - 1}$ 。

在命令窗输入:

```
>> syms n a
>> f = a^n;
>> F = ztrans(f)
iztrans(F)
```

运行结果:

```
F =
(z/a)/(z/a-1)
ans =
a^n
```

第 7 章 基本绘图功能

MATLAB 提供了丰富的实现数据可视化的方法。使用交互式工具可以方便地将数据以图形形式显示出来。MATLAB 还提供了图形的注释和打印功能，甚至可以将图形结果输出成为各种形式的图片。

本章主要介绍 MATLAB 常用图形如二维或三维基本图形和特殊图形的绘制及处理方法。由于这些绘图函数的用法和功能很多，为使读者更好地掌握各种图形的绘制和处理方法，本章将对这些函数的调用格式作详细的介绍。

7.1 图形窗口

当用户使用一个简单的绘图命令绘图时，MATLAB 会自动输出一个显示图形的窗口。事实上，MATLAB 先创建了一个图形窗口，然后才在该窗口上进行绘图操作。通常不仅需要数据图形显示出来，还需要对图形进行适当的操作处理才能够满足特定的需要。本节将主要介绍图形窗口的基本操作功能。

7.1.1 图形窗口的创建与设置

创建一个图形对象时，MATLAB 将自动选择该图形对象的属性值。利用 `get` 函数可以获得当前图形对象的属性，如果需要修改某项属性，则可以通过 `set` 函数来实现。通常使用 `gcf` 命令获得当前图形的句柄以作为 `get`、`set` 函数的输入参量。

1. figure 函数

`figure` 函数用于创建一个新的图形对象，其调用格式如下：

- `figure`：使用默认属性值新建一个图形对象。
- `figure('PropertyName',propertyvalue, ...)`：使用指定属性值新建一个图形对象。对于未指定的属性，MATLAB 则使用默认属性。
- `figure(h)`：如果 `h` 为现有图形的句柄，`figure(h)` 则使由 `h` 确定的图形成为当前图形，使该图形可视并显示于屏幕的最前端，当前图形为图形输出的目标；如果 `h` 不是现有图形的句柄而是一个整数，`figure(h)` 则创建一个图形并将其赋值为句柄 `h`。
- `h = figure(...)`：返回图形对象的句柄。

2. gcf 函数

`gcf` 函数用于获取当前图形的句柄，其调用格式为 `h = gcf`。

3. get 函数

get 函数用于查询对象的属性，其常用的调用格式如下：

- get(h): 返回句柄 h 指定的图形对象的所有属性和当前值。
- get(h,'PropertyName'): 返回由 h 指定的图形对象的属性 PropertyName 的属性值。

4. set 函数

set 函数用于设置对象的属性，其常用的调用格式为

set(h,'PropertyName',PropertyValue, ...)

该函数设置由 h 指定的对象的属性名 PropertyName 的属性值为 PropertyValue。h 可以为句柄向量，这种情况下将设置所有对象的属性值。

【例】 图形窗口的创建、查看与设置。

在命令窗输入：

```
>> figure
>> x = 0:pi/10:2*pi;
>> y = sin(x);
>> plot(x,y,'k-*')
```

运行结果如图 7-1 所示。

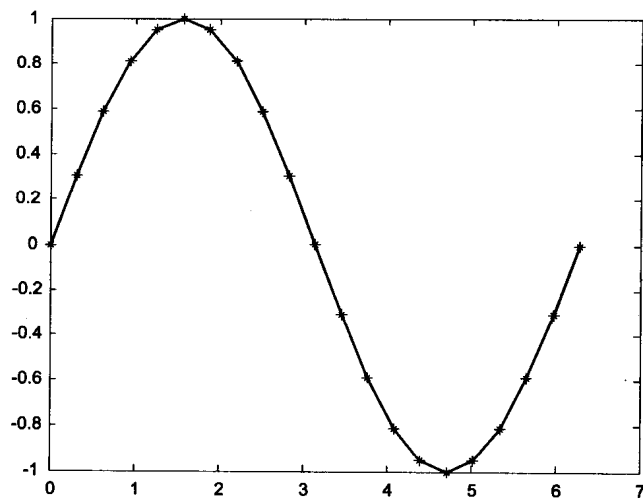


图 7-1 函数 $y = \sin(x)$ 的波形图

继续在命令窗输入：

```
>> get(findobj('Type','line'),'color')
```

运行结果：

```
ans =
     0     0     0
```

再在命令窗输入：

```
>> set(findobj('Type','line'),'Color','b')
>> set(findobj('Type','line'),'linestyle',':')
```

运行结果如图 7-2 所示。

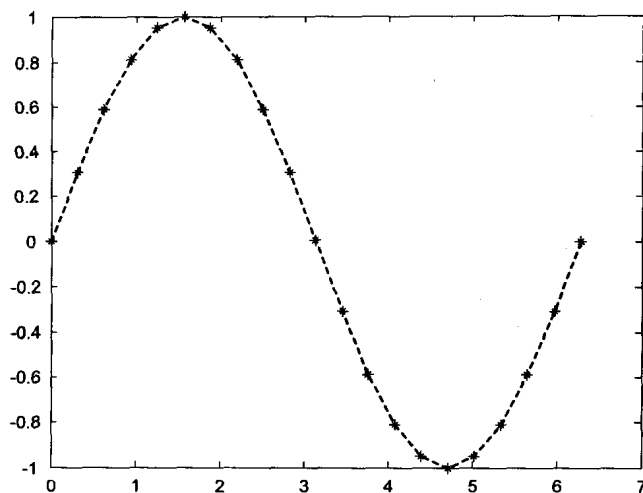


图 7-2 属性设置后的波形图

继续在命令窗输入：

```
>> get(findobj('Type','line'),'color')
```











运行结果：

```
ans =  
      0      0      1
```



7.1.2 图形窗口的工具栏













图形窗口的工具栏由图形工具栏、照相机工具栏和绘图编辑工具栏三部分组成。个别工具栏按钮功能与 MATLAB 标准工具栏按钮的功能相同，这里不作介绍。本小节将对作图窗口特有的工具栏按钮进行详细介绍。

1. 图形工具栏

- ：进入图形编辑模式。
- ：放大视图。
- ：缩小视图。
- ：平移图形。
- ：三维旋转。
- ：光标取点。
- ：插入颜色条。
- ：插入绘图标记。
- ：隐藏绘图工具。
- ：显示绘图工具。

2. 照相机工具栏

- ：旋转照相机。
- ：改变光照位置偏移。

-  : 移动照相点。
-  : 水平或竖直移动照相机。
-  : 前后移动照相机。
-  : 缩放照相机。
-  : 旋转照相机。
-  : 以 X 轴为主。
-  : 以 Y 轴为主。
-  : 以 Z 轴为主。
-  : 无主轴。
-  : 切换景物光。
-  : 正交投影。
-  : 透视投影。
-  : 复位照相机和景物光。
-  : 停止照相机和光线变动。

3. 绘图编辑工具栏

-  : 插入直线。
-  : 插入箭头。
-  : 插入双箭头。
-  : 插入带文字的箭头。
-  : 插入文字。
-  : 插入矩形。
-  : 插入椭圆。
-  : 增加坐标轴 pin。
-  : 排列分布。

7.1.3 图形窗口的主菜单

本小节将详细介绍图形窗口的主菜单功能。

1. File 菜单

- New: 新建 M 文件、图形窗口、工作区间变量、GUI。
- Open: 打开图形文件。
- Close: 关闭当前图形窗口。
- Save: 保存当前图形文件。
- Save As: 将当前图形文件另存为。
- Generate M-File: 生成 M 文件绘图函数。
- Import Data: 导入数据。
- Save Workspace As: 将图形数据保存为.mat 文件。
- Preferences: 环境设置。

- Export Setup: 图形输出属性设置。

- Print Preview: 打印预览。

- Print: 打开打印对话框。

2. Edit 菜单

- Undo: 撤消。

- Redo: 恢复。

• Cut、Copy、Paste、Clear Clipboard、Delete、Select All: 分别实现图形数据的剪切、复制、粘贴、清除剪贴板、删除、全选操作, 进入图形编辑模式时才可用。

- Copy Figure: 复制整个图形。

- Copy Option: 设置复制图形选项。

- Figure Properties: 打开图形属性编辑器, 如图 7-3 所示。

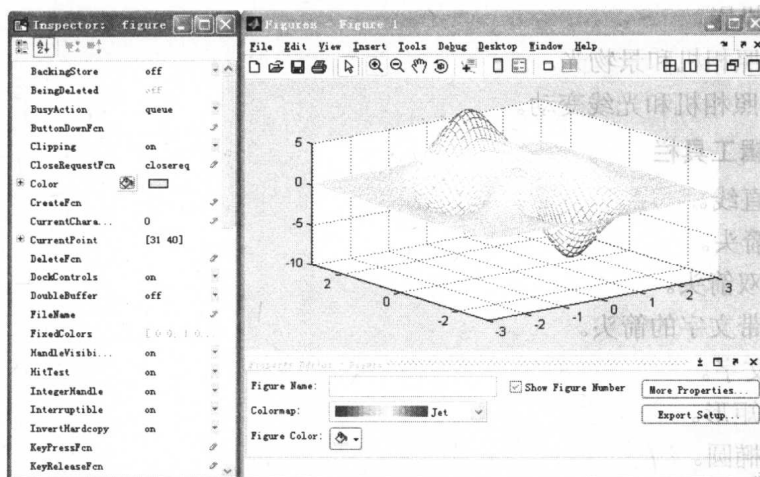


图 7-3 图形属性编辑器

- Axes Properties: 打开坐标轴属性编辑器, 如图 7-4 所示。

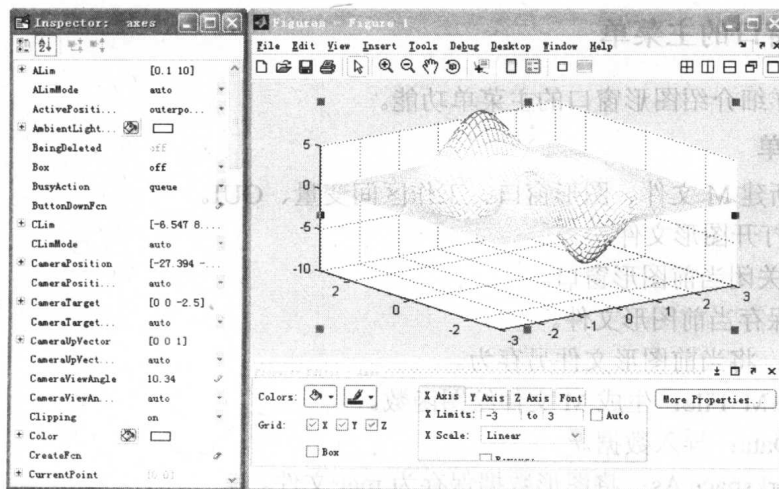


图 7-4 坐标轴属性编辑器

- Current Object Properties: 打开当前对象的属性编辑器。

- Colormap: 打开彩色图编辑器。
- Find Files: 查找文件。
- Clear Figure、Clear Command Window、Clear Command History、Clear Workspace: 分别用于清除图形、清除命令窗、清除历史命令记录、清除工作区间。

3. View 菜单

- Figure Toolbar: 显示图形工具栏。
- Camera Toolbar: 显示照相机工具栏。
- Plot Edit Toolbar: 显示绘图编辑工具栏。
- Figure Palette: 打开图形调色板, 见图 7-5。
- Plot Browser: 打开绘图浏览器, 见图 7-5。
- Property Editor: 打开属性编辑器, 见图 7-5。

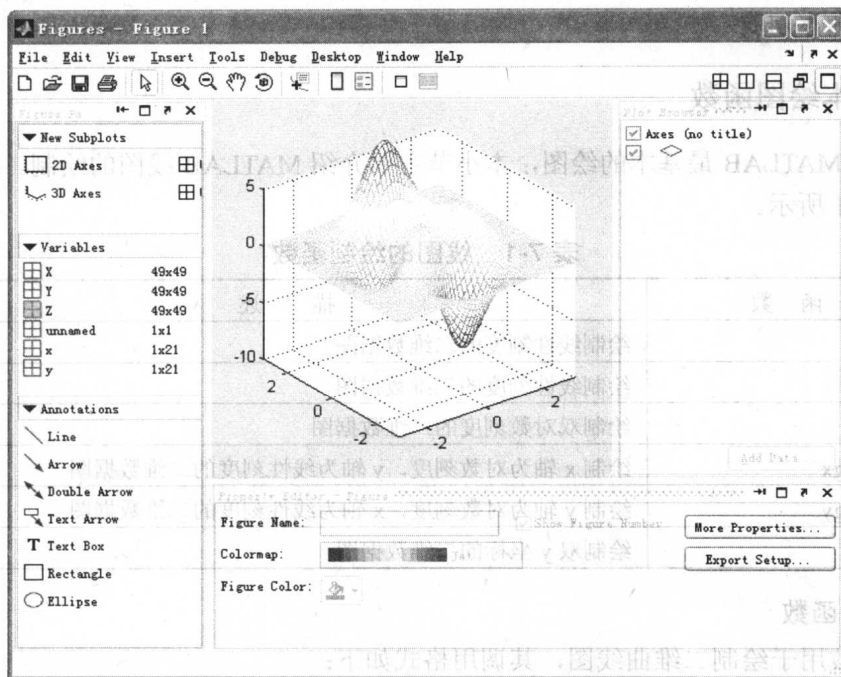


图 7-5 图形调色板、绘图浏览器和属性编辑器

4. Insert 菜单

该菜单中大部分功能与工具栏按钮功能相同, 这里不重复介绍。

- X Label: 标注 X 坐标。
- Y Label: 标注 Y 坐标。
- Z Label: 标注 Z 坐标。
- Title: 标注标题。
- Axes: 添加坐标轴。
- Light: 添加光照。

5. Tools 菜单

该菜单绝大部分功能与工具栏按钮功能相同。

6. Window 菜单

该菜单用于在命令窗和图形窗之间切换。

7. Help 菜单

该菜单用于查询有关图形功能的相关帮助。

7.2 绘制二维图形

本节主要介绍最基本的二维绘图函数以及通用的图形处理方法，这些处理方法同样适用于特殊图形的处理。

7.2.1 基本绘图函数

线图是 MATLAB 最基本的绘图，本小节主要介绍 MATLAB 线图的绘制。绘制线图的函数如表 7-1 所示。

表 7-1 线图的绘制函数

函 数	描 述
plot	绘制线性刻度的二维数据图
plot3	绘制线性刻度的三维数据图
loglog	绘制双对数刻度的二维数据图
semilogx	绘制 x 轴为对数刻度，y 轴为线性刻度的二维数据图
semilogy	绘制 y 轴为对数刻度，x 轴为线性刻度的二维数据图
plotyy	绘制双 y 坐标的二维数据图

1. plot 函数

plot 函数用于绘制二维曲线图，其调用格式如下：

- plot(Y)：如果 Y 为实数，绘制纵坐标为 Y 的列向量，横坐标为列索引的二维图像；如果 Y 为复数，plot(Y)等价于 plot(real(Y),imag(Y))。在所有 plot 函数的其他用法中，虚数部分将被忽略。

- plot(X1,Y1,...)：绘制所有由数据对 Xn 与 Yn 构成的曲线。如果只有 Xn 或 Yn 是矩阵，则根据向量与矩阵的行或列是否匹配绘制向量对于矩阵的行或列构成的图像。如果 Xn 是标量且 Yn 为向量，则在过 Xn 的垂线上绘制离散点 Yn。

- plot(X1,Y1,LineSpec,...)：绘制所有由 Xn、Yn、LineSpec 定义的曲线，其中 LineSpec 用于指定线型、标记符号和曲线颜色。可以混合使用 Xn、Yn、LineSpec 与 Xn、Yn，如 plot(X1,Y1,X2,Y2,LineSpec,X3,Y3)。

- plot(...,'PropertyName',PropertyValue,...)：对所有由 plot 函数创建的图形的属性进行设置。

- `plot(axes_handle, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。
- `h = plot(...)`: 返回图形对象的句柄列向量，一条线对应一个句柄。

【例】 `plot` 函数绘图。

在命令窗输入：

```
>> x = -pi:pi/10:pi;
```

```
>> y = sin(x);
```

```
>> plot(x,y,'-p','LineWidth',2,'MarkerEdgeColor','b','MarkerFaceColor','k','MarkerSize',6)
```

运行结果如图 7-6 所示。

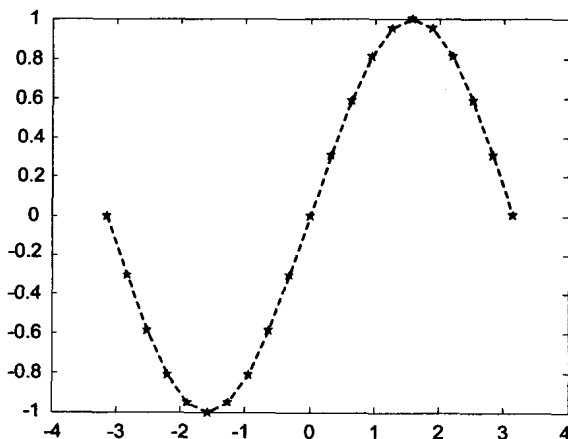


图 7-6 `plot` 函数绘图

2. `plotyy` 函数

`plotyy` 函数用于绘制双 y 坐标的二维曲线图，其调用格式如下：

- `plotyy(X1,Y1,X2,Y2)`: 绘制由 `X1`、`Y1` 和 `X2`、`Y2` 确定的两组曲线，其中 `X1`、`Y1` 的坐标轴在图形窗口的左侧，`X2`、`Y2` 的坐标轴在图形窗口的右侧。
- `plotyy(X1,Y1,X2,Y2,function)`: 使用 `function` 指定的绘图函数绘图，`function` 可以是函数句柄或字符串 `plot`、`semilogx`、`semilogy`、`loglog`、`stem` 或符合语法 `h = function(x,y)` 的 MATLAB 函数。
- `plotyy(X1,Y1,X2,Y2,'function1','function2')`: 在左侧坐标轴使用 `function1(X1,Y1)` 绘图，在右侧坐标轴使用 `function2(X1,Y1)` 绘图。
- `[AX,H1,H2] = plotyy(...)`: 返回在 `AX` 中创建的两个坐标轴的句柄和 `H1` 及 `H2` 中每个图形绘图对象的句柄，`AX(1)` 为左侧轴，`AX(2)` 为右侧轴。

【例】 `plotyy` 函数绘图。

在命令窗输入：

```
>> x = 0:0.01:10;
```

```
>> y1 = 100*exp(-0.1*x).*sin(2*x);
```

```
>> y2 = 0.4*exp(-x).*sin(20*x);
```

```
>> [AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
```

```
>> set(get(AX(1),'Ylabel'),'String','Slow Decay')
```

```
>> set(get(AX(2),'Ylabel'),'String','Fast Decay')
```

```
>> xlabel('Time (\musec)')
>> title('Multiple Decay Rates')
>> set(H1,'LineStyle','-.')
>> set(H2,'LineStyle',':')
```

运行结果如图 7-7 所示。

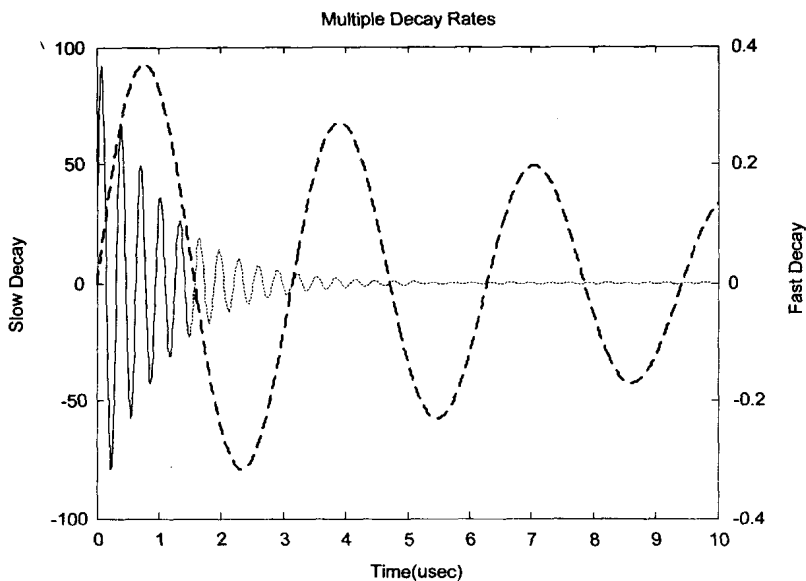


图 7-7 plotyy 函数绘图

3. semilogx 函数与 semilogy 函数

semilogx 函数或 semilogy 函数分别用于绘制 x 轴为对数坐标或 y 轴为对数坐标的二维曲线图，其调用格式如下：

- **semilogx(Y)**: 绘制 x 轴刻度是以 10 为底的对数，y 轴刻度是线性刻度的二维曲线图。如果 Y 为实数，则绘制纵坐标为 Y 的列向量，横坐标为列索引的二维 x 轴对数图；如果 Y 为复数，semilogx(Y) 等价于 semilogx(real(Y), imag(Y))。在所有 semilogx 函数的其他用法中，虚数部分将被忽略。

- **semilogy(...)**: 绘制 y 轴刻度是以 10 为底的对数，x 轴刻度是线性刻度的二维曲线图。

- **semilogx(X1, Y1, ...)**: 绘制所有由数据对 Xn 与 Yn 构成的曲线。如果只有 Xn 或 Yn 是矩阵，则根据向量与矩阵的行或列是否匹配绘制向量对于矩阵的行或列构成的图像。

- **semilogx(X1, Y1, LineSpec, ...)**: 绘制所有由 Xn、Yn、LineSpec 定义的曲线，其中 LineSpec 用于指定线型、标记符号和曲线颜色。

- **semilogx(..., 'PropertyName', PropertyValue, ...)**: 对所有由 semilogx 函数创建的图形的属性进行设置。

- **h = semilogx(...)** 与 **h = semilogy(...)**: 返回图形对象的句柄向量，一条线对应一个句柄。

【例】 semilogx 函数绘图。

在命令窗输入：

```
>> x = 0:1:10;
>> semilogx(x,1./(1+x))
>> grid
```

运行结果如图 7-8 所示。

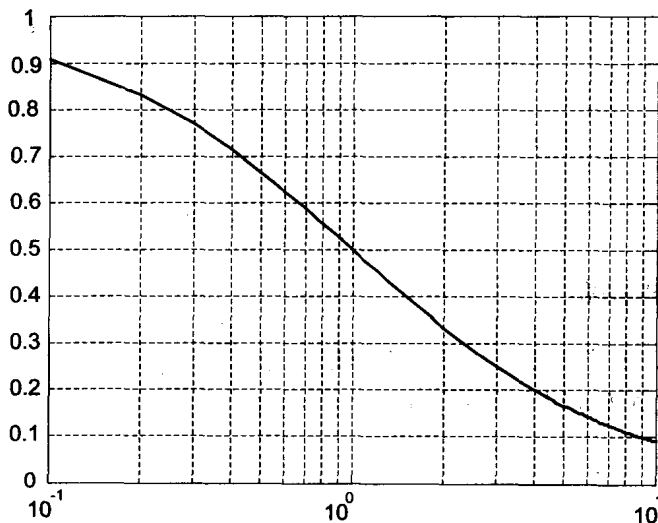


图 7-8 semilogx 函数绘图

4. loglog 函数

- **loglog(Y)**: 绘制 x 轴 y 轴刻度均是以 10 为底的对数的二维曲线图。如果 Y 为实数，则绘制纵坐标为 Y 的列向量，横坐标为列索引的二维对数图；如果 Y 为复数，则 **loglog(Y)** 等价于 **loglog(real(Y),imag(Y))**。在所有 **loglog** 函数的其他用法中，虚数部分将被忽略。

- **loglog(X1,Y1,...)**: 绘制所有由数据对 X_n 与 Y_n 构成的曲线。如果只有 X_n 或 Y_n 是矩阵，则根据向量与矩阵的行或列是否匹配绘制向量对于矩阵的行或列构成的图像。

- **loglog(X1,Y1,LineSpec,...)**: 绘制所有由 X_n 、 Y_n 、**LineSpec** 定义的曲线，其中 **LineSpec** 用于指定线型、标记符号和曲线颜色。可以混合使用 X_n 、 Y_n 、**LineSpec** 与 X_n 、 Y_n ，如 **loglog(X1,Y1,X2,Y2,LineSpec,X3,Y3)**。

- **loglog(...,'PropertyName',PropertyValue,...)**: 对所有由 **loglog** 函数创建的图形的属性进行设置。

- **h = loglog(...)**: 返回图形对象的句柄列向量，一条线对应一个句柄。

【例】 loglog 函数绘图。

在命令窗输入：

```
>> x = 0:0.01:1;
>> loglog(x,1./(1+x))
>> grid
```

运行结果如图 7-9 所示。

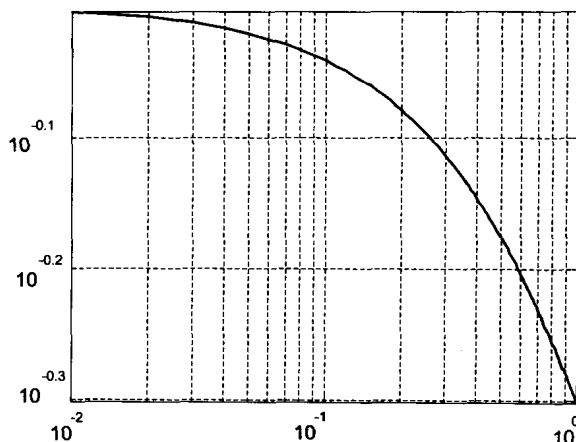


图 7-9 loglog 函数绘图

7.2.2 图形处理函数

实际应用中，通常需要对显示的数据图形进行一定的处理，如改变线型、颜色以及图形标注等。MATLAB 为用户提供了丰富的图形处理函数，本小节将主要介绍这些函数通常的用法和功能。

1. 图形标注

函数 `xlabel(ylabel, zlabel)` 分别用于实现图形的 $x(y, z)$ 坐标轴标注，其常用调用格式为
`xlabel('string', 'PropertyName', PropertyValue, ...)`

其中，`string` 为待标注的字符串，并用 `Property Name` 和 `Property Value` 指定文字属性。

函数 `title` 用于实现图形的标题标注，其常用调用格式为

`title('string', 'Property Name', Property Value, ...)`

其中，`string` 为待标注的字符串，并用 `Property Name` 和 `Property Value` 指定文字属性。

【例】坐标轴标注和标题标注。

在命令窗输入：

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine Function','FontSize',12)
```

运行结果如图 7-10 所示。

函数 `text` 用于实现图形任意位置的文字标注，其常用调用格式如下：

- `text(x,y,'string')`：在指定点 (x,y) 处添加文字标注。
- `text(x,y,z,'string')`：在三维坐标系内添加文字标注。
- `text(x,y,z,'string','PropertyName',PropertyValue...)`：在坐标系内添加文字标注，并指定文字属性。

- `text('PropertyName',Property Value...)`: 完全忽略坐标系并用 Property Name 和 Property Value 指定所有属性。
- `h = text(...)`: 返回一个 text 对象的句柄列向量, 一个对象对应一个句柄。

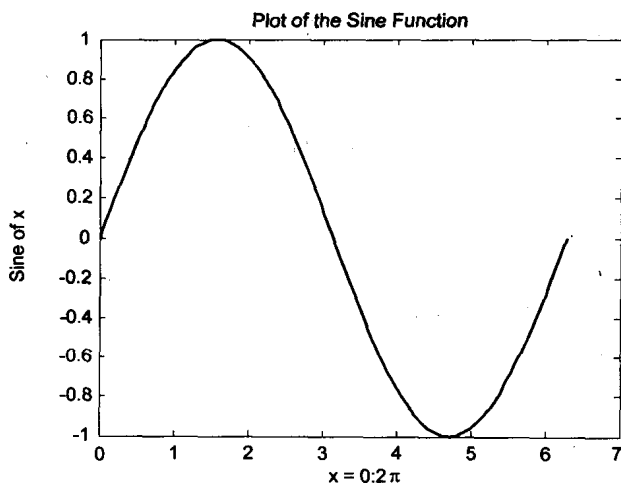


图 7-10 坐标轴标注和标题标注

【例】 在最大值和最小值处标注。

在命令窗输入:

```
>> Z = peaks;
>> h = plot(Z(:,33));
>> x = get(h,'XData');           %获取绘图数据
>> y = get(h,'YData');
>> imin = find(min(y) == y);      %寻找最大值和最小值索引
>> imax = find(max(y) == y);
>> text(x(imin),y(imin),[' Minimum = ',num2str(y(imin))], ...
'VerticalAlignment','middle', 'HorizontalAlignment','left', 'FontSize',10)
>> text(x(imax),y(imax),['Maximum = ',num2str(y(imax))], ...
'VerticalAlignment','bottom', 'HorizontalAlignment','right', 'FontSize',10)
```

运行结果如图 7-11 所示。

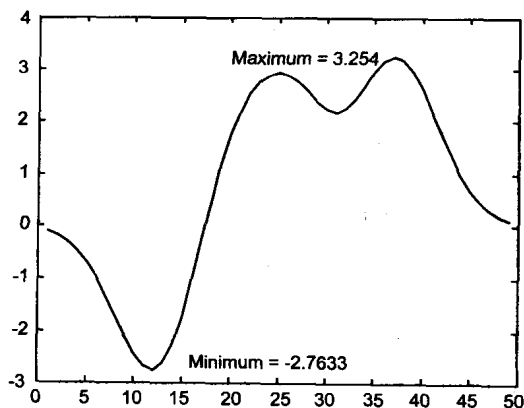


图 7-11 标注最大值和最小值

2. 指定线型和颜色

如果在绘图时需要指定图形的线型和颜色,则可以使用类似 `plot(x,y,'color_style_marker')` 形式的语句来直接实现。`color_style_marker` 是一个包含 1~4 个字符的字符串,它由颜色、线型和标记类型组合而成,它们的符号和含义如下:

- 颜色字符串: 'c'、'm'、'y'、'r'、'g'、'b'、'w'、'k' 分别对应青色、品红色、黄色、红色、绿色、蓝色、白色、黑色。

- 线型字符串: '-'、'--'、'.'、':-' 分别对应实线、虚线、点线、点划线。可以通过 `LineWidth` 属性设置线宽。

- 直接标记类型: '+'、'o'、'*'、'x' 分别对应加号、圆圈、星号、叉号; 特殊标记类型: 's'、'd'、'^'、'v'、'>'、'<'、'p'、'h' 分别对应正方形、钻石、上三角、下三角、右三角、左三角、五角星、六角星。可以通过 `MarkerEdgeColor`、`MarkerFaceColor` 和 `MarkerSize` 属性设置标记的边框色、填充色和大小。

【例】 指定线型、颜色和标记。

在命令窗输入:

```
>> x = 0:pi/20:2*pi;
```

```
>> plot(x,sin(x),'r:h')
```

运行结果如图 7-12 所示。

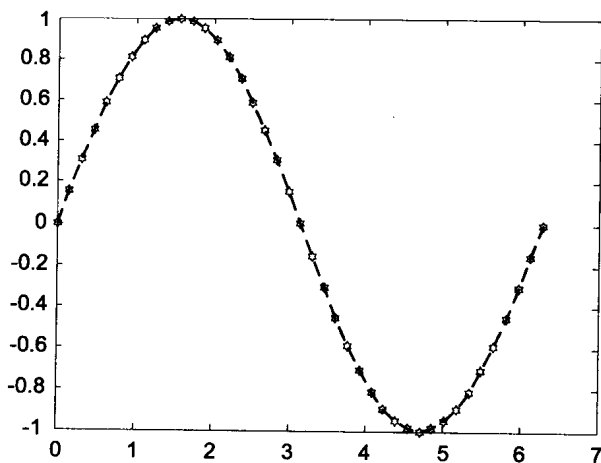


图 7-12 指定线型、颜色和标记

3. 多张图形的绘制

如果要显示多张图形,则可以通过创建多个图形窗口来分别显示或者在一个图形窗口内同时显示多张图形。

如果要新建一个图形窗口并在该窗口中显示下一张图,则可以使用 `figure` 命令来实现。更为一般的方法是,在将要绘制的每张图形之前,使用命令 `figure(n)` 来创建第 `n` 个图形窗口, `n` 从 1 开始。

如果需要在在一个图形窗口内同时显示多张图形,则可以使用 `subplot(m,n,p)` 命令(或 `subplot(mnp)` 命令)来实现。该命令将一个图形窗口分割成 `m×n` 个小窗口,并指定第 `p` 个窗

口(p 也可以是几个序号的组合)为当前图形窗口。窗口序号为从第一行开始,依次向下一行计数。

【例】 多张图形的绘制。

在命令窗输入:

```
>> figure;  
>> x = 0:pi/30:2*pi;  
>> subplot(2,2,[1 3])  
>> plot(x,3*x)  
>> subplot(2,2,2)  
>> plot(x,sin(x))  
>> subplot(2,2,4)  
>> plot(x,cos(x))
```

运行结果如图 7-13 所示。

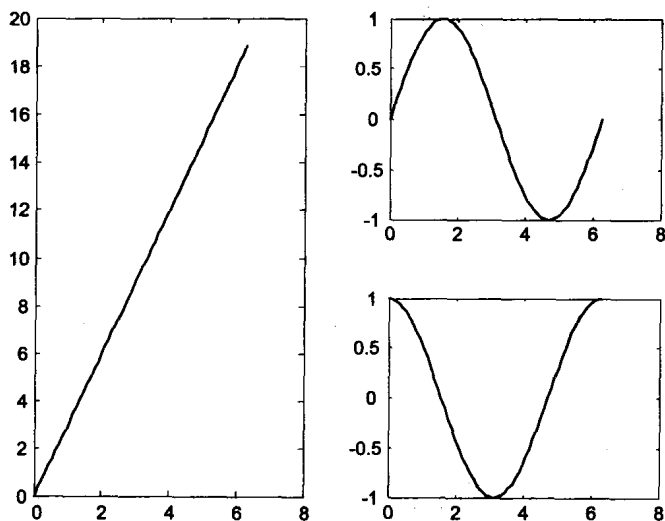


图 7-13 多张图形的绘制

4. 坐标轴控制

当绘制一个图形时, MATLAB 会自动完成坐标轴的范围设定和刻度划分。为满足某些特殊需要, 用户可以使用坐标轴控制函数实现坐标的设置。

坐标轴范围的设置可以通过 axis 函数实现, 其常用调用格式为

```
axis([xmin,xmax,ymin,ymax])
```

其中, 坐标轴范围的最大值必须大于最小值。如果不需要设置某个最大值(最小值), 则可以使用 Inf(-Inf), MATLAB 会根据数据范围进行自动设置。

如果需要对图形的坐标轴刻度进行指定, 则可以通过对 xtick 或 ytick 属性的设置来实现。例如, set(gca,'ytick',v)用递增的向量 v 来表示 y 刻度点, 刻度可以不是均匀划分的。下面通过实例来说明坐标轴的控制方法。

【例】 坐标轴的控制。

在命令窗输入：

```
>> x = -pi:1:pi;  
>> y = sin(x);  
>> plot(x,y)  
>> axis([-pi pi -inf inf])  
>> set(gca,'xtick',[-3.1:5:4])
```

运行结果如图 7-14 所示。

默认情况下，MATLAB 在一个与图形窗口具有相同纵横比的矩形坐标轴区域内显示图形。用户可以根据需要使用 axis 函数改变显示图形的纵横比。常用的纵横比设置命令如下：

- axis square: 设置图形显示区域为正方形。
- axis equal: 设置坐标轴具有相同长度的刻度。
- axis tight: 设置坐标轴的范围等于数据的范围。

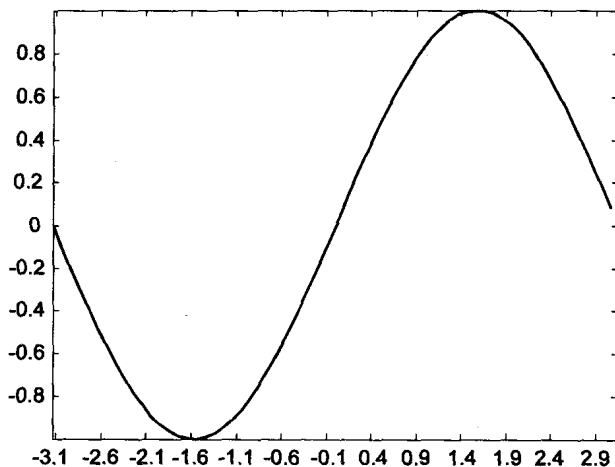


图 7-14 控制坐标轴

使用 grid 函数可以添加或者取消坐标的网格线。命令 grid on 用于添加网格线，命令 grid off 用于取消网格线，命令 grid 可在添加和取消网格线之间进行切换。

【例】 设置坐标轴的纵横比。

在命令窗输入：

```
>> t = 0:pi/20:2*pi;  
>> subplot(221)  
>> plot(sin(t),2*cos(t))  
>> grid on  
>> subplot(222)  
>> plot(sin(t),2*cos(t))  
>> grid on  
>> axis square
```

```
>> subplot(223)
>> plot(sin(t),2*cos(t))
>> grid on
>> axis equal
>> subplot(224)
>> plot(sin(t),2*cos(t))
>> grid on
>> axis equal tight
```

运行结果如图 7-15 所示。

使用 `box` 函数可以显示或隐藏坐标边界。命令 `box on` 用于显示坐标边界，命令 `box off` 用于隐藏坐标边界。

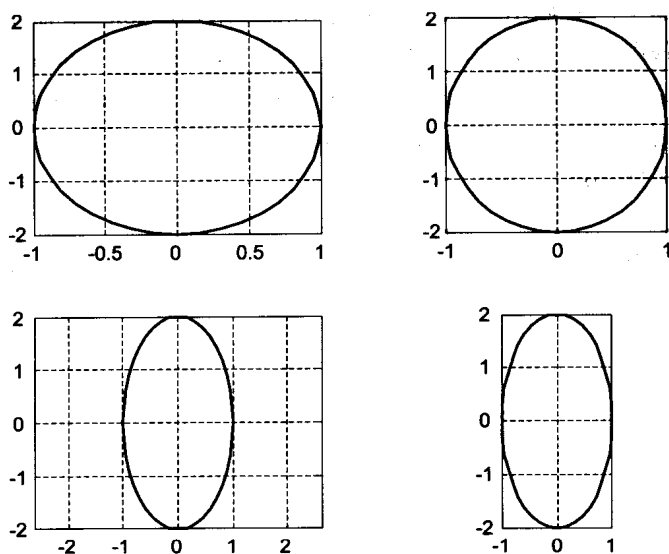


图 7-15 设置坐标轴的纵横比

【例】 使用 `box` 函数。

在命令窗输入：

```
>> x = -pi:1:pi;
>> y = sin(x);
>> subplot(211)
>> plot(x,y)
>> box on
>> subplot(212)
>> plot(x,y)
>> box off
```

运行结果如图 7-16 所示。

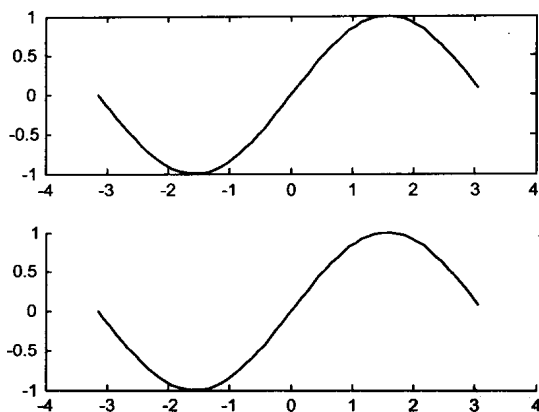


图 7-16 使用 box 函数

如果要在一张图上同时显示多个二维曲线，则可以使用 `hold on` 命令，接着绘制所有待显示的曲线。当这张图绘制完毕时，则使用 `hold off` 关闭该模式。如果需要标明这些曲线，则可以使用 `legend` 函数加以区分。`legend` 函数的常用调用格式为 `legend('string1','string2', ...)`，如果有 n 条曲线，则用 `'string1'、'string2'... 'stringn'` 来标注。

【例】 绘制多条曲线。

在命令窗输入：

```
>> x = 0:pi/10:2*pi;  
>> y = sin(x);  
>> y2 = sin(x-.25);  
>> y3 = sin(x-.5);  
>> plot(x,y,'k-*')  
>> hold on  
>> plot(x,y2,'b:o')  
>> plot(x,y3,'r-p')  
>> hold off  
>> legend('sin(x)','sin(x-.25)','sin(x-.5)')
```

运行结果如图 7-17 所示。

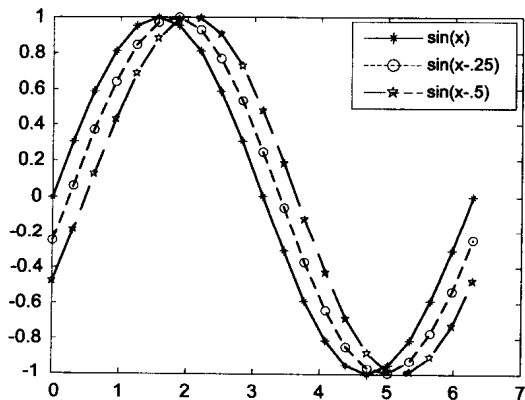


图 7-17 绘制多条曲线

7.3 绘制三维图形

本节主要介绍基本三维绘图的方法。有关三维基本绘图的函数如表 7-2 所示。

表 7-2 三维基本绘图函数

函 数	描 述
mesh, surf	绘制网格图(曲面图)
meshc, surfc	同时绘制网格图(曲面图)和等高线图
meshz	绘制带垂帘的网格图
surf1	绘制带光照的曲面图

7.3.1 三维曲线图

plot3 函数用于绘制三维曲线图，其调用格式如下：

- plot3(X1,Y1,Z1, ...): 在三维空间绘制由数据点 X1、Y1、Z1 确定的一条或多条曲线，其中 X1、Y1、Z1 可以是向量或矩阵。
- plot3(X1,Y1,Z1,LineSpec, ...): 绘制所有由 Xn、Yn、Zn、LineSpec 定义的三维曲线，其中 LineSpec 用于指定线型、标记符号和曲线颜色。
- plot3(...,'PropertyName',PropertyValue, ...): 对所有由 plot3 函数创建的图形的属性进行设置。
- h = plot3(...): 返回图形对象的句柄列向量，一个对象对应一个句柄。

【例】 使用 plot3 函数绘图。

在命令窗输入：

```
>> t = 0:pi/50:6*pi;
>> plot3(t.*sin(3*t),t.*cos(3*t),t)
>> grid
```

运行结果如图 7-18 所示。

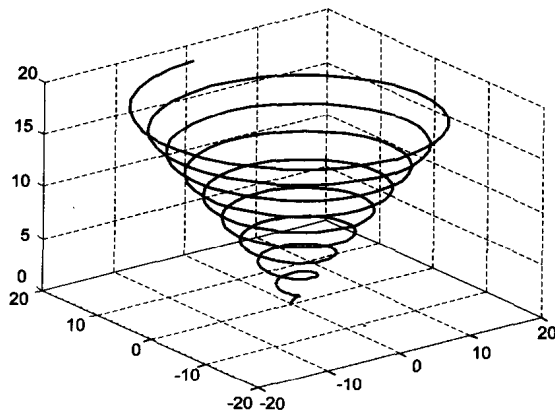


图 7-18 使用 plot3 函数绘图

7.3.2 三维网格图

mesh 函数用于绘制三维网格图, meshc 函数用于同时绘制三维网格图和等高线图, meshz 函数用于绘制带垂帘的三维网格图, 它们的调用格式如下:

- mesh(X,Y,Z): 绘制颜色和曲面的高度 Z 成正比的三维网格图。如果 X 与 Y 均为向量, $\text{length}(X) = n$ 、 $\text{length}(Y) = m$, 其中 $[m,n] = \text{size}(Z)$, $(X(j), Y(i), Z(i,j))$ 则为网格线的交点。X 与 Y 分别对应 Z 的列向量; 如果 X 与 Y 均为矩阵, $(X(i,j), Y(i,j), Z(i,j))$ 则为网格线的交点。

- mesh(Z): 绘制由 $X = 1:n$ 、 $Y = 1:m$ 以及 Z 确定的网格图。其中, $[m,n] = \text{size}(Z)$, 高度 Z 为定义在直角坐标网上的单值函数, 网格颜色和曲面的高度成正比。

- mesh(...,C): 绘制颜色由矩阵 C 确定的三维网格图。MATLAB 通过对 C 的数据进行线性处理, 从当前色彩图中获得颜色。如果 X、Y、Z 为矩阵, 则必须与 C 具有相同的尺寸。

- mesh(...,'PropertyName',PropertyValue, ...): 对指定的曲面属性进行设置, 允许一次设置多个属性。

- mesh(axes_handles, ...): 在句柄 axes_handle 指定的坐标轴内绘图。

- meshc(...): 在网格图下绘制等高线。

- meshz(...): 在网格图周围绘制垂帘。

- h = mesh(...), h = meshc(...), h = meshz(...): 返回曲面图形对象的句柄。

【例】 绘制网格图。

在命令窗输入:

```
>> [X,Y] = meshgrid(-3:125:3);
```

```
>> Z = peaks(X,Y);
```

```
>> mesh(X,Y,Z);
```

```
>> figure;
```

```
>> meshc(X,Y,Z);
```

```
>> figure;
```

```
>> meshz(X,Y,Z);
```

运行结果如图 7-19~图 7-21 所示。

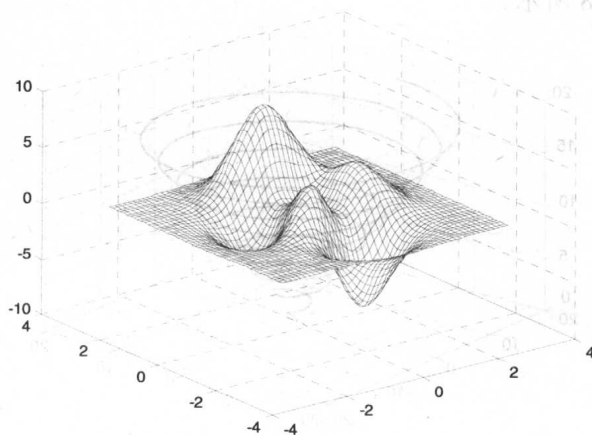


图 7-19 网格图

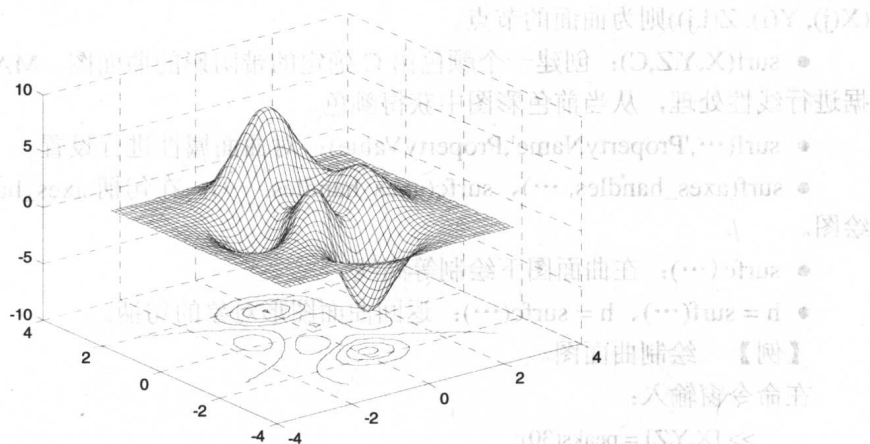


图 7-20 网格图与等高线图

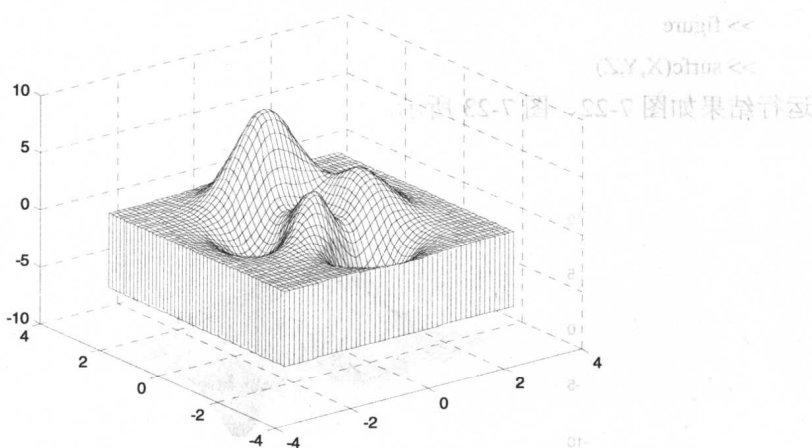


图 7-21 带垂帘的网格图

7.3.3 三维曲面图

`surf` 函数和 `surfc` 函数用于绘制带阴影的三维曲面图，而 `surf1` 函数则用于绘制带彩色光照的三维曲面图。

1. `surf`、`surfc` 函数

`surf` 函数用于绘制带阴影的三维曲面图，`surfc` 函数用于同时绘制带阴影的曲面图和等高线图。它们的调用格式如下：

- `surf(Z)`：创建一个由矩阵 `Z` 的 `z` 部分确定的三维带阴影的曲面图，`x = 1:n`、`y = 1:m`，其中 `[m,n] = size(Z)`。高度 `Z` 为定义在几何直角坐标网上的单值函数，且曲面颜色和高度 `Z` 成正比。

- `surf(Z,C)`：绘制高度 `Z` 为定义在几何直角坐标网上的单值函数的曲面图，其颜色由与 `Z` 尺寸相同的矩阵 `C` 确定。

- `surf(X,Y,Z)`：创建一个高度和颜色由 `Z` 确定的曲面图。`X`、`Y` 为确定曲面的 `x` 或 `y` 部分的向量或矩阵。如果 `X` 与 `Y` 均为向量，`length(X) = n`、`length(Y) = m`，其中 `[m,n] = size(Z)`，

(X(j), Y(i), Z(i,j))则为曲面的节点。

- `surf(X,Y,Z,C)`: 创建一个颜色由 `C` 确定的带阴影的曲面图。MATLAB 通过对 `C` 的数据进行线性处理, 从当前色彩图中获得颜色。

- `surf(...,'PropertyName',PropertyValue)`: 对曲面属性进行设置。

- `surf(axes_handles, ...)`、`surfc(axes_handles, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。

- `surfc(...)`: 在曲面图下绘制等高线。

- `h = surf(...)`、`h = surfc(...)`: 返回曲面图形对象的句柄。

【例】 绘制曲面图。

在命令窗输入:

```
>> [X,Y,Z] = peaks(30);
```

```
>> surf(X,Y,Z)
```

```
>> figure
```

```
>> surfc(X,Y,Z)
```

运行结果如图 7-22、图 7-23 所示。

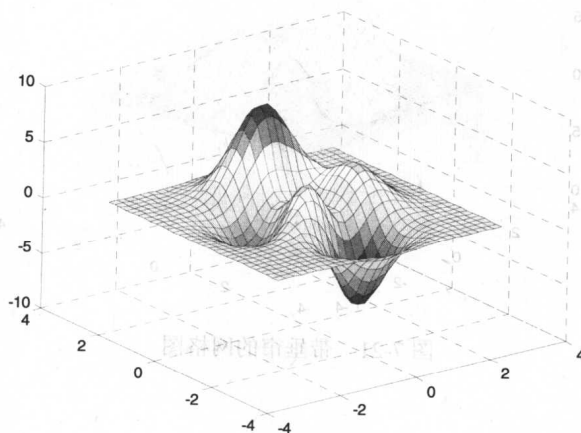


图 7-22 曲面图

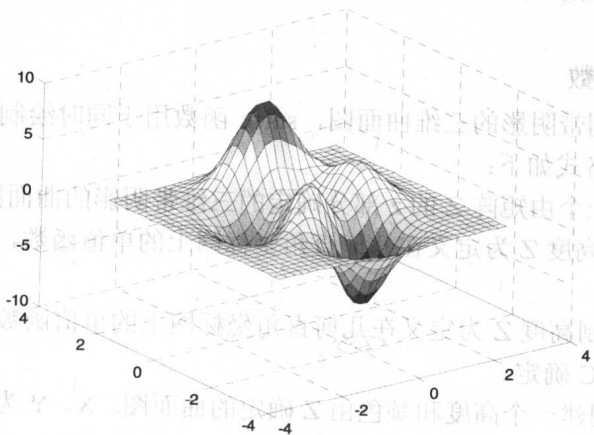


图 7-23 曲面图与等高线图

2. surf1 函数

surf1 函数用于绘制带彩色光照的三维曲面图，其调用格式如下：

- `surf1(Z)`、`surf1(X,Y,Z)`：创建一个带阴影的三维曲面，光源方向和光照系数为默认值。
`X`、`Y`、`Z` 为定义曲面 `x`、`y`、`z` 部分的向量或矩阵。
- `surf1(...,'light')`：利用 MATLAB 光照对象创建一个有颜色和光照的曲面。该结果不同于默认光照法 `surf1(...,'cdata')`，因为后者将曲面的颜色数据改为表面的反射系数。
- `surf1(...,s)`：指定光源的方向。`s` 为指定曲面到光源方向的二维向量[azimuth elevation]或三维向量[sx sy sz]。`s` 的默认值为从当前视角开始逆时针 45° 。
- `surf1(X,Y,Z,s,k)`：指定反射系数常数 `k`。`k` 为定义环境光系数、漫反射系数、镜面反射系数与镜面光亮系数相对作用的四元素向量。`k = [ka kd ks shine]`并且默认值为 `[.55,.6,.4,10]`。
- `h = surf1(...)`：返回曲面图形对象的句柄。

【例】 绘制光照曲面图。

在命令窗输入：

```
>> [X,Y,Z] = peaks(30);
>> surf1(X,Y,Z)
>> shading interp
>> colormap copper
```

运行结果如图 7-24 所示。

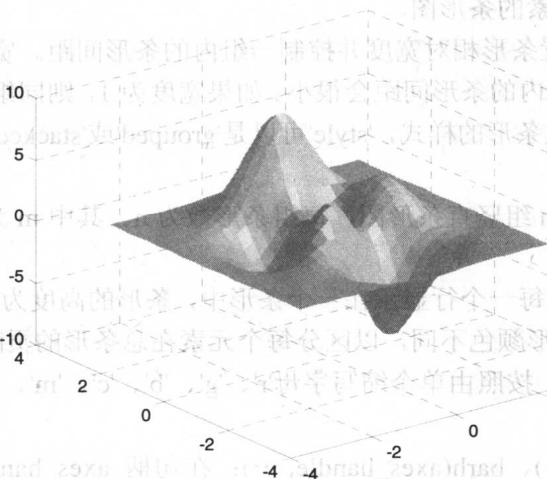


图 7-24 光照曲面图

7.4 绘制特殊图形

除了基本二维和三维绘图，MATLAB 还提供了丰富的特殊图形的绘制函数。本节将介绍如何利用这些函数实现特殊图形的绘制。

7.4.1 条形图与区域图

条形图和区域图常用于查看和对比周期性的数据组，条形图适于显示离散数据，区域图则适于显示连续数据。有关条形图和区域图的函数如表 7-3 所示。

表 7-3 条形图和区域图绘制函数

函 数	描 述
bar	绘制竖直显示的二维条形图
barh	绘制水平显示的二维条形图
bar3	绘制竖直显示的三维条形图
bar3h	绘制水平显示的三维条形图
area	绘制二维区域图

1. bar、barh 函数

bar 函数用于在二维竖直条形图上显示向量或矩阵的值，barh 函数用于在二维水平条形图上显示向量或矩阵的值。它们的调用格式如下：

- **bar(Y)**: 绘制 Y 的每个元素的条形图。如果 Y 为矩阵，则将 Y 的元素按行分成组，x 轴坐标从 1 到 size(Y,1)。如果 Y 为向量，则 x 轴坐标从 1 到 length(Y)。
- **bar(x,Y)**: 在以向量 x 指定的 x 轴上绘制 Y 元素的竖直条形图。x 的值可以是非单调的，但是必须保证不含有重复值。如果 Y 为矩阵，则将 Y 的元素按行分成组，再在以 x 指定的 x 轴上绘制 Y 的行元素的条形图。
- **bar(...,width)**: 设置条形相对宽度并控制一组内的条形间距。宽度默认值为 0.8，所以如果没有指定 x，则同组内的条形间距会很小。如果宽度为 1，则同组内的条形相互接触。
- **bar(...,'style')**: 指定条形的样式，'style'可以是'grouped'或'stacked'，'grouped'为默认显示模式。
 - 'grouped': 显示 m 组竖直条形图，每组条形数为 n。其中 m 为 Y 的行数，n 为 Y 的列数。
 - 'stacked': 将 Y 的每一个行显示在一个条形中，条形的高度为该行元素之和。一行内不同元素的条形颜色不同，以区分每个元素在总条形的贡献。
- **bar(...,'bar_color')**: 按照由单个缩写字母'r'、'g'、'b'、'c'、'm'、'y'、'k'或'w'指定的颜色绘制条形图。
- **bar(axes_handle,...)**、**barh(axes_handle,...)**: 在句柄 axes_handle 指定的坐标轴内绘图。
- **h = bar(...)**: 返回一个 barseries 图形对象的句柄向量，每个条形对应一个句柄。如果 Y 为矩阵，Y 的一列对应一个条形图形对象。
- **barh(...)**、**h = barh(...)**: 创建水平条形图。条形长度由 Y 决定。向量 x 定义水平条形的 y 轴间隔。x 的值可以是非单调的，但是必须保证不含有重复值。

【例】 绘制二维条形图。

在命令窗输入：

```
>> Y = round(rand(5,3)*7);
```

```
>> subplot(2,2,1)
```

```
>> bar(Y,'group')
```

```
>> title 'Group'
```

```
>> subplot(2,2,2)
```

```
>> bar(Y,'stack')
```

```
>> title 'Stack'
```

```
>> subplot(2,2,3)
```

```
>> barh(Y,'stack')
```

```
>> title 'Stack'
```

```
>> subplot(2,2,4)
```

```
>> bar(Y,1.5)
```

```
>> title 'Width = 1.5'
```

运行结果如图 7-25 所示。

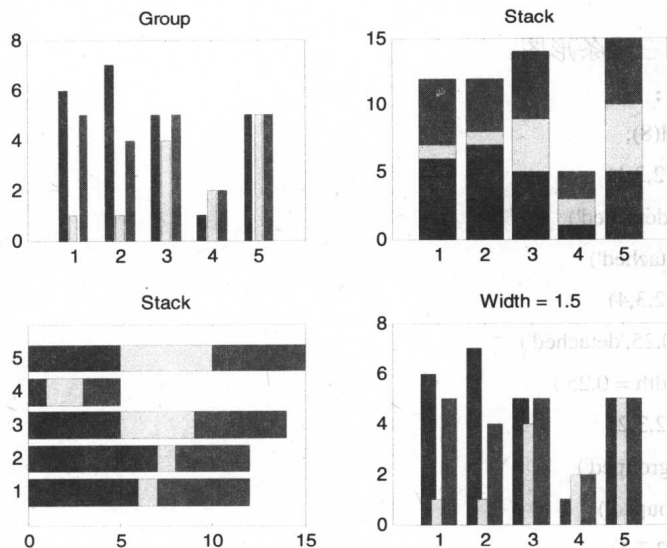


图 7-25 二维条形图

2. bar3、bar3h 函数

bar3 函数用于在三维垂直条形图上显示向量或矩阵的值，bar3h 函数用于在三维水平条形图上显示向量或矩阵的值。它们的调用格式如下：

- bar3(Y): 绘制 Y 的每个元素的三维条形图。如果 Y 为向量，则 x 轴坐标从 1 到 length(Y)。如果 Y 为矩阵，则 x 轴坐标从 1 到 size(Y,2)，并且将 Y 的元素按行分成组。

- bar3(x,Y): 在以向量 x 指定的 x 轴上绘制 Y 元素的垂直条形图。x 的值可以是非单调的，但是必须保证不含有重复值。如果 Y 为矩阵，则将 Y 的元素按行分成组。再在以 x 指定的 x 轴上绘制 Y 的行元素的条形图。

- bar3(...,width): 设置条形相对宽度并控制一组内的条形间距。宽度默认值为 0.8，

所以如果没有指定 x ，则同组内的条形间距会很小。如果宽度为 1，则同组内的条形相互接触。

- `bar3(...,'style')`: 指定条形的样式，'style'可以是'detached'、'grouped'或'stacked'，'detached'为默认显示模式。

- 'detached': 显示 Y 的每行元素，在 x 方向前后分块。

- 'grouped': 显示 n 组竖直条形图，每组条形数为 m 。其中 n 为 Y 的行数， m 为 Y 的列数。

- 'stacked': 将 Y 的每一个行显示在一个条形中，条形的高度为该行元素之和。一行内不同元素的条形颜色不同，以区分每个元素在总条形的贡献。

- `bar3(...,LineStyle)`: 用指定的颜色 LineSpec 绘制所有条形。

- `bar3(axes_handle, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。

- `h = bar3(...)`: 返回一个 patch 图形对象的句柄向量，每个条形对应一个句柄。如果 Y 为矩阵， Y 的一列对应一个条形图形对象。

- `bar3h(...)`、`h = bar3h(...)`: 创建水平条形图。条形长度由 Y 决定。向量 x 定义水平条形的 y 轴间隔。

【例】 绘制三维条形图。

在命令窗输入：

```
>> Y = cool(8);
>> subplot(2,3,1)
>> bar3(Y,'detached')
>> title('Detached')
>> subplot(2,3,4)
>> bar3(Y,0.25,'detached')
>> title('Width = 0.25')
>> subplot(2,3,2)
>> bar3(Y,'grouped')
>> title('Grouped')
>> subplot(2,3,5)
>> bar3(Y,0.5,'grouped')
>> title('Width = 0.5')
>> subplot(2,3,3)
>> bar3(Y,'stacked')
>> title('Stacked')
>> subplot(2,3,6)
>> bar3(Y,0.4,'stacked')
>> title('Width = 0.4')
>> colormap([1 0 0;0 1 0;0 0 1])
```

运行结果如图 7-26 所示。

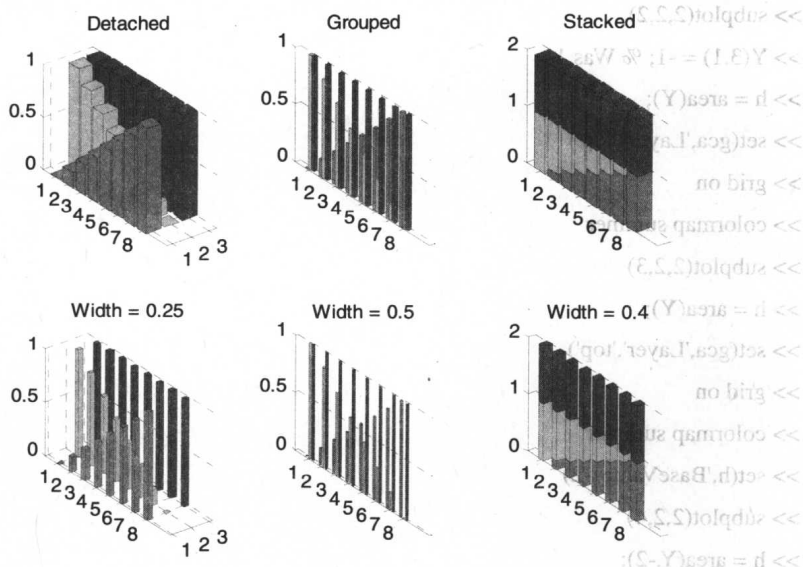


图 7-26 三维条形图

3. area 函数

area 函数用于绘制二维区域图，其调用格式如下：

- area(Y): 绘制向量 Y 或矩阵 Y 列之和的区域图，x 轴自动定义为 1:size(Y,1)。
- area(X,Y): 对于向量 X 和 Y，area(X,Y)等同于 plot(X,Y)，不同的是 area(X,Y)将 0 到 Y 区域用颜色填充。如果 Y 为矩阵，area(X,Y)绘制 Y 的列向量构成的区域图。对于每个 X，最终结果为相应 Y 的列元素之和。如果 X 为向量，length(X)必须等于 length(Y)；如果 X 为矩阵，size(X)必须等于 size(Y)。
- area(...,basevalue): 指定填充区域的基值，默认基值为 0。
- area(...,'PropertyName',PropertyValue, ...): 对 area 创建的区域图的属性名和属性值进行设置。
- area(axes_handle, ...): 在句柄 axes_handle 指定的坐标轴内绘图。
- h = area(...): 返回 areaseries 图形对象的句柄。

【例】 绘制二维区域图。

在命令窗输入：

```
>> Y = [2, 5, 3;
        1, 2, 7;
        2, 5, 3;
        3, 6, 5];
```

```
>> subplot(2,2,1)
```

```
>> area(Y)
```

```
>> grid on
```

```
>> colormap summer
```

```
>> set(gca,'Layer','top')
```

```

>> subplot(2,2,2)
>> Y(3,1) = -1; % Was 1
>> h = area(Y);
>> set(gca,'Layer','top')
>> grid on
>> colormap summer
>> subplot(2,2,3)
>> h = area(Y);
>> set(gca,'Layer','top')
>> grid on
>> colormap summer
>> set(h,'BaseValue',-2)
>> subplot(2,2,4)
>> h = area(Y,-2);
>> set(h(1),'FaceColor',[.5 0 0])
>> set(h(2),'FaceColor',[.7 0 0])
>> set(h(3),'FaceColor',[1 0 0])
>> set(h,'LineStyle','-', 'LineWidth',2)

```

运行结果如图 7-27 所示。

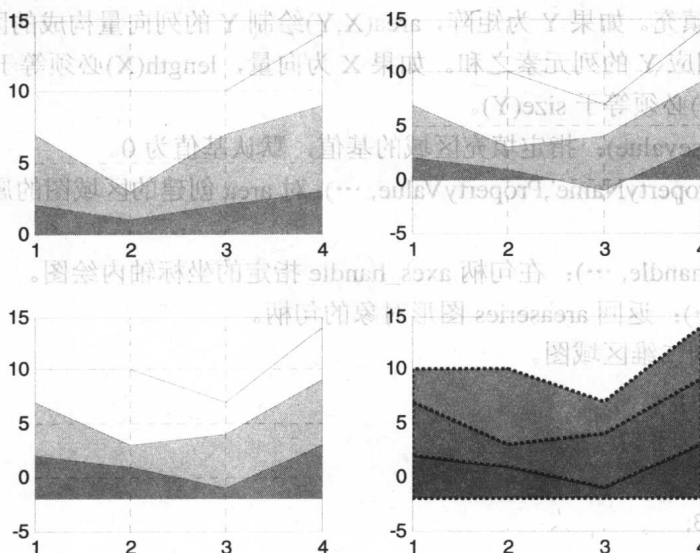


图 7-27 二维区域图

7.4.2 饼形图

饼形图常用于表示每个元素在向量或矩阵元素总和中所占的比例,通常用 `pie` 函数绘制二维饼形图,用 `pie3` 函数绘制三维饼形图。

1. pie 函数

pie 函数用于绘制二维饼形图，其调用格式如下：

- `pie(X)`: 绘制由数据 `X` 构成的二维饼形图，每个 `X` 中的元素代表饼形图的一个切片。
- `pie(X,explode)`: 从饼形图中分离出一个切片。`explode` 为与 `X` 对应的由零元素和非零元素组成的向量或矩阵。非零值将从饼形图的中心分离出对应切片，即如果 `explode(i,j)` 非零，则从饼形图中心分离出 `X(i,j)`。`explode` 与 `X` 尺寸必须相同。

● `pie(...,labels)`: 为切片指定文本标签，标签数必须等于 `X` 的元素数。例如，`pie(1:3,{'Taxes','Expenses','Profit'})`。

● `pie(axes_handle,...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。

● `h = pie(...)`: 返回 `patch` 和 `text` 图形对象的句柄向量。

【例】 绘制二维饼形图。

在命令窗输入：

```
>> x = [1 3 2 1.5 2.5];
>> explode = [0 0 1 0 1];
>> pie(x,explode)
```

运行结果如图 7-28 所示。

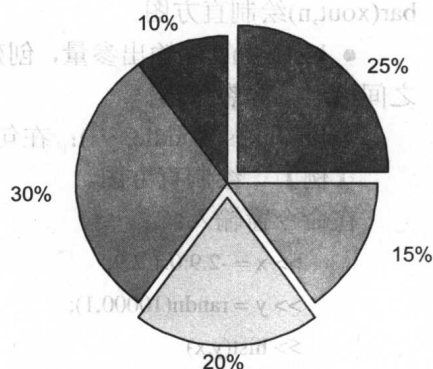


图 7-28 二维饼形图

2. pie3 函数

pie3 函数用于绘制三维饼形图，其调用格式如下：

- `pie3(X)`: 绘制由数据 `X` 构成的三维饼形图，每个 `X` 中的元素代表饼图的一个切片。
- `pie3(X,explode)`: 从饼形图中分离出一个切片。如果 `explode(i,j)` 非零，则从饼形图中心分离出 `X(i,j)`。`explode` 与 `X` 尺寸必须相同。

● `pie3(...,labels)`: 为切片指定文本标签，标签数必须等于 `X` 的元素数。例如，`pie3(1:3,{'Taxes','Expenses','Profit'})`。

● `pie3(axes_handle,...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。

● `h = pie3(...)`: 返回 `patch`、`surface` 和 `text` 图形对象的句柄向量。

【例】 绘制三维饼形图。

在命令窗输入：

```
>> x = [1 3 2 1.5 2.5];
>> explode = [0 0 1 0 1];
>> pie3(x,explode)
```

运行结果如图 7-29 所示。

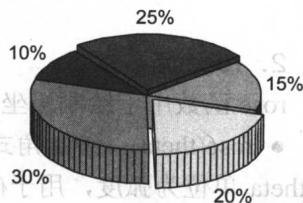


图 7-29 三维饼形图

7.4.3 直方图

`hist` 函数通常用于绘制笛卡尔坐标系下的直方图，`rose` 函数通常用于绘制极坐标系下的直方图。

1. hist 函数

hist 函数用于绘制直方图，其调用格式如下：

- **n = hist(Y)**: 将向量 Y 中的元素放入等距的 10 个条形中，并以行向量形式返回每一个条形中的元素个数。若 Y 为 $m \times p$ 矩阵，则该按列向量对 Y 进行处理并返回一个 $10 \times p$ 的矩阵 n。n 的每一列包含相应 Y 列的结果，Y 的元素不能为复数。
- **n = hist(Y,x)**: x 为向量，按照 length(x) 分裂 Y 并放到中心由 x 元素指定的条形中。
- **n = hist(Y,nbins)**: nbins 为标量，使用 nbins 个条形。
- **[n,xout] = hist(...)**: 返回向量 n 和包含频率计数与条形的位置的向量 xout。可以使用 bar(xout,n) 绘制直方图。
- **hist(...)**: 无输出参量，创建上述方法中输出参量的直方图。在 Y 的最小值和最大值之间沿 x 轴分裂条形。
- **hist(axes_handle, ...)**: 在句柄 axes_handle 指定的坐标轴内绘图。

【例】绘制直方图。

在命令窗输入：

```
>> x = -2.9:0.1:2.9;  
>> y = randn(10000,1);  
>> hist(y,x)
```

运行结果如图 7-30 所示。

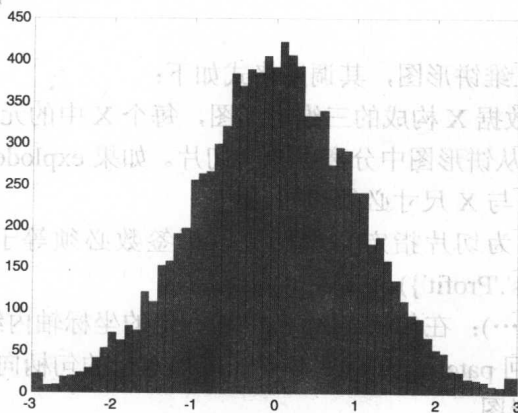


图 7-30 直方图

2. rose 函数

rose 函数用于绘制极坐标下的直方图，其调用格式如下：

- **rose(theta)**: 绘制角式直方图，显示 theta 数据在 20 个或更少的角区间内的分布。向量 theta 单位为弧度，用于确定每一区间与原点的夹角。每个区间的长度反映出 theta 的元素落入一组的个数。
- **rose(theta,x)**: 用向量 x 指定每一区间的号码和位置。length(x) 等于区间数，x 的值指定每个区间的中心角度。
- **rose(theta,nbins)**: 在 $[0, 2\pi]$ 内绘制出 nbins 个等距的区间，默认值为 20。
- **rose(axes_handle, ...)**: 在句柄 axes_handle 指定的坐标轴内绘图。
- **h = rose(...)**: 返回用于创建图形的 line 对象句柄。

● `[tout, rout] = rose(...)`: 返回向量 `tout` 与 `rout`, 以便于用 `polar(tout, rout)` 绘制直方图。该语句不产生图形。

【例】 绘制极坐标下的直方图。

在命令窗输入:

```
>> theta = 2*pi*rand(1,50);
```

```
>> rose(theta)
```

运行结果如图 7-31 所示。

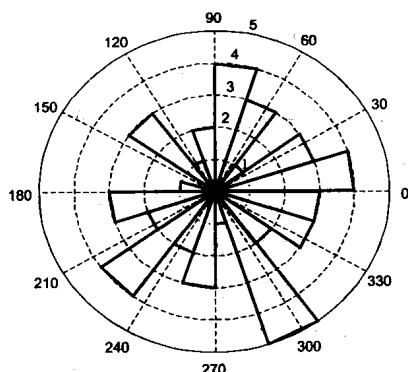


图 7-31 极坐标下的直方图

7.4.4 离散数据图

绘制离散数据图的函数有 `stem`、`stem3`、`stairs`, 其中 `stem`、`stem3` 函数分别用于绘制二维、三维离散序列图, 而 `stairs` 函数用于绘制二维阶梯图。

1. stem 函数

`stem` 函数用于绘制二维离散序列图, 其调用格式如下:

- `stem(Y)`: 在 `x` 轴上等距离绘制离散序列 `Y`。如果 `Y` 为矩阵, 则将 `Y` 分成行向量, 在同一 `x` 位置处绘制一行向量。
- `stem(X,Y)`: 在 `x` 轴绘制 `X`, `y` 轴绘制 `Y` 的列。`X` 和 `Y` 必须是尺寸相同的向量或矩阵。`X` 可以为行或列向量, `Y` 是行数为 `length(X)` 的矩阵。
- `stem(...,'fill')`: 用颜色填充序列图末端的圆圈。
- `stem(...,LineStyle)`: 指定线型、标记符号以及序列图末端的小圆圈颜色。
- `stem(axes_handle, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。
- `h = stem(...)`: 返回 `stemseries` 对象的句柄向量, 一个句柄对应一个 `Y` 的数据列。

【例】 绘制二维离散序列图。

在命令窗输入:

```
>> x = sin(2*pi/20*[1:20]);
```

```
>> stem(x,'fill')
```

运行结果如图 7-32 所示。

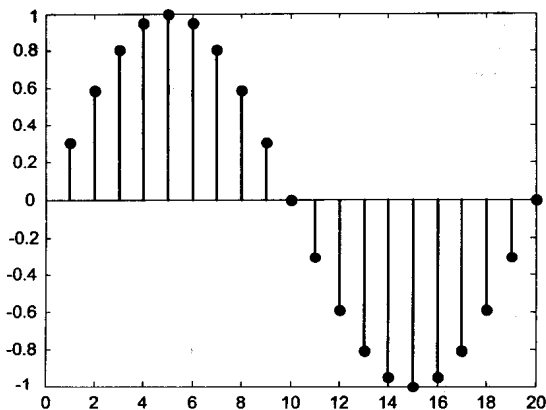


图 7-32 二维离散序列图

2. stem3 函数

stem3 函数用于绘制三维离散序列图，其调用格式如下：

- stem3(Z): 在 $x-y$ 平面上绘制数据序列 Z 的离散序列图。x 与 y 是自动产生的。如果 Z 为行向量，stem3 将沿 x 轴平行方向等间距地绘制 Z 的所有元素；如果 Z 为列向量，stem3 将沿 y 轴平行方向等间距地绘制 Z 的所有元素。

- stem3(X,Y,Z): 在 X 和 Y 值指定的位置绘制数据序列 Z 的离散序列图。X、Y、Z 必须是尺寸相同的向量或矩阵。

- stem3(...,'fill'): 用颜色填充序列图末端的圆圈。

- stem3(...,LineStyle): 指定线型、标记符号以及序列图末端的小圆圈颜色。

- h = stem3(...): 返回 stemseries 对象的句柄。

【例】 绘制三维离散序列图。

在命令窗输入：

```
>> th = (0:127)/128*2*pi;
>> x = cos(th);
>> y = sin(th);
>> f = abs(fft(ones(10,1),128));
>> stem3(x,y,f,'d','fill')
>> view([-60 30])
```

运行结果如图 7-33 所示。

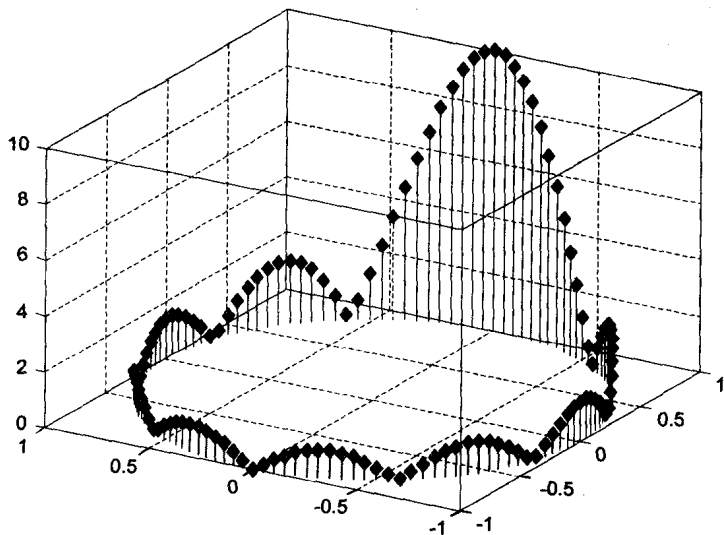


图 7-33 三维离散序列图

3. stairs 函数

stairs 函数用于绘制阶梯图，其调用格式如下：

- stairs(Y): 绘制 Y 元素的阶梯图，一条线对应矩阵的一列。轴的 ColorOrder 属性决定每条线的颜色。如果 Y 为向量，则 x 轴刻度从 1 变化到 length(Y)；如果 Y 为矩阵，则 x 轴刻度从 1 变化到 Y 的行数。

- `stairs(X,Y)`: 在 x 轴绘制 X , y 轴绘制 Y 的元素。 X 与 Y 必须具有相同的尺寸, 或者如果 Y 是矩阵, 则 X 可以是行或列向量且满足 $\text{length}(X) = \text{size}(Y,1)$ 。
- `stairs(...,LineSpec)`: 指定图形的线型、标记符号和颜色。
- `stairs(...,'PropertyName',propertyvalue)`: 创建阶梯图并对指定属性进行设置。
- `stairs(axes_handle, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。
- `h = stairs(...)`: 返回 `stairs` 对象创建的句柄。
- `[xb,yb] = stairs(Y, ...)`: 不产生图形, 返回向量 xb 和 yb 以便于使用 `plot(xb,yb)` 绘制阶梯图。

【例】 绘制阶梯图。

在命令窗输入:

```
>> x = linspace(-2*pi,2*pi,40);
```

```
>> stairs(x,sin(x))
```

运行结果如图 7-34 所示。

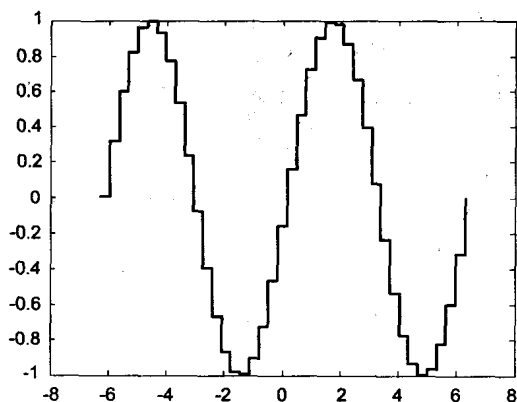


图 7-34 阶梯图

7.4.5 方向和速度向量图

MATLAB 提供了四个绘制方向和速度向量图的函数, 如表 7-4 所示。

表 7-4 方向和速度向量图绘制函数

函 数	描 述
<code>compass</code>	从极坐标原点绘制向量
<code>feather</code>	从水平线上等距地绘制向量
<code>quiver</code>	绘制二维向量图
<code>quiver3</code>	绘制三维向量图

1. compass 函数

`compass` 函数用于绘制从原点发散出的箭头图, 其调用格式如下:

- `compass(U,V)`: 绘制有 n 个箭头的罗盘图, n 为 U 和 V 的元素数。每个箭头的起点为原点, 箭头位置由 $[U(i),V(i)]$ 确定。
- `compass(Z)`: 绘制有 n 个箭头的罗盘图, n 为 Z 的元素数。每个箭头的起点为原点,

箭头位置由 Z 的实部和虚部共同确定。该调用格式等价于 `compass(real(Z),imag(Z))`。

- `compass(...,LineSpec)`: 绘制线型、标记符号和颜色由 `LineSpec` 指定的罗盘图。
- `compass(axes_handle,...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。
- `h = compass(...)`: 返回 `line` 对象的句柄 `h`。

【例】 绘制罗盘图。

在命令窗输入：

```
>> Z = eig(randn(20,20));
```

```
>> compass(Z)
```

运行结果如图 7-35 所示。

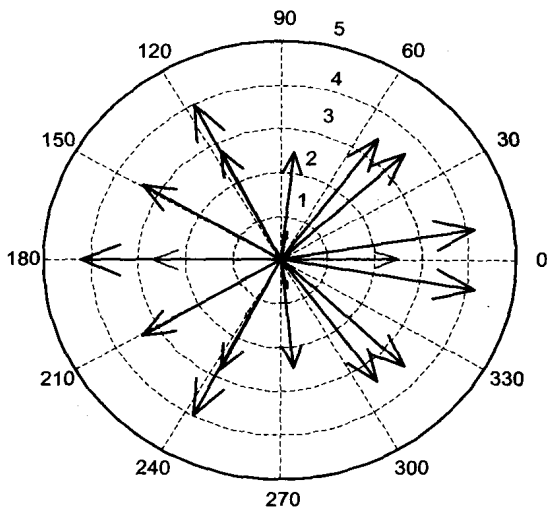


图 7-35 罗盘图

2. feather 函数

`feather` 函数用于绘制速度向量图，其调用格式如下：

• `feather(U,V)`: 绘制由 U 和 V 指定的向量， U 包含作为相对坐标系中的 x 部分， V 包含作为相对坐标系中的 y 部分。

- `feather(Z)`: 绘制由复数 Z 指定的向量，等价于 `feather(real(Z),imag(Z))`。
- `feather(...,LineSpec)`: 绘制线型、标记符号和颜色由 `LineSpec` 指定的羽毛图。
- `feather(axes_handle,...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。
- `h = feather(...)`: 返回 `line` 对象的句柄 `h`。

【例】 绘制速度向量图。

在命令窗输入：

```
>> theta = 90:-10:0;
```

```
>> r = ones(size(theta));
```

```
>> [u,v] = pol2cart(theta*pi/180,r*10);
```

```
>> feather(u,v)
```

运行结果如图 7-36 所示。

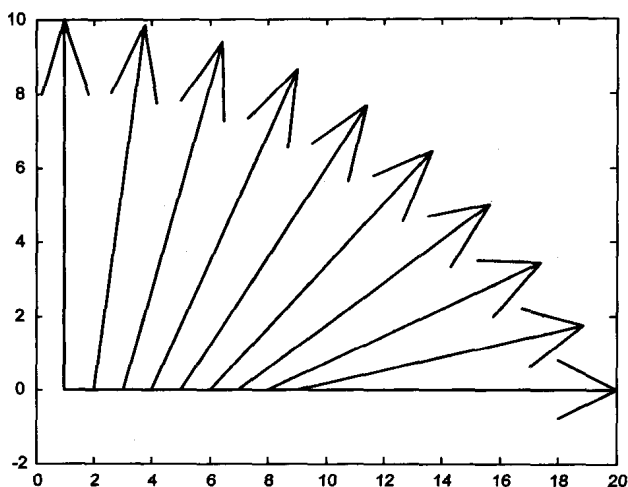


图 7-36 速度向量图

3. quiver 函数

quiver 函数用于绘制二维向量图，其调用格式如下：

- **quiver(x,y,u,v)**: 绘制箭头坐标由 x 、 y 中元素指定的向量图。矩阵 x 、 y 、 u 、 v 必须具有相同的尺寸并且包含相应的位置和速度分量。 x 和 y 也可以是向量。默认情况下箭头不会重叠，但是可以根据需要对箭头进行伸长或缩短。

- **quiver(u,v)**: 在等距划分的 x - y 平面点上绘制由 u 和 v 指定的向量图。

- **quiver(...,scale)**: 自动处理箭头使之适应网格并按照标度因子延伸。如果 $scale = 2$ ，则向量相对长度变为 2 倍；如果 $scale = 0.5$ ，则向量长度减半；如果 $scale = 0$ ，则对向量不作处理。

- **quiver(...,LineStyle)**: 由 LineSpec 指定线型、标记符号和颜色。quiver 函数在向量的原点绘制标记。

- **quiver(...,LineStyle,'filled')**: 填充由 LineSpec 指定的记号。

- **quiver(axes_handle,...)**: 在句柄 axes_handle 指定的坐标轴内绘图。

- **h = quiver(...)**: 返回 quivergroup 对象的句柄。

【例】 绘制函数 $z = xe^{(-x^2-y^2)}$ 的梯度场向量图。

在命令窗输入：

```
>> [X,Y] = meshgrid(-2:.2:2);
>> Z = X.*exp(-X.^2 - Y.^2);
>> [DX,DY] = gradient(Z,.2,.2);
>> contour(X,Y,Z)
>> hold on
>> quiver(X,Y,DX,DY)
>> colormap hsv
>> hold off
```

运行结果如图 7-37 所示。

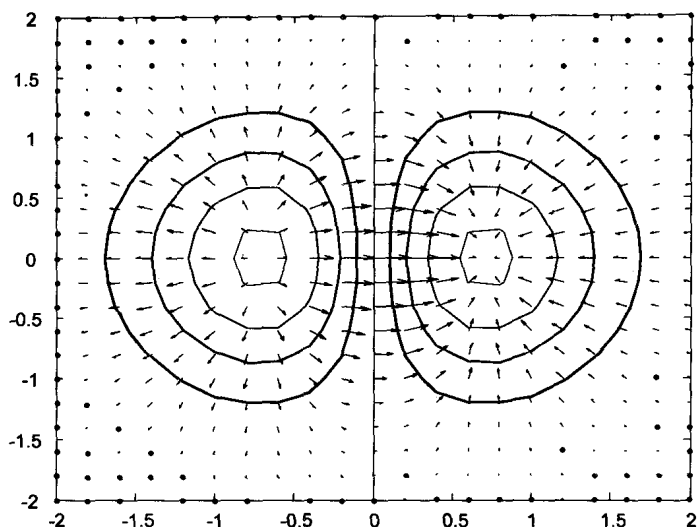


图 7-37 梯度场向量图

4. quiver3 函数

quiver3 函数用于绘制三维向量图，其调用格式如下：

- **quiver3(x,y,z,u,v,w)**: 在点(x,y,z)处由(u,v,w)分量绘制向量。矩阵 x、y、z、u、v、w 必须具有相同的尺寸并包含相应的位置和速度分量。

- **quiver3(z,u,v,w)**: 在由 z 指定的等距划分的曲面点上绘制向量图。quiver3 自动处理向量距离使它们不会重合。

- **quiver3(...,scale)**: 自动处理向量使之相互不重合，然后乘以 scale。如果 scale = 2，则向量相对长度变为 2 倍；如果 scale = 0.5，则向量长度减半；如果 scale = 0，则对向量不作处理。

- **quiver3(...,LineStyle)**: 由 LineSpec 指定线型和颜色。

- **quiver3(...,LineStyle,'filled')**: 填充由 LineSpec 指定的记号。

- **quiver3(axes_handle, ...)**: 在句柄 axes_handle 指定的坐标轴内绘图。

- **h = quiver3(...)**: 返回 line 句柄的向量。

【例】 绘制函数 $z = xe^{(-x^2-y^2)}$ 的曲面法线向量图。

在命令窗输入：

```
>> [X,Y] = meshgrid(-2:0.25:2,-1:0.2:1);
>> Z = X.* exp(-X.^2 - Y.^2);
>> [U,V,W] = surfnorm(X,Y,Z);
>> quiver3(X,Y,Z,U,V,W,0.5);
>> hold on
>> surf(X,Y,Z);
>> colormap hsv
>> view(-35,45)
```



```
>> axis([-2 2 -1 1 -.6 .6])
```

```
>> hold off
```

运行结果如图 7-38 所示。

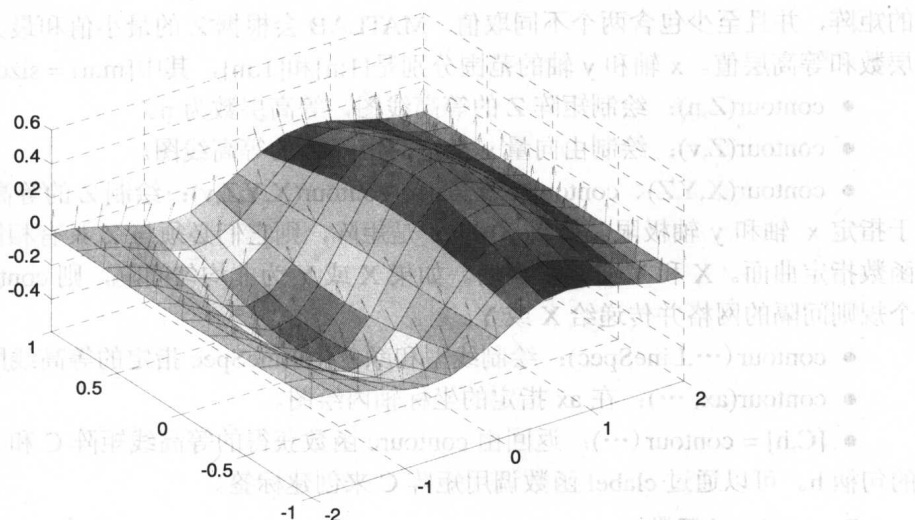


图 7-38 曲面法线向量图

7.4.6 等高线图

常用有关绘制等高线图的函数如表 7-5 所示。

表 7-5 等高线图绘制函数

函 数	描 述
clabel	在当前等高线图上添加标签
contour	绘制二维等高线图
contour3	绘制三维等高线图
contourf	绘制带填充色的二维等高线图
contourc	计算等高线矩阵

1. contourc 函数

contourc 函数用于计算出等高线绘图函数 contour、contour3 和 contourf 所需的等高线矩阵 C，其调用格式如下：

- $C = \text{contourc}(Z)$ ：由矩阵 Z 的数据计算等高线矩阵 C。Z 至少是一个 2×2 的矩阵，等高线为 Z 中单元的等值线。MATLAB 会自动选择等高线数和等高线值。
- $C = \text{contourc}(Z, n)$ ：计算矩阵 Z 的等高线，等高层数为 n。
- $C = \text{contourc}(Z, v)$ ：计算矩阵 Z 的等高线，等高线由向量 v 的值确定，等高层数由 v 的长度确定。如计算第 i 层等高线，使用命令 $\text{contourc}(Z, [i \ i])$ 。
- $C = \text{contourc}(x, y, Z)$ 、 $C = \text{contourc}(x, y, Z, n)$ 、 $C = \text{contourc}(x, y, Z, v)$ ：计算 Z 的等高线。x 和 y 用于指定 x 轴和 y 轴极限，x 和 y 必须单调增。

2. contour 函数

contour 函数用于绘制二维等高线图，其调用格式如下：

- contour(Z): 绘制矩阵 Z 的等高线图，Z 为相对 x-y 平面的高度。Z 至少是一个 2×2 的矩阵，并且至少包含两个不同取值。MATLAB 会根据 Z 的最小值和最大值自动选择等高层数和等高层值。x 轴和 y 轴的范围分别是 [1:n] 和 [1:m]，其中 [m,n] = size(Z)。

- contour(Z,n): 绘制矩阵 Z 的等高线图，等高层数为 n。

- contour(Z,v): 绘制由向量 v 指定的矩阵 Z 的等高线图。

- contour(X,Y,Z)、contour(X,Y,Z,n)、contour(X,Y,Z,v): 绘制 Z 的等高线图。X 和 Y 用于指定 x 轴和 y 轴极限。如果 X 和 Y 是矩阵，则它们必须与 Z 具有相同的尺寸，由 surf 函数指定曲面。X 和 Y 必须单调增。如果 X 或 Y 为非均匀间隔，则 contour 函数计算出一个规则间隔的网格并传递给 X 或 Y。

- contour(...,LineStyle): 绘制线型和颜色由 LineSpec 指定的等高线图。

- contour(ax, ...): 在 ax 指定的坐标轴内绘图。

- [C,h] = contour(...): 返回由 contourc 函数获得的等高线矩阵 C 和 contourgroup 对象的句柄 h。可以通过 clabel 函数调用矩阵 C 来创建标签。

3. contourf 函数

contourf 函数用于绘制带填充色的二维等高线图，其调用格式如下：

- contourf(Z): 绘制矩阵 Z 的等高线图，Z 为相对于平面的高度。Z 至少是一个 2×2 的矩阵，并且至少包含两个不同取值。MATLAB 会根据 Z 的最小值和最大值自动选择等高层数和等高层值。

- contourf(Z,n): 绘制矩阵 Z 的等高线图，等高层数为 n。

- contourf(Z,v): 绘制由向量 v 指定的矩阵 Z 的等高线图。等高层数为 length(v)。当使用该命令指定等高层向量时，Z 值小于 v(1) 的等高区域将不被填充。如果需要填充，则须保证 v(1) 不大于 Z 的最小值。

- contourf(X,Y,Z)、contourf(X,Y,Z,n)、contourf(X,Y,Z,v): 绘制 Z 的等高线图。X 和 Y 用于指定 x 轴和 y 轴极限。如果 X 和 Y 是矩阵，则它们必须与 Z 具有相同的尺寸，由 surf 函数指定曲面。X 和 Y 必须单调增。

- contourf(axes_handle,...): 在句柄 axes_handle 指定的坐标轴内绘图。

- C = contourf(...): 返回由 contourc 函数获得的等高线矩阵 C。可以通过 clabel 函数调用矩阵 C 来创建标签。

- [C,h] = contourf(...): 同上，且返回一个 contourgroup 对象的句柄 h。

4. contour3 函数

contour3 函数用于绘制三维等高线图，其调用格式如下：

- contour3(Z): 绘制矩阵 Z 的三维等高线图，Z 为相对 x-y 平面的高度。Z 至少是一个 2×2 的矩阵，并且至少包含两个不同的取值。MATLAB 会根据 Z 的最小值和最大值自动选择等高层数和等高层值。x 轴和 y 轴的范围分别是 [1:n] 和 [1:m]，其中 [m,n] = size(Z)。

- contour3(Z,n): 绘制矩阵 Z 的三维等高线图，等高层数为 n。

- contour3(Z,v): 绘制由向量 v 指定的矩阵 Z 的等高线图。等高层数为 length(v)。

• `contour3(X,Y,Z)`、`contour3(X,Y,Z,n)`、`contour3(X,Y,Z,v)`: 绘制 Z 的等高线图。 X 和 Y 用于指定 x 轴和 y 轴极限。如果 X 是矩阵, 则 x 轴由 $X(1,:)$ 定义; 如果 Y 是矩阵, 则 y 轴由 $Y(:,1)$ 定义。如果 X 和 Y 是矩阵, 则它们必须与 Z 具有相同的尺寸, 由 `surf` 函数指定曲面。

• `contour3(...,LineStyle)`: 绘制线型和颜色由 `LineStyle` 指定的三维等高线图。

• `contour3(axes_handle, ...)`: 在句柄 `axes_handle` 指定的坐标轴内绘图。

• `[C,h] = contour3(...)`: 返回由 `contourc` 函数获得的等高线矩阵 C 和图形对象的句柄列向量 h 。`contour3` 函数将创建 `patch` 图形对象, 除非指定 `LineStyle`, 这种情况下则创建 `line` 图形对象。

【例】 绘制函数 $z = xe^{(-x^2-y^2)}$ 的等高线图。

在命令窗输入:

```
>> [X,Y] = meshgrid([-2:.1:2]);  
>> Z = X.*exp(-X.^2-Y.^2);  
>> figure;contour(Z)  
>> figure;contourf(Z)  
>> colormap winter  
>> figure; contour3(X,Y,Z,30)  
>> surface(X,Y,Z,'EdgeColor',[.8 .8 .8],'FaceColor','none')  
>> grid off  
>> view(-15,25)
```

运行结果如图 7-39~图 7-41 所示。

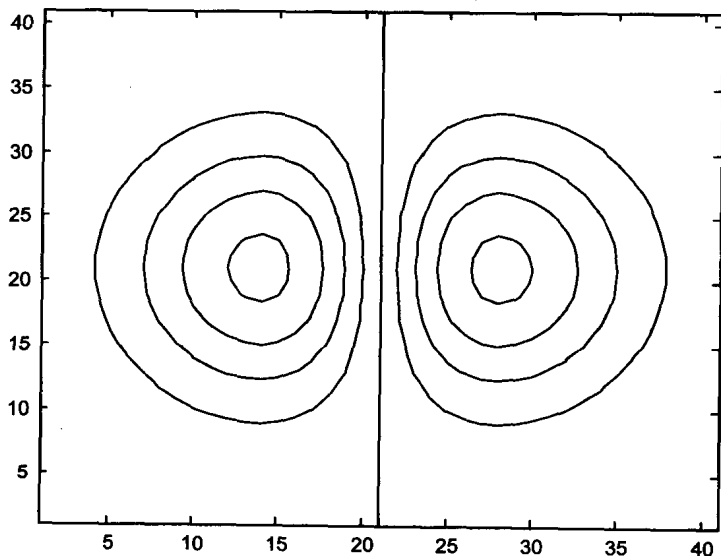


图 7-39 二维等高线图

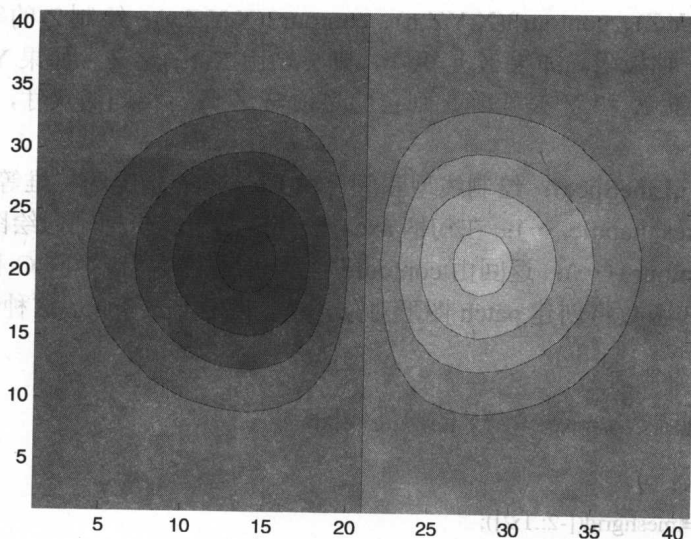


图 7-40 带填充色的二维等高线图

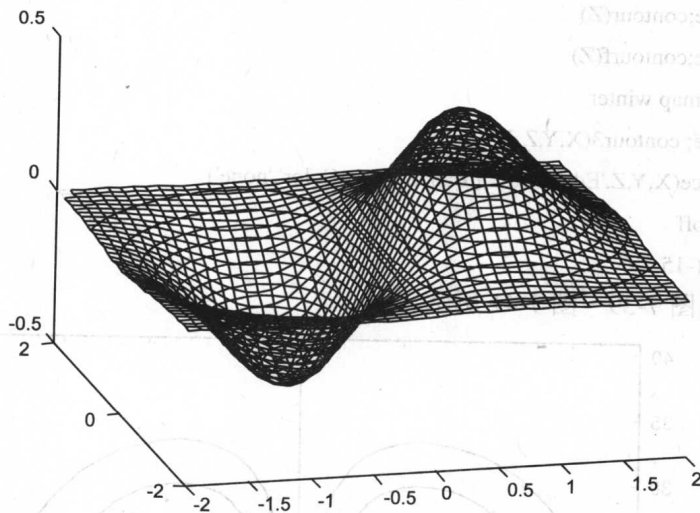


图 7-41 三维等高线图

5. clabel 函数

clabel 函数用于为二维等高线添加标签，其调用格式如下：

- clabel(C,h): 旋转标签并将其插入等高线中。该函数只在有足够空隙的等高线间插入标签，这取决于等高线的尺寸。
- clabel(C,h,v): 为向量 v 指定的等高层创建标签，旋转标签并将其插入等高线。
- clabel(C,h,'manual'): 在鼠标选择的位置放置等高线标签。按下鼠标左键或空格键就在最近处放置标签。按下回车键终止标记操作，标签被旋转并插入等高线。
- clabel(C): 使用 contour 生成的等高线数组 C 在当前等高线上增加标签。该函数将在随机位置标记所有等高线。
- clabel(C,v): 只标记向量 v 指定的等高层。

- `clabel(C,'manual')`: 用鼠标选择等高线的标签位置。
- `text_handles = clabel(...)`: 返回 `clabel` 函数创建的 `text` 对象句柄。`text` 对象的 `UserData` 属性包含等高线显示的值。如果不使用参数 `h` 调用 `clabel`, `text_handles` 同样包含用于创建 '+' 符号的 `line` 对象句柄。
- `clabel(...,'PropertyName',propertyvalue, ...)`: 为标签字符串指定 `text` 对象的属性。
- `clabel(...,'LabelSpacing',points)`: 指定一条等高线上的标签间隔。

【例】 绘制带标签的等高线图。

在命令窗输入:

```
>> [x,y,z] = peaks;  
>> [C,h] = contour(x,y,z);  
>> clabel(C,h,'FontSize',15,'Color','r','Rotation',0)
```

运行结果如图 7-42 所示。

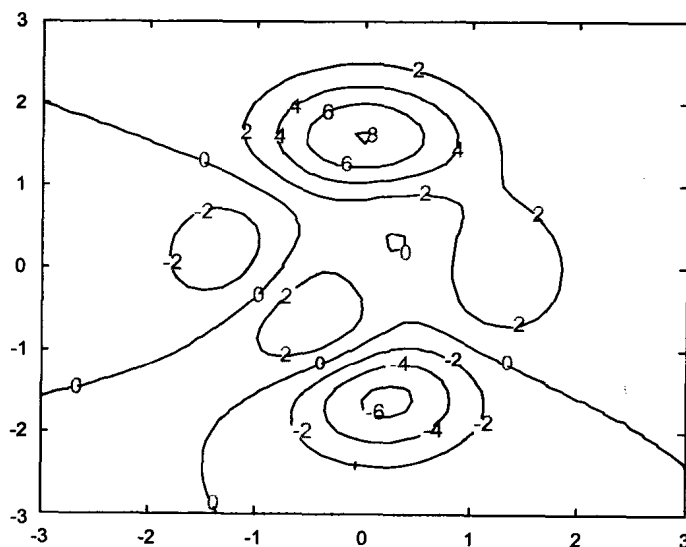


图 7-42 带标签的等高线图

第 8 章 Simulink 仿真环境

Simulink 是实现动态系统建模和仿真的集成环境, 它可以针对任何能用数学描述的系统进行建模, 如航空航天动力学系统、卫星控制制导系统、通信系统、船舶及汽车等, 这些系统包括连续、离散、条件执行、事件触发、单速率、多速率和混杂系统等。由于 Simulink 提供了鼠标拖放建立系统框图模型的方法, 而且还提供了丰富的功能模块和专业模型库, 因此使用 Simulink 可以避免书写复杂的代码而实现整个动态系统的建模。

本章主要介绍 Simulink 的工作环境、基本操作步骤、操作细节以及操作技巧, 并结合实例进行说明。


8.1 Simulink 基础

利用 Simulink 仿真一个动力学系统通常分为两步: 首先使用 Simulink 模型编辑器创建一个描述时变系统输入、状态以及输出之间数学关系的框图, 然后指定仿真起始时间和终止时间, 运行该系统仿真模型。

本节主要介绍 Simulink 的工作环境和 Simulink 仿真基础。


8.1.1 Simulink 的启动

要启动 Simulink, 首先必须启动 MATLAB。当 MATLAB 启动之后, 可以通过以下两种途径打开 Simulink 模块库浏览器:

- 单击 MATLAB 工具栏上的 Simulink 按钮 。
- 在 MATLAB 命令窗输入 `simulink` 命令。

运行结果如图 8-1 所示。

Simulink 模块库浏览器打开之后, 可以通过以下两种途径新建一个模型文件:

- 单击 Simulink 模块库浏览器工具栏按钮 。
- 在 Simulink 模块库浏览器主菜单选择 `File > New > Model`。

打开新建模型文件, 出现如图 8-2 所示的窗口。

新建模型文件名默认为 `untitled.mdl`, 用户可根据需要对文件重命名。

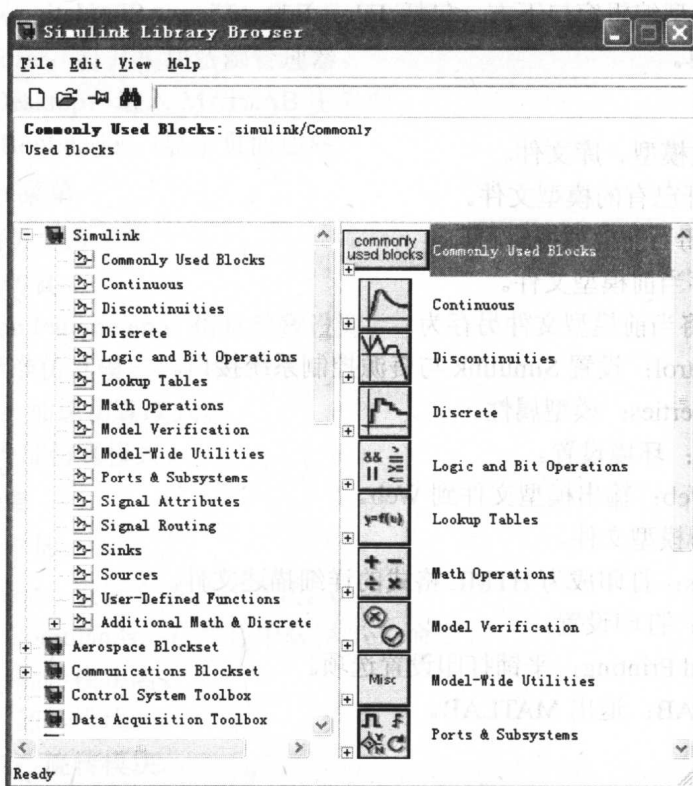


图 8-1 Simulink 模块库浏览器

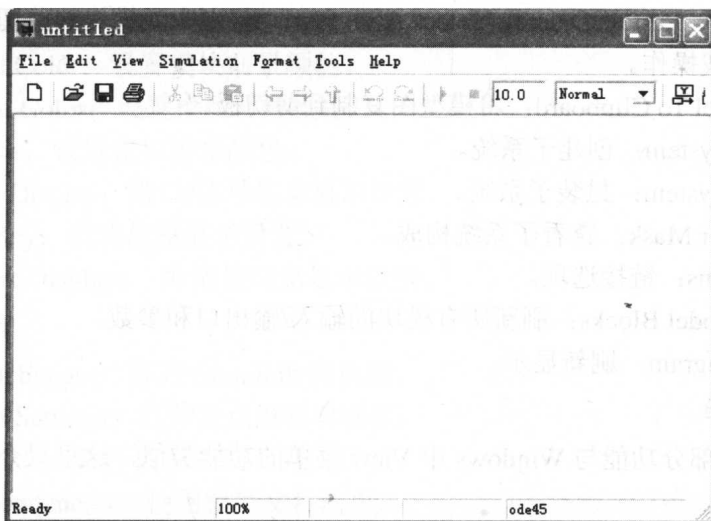


图 8-2 模型编辑窗口

8.1.2 Simulink 工作环境

Simulink 模块库浏览器的操作界面不难掌握, 本小节主要介绍图 8-2 所示的模型编辑窗口的菜单功能, 其中包括主菜单和工具栏。

主菜单位于模型编辑窗口下方, 包括 File、Edit、View、Simulation、Format、Tools、Help 七个功能菜单。

1. File 菜单

- New: 新建模型、库文件。
- Open: 打开已有的模型文件。
- Close: 关闭当前模型编辑窗口。
- Save: 保存当前模型文件。
- Save As: 将当前模型文件另存为。
- Source Control: 设置 Simulink 与资源控制系统接口。
- Model Properties: 模型属性。
- Preferences: 环境设置。
- Export to Web: 输出模型文件到 Web。
- Print: 打印模型文件。
- Print Details: 打印成为 HTML 格式的详细描述文件。
- Print Setup: 打印设置。
- Enable Tiled Printing: 平铺打印设置选项。
- Exit MATLAB: 退出 MATLAB。

2. Edit 菜单

- Undo: 撤消。
- Redo: 恢复。
- Cut、Copy、Paste、Delete、Select All、Find: 分别实现模块的剪切、复制、粘贴、删除、全选、查找操作。
- Copy Model To Clipboard: 将模型图复制到剪切板。
- Creat Subsystem: 创建子系统。
- Mask Subsystem: 封装子系统。
- Look Under Mask: 查看子系统构成。
- Link Options: 链接选项。
- Refresh Model Blocks: 刷新所有模块的输入/输出口和参数。
- Update Diagram: 刷新显示。

3. View 菜单

View 菜单的部分功能与 Windows 中 View 菜单的功能类似, 这里只介绍模型编辑窗口的特殊功能菜单。

- Back: 后退。
- Forward: 前进。
- Go To Parent: 返回子系统上一级。
- Model Browser Options: 模型浏览器选项。
- Block Data Tips Options: 模块数据提示选项。
- System Requirments: 系统要求。

- Library Browser: 打开 Simulink 模块库浏览器。
- Model Explorer: 打开模块资源管理器。
- MATLAB Desktop: 进入 MATLAB 主界面。
- Show Page Boundaries: 显示页面边界。

4. Simulation 菜单

- Start: 启动仿真。
- Stop: 停止仿真。
- Configuration Parameters: 仿真参数设置。
- Normal: 标准仿真模式。
- Accelerator: 加速仿真模式。
- External: 外部仿真模式。

5. Format 菜单

- Font: 设置字体。
- Text Alignment: 设置文字对齐方式。
- Enable TeX Commands: 在注释中解释 TeX 命令。
- Flip Name: 翻转模块名。
- Flip Block: 翻转模块。
- Rotate Block: 旋转模块。
- Show Name: 显示模块名。
- Show Drop Shadow: 显示模块阴影。
- Show Port Labels: 端口标签显示模式。
- Foreground Color: 设置模块前景颜色。
- Background Color: 设置模块背景颜色。
- Screen Color: 设置窗口屏幕颜色。
- Port/Signal Display: 端口/信号信息显示设置。
- Block Display: 模块信息显示设置。
- Library Link Display: 库链接信息显示设置。

6. Tools 菜单

- Simulink Debugger: 打开 Simulink 调试器。
- Fixed-Point Settings: 打开定点设置对话框。
- Model Advisor: 打开模型咨询工具。
- Model Dependencies: 使用模型文件清单。
- Lookup Table Editor: 打开查表编辑器。
- Data Class Designer: 打开 Simulink 数据类型设计器。
- Bus Editor: 打开总线编辑器查看或编辑总线对象。
- Profiler: 触发 Simulink 优化工具。
- Coverage Settings: 打开模型设置对话框。
- Signal & Scope Manager: 打开信号与示波器管理器。

- Real-Time Workshop: 实时工作区间选项。
- External Mode Control Panel: 打开外部模式控制面板。
- Control Design: 用于控制设计。
- Parameter Estimation: 用于参数估计。
- Report Generator: 打开报告产生器。
- HDL Coder: HDL 代码选项。
- Link for TASKING: TASKING 链接选项。
- Data Object Wizard: 打开数据对象向导。
- SystemTest: 系统测试。
- MPlay Video Viewer: 打开 MPlay 视频观察器。

7. Help 菜单

- Using Simulink: 在 MATLAB 帮助浏览器中显示 Simulink 用户指导。
- Blocks: 在帮助浏览器中显示选中模块或所有模块的帮助。
- Blocksets: 分类选择查看帮助。
- Block Support Table: 查看 Simulink 各模块支持的数据类型表。
- Shortcuts: 在帮助浏览器中显示 Simulink 命令的快捷键。
- S-Functions: 在帮助浏览器中显示 S-函数的帮助。
- Demos: 在帮助浏览器中显示 Demos 标签页的 Simulink 部分。
- Terms of Use: 打开 Licence.txt 文件。
- Patents: 打开 patents.txt 文件。
- About Simulink: 显示 Simulink 版本信息。

工具栏位于主菜单之下, 为用户提供了快捷的功能操作。由于主菜单中都包含了工具栏的基本功能, 这里就不再重复介绍了。

8.1.3 Simulink 模块的基本操作

有关模块的操作都可以通过主菜单、鼠标或右键菜单来完成。本小节将介绍最重要且最基本的模块操作。

1. 模块的添加和重命名

建立系统框图时首先需要添加模块。用户可以从模块库浏览器中选中某个模块并选择菜单中的 Add to the Current Model 项, 将其添加到当前模型编辑窗口, 或者直接将模块库浏览器中的模块拖放到模型编辑窗口的某个位置。

默认情况下, 添加的模块名为模块库浏览器中的模块名。在模型编辑窗口单击模块名可以对模块进行重命名。

2. 模块的移动和缩放

为调整模型编辑窗口内模块的位置, 可以用鼠标左键拖动模块到指定位置, 也可以选中模块后通过键盘的上、下、左、右键调整模块的位置。

为调整模块的大小, 可以先选中模块, 然后将鼠标移动到该模块的四个角中的任意一个角, 这时鼠标变成双向斜箭头, 按下鼠标左键就可以对模块的大小进行调整了。

3. 模块的剪切、复制和粘贴

选中一个模块或一块区域后, 可以通过主菜单、工具栏或右键菜单对其进行剪切(Cut)或复制(Copy)操作。当剪贴板中有内容时, 可以进行粘贴(Paste)操作。

在一个模型文件内还有更为快捷的复制和粘贴操作, 即选中需要复制的部分, 使用右键或 Ctrl+左键拖动该部分, 在指定位置松开右键或 Ctrl+左键后, 则将该部分复制并粘贴到该位置。

4. 模块的旋转与翻转

为了布局设计和连线方便, 常常需要对模块进行旋转或翻转操作。可以先选中模块, 然后在右键菜单或主菜单中的 Format 选项内选择 Rotate Block 或 Flip Block, 对模块进行旋转或翻转操作。

5. 模块的参数设置

当添加一个模块后, 还需要对该模块的相关参数进行设置。双击需要设置参数的模块, 弹出一个参数设置对话框, 可以将参数设置为常量, 也可以设置为变量(运行前先对变量赋值)。采样时间(Sample time)通常设为 -1(继承与输入相连的模块的采样时间), 如图 8-3 所示。

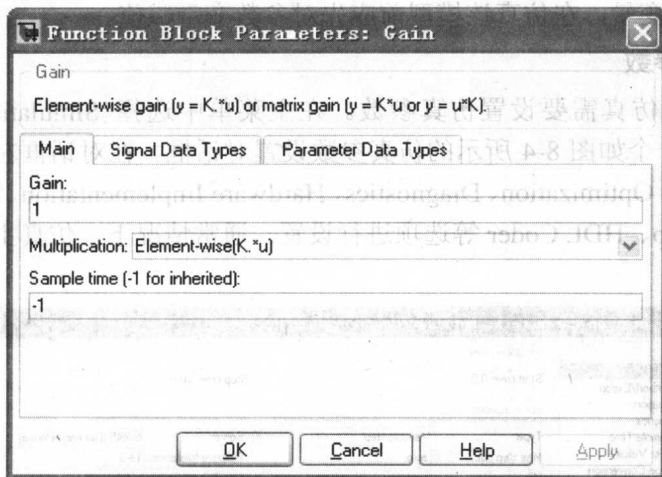


图 8-3 模块参数设置对话框

6. 模块的连接

模块之间信号线的连接可以用鼠标来完成。光标在输入端(或输出端)变为十字形状时, 按下鼠标左键并拖动鼠标至输出端(或输入端), 松开左键即可完成一条信号线的连接。

如果要在一条信号线上引出分支线并连接到其他模块, 可以在需要引出分支处按下鼠标右键或 Ctrl+左键, 拖动支线的箭头到指定位置或其他模块的输入端口, 然后松开右键或 Ctrl+左键即可。

7. 信号线的弯折、移动和删除

如果需要弯折信号线, 则拖动信号线时在需要弯折处松开鼠标, 再按下鼠标继续拖动即可。

如果需要移动已有的信号线, 则将鼠标指向该信号线, 按下左键进行移动, 然后松开

即可。

如果需要删除已有的信号线,则选中该信号线,通过主菜单或右键菜单选择 Delete,或者在键盘上按下 Delete 键即可。

8.1.4 Simulink 仿真步骤

Simulink 仿真步骤通常由以下几个部分组成。

1. 创建系统模型

要仿真一个系统,首先要新建一个模型文件并打开 Simulink 模型库浏览器,然后在模型编辑窗口添加系统模块并以适当的方式将这些模块连接起来,形成系统模型。一个仿真模型应该至少有一个输入(Sources 库)和一个输出(Sinks 库),创建完毕后应将模型文件保存在当前路径下。

2. 设置模块参数

用户可以根据特定需要设置系统各模块的参数。双击需要进行参数设置的模块,在弹出的对话框中填写模块参数,单击 OK 按钮即可完成设置。填入的参数可以是常数,也可以是变量。如果是变量,在仿真该模型前应先对参数变量赋值。

3. 设置仿真参数

仿真前应根据仿真需要设置仿真参数。在主菜单中选择 Simulation > Configuration Parameters,弹出一个如图 8-4 所示的仿真参数设置对话框。在对话框左侧可以对 Solver、Data Import/Export、Optimization、Diagnostics、Hardware Implementation、Model Referencing、Real-Time Workshop、HDL Coder 等选项进行设置。通常情况下,仿真主要设置的是 Solver 选项。






图 8-4 仿真参数设置对话框

Solver 选项的主要设置项如下:

- Simulation time: Start time(起始时间)和 Stop time(终止时间)。
- Solver options:

- Type(是否采用变步长算法)。
- Max step size(最大时间步长)和 Min step size(最小时间步长)。
- Solver(求解器)。
- Relative tolerance(相对允许误差)和 Absolute tolerance(绝对允许误差)。
- Initial step size(初始步长)。
- Zero crossing control(过零控制选项)。
- Solver diagnostic controls:
 - Number of consecutive min step size violations allowed(允许违反连续最小步长的最大次数)。
 - Consecutive zero crossings relative tolerance(连续过零相对允许误差)。
 - Number of consecutive zero crossings allowed(连续过零允许次数)。

4. 运行仿真

以上操作步骤完成后就可以对模型进行仿真了。选择主菜单中的 **Simulation > Start** 或单击工具栏按钮  启动仿真。仿真进行时, 模型编辑窗口状态栏上会显示仿真的进度, 用户可以等待仿真进行到终止时间或单击工具栏按钮  暂停仿真(暂停之后可以恢复仿真), 甚至可以单击工具栏按钮  强行终止仿真。

5. 仿真结果分析

仿真终止后即可对仿真结果进行观察和分析。通常的做法是使用示波器(Scope)观测, 即双击示波器观察波形, 再利用波形缩放工具实现更进一步的观察。另外, 还可以将仿真得到的数据输出到 MATLAB 工作区间, 然后对这些数据进行观察或分析。

下面通过实例来说明仿真的具体步骤。

【例】二阶振荡环节 $G(s) = \frac{1}{s^2 + 6s + 100}$ 的阶跃响应仿真。

(1) 新建一个模型文件, 保存并命名为 **untitled.mdl**。从模块库浏览器中选择 **Step**(阶跃输入)、**Transfer Fcn**(振荡环节传递函数)、**Scope**(示波器)并添加到模型编辑窗口, 然后连接各模块, 如图 8-5 所示。

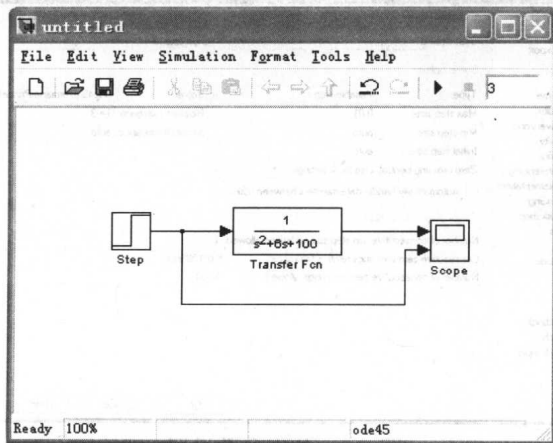


图 8-5 二阶振荡系统连接框图

(2) 设置各模块的参数。其中, Step 模块的参数设置如图 8-6 所示, Transfer Fcn 模块的参数设置如图 8-7 所示, Scope 模块的参数设置如图 8-8 所示。

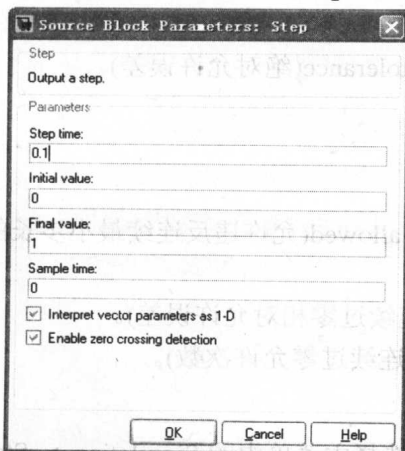


图 8-6 Step 模块的参数设置

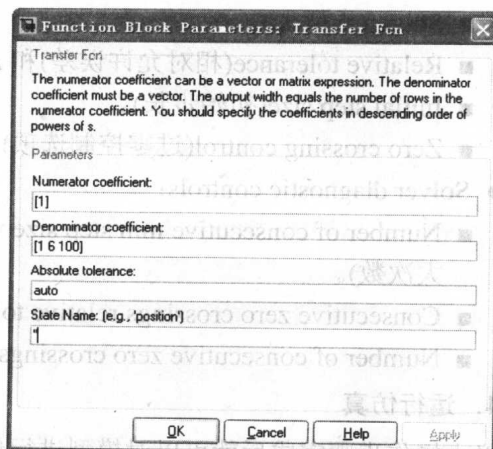


图 8-7 Transfer Fcn 模块的参数设置

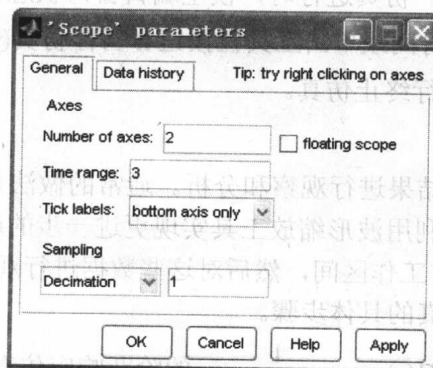


图 8-8 Scope 模块的参数设置

(3) 设置仿真参数。打开仿真参数设置对话框, 将 Stop time 设置为 3, Max step size 设置为 0.01, 其他参数保留默认值。设置结果如图 8-9 所示。

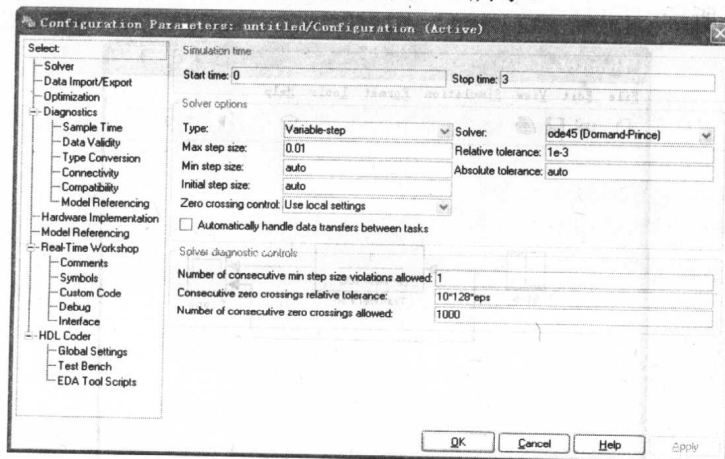


图 8-9 仿真参数设置

(4) 单击工具栏按钮 ▶ 运行仿真, 阶跃输入信号(下)及振荡环节的输出信号(上)的波形如图 8-10 所示。可以看出, 系统输出经过一段振荡后输出幅值稳定在 $1/100$ 。

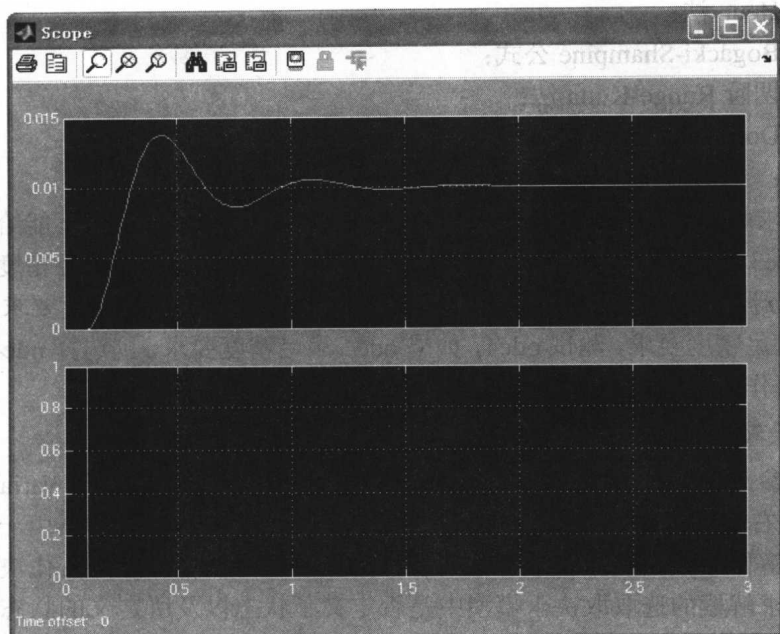


图 8-10 示波器显示的波形结果

8.1.5 Simulink 求解算法

Simulink 仿真是通过求解器来计算下一个时刻及其状态的。Simulink 的求解器可分为两类: 固定步长求解器和变步长求解器。对于固定步长求解器, 在仿真期间步长保持为常数; 而对于变步长求解器, 步长的大小取决于模型的动力学特性。当模型的状态变化迅速时, 变步长求解器将减小步长以保证计算精度; 当模型的状态变化缓慢时, 变步长求解器将增大步长以避免不必要的计算。

求解器的选择取决于模型的动力学特性和用户的配置计划。如果用户需要生成模型代码并在实时计算机系统上运行该代码, 则应该选择固定步长求解器来仿真模型, 这是因为实时计算机系统以固定的信号采样率运行, 而变步长求解器可能导致实时计算机系统的仿真丢失误差条件。如果不需要生成代码, 则求解器的选择取决于系统模型的动力学特性。

如果模型的状态变化迅速或包含不连续性, 则变步长求解器可以减小模型仿真所用的时间步长。这是因为对于一个模型, 变步长求解器比固定步长求解器需要更短的时间步长以达到可接受的精度级别。

1. 固定步长求解器

如果仿真参数设置对话框的 Solver options 选项区中的 Type 选项为 Fixed-step, 则可以选择 Solver 右侧下拉列表框列出的固定步长求解器。这类固定步长求解器由离散求解器和连续求解器两部分组成。固定步长求解器有如下几种:

- discrete: 无连续状态的离散状态模型算法;
- ode1: 欧拉法;
- ode2: Heun 法;
- ode3: Bogacki-Shampine 公式;
- ode4: 四阶 Runge-Kutta 法;
- ode5: Dormand-Prince 公式;
- ode14x: 结合牛顿法与外推法。

固定步长求解器的选择方法通常采用二进制查找法,即如果 ode1 不能给出精确的计算结果,测试 ode3,如果 ode3 满足精度要求,测试 ode2,如果 ode2 满足精度要求,则选择 ode2 作为求解器,否则选择 ode3 作为求解器;如果 ode3 不能满足精度要求,测试 ode5,如果 ode5 能满足精度要求,测试 ode4,如果 ode4 满足精度要求,则选择 ode4 作为求解器,否则选择 ode5 作为求解器。

2. 变步长求解器

如果仿真参数设置对话框的 Solver options 选项区中的 Type 选项为 Variable-step,则可以选择 Solver 右侧下拉列表框列出的变步长求解器。这类变步长求解器由一个离散求解器和几个连续求解器组成。变步长求解器的步长选择取决于模型状态的变化快慢程度。离散求解器或连续求解器的选取取决于模型中是否定义了状态以及所定义的状态类型。如果模型定义无状态或者只定义了离散状态,则应当选择离散求解器。事实上,即使这种情况下选择的是连续求解器,Simulink 仍会使用离散求解器来求解。变步长求解器有如下几种:

- discrete: 无连续状态的离散状态模型算法。
- ode45: 基于显式 Runge-Kutta (4,5)公式和 Dormand-Prince 的单步算法。例如,计算 $y(t_n)$,只需要知道前一个时刻处的解 $y(t_{n-1})$ 。通常 ode45 是大多数问题首选的最佳求解器,因此 ode45 也是 Simulink 连续状态模型的默认求解器。
- ode23: 基于显式 Runge-Kutta (2,3)与 Bogacki 和 Shampine 的单步算法。在处理粗略允许误差和弱刚性问题时较 ode45 有效。
- ode113: 变阶 Adams-Bashforth-Moulton PECE 多步算法。在允许误差有严格要求时较 ode45 有效。多步算法在计算当前时刻的值时需要知道之前多个时刻的解。
- ode15s: 基于数值微分公式(NDFs)的变阶多步求解算法。它比反向微分公式(BDFs 或 Gear 法)更有效。对于一个可能为刚性的问题,或者 ode45 不能有效解决的问题,可以尝试采用 ode15s 进行求解。
- ode23s: 基于改进的二阶 Rosenbrock 公式的单步算法。ode23s 在处理粗略允许误差问题时较 ode15s 有效。它还可以求解一些 ode15s 不能有效解决的刚性问题。
- ode23t: 使用“自由”内插式实现的梯形算法。如果问题为适度刚性且希望得到一个不存在数值阻尼的解,则使用该求解器。
- ode23tb: 由二阶反向微分公式的梯形法(TR-BDF2),即第一阶段采用梯形法而第二阶段采用二阶反向微分公式的隐式 Runge-Kutta 公式实现的算法。从构造上讲,两个阶段的估计都使用了相同的迭代矩阵。类似 ode23s, ode23tb 在处理粗略允许误差问题时较 ode15s 有效。

8.2 Simulink 的模块库

Simulink 模块库中的 Simulink 子库包含有 16 个模块库。这些模块库按照树状结构分类列出各模块，以便于用户查找和选择。掌握模块功能是 Simulink 仿真的关键，本节将对 Simulink 子库中的基本模块进行分类介绍。

8.2.1 Commonly Used Blocks 模块库

Commonly Used blocks 模块库中包含了仿真最常用的模块，这些模块在其他模块库中都可以找到，但是 Commonly Used Blocks 模块库提供了快捷的模块选取方法。该模块库中的各模块描述如表 8-1 所示。

表 8-1 Commonly Used Blocks 模块库

模 块 名	功 能 描 述
Bus Creator	将多个信号输入到总线
Bus Selector	将总线信号分解成多个信号
Contant	常数信号
Data Type Conversion	将输入信号转化为指定数据类型
Demux	将向量信号分离后输出
Discrete-Time Integrator	对离散时间信号进行积分或累加
Gain	常数增益
Ground	信号接地
In1	子系统或模型的输入端
Integrator	连续时间信号积分器
Logical Operator	逻辑运算
Mux	标量或向量信号合成为一路信号
Out1	子系统或模型的输出端
Product	乘法/除法器
Relational Operator	关系运算
Saturation	限制输入信号的最大值和最小值
Scope	示波器
Subsystem	子系统
Sum	输入信号相加减
Switch	由第二个输入信号选择在第一路或第三路之间切换
Terminator	未连接输出端口的终端
Unit Delay	离散信号延迟一个采样周期

8.2.2 Continuous 模块库

Continuous 模块库中主要包含了连续系统的仿真模块。该模块库中的各模块描述如表 8-2 所示。

表 8-2 Continuous 模块库

模 块 名	功 能 描 述
Derivative	数值微分器
Integrator	连续时间信号积分器
State-Space	状态空间模型
Transfer Fcn	传递函数
Transport Delay	以固定时间对输入信号进行延迟
Variable Time Delay	以可变时间对输入信号进行延迟
Variable Transport Delay	以可变时间对输入信号进行延迟
Zero-Pole	以零极点表示的传递函数

8.2.3 Discontinuities 模块库

Discontinuities 模块库中主要包含了不连续系统的仿真模块。该模块库中的各模块描述如表 8-3 所示。

表 8-3 Discontinuities 模块库

模 块 名	功 能 描 述
Backlash	指定系统死区宽度运行量
Coulomb and Viscous Friction	在零点的不连续性, 在其他处具有线性增益
Dead Zone	在死区内输出为零
Dead Zone Dynamic	在死区内输出为零, 死区宽度取决于上下限输入
Hit Crossing	检测输入信号是否穿越某个值
Quantizer	以指定间隔对信号进行量化
Rate Limiter	限制信号的变化率
Rate Limiter Dynamic	限制信号的变化率, 取决于变化率上下限输入
Relay	与上下阈值比较进行切换, 阈值之间保持不变
Saturation	限制信号的变化范围
Saturation Dynamic	限制信号的变化范围, 范围取决于上下限输入
Wrap To Zero	如果输入超过阈值, 输出为零; 否则输出等于输入

8.2.4 Discrete 模块库

Discrete 模块库中主要包含了不连续系统的仿真模块。该模块库中的各模块描述如表 8-4 所示。

表 8-4 Discrete 模块库

模 块 名	功 能 描 述
Difference	离散差分器
Discrete Derivative	离散微分器
Discrete Filter	离散滤波器
Discrete State-Space	离散状态空间模型
Discrete Transfer Fcn	离散传递函数
Discrete Zero-Pole	以零极点表示的离散传递函数
Discrete-Time Integrator	离散时间信号积分器
First-Order Hold	一阶采样保持器
Integer Delay	对信号进行 N 周期延时
Memory	输出是前一个时间步长时刻的输入
Tapped Delay	对信号进行 N 周期延时并输出所有延时版本
Transfer Fcn First Order	离散时间一阶传递函数
Transfer Fcn Lead or Lag	离散时间超前或滞后补偿器
Transfer Fcn Real Zero	只有实数零点而没有极点的离散传递函数
Unit Delay	对信号进行单周期延时
Weighted Moving Average	加权移动平均器
Zero-Order Hold	零阶保持器

8.2.5 Logic and Bit Operations 模块库

Logic and Bit Operations 模块库中的仿真模块主要用于实现逻辑运算和位运算。该模块库中的各模块描述如表 8-5 所示。

表 8-5 Logic and Bit Operations 模块库

模 块 名	功 能 描 述
Bit Clear	将整数的某一位设置为 0
Bit Set	将整数的某一位设置为 1
Bitwise Operator	对输入信号进行按位操作
Combinatorial Logic	实现一个真值表
Compare To Constant	确定信号与指定常数的比较方式
Compare To Zero	确定信号与零的比较方式
Detect Change	检测信号是否等于前一个采样时刻的值
Detect Decrease	检测信号是否小于前一个采样时刻的值
Detect Fall Negative	如果当前输入为负、前一时刻为非负，输出为真
Detect Fall Nonpositive	如果当前输入为非正、前一时刻为正，输出为真
Detect Increase	检测信号是否大于前一个采样时刻的值
Detect Rise Nonnegative	如果当前输入为非负、前一时刻为负，输出为真
Detect Rise Positive	如果当前输入为正、前一时刻为非正，输出为真
Extract Bits	输出为输入的相邻几位
Interval Test	检测输入是否在指定区间
Interval Test Dynamic	检测输入是否在指定区间，区间取决于上下限输入
Logical Operator	逻辑运算器
Relational Operator	关系运算器
Shift Arithmetic	算术移位

8.2.6 Lookup Tables 模块库

Lookup Tables 模块库提供了各种用于查表运算的模块。该模块库中的各模块描述如表 8-6 所示。

表 8-6 Lookup Tables 模块库

模 块 名	功 能 描 述
Cosine	定点查表余弦函数
Direct Lookup Table (n-D)	n 维直接查表器
Interpolation Using PreLookup	使用常数插值或线性插值实现的 n 维查表器
Lookup Table	使用线性插值实现的一维查表器
Lookup Table (2-D)	使用线性插值实现的二维查表器
Lookup Table (n-D)	使用插值实现的 n 维查表器
Lookup Table Dynamic	由给定数据生成一维近似函数
Prelookup	使用常数插值或线性插值对间断点序列进行检索
Sine	定点查表正弦函数

8.2.7 Math Operations 模块库

Math Operations 模块库中包含了大量的用于实现数学运算的模块。该模块库中的各模块描述如表 8-7 所示。

表 8-7 Math Operations 模块库

模 块 名	功 能 描 述
Abs	绝对值运算
Add	加减运算
Algebraic Constraint	建立输入状态表达式等于零的代数方程
Assignment	对输入信号的指定元素赋值
Bias	为输入增加一个偏移
Complex to Magnitude-Angle	计算复数输入信号的幅值及相角
Complex to Real-Imag	计算复数输入信号的实部及虚部
Divide	乘法/除法器
Dot Product	求内积
Gain	常数增益
Magnitude-Angle to Complex	由信号的幅值及相角输入组成复数
Math Function	数学函数
Matrix Concatenate	相同数据类型的矩阵输入信号串接
MinMax	输出为输入的最大值或最小值
MinMax Running Resettable	输出为输入的最大值或最小值，可由 R 复位
Permute Dimensions	重整多维数组的维数
Polynomial	由指定多项式系数估计多项式的值

续表

模 块 名	功 能 描 述
Product	乘法/除法器
Product of Elements	元素连乘器
Real-Imag to Complex	由信号的实部及虚部输入组成复数
Reshape	改变数据的维形状
Rounding Function	舍入运算
Sign	符号函数
Sine Wave Function	正弦函数
Slider Gain	使用滚动条设置增益
Squeeze	去除多维数组的单一维
Subtract	加减运算
Sum	求和运算
Sum of Elements	元素求和
Trigonometric Function	三角或双曲函数
Unary Minus	取负运算
Vector Concatenate	相同数据类型的向量输入信号串接
Weighted Sample Time Math	离散信号与采样时间的数学运算

8.2.8 Model Verification 模块库

Model Verification 模块库主要用于实现信号的检测。该模块库中的各模块描述如表 8-8 所示。

表 8-8 Model Verification 模块库

模 块 名	功 能 描 述
Assertion	检测信号是否非零
Check Discrete Gradient	检测离散信号相邻采样点的绝对值差是否小于上限
Check Dynamic Gap	检测信号幅值范围内的可能变化宽度
Check Dynamic Lower Bound	检测一个信号是否始终低于另一个信号
Check Dynamic Range	检测一个信号是否处于一个动态范围内
Check Dynamic Upper Bound	检测一个信号是否始终低于上限信号
Check Input Resolution	检测一个信号是否满足指定精度
Check Static Gap	检测信号的每个元素是否处于静态范围之外
Check Static Lower Bound	检测信号的每个元素是否大于指定的静态下限
Check Static Range	检测信号的每个元素是否处于指定的范围之内
Check Static Upper Bound	检测信号的每个元素是否小于指定的静态上限

8.2.9 Model-Wide Utilities 模块库

Model-Wide Utilities 模块库中的各模块描述如表 8-9 所示。

表 8-9 Model-Wide Utilities 模块库

模 块 名	功 能 描 述
Block Support Table	查看 Simulink 模块支持的数据类型
DocBlock	用于保存描述模块的大量文字信息
Model Info	用于显示模型相关版本信息
Time-Based Linearization	在指定时间创建该时刻系统的线性模型
Trigger-Based Linearization	创建触发时刻系统的线性模型

8.2.10 Ports & Subsystems 模块库

Ports & Subsystems 模块库主要用于端口和子系统的设计。该模块库中的各模块描述如表 8-10 所示。

表 8-10 Ports & Subsystems 模块库

模 块 名	功 能 描 述
Configurable Subsystem	表示指定模块库中的一系列模块
Atomic Subsystem	子系统
CodeReuse Subsystem	子系统
Enable	用在子系统中，用于为子系统添加使能端
Enabled and Triggered Subsystem	有触发端和使能端的子系统
Enabled Subsystem	有使能端的子系统
For Iterator Subsystem	循环执行子系统
Function-Call Generator	重复操作发生器，控制子系统执行顺序
Function-Call Subsystem	可由重复操作发生器控制的子系统
If	if-else 条件控制器
If Action Subsystem	可由 if-else 条件控制器控制的子系统
In1	子系统输入端
Model	将其他模型文件作为一个模块
Out1	子系统输出端
Subsystem	子系统
Subsystem Examples	其中包括不同类型的子系统实例
Switch Case	switch-case 条件控制器
Switch Case Action Subsystem	可由 switch-case 条件控制器控制的子系统
Trigger	用在子系统中，用于为子系统添加触发端
Triggered Subsystem	有触发端的子系统
While Iterator Subsystem	while 条件循环控制子系统

8.2.11 Signal Attributes 模块库

Signal Attributes 模块库主要用于信号和数据类型的转换。该模块库中的各模块描述如表 8-11 所示。

表 8-11 Signal Attributes 模块库

模 块 名	功 能 描 述
Bus to Vector	将虚拟总线转化为向量信号
Data Type Conversion	将输入信号转化为指定数据类型
Data Type Conversion Inherited	将 u 端输入的数据转化为第一个输入端数据类型
Data Type Duplicate	强制所有输入为相同数据类型
Data Type Propagation	根据参考信号设置传播信号的数据类型和标定
Data Type Propagation Examples	其中包括数据类型传播的实例
Data Type Scaling Strip	去除数据定点信号的标定
IC	设置信号的初始值
Probe	输出信号的宽度、维数、采样时间、复数标志等属性
Rate Transition	运行在不同速率的模块之间的数据句柄传递
Signal Conversion	不改变信号的值, 将信号转化为新的类型
Signal Specification	指定信号的维数、采样时间、数据类型、数值类型以及其他属性
Weighted Sample Time	信号与采样时间进行运算
Width	以指定数据类型输出输入信号的宽度

8.2.12 Signal Routing 模块库

Signal Routing 模块库主要用于实现信号路径的控制。该模块库中的各模块描述如表 8-12 所示。

表 8-12 Signal Routing 模块库

模 块 名	功 能 描 述
Bus Assignment	对总线的指定元素赋值
Bus Creator	将多个信号输入到总线
Bus Selector	将总线信号分解成多个信号
Data Store Memory	定义数据存储器
Data Store Read	从数据存储器读取数据
Data Store Write	向数据存储器写入数据
Demux	将向量信号分离后输出
Environment Controller	创建模块结构图分支, 用于仿真或代码生成
From	从 Goto 模块接收信号输入
Goto	将信号发送到指定标记的 From 模块
Goto Tag Visibility	定义 Goto 模块标签范围
Index Vector	根据第一个输入值切换开关
Manual Switch	手动切换开关, 可通过双击进行切换
Merge	将多路信号合并为单路信号
Multiport Switch	由第一个输入端控制在多个输入之间进行切换
Mux	标量或向量信号合成为一路信号
Selector	从向量、矩阵或多维信号中选取部分元素
Switch	由第二个输入信号选择在第一路或第三路之间切换

8.2.13 Sinks 模块库

Sinks 模块库主要提供信号的显示或信号的输出连接。该模块库中的各模块描述如表 8-13 所示。

表 8-13 Sinks 模块库

模 块 名	功 能 描 述
Display	数字显示输入值
Floating Scope	浮动示波器
Out1	子系统或模型的输出端
Scope	示波器
Stop Simulation	当输入非零时终止仿真
Terminator	未连接输出端口的终端
To File	将数据写入 MAT 文件
To Workspace	将数据写入 MATLAB 工作区间
XY Graph	在图形窗口中绘制 X-Y 图像

8.2.14 Sources 模块库

Sources 模块库提供了大量的信号发生器模块。该模块库中的各模块描述如表 8-14 所示。

表 8-14 Sources 模块库

模 块 名	功 能 描 述
Band-Limited White Noise	连续系统引入白噪声
Chirp Signal	产生频率递增的正弦波信号
Clock	时钟信号
Constant	常数信号
Counter Free-Running	加法计数器，溢出后清零
Counter Limited	加法计数器，超过上限后清零
Digital Clock	以指定采样间隔输出仿真时间的数字钟
From File	从 MAT 文件读取数据
From Workspace	从 MATLAB 工作区间读取数据
Ground	接地
In1	子系统或模型的输入端
Pulse Generator	脉冲信号发生器
Ramp	固定斜率的斜坡信号发生器
Random Number	产生正态分布随机数
Repeating Sequence	由数据序列产生周期信号
Repeating Sequence Interpolated	重复输出离散时间序列，数据点之间插值
Repeating Sequence Stair	重复输出离散时间序列，数据点之间为常数
Signal Builder	产生具有分段线性波形的可交换信号组
Signal Generator	产生多种波形
Sine Wave	正弦信号发生器
Step	产生阶跃信号
Uniform Random Number	产生均匀分布随机数

8.2.15 User-Defined Functions 模块库

User-Defined Functions 模块库主要用于实现自定义函数功能，包括 MATLAB 函数和 S-函数。该模块库中的各模块描述如表 8-15 所示。

表 8-15 User-Defined Functions 模块库

模 块 名	功 能 描 述
Embedded MATLAB Function	MATLAB 函数，可通过编辑 M 文件实现函数功能
Fcn	表达式函数模块
Level-2 M-File S-Function	Level-2 M 文件 S-函数
MATLAB Fcn	MATLAB 函数库函数
S-Function	S-函数
S-Function Builder	S-函数编译器，由 C 代码生成 S-函数
S-Function Examples	其中包括 S-函数设计实例

8.3 子系统及封装技术

使用子系统可以将复杂的系统划分为多个模块，以便于对整个系统进行观察和操作。MATLAB 还提供了子系统的封装功能，可以为子系统创建图标、隐藏内容，使子系统的操作更为方便。此外，还可以将子系统封装成为自定义的功能模块，或者创建自定义的模块库。

8.3.1 创建子系统

当模型的规模和复杂性增加时，可以通过将模块组成子系统来简化模型。使用子系统有如下优点：

- 减小模型窗口内显示的模块数；
- 可将功能关系密切的模块集合在一起；
- 实现层次化的模型图，子系统处于某一层，组成该子系统的模块处于另一层。

创建一个子系统通常有以下两种方法：

- 在模型内添加一个 Subsystem 模块，打开该模块并在该子系统窗口内添加模块；
- 在模型内添加构成子系统的模块，然后由这些模块创建组成子系统。

1. 通过添加 Subsystem 模块创建子系统

在添加组成子系统的各模块之前，首先在模型内添加一个子系统，然后再在子系统中添加组成子系统的模块。具体操作步骤如下：

- (1) 从 Ports & Subsystems 库将 Subsystem 模块复制到当前模型内。
- (2) 双击打开 Subsystem 模块。
- (3) 在子系统窗口内创建子系统，使用输入端模块和输出端模块分别表示子系统的外部输入和外部输出。

【例】 利用 Subsystem 模块创建子系统。

创建一个名为 untitled.mdl 的模型文件，添加信号源、示波器和子系统模块，如图 8-11 所示。

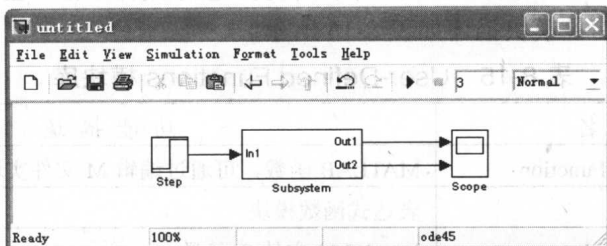


图 8-11 系统模块连接

打开子系统模块，在子系统中添加传递函数模块，如图 8-12 所示。

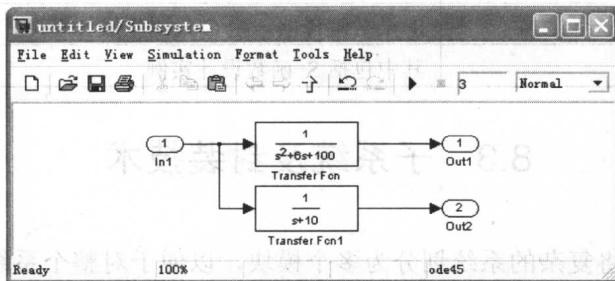


图 8-12 子系统模块连接

2. 通过模型中已有模块创建组成子系统

如果模型中存在一些相关模块，并且希望将这些模块组成子系统，则可以通过选中这些模块来创建子系统。具体操作步骤如下：

(1) 按下鼠标左键，拖动鼠标形成包围框，框中用于创建子系统的模块以及相关连接线，松开鼠标完成选定。需要注意的是，不可以一个接一个地选择模块或者使用 Select All 命令。

(2) 从右键菜单或 Edit 菜单中选择 Create Subsystem，Simulink 则使用子系统模块取代所选中的这些模块。

【例】 通过模型中已有模块创建子系统。

创建一个名为 untitled.mdl 的模型文件，添加如图 8-13 所示的各模块，并用鼠标选中需要创建生成子系统的部分。

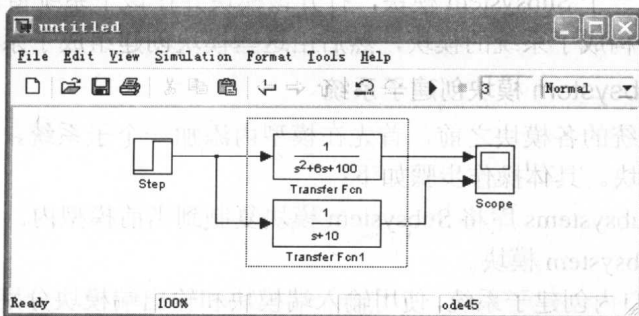


图 8-13 选中部分模块

松开鼠标左键, 在 Edit 菜单下选择 Create Subsystem, 生成的子系统模块如图 8-14 所示。

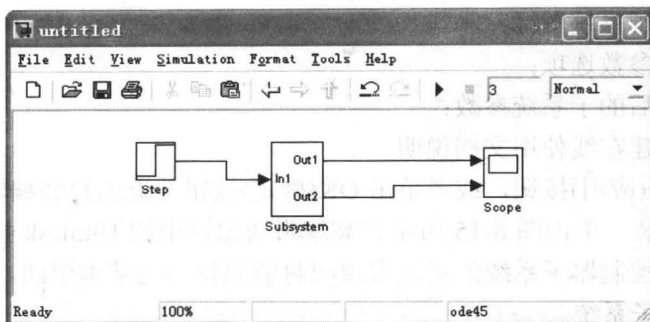


图 8-14 生成的子系统模块

8.3.2 封装子系统

子系统封装是自定义的用户界面, 用于隐藏子系统内容, 并且使子系统具有类似 Simulink 原型模块的外部特性, 如显示子系统图标、参数设置对话框等。Simulink 的子系统封装编辑器提供了以下封装功能:

- 只需要用一个参数对话框就可以实现对整个子系统的描述、参数设置以及帮助信息的显示。
- 使用自定义图标取代标准子系统图标来表示子系统。
- 隐藏子系统的内部模块构成。
- 通过在封装子系统中定义模块的行为来创建自定义模块, 并将封装后的子系统放入一个模块库。

通常封装一个子系统的步骤如下:

- (1) 选择需要封装的子系统模块。
- (2) 通过模型右键菜单或者模型编辑器的 Edit 菜单选择 Mask Subsystem, 出现如图 8-15 所示的封装编辑器窗口。

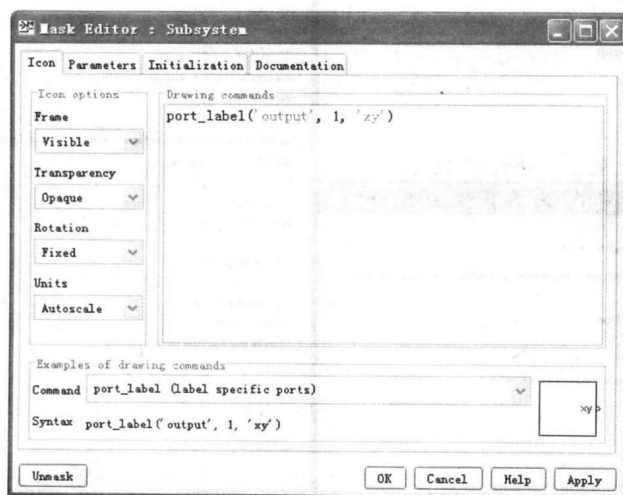


图 8-15 封装编辑器窗口

(3) 设置封装编辑器的各项标签页，实现如下功能：

- 创建自定义子系统图标；
- 创建子系统参数选项；
- 初始化封装后的子系统参数；
- 为子系统创建在线使用文档说明。

(4) 单击 Apply(应用)按钮，或者单击 OK(确定)按钮并退出封装编辑器。

如果想取消封装，单击图 8-15 所示封装编辑器窗口中的 Unmask 按钮即可。下面通过封装一个离散 PID 控制器子系统的实例来说明封装编辑器的基本用法。

【例】 封装子系统。

创建一个模型，其中包含名为 Subsystem 的子系统，如图 8-16、图 8-17 所示。

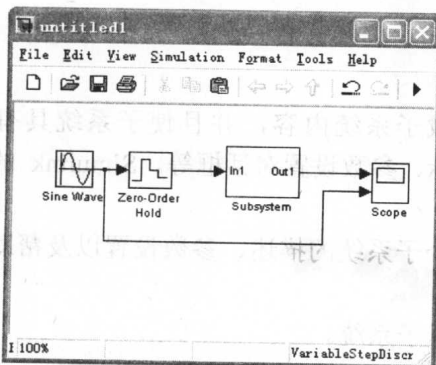


图 8-16 系统模块连接

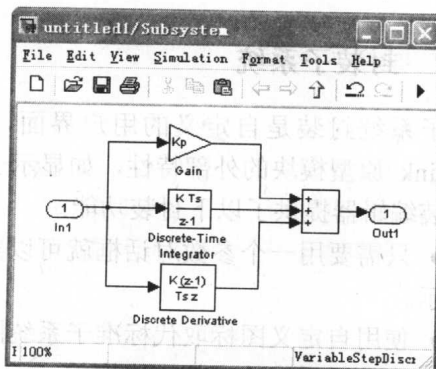


图 8-17 子系统模块连接

打开封装编辑器，将比例器、离散时间积分器、离散微分器的增益分别设为 K_p 、 K_i 、 K_d ，如图 8-18 所示。

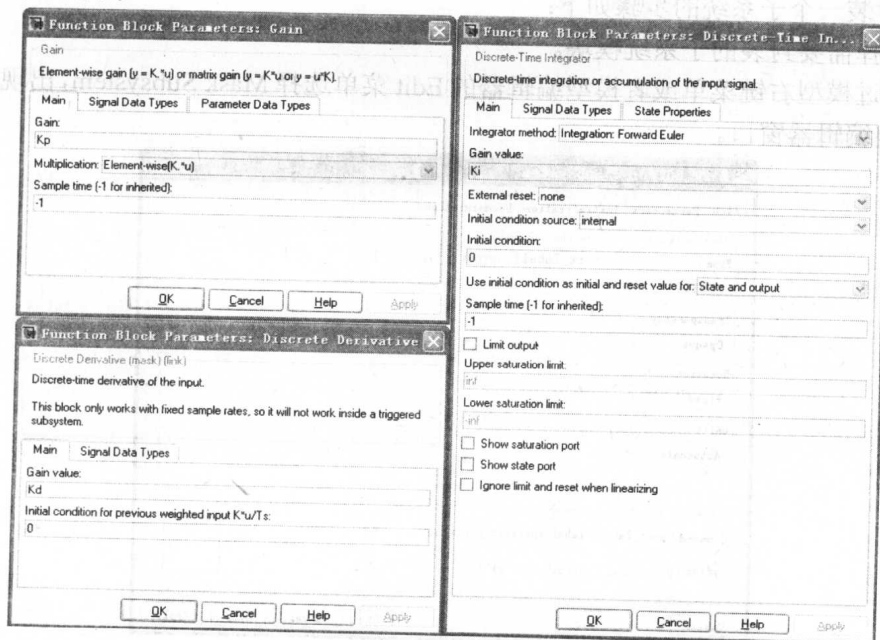


图 8-18 模块参数设置

打开封装编辑器，在Icon标签页的Drawing commands文本命令区输入命令disp('PID')，则子系统的图标变成居中显示的文字“PID”，如图8-19所示。

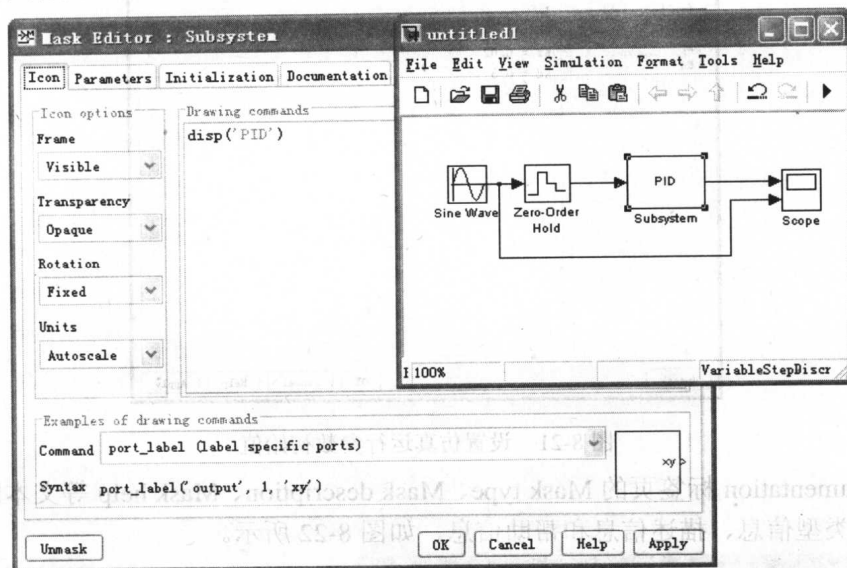


图 8-19 更改图标

在 Parameters 标签页下设置参数 Kp、Ki、Kd，它们在参数对话框中的名称分别为 Proportion、Integral、Derivative，如图 8-20 所示。

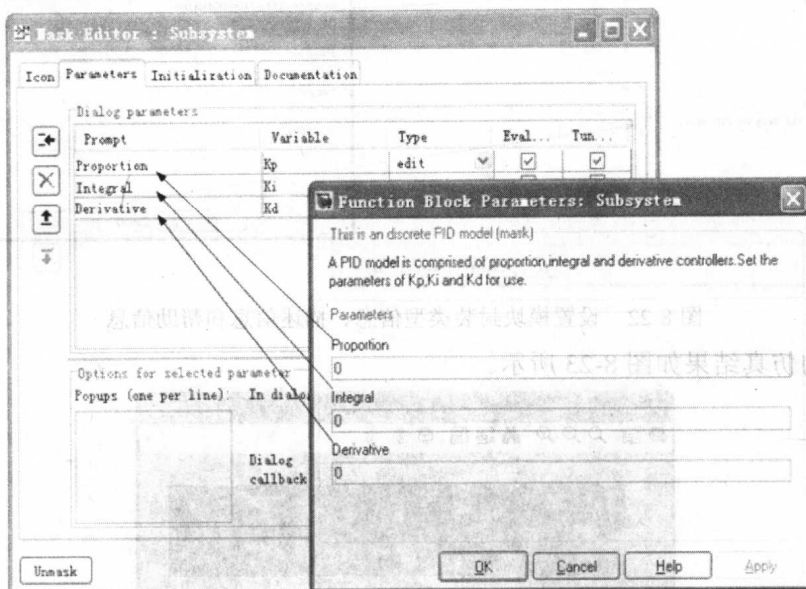


图 8-20 设置参数名称

在 Initialization 标签页的 Initialization commands 文本命令区输入仿真运行参数初始化代码，如图 8-21 所示。每行命令后无分号，运行时 MATLAB 命令窗中会显示这些命令的运行结果。

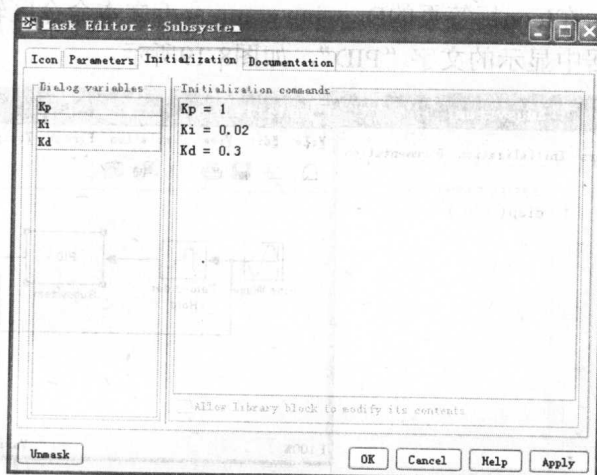


图 8-21 设置仿真运行参数初始值

在 Documentation 标签页的 Mask type、Mask description、Mask help 等文本区分别输入模块的封装类型信息、描述信息和帮助信息，如图 8-22 所示。

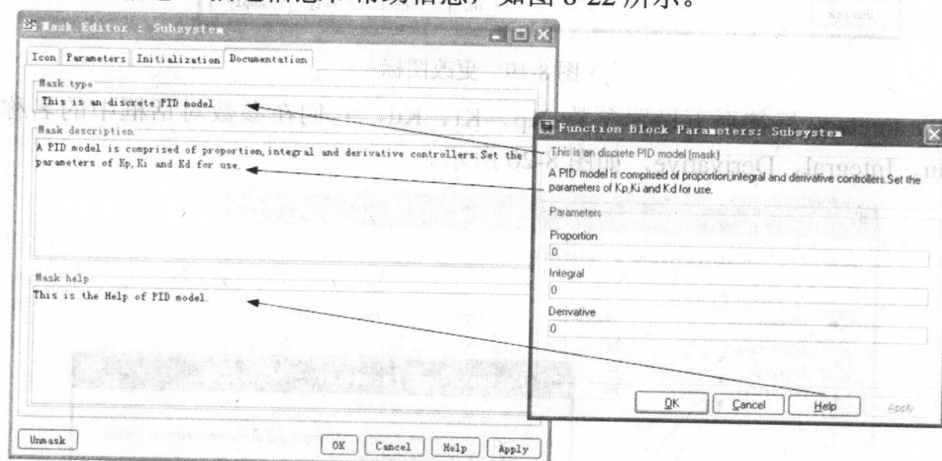


图 8-22 设置模块封装类型信息、描述信息和帮助信息

运行后的仿真结果如图 8-23 所示。

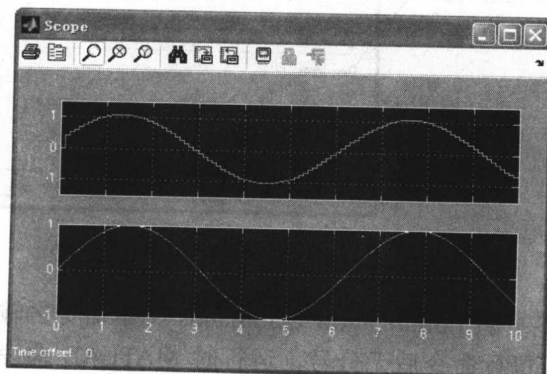


图 8-23 系统仿真波形结果

8.3.3 自定义模块库

Simulink 允许用户创建自己的模块库,并将封装好的子系统模块放入其中。创建一个模块库时,首先要通过 File 菜单新建一个模块库文件,接下来将显示一个标题为 Library: untitled 的窗口。如果已经存在名为 untitled 的窗口,则创建的文件名后按顺序附加数字。模块库名不得与模型文件重名。

当创建好一个库文件时,可以将模型或其他模块库中的模块拖动到新建的库文件中。这个自定义模块库中包含了所有经常用到的模块或子系统,将这个模块库保存后,如果需要新建一个系统模型,不需要打开 Simulink 模块库浏览器,而只需要打开自定义模块库,直接从自定义的模块库中选择模块并拖到模型编辑器中就可以实现模型的创建。

8.4 仿真运行与分析

仿真的运行与分析在前面的小节中曾作过简要的介绍,本节将对这个环节作更为详细的阐述,以便使读者掌握更全面的仿真运行控制、参数设置以及对仿真结果进行分析的方法。

8.4.1 仿真的运行控制

通过使用 Simulink 仿真界面主菜单、工具栏可以方便地实现仿真的启动、停止和暂停,这在 8.1.4 小节中已经作过简要的介绍。本小节主要介绍使用模块实现仿真运行控制的方法。

1. 使用仿真停止模块

使用 Stop Simulation 模块可以实现有条件的仿真停止。当该模块的输入不为零时,仿真将会终止。使用该模块的步骤如下:

- (1) 从 Sinks 模型库中选中 Stop Simulation 模块并添加到当前模型内。
- (2) 将控制信号接入该模块的输入端,当信号非零时,仿真停止。如果输入为向量,只要有一个元素不为零,仿真都会停止。

例如,图 8-24 所示的模型将在时钟信号到达 5 时停止仿真(Compare To Constant 模块的输出数据为布尔类型)。

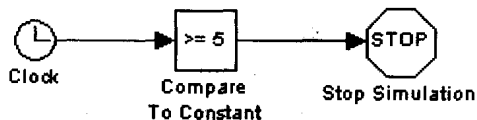


图 8-24 使用 Stop Simulation 模块

2. 使用仿真暂停模块

使用 Assertion 模块可以实现有条件的仿真暂停。当该模块的输入为零时,仿真将会暂停。使用该模块的步骤如下:

- (1) 从 Model Verification 模型库中选中 Assertion 模块并添加到当前模型内。
- (2) 将控制信号接入该模块的输入端,当信号为零时,仿真暂停。
- (3) 打开 Assertion 模块参数对话框,在 Simulation callback when assertion fails 项下输入如下命令:


```
set_param(bdroot,'SimulationCommand','pause'),
```

```
disp(sprintf('\nSimulation paused.'))
```

不选中 Stop simulation when assertion fails 选项。

(4) 单击 OK 完成设置。

例如, Assertion 模块的使用如图 8-25 所示, 模型设置如上。

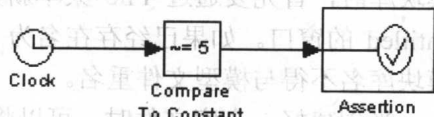


图 8-25 使用 Assertion 模块

当运行时间等于 5 时, 仿真将会暂停, 并且

MATLAB 命令窗会显示如下信息:

```
Simulation paused.
```

```
Warning: Assertion detected in 'stop/Assertion' at time 5.000000.
```

仿真暂停之后, 从模型编辑器主菜单的 Simulation 菜单中选择 Continue 或在工具栏上按下 Continue 按钮可以继续完成仿真。

8.4.2 仿真数据的输入和输出

Simulink 允许用户由 MATLAB 工作区间输入信号和初始状态, 并且可以在仿真期间将信号和状态数据输出到 MATLAB 工作区间。本小节将通过实例介绍 Simulink 仿真数据和状态的输入和输出方法。

1. 数据输入

以数组数据形式输入的方法是最常用的数据输入方法, 其中的数据必须是 double 类型的实数矩阵。矩阵的第一列是按升序排列的时间向量, 其余的列为对应时间向量的输入值向量。这些输入值向量分别代表按数字顺序排列的 In1 模块的输入值, 而矩阵的行则代表某个时刻的输入值。

如果 n 为总的输入端数目, 则矩阵的列数等于 $n + 1$ 。

默认的输入表达式为 $[t,u]$, 默认的输入形式为数组。可以设置输入模块的维数, 使输入数据为数组。下面通过一个具体例子来说明。

【例】 数组的输入。

在命令窗输入:

```
>> t = (0:0.01:10)';
```

```
>> u = [sin(t), cos(3*t), cos(5*t)];
```

创建一个模型文件, 系统框图如图 8-26 所示。

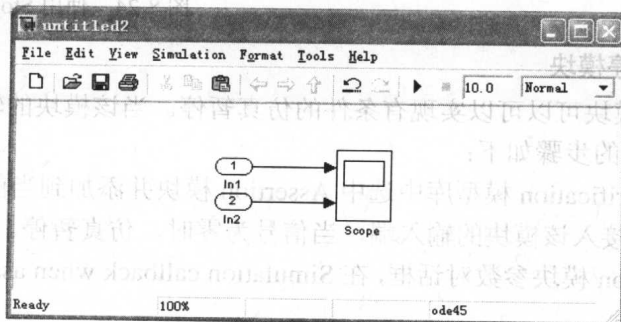


图 8-26 系统模块连接

将 In1 模块参数设置对话框中 Signal Specification 标签页的 Port dimensions 项设置为 2, 如图 8-27 所示。

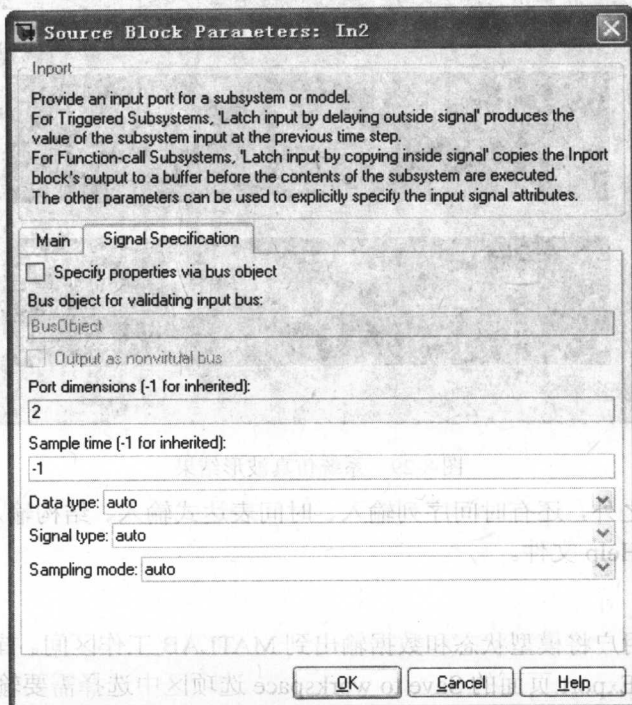


图 8-27 In1 模块参数设置

在参数设置对话框中选择 Data Import/Export, 选中 Input, 具体设置如图 8-28 所示。

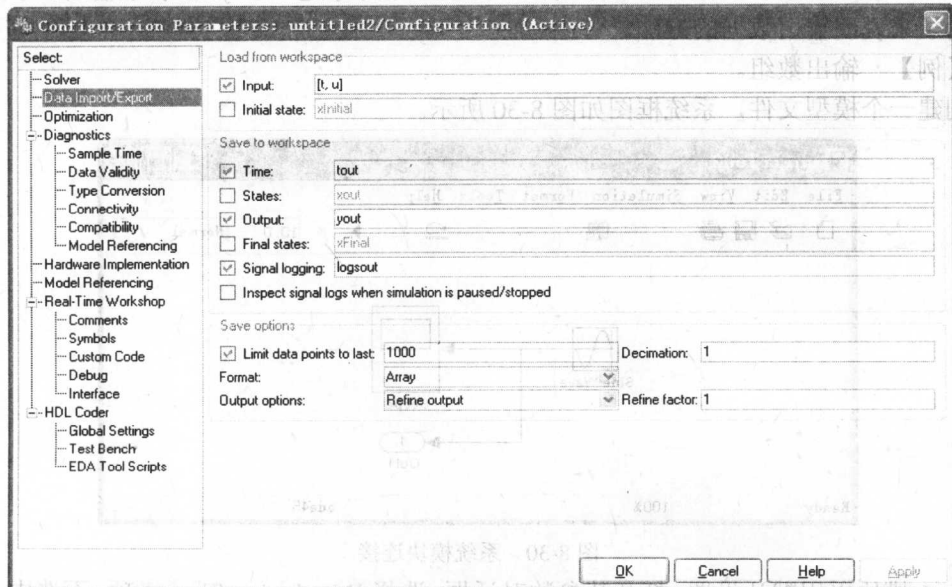


图 8-28 Data Import/Export 参数设置

运行仿真，结果如图 8-29 所示。

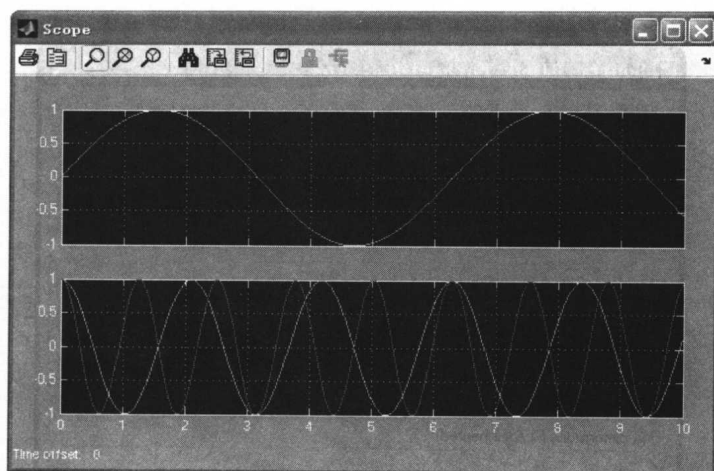


图 8-29 系统仿真波形结果

除过数组输入之外，还有时间序列输入、时间表达式输入、结构输入等数据输入方法。具体用法可以参考 Help 文件。

2. 数据输出

Simulink 允许用户将模型状态和数据输出到 MATLAB 工作区间。首先要在参数设置对话框的 Data Import/Export 页面的 Save to workspace 选项区中选择需要输出的类型以及保存到工作区间的变量名称，然后可在 Save options 选项区指定输出数据的保存格式和数量，数据格式可以是数组、结构或包含时间字段的结构。

输出的数组数据默认以 tout 表示时间、以 xout 表示状态、以 yout 表示输出。下面通过一个例子来说明输出数据的方法。

【例】 输出数组。

创建一个模型文件，系统框图如图 8-30 所示。

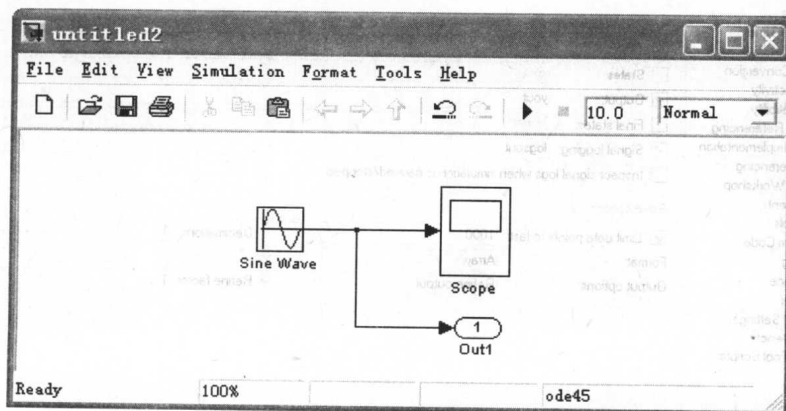


图 8-30 系统模块连接

Out1 模块采用默认设置，打开其参数对话框，选择 Data Import/Export 项，不选中 Input，选中 Output，如图 8-31 所示。

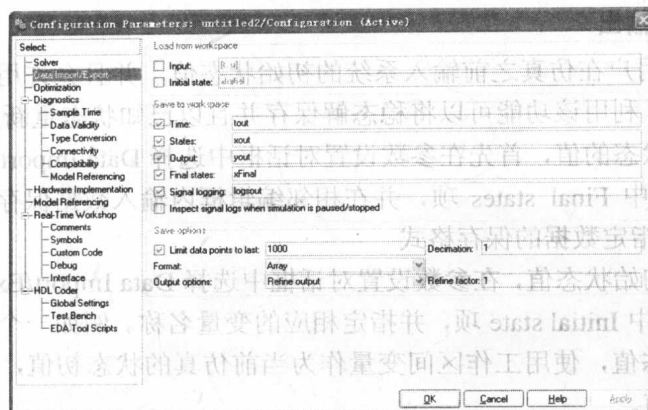


图 8-31 Data Import/Export 参数设置

运行仿真，结果如图 8-32 所示。

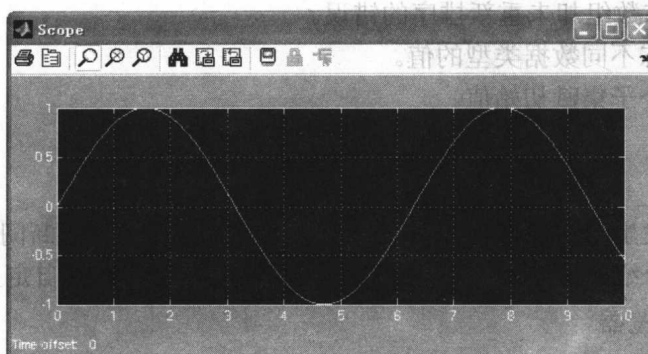


图 8-32 系统仿真波形结果

MATLAB 工作区间中会出现 tout、yout 等变量，在命令窗输入：

```
>> plot(tout,yout)
```

运行结果如图 8-33 所示。

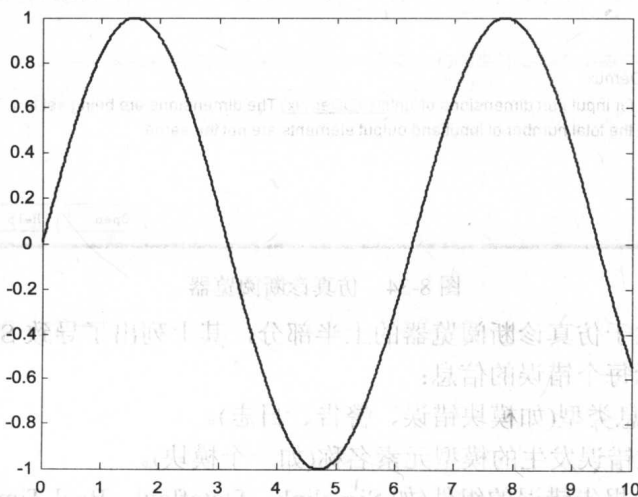


图 8-33 利用输出数据绘图结果

3. 状态输入和输出

Simulink 允许用户在仿真之前输入系统的初始状态值, 并且允许用户在仿真结束后输出系统的末状态值。利用该功能可以将稳态解保存并且以已知状态重新启动仿真。

要保存模型末状态的值, 首先在参数设置对话框中选择 Data Import/Export, 在 Save to workspace 选项区选中 Final states 项, 并在相邻编辑框内输入数据保存后的名称, 然后在 Save options 选项区指定数据的保存格式。

要载入模型的初始状态值, 在参数设置对话框中选择 Data Import/Export, 在 Load from workspace 选项区选中 Initial state 项, 并指定相应的变量名称。例如一个变量包含了前一个仿真保留下来的状态值, 使用工作区间变量作为当前仿真的状态初值, 将覆盖模型中各模块设置过的仿真初值。

如果需要完成如下功能, 则可以使用结构或包含时间字段的结构来指定初始状态值:

- 将初始状态值直接与状态的完全路径名相关联, 这样可以避免可能发生于模型状态重新排序而初始状态数组却未重新排序的错误。
- 赋给初始状态不同数据类型的值。
- 给状态的一个子集赋初始值。

8.4.3 错误诊断

如果仿真期间发生了错误, 打开导致错误的子系统, 在仿真诊断浏览器中会显示出错误信息。本小节将介绍如何使用该阅读器检查错误, 以及如何创建自定义错误信息。

1. 仿真诊断浏览器

仿真诊断浏览器由错误摘要页面和错误信息页面构成, 如图 8-34 所示。

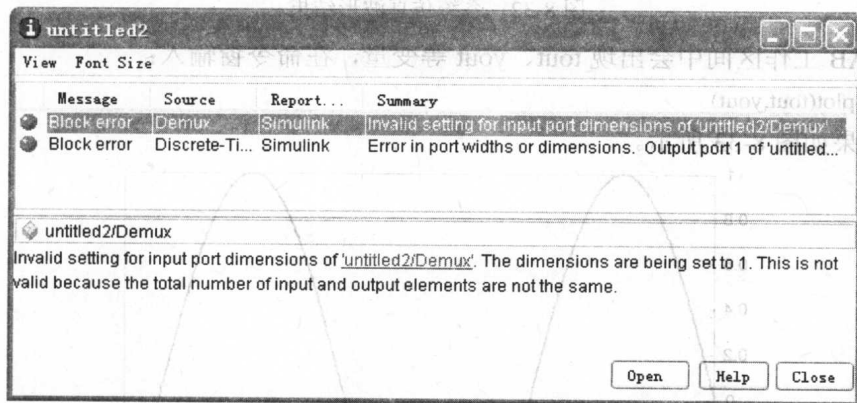


图 8-34 仿真诊断浏览器

错误摘要页面位于仿真诊断浏览器的上半部分, 其上列出了导致 Simulink 仿真终止的错误。该页面将显示每个错误的信息:

- Message: 信息类型(如模块错误、警告、日志)。
- Source: 导致错误发生的模型元素名称(如一个模块)。
- Reported by: 报告错误的组件(如 Simulink、Stateflow、Real-Time Workshop 等)。
- Summary: 错误信息。

通过 View 菜单可以选择显示或不显示其中的某列信息。

错误信息页面位于仿真诊断浏览器的下半部分，其上显示的是对应错误摘要条目的错误摘要，单击错误摘要页面上的其他错误条目可以浏览每个错误的摘要。

如果错误存在于子系统中，Simulink 将打开存在第一个出错源的子系统并突出显示该错误源。

点击以蓝色突出显示的错误信息超链接或者单击 Open 按钮将定位到出错源。

2. 创建自定义错误信息

在回调 S-函数或 MATLAB Fcn 模块中使用 MATLAB 的 error 函数可以创建自定义错误信息，其方法有以下几种：

- 显示文本字符串；
- 包含对象的超链接；
- 链接到 HTML 文件。

如果要在出错时显示文本字符串，则将字符串用引号包括起来，作为 error 函数的输入参量。下面通过一个具体的例子来说明。

【例】 出错时显示字符串 Signal is zero.

创建一个模型文件，模块连接如图 8-35 所示。

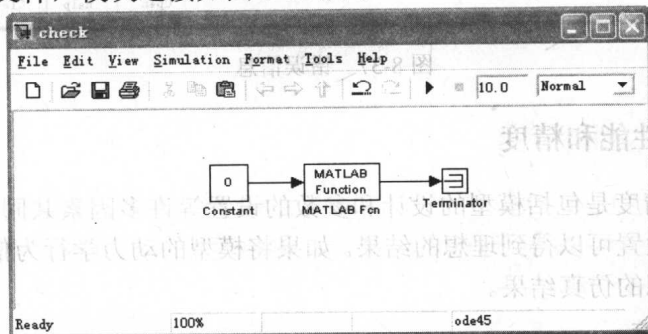


图 8-35 系统模块连接

设置 MATLAB Fcn 模块参数，如图 8-36 所示。

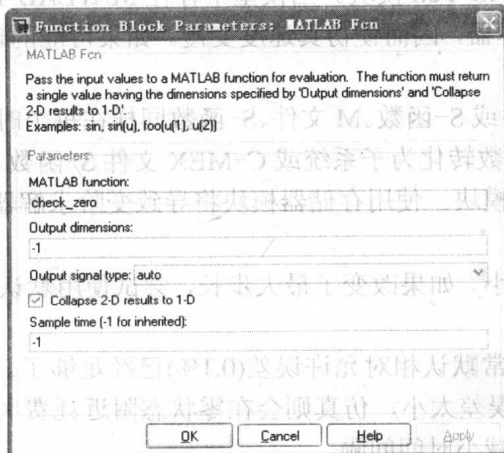


图 8-36 MATLAB Fcn 模块参数设置

新建一个名为 check_zero.m 的函数文件，输入如下代码：

```
function y = check_zero(x)
```

```
if x == 0
```

```
error('Signal is zero');
```

```
else
```

```
y = x;
```

```
end
```

运行仿真，弹出错误信息，如图 8-37 所示。

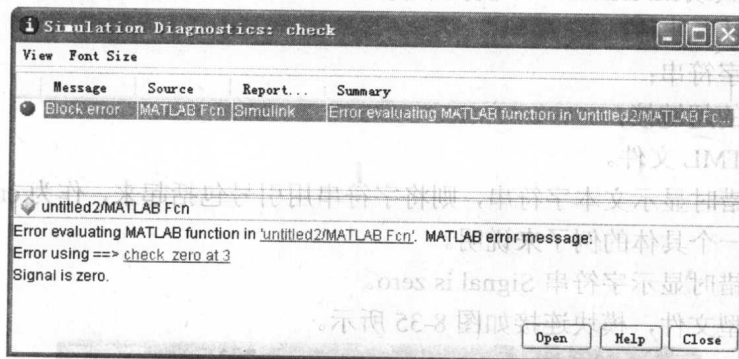


图 8-37 错误信息

8.4.4 改善仿真性能和精度

仿真的性能和精度是包括模型的设计和参数的设置等许多因素共同决定的结果。改变求解器的默认参数设置可以得到理想的结果。如果将模型的动力学行为信息提供给求解器，那么可以得到更理想的仿真结果。

1. 提高仿真速度

仿真速度缓慢可能由多种因素引起，下面给出了其中的一部分因素：

- 模型中存在 MATLAB Fcn 模块。当模型中存在 MATLAB Fcn 模块时，每个时间步长都将调用 MATLAB 解释器，因而使仿真速度变慢。如果可能的话，建议尽量使用内建函数模块或数学函数模块。

- 模型中存在 M 文件或 S-函数。M 文件、S-函数同样在每个时间步长处调用 MATLAB 解释器。可以考虑将 S-函数转化为子系统或 C-MEX 文件 S-函数。

- 模型中存在存储器模块。使用存储器模块将导致变阶求解器(ode15s 和 ode113)在每个时间步长处复位为 1 阶。

- 最大步长设置得太小。如果改变了最大步长，尝试使用默认最大步长设置(auto)运行仿真。

- 精度要求过高。通常默认相对允许误差(0.1%)已经足够了。对于存在状态可变化到零的模型，如果绝对允许误差太小，仿真则会在零状态附近耗费大量的步进。

- 时间标度过长，可减小时间间隔。

- 使用非刚性求解器求解可能为刚性的问题，可尝试使用 ode15s。

- 模型使用的各采样时间之间不存在倍数关系。混合采样时间将导致求解器使用足够小的步长以保证采样时间满足模型中所有的采样时间。

- 模型中存在代数环。代数环是通过在每个时间步长处迭代计算得出解，因此会严重地降低仿真性能。

- 模型中存在随机数模块馈入积分模块，对于连续系统，建议使用 Sources 模块库中的 Band-Limited White Noise 模块。

2. 提高仿真精度

为选择仿真精度，应该以合理的时间间隔运行仿真。将相对允许误差减小到 $1e-4$ 或者减小绝对允许误差，再次运行仿真，然后比较两次仿真的结果，如果两次的结果没有太大的差异，则可以确认解是收敛的。

如果仿真在启动时就丢失了关键行为，应减小初始步长以确保仿真没有跳过关键行为。

如果仿真结果随着时间变化到不稳定状态，则

- 系统可能是不稳定的。
- 如果使用的是 ode15s 求解器，可能需要限制最大阶数为 2 或者尝试使用 ode23s 求解器。

如果仿真结果不能满足精度要求，则

- 对于一个状态值接近零的模型，如果绝对允许误差参数太大，仿真则会在状态值接近零的区域使用过少的步进。因此要减小该参数值，或者在 Integrator 对话框中将其调整为独立状态。

- 如果减小绝对允许误差不能完全改善精度，则应减小相对允许误差以减少可以接受的错误并实现更小的步长和更多的步进。

特定建模结构也可以产生不能满足精度要求的仿真结果：

- 一个 Source 模块继承自身的采样时间可能产生不同的仿真结果。
- 代数环中的一个 Derivative 模块可能导致求解器失去精度。

8.4.5 使用命令运行仿真

在 MATLAB 命令窗或 M 文件中输入一个仿真命令，可以实现仿真模型的自动运行。通过随机改变参数和循环运行仿真还可以实现 Monte Carlo 分析。使用 sim 命令或 set_param 命令可以有计划地进行仿真。

1. 使用 sim 命令

sim 命令用于控制仿真的运行，其调用格式如下：

```
[t,x,y] = sim(model, timespan, options, ut);
```

模型的参数不能通过该命令设置，而要通过参数对话框来实现。参数 options 是一个提供附加参数设置的结构，包含求解器名称和允许误差。使用 simset 命令可以定义 options 结构中的参数。

2. 使用 set_param 命令

使用 set_param 命令可以实现仿真的启动、停止、暂停、继续、模块图更新或者将所有记录变量写入基本工作区间，其调用格式如下：

```
set_param('sys','SimulationCommand','cmd')
```

其中, 'sys' 为系统名称, 'cmd' 可以为 'start'、'stop'、'pause'、'continue'、'update'、'WriteDataLogs'。

类似地, 可以使用 `get_param` 命令检测仿真状态, 其调用格式如下:

```
get_param('sys','SimulationStatus')
```

Simulink 将返回 'stopped'、'initializing'、'running'、'paused'、'updating'、'terminating'、'external' 等信息。

S-函数中可以使用 `set_param` 命令来控制仿真的执行。一个 C MEX S-函数可以使用 `mexCallMatlab` 宏来调用 `set_param` 命令。

8.4.6 观察输出轨迹

Simulink 的输出轨迹可以通过以下几种方法绘制:

- 将信号接到 Scope 模块或 XY Graph 模块的输入端。
- 将输出写入返回变量并使用 MATLAB 绘图命令。
- 使用 To Workspace 模块将输出写入工作区间, 使用绘图命令绘制结果。

1. 使用 Scope 模块

使用 Scope 示波器模块可以直接显示仿真输出的波形。Scope 模块不但可以显示输出轨迹, 还可以放大或缩小波形图。用户也可以利用 Scope 模块将输出数据保存到 MATLAB 工作区间。

利用 XY Graph 模块可以实现相图轨迹的显示。

这些模块的应用实例在前面已经作过介绍, 此处不再重复。

2. 使用返回变量

使用返回变量可以保存输出数据, 还可以使用 MATLAB 绘图命令以各种形式显示输出轨迹。从 Ports & Subsystems 模块库中选择 Out1 模块添加到当前模型并打开参数设置对话框进行设置, 具体操作参见 8.4.2 小节。

3. 使用 To Workspace 模块

利用 To Workspace 模块可以将输出轨迹返回到 MATLAB 工作区间, 下面通过一个实例来说明。

【例】 使用 To Workspace 模块观察输出。

创建一个模型文件, 系统连接框图如图 8-38 所示。

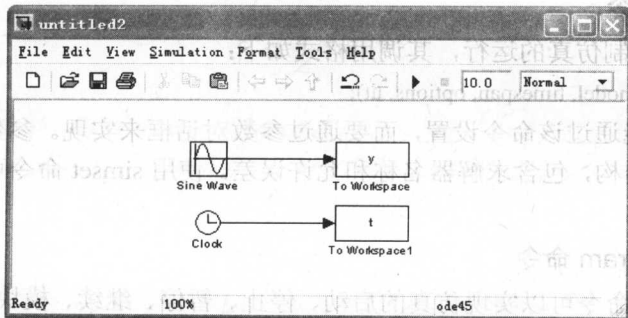


图 8-38 系统模块连接

打开 To Workspace 模块的参数设置对话框，按图 8-39 所示设置参数。

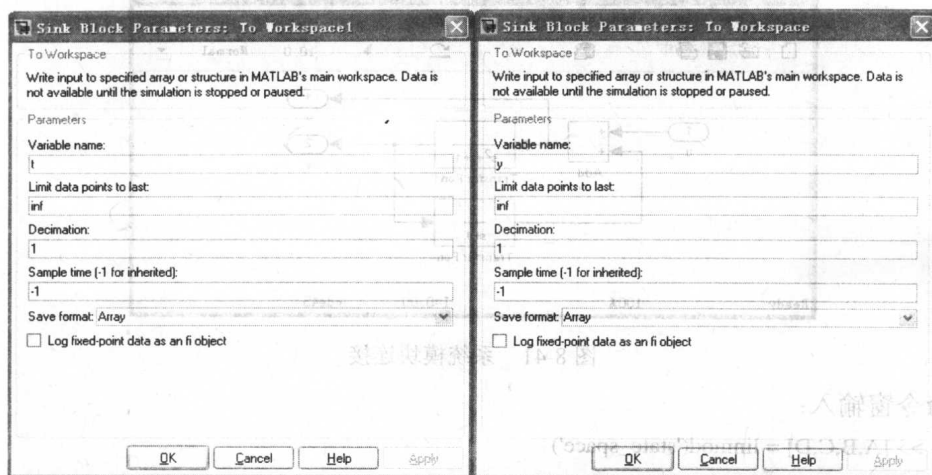


图 8-39 To Workspace 模块参数设置对话框

运行仿真，MATLAB 工作区间中则会增加 t 和 y 两个变量，在命令窗输入：

```
>> plot(t,y)
```

运行结果如图 8-40 所示。

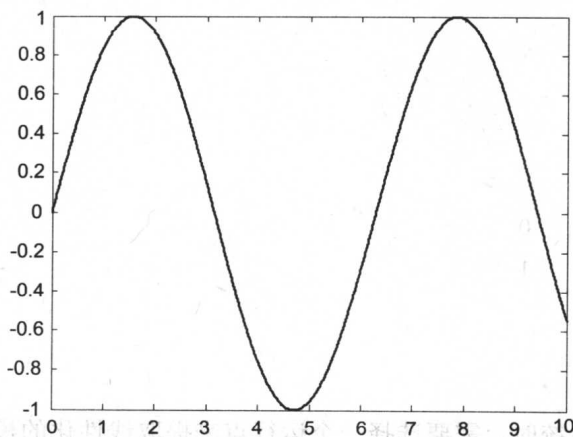


图 8-40 利用输出数据绘图

8.4.7 线性化模型

Simulink 提供了 linmod、linmod2 以及 dlinmod 等函数，用于提取以状态空间矩阵 A、B、C、D 表示的线性系统的模型。状态空间矩阵描述系统输入输出关系如下：

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

x、u、y 分别为状态向量、输入向量、输出向量。

【例】 使用 linmod 命令获取线性模型。

创建一个名为 state_space.mdl 的模型文件，系统连接框图如图 8-41 所示。

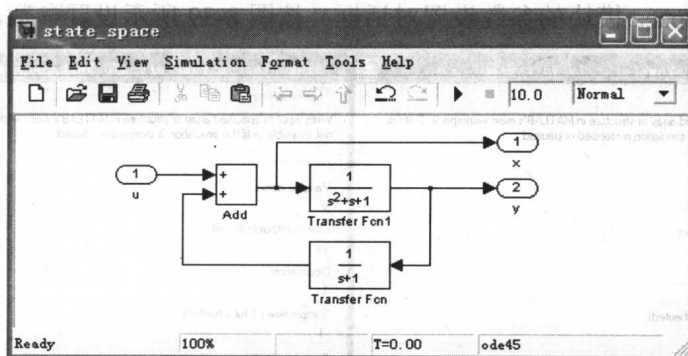


图 8-41 系统模块连接

在命令窗输入:

```
>> [A,B,C,D] = linmod('state_space')
```

运行结果:

A =

```
-1 0 1
1 -1 -1
0 1 0
```

B =

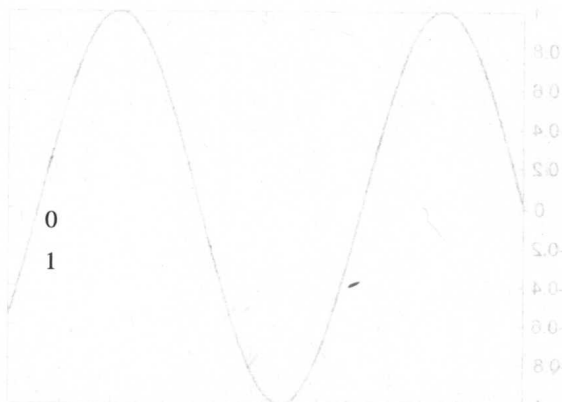
```
0
1
0
```

C =

```
1 0 0
0 0 1
```

D =

```
1
0
```



当模型为非线性系统时, 需要选择一个运行点来提取线性化的模型, 即在 `linmod` 函数的输入参数中加入运行点:

```
[A,B,C,D] = linmod('sys', x, u)
```

对于离散系统或连续离散混合型系统, 可以使用 `dlinmod` 函数实现线性化。`dlinmod` 函数的调用格式与 `linmod` 函数的调用格式类似, 不同的是第二个输入参数必须包含采样时间。

8.4.8 寻找稳态工作点

利用 Simulink 的 `trim` 函数可以在指定输入、输出和状态条件的前提下寻找动力学系统的稳态工作点。本小节将通过一个实例来说明稳态工作点的寻找方法。

【例】 寻找稳态工作点。

创建一个名为 `steady_state.mdl` 的模型文件, 系统连接框图如图 8-42 所示。

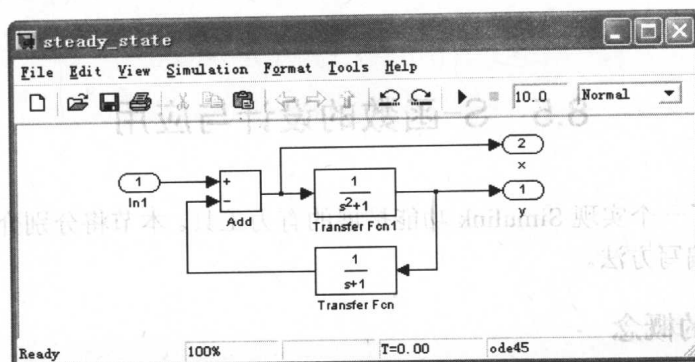


图 8-42 系统模块连接

使用 `trim` 函数寻找使输出均为 1 的输入和状态。首先对状态变量 x 和输入量 u 进行猜测，然后设置输出量 y 的值：

```
>> x = [0; 0; 0];
```

```
>> u = 0;
```

```
>> y = [1; 1];
```

使用索引变量表示哪些变量(iy)需要固定，哪些变量(ix 、 iu)不需要固定：

```
ix = [];
```

```
iu = [];
```

```
iy = [1; 2];
```

调用 `trim` 函数求解，在命令窗输入：

```
>> [x,u,y,dx] = trim('steady_state',x,u,y,ix,iu,iy)
```

运行结果：

```
x =
```

```
0.0000
```

```
1.0000
```

```
1.0000
```

```
u =
```

```
2.0000
```

```
y =
```

```
1.0000
```

```
1.0000
```

```
dx =
```

```
1.0e-015 *
```

```
0.3331
```

```
0.1586
```

```
-0.3331
```

需要注意的是，有些问题可能不存在稳态工作点，如果出现这种情况，`trim` 函数将在初次设置导数为零之后，从需要的解中返回最小的最大偏差解。

8.5 S-函数的设计与应用

S-函数提供了一个实现 Simulink 功能扩展的有力工具。本节将分别介绍 S-函数的工作原理和 S-函数的编写方法。

8.5.1 S-函数的概念

S-函数是 Simulink 模块的计算机描述语言。S-函数可以通过 MATLAB、C、C++、Ada 或 Fortran 语言来编写。C、C++、Ada 以及 Fortran 语言编写的 S-函数将通过 mex 应用程序编译成为 MEX 文件, 就像其他 MEX 文件一样, 在需要时会动态链接到 MATLAB。

S-函数使用一种特定的调用格式与 Simulink 方程求解器相互作用。这种交互作用与求解器和内建 Simulink 模块之间的交互作用非常类似。S-函数的形式非常通用, 而且 S-函数适用于连续、离散以及混合系统。

使用 S-函数可以将用户自定义模块加入到 Simulink 模型之中。用户可以通过 MATLAB、C、C++、Fortran 或 Ada 语言来创建自定义模块。只要遵循一系列的规则, 就可以通过 S-函数实现自定义的算法。编写完一个 S-函数并将它的名称放入 S-函数模块 (User-Defined Functions 模块库中) 之后, 可以通过封装模块实现自定义用户界面。

Real-Time Workshop 可与 S-函数结合使用。此外还可以通过编写一个 Target Language Compiler (TLC) 文件自定义由 Real-Time Workshop 生成的 S-函数代码。

8.5.2 S-函数的使用

要想添加一个 S-函数模块, 首先要从 User-Defined Functions 模块库中选取 S-Function 模块并将其拖动到当前模型窗口内, 然后再在 S-Function 模块对话框的 S-function name 字段指定 S-函数文件名, 如图 8-43 所示。

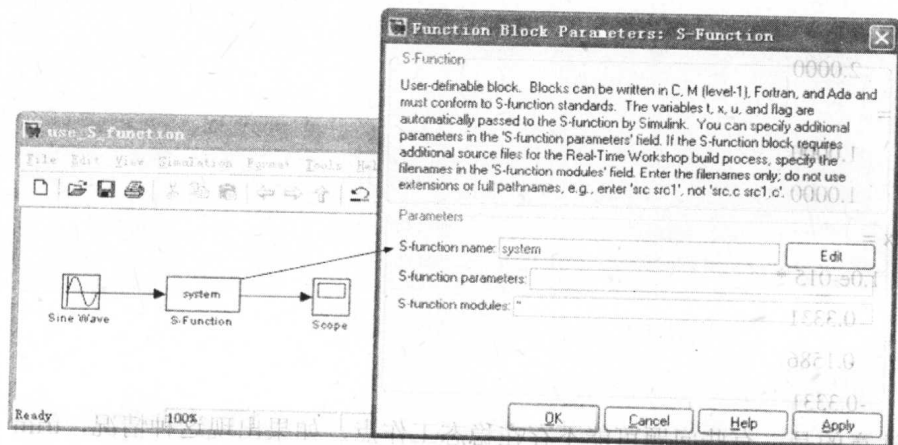


图 8-43 S-Function 模块及其参数设置对话框

S-Function 模块对话框的 S-function parameters 字段用于指定传递到相应的 S-函数的参数数值。要填写这个字段,首先要知道 S-函数中所需的参数以及这些参数在 S-函数中的排列顺序,然后再按照这个顺序填写这些参数,并用逗号隔开。参数可以是常数、MATLAB 工作区间变量或者 MATLAB 表达式。

S-函数一般用于创建自定义 Simulink 模块。S-函数还有以下多种应用场合:

- 为 Simulink 添加一个新的通用模块。
- 添加代表硬件设备驱动模块。
- 将已有的 C 代码并入仿真当中。
- 以一系列数学方程来描述系统。
- 使用图形动画。

S-函数的优点是可以帮助用户创建一个通用模块,以便在模型中多次使用,甚至不断地改变模块的参数值。

8.5.3 S-函数的工作方式

创建一个 S-函数,首先需要理解 S-函数的工作方式。而理解 S-函数的工作方式,就需要理解 Simulink 如何仿真一个模型,以及理解模块之间的数学关系。

1. 模块之间的数学关系

一个 Simulink 模块由一系列的输入、状态和输出组成,如图 8-44 所示,输出是采样时间、输入和模块状态的函数。

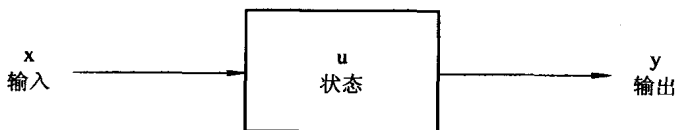


图 8-44 Simulink 模块数学关系

模块输入、状态和输出之间的数学关系如下:

$$\begin{cases} y = f_0(t, x, u) \\ \dot{x}_c = f_d(t, x, u) \\ x_{d_{k+1}} = f_u(t, x, u) \end{cases}, \quad x = x_c + x_d$$

2. 仿真进程

仿真的执行是按照阶段进行的。首先是初始化阶段, Simulink 将模块库并入模型,传递宽度、数据类型以及采样时间,估计模块参数,确定模块的执行顺序,以及分配存储空间。接下来, Simulink 进入一个仿真循环,每次循环为一个仿真步。每个仿真步期间, Simulink 按照初始化时确定的顺序执行模型的每个模块。对于每个模块, Simulink 调用计算模块当前采样时刻的状态、导数以及输出的函数,直到仿真结束。

Simulink 的仿真流程如图 8-45 所示。

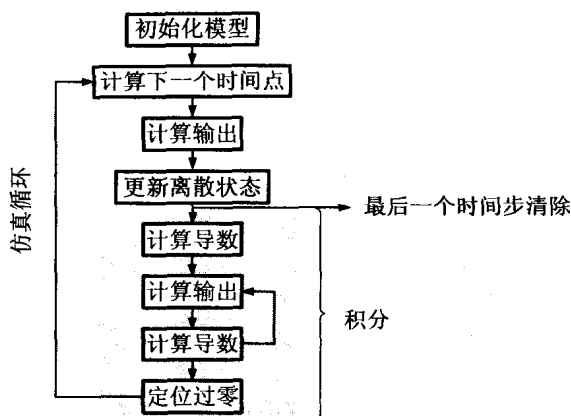


图 8-45 Simulink 仿真流程

3. S-函数回调方法

一个 S-函数由一系列在每个仿真阶段执行任务需求的 S-函数回调方法组成。仿真一个模型时，在每一个仿真阶段，Simulink 调用适当的方法处理模型中的每一个 S-函数模块。S-函数方法的执行任务包括：

- 初始化：第一次仿真循环前执行，初始化 S-函数，该阶段 Simulink 将执行以下操作：
 - 初始化 SimStruct(一个包含 S-函数信息的仿真结构)。
 - 设置输入端和输出端的数目和维数。
 - 设置模块采样时间。
 - 分配存储空间和 sizes 数组。
- 计算下一个时间点：如果创建了一个可变采样时间模块，该阶段将计算下一个采样点的时间，即计算下一个步长。
- 计算主要时间步的输出：该调用完成后，所有模块输出端在当前时间步有效。
- 更新主要时间步的离散状态：该调用中所有模块应当在每一个时间步执行一次，如在仿真循环内更新下一个时刻的离散状态。
- 积分：用于包含连续状态以及非采样过零的模型。如果 S-函数中包含连续状态，Simulink 将以次要时间步调用 S-函数的输出和导数部分。这就是 Simulink 能够计算 S-函数状态的原因。如果 S-函数(只有 C MEX)包含非采样过零，Simulink 将以次要时间步调用 S-函数的输出和过零部分，以实现过零定位。

8.5.4 编写 S-函数

S-函数可以通过多种格式的文件来实现，如 M 文件和 MEX 文件。本小节将介绍如何使用 MATLAB 语言编写 M 文件 S-函数。

1. Level-1 M 文件 S-函数

一个 Level-1 M 文件 S-函数由如下形式的 MATLAB 函数组成：

`[sys,x0,str,ts]=f(t,x,u,flag,p1,p2, ...)`

其中，f 为 S-函数名，t 为当前时间，x 为相应 S-函数模块的状态向量，u 为模块的输入，

flag 表示将要执行的任务, p1, p2, ... 为模块的参数。在模型的仿真期间, Simulink 将重复调用 f, 使用 flag 来表示特定调用中将要执行的任务。每当 S-函数执行任务时, 将返回一个结构。

在 MATLAB 根目录下的 toolbox\simulink\blocks 文件夹中有一个 M 文件 S 函数模板文件 sfuntmpl.m。该模板由一个最高级主函数和一系列称作 S-函数回调方法的框架函数构成, 每个子函数对应一个 flag 值。在仿真期间最高级主函数调用由 flag 指示的子函数, 由子函数执行 S-函数的实际任务。模板文件 sfuntmpl.m 的内容如下(为简洁起见, 删除了注释和空行):

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

```

表 8-16 列出了 flag 的可能取值及其对应于模板中的方法。

表 8-16 flag 的取值及对应方法

flag	S-函数程序	描 述
0	mdlInitializeSizes	定义 S-函数模块的基本特征, 包含采样时间、连续或离散状态的初始条件以及 sizes 数组
1	mdlDerivatives	计算连续状态变量的导数
2	mdlUpdate	更新离散状态、采样时间以及主要时间步要求
3	mdlOutputs	计算 S-函数的输出
4	mdlGetTimeOfNextVarHit	计算下一时刻的绝对时间, 仅当 mdlInitializeSizes 中指定了可变离散时间采样时可用
9	mdlTerminate	执行仿真结束

S-函数的输出向量为 sys、x0、str 和 ts, 它们的含义如下:

- sys: 通用返回变量, 返回值取决于 flag 的值。
- x0: 初始状态值, 除 flag = 0 外 x0 将被忽略。
- str: 保存值, 以便将来应用。M 文件 S-函数必须令其为空矩阵[]。
- ts: 包含模块采样时间和偏移的二列矩阵。如果希望 S-函数每个时间步运行一次, 设为[0 0]; 如果希望 S-函数继承其他模块的采样时间, 设为[-1 0]; 如果希望 S-函数每 0.2 秒运行一次并且在仿真启动后 0.1 秒开始运行, 设为[0.2 0.1]; 如果希望 S-函数以[0 0.1 0.25 0.5 0.75 1 1.1 ...]时间序列运行, 设为[.25 0; 1.0 .1]。

要让 Simulink 认出一个 M 文件 S-函数, 必须提供 S-函数的指定信息, 包括输入数、输出数、状态数以及模块的其他特性。

如果要给 Simulink 提供这些信息, 则需在 mdlInitializeSizes 的开头调用 simsizes 函数:

```
sizes = simsizes;
```

该函数返回一个未初始化的 sizes 结构, 其各字段描述如表 8-17 所示。

表 8-17 sizes 结构各字段描述

字 段 名	描 述
sizes.NumContStates	连续状态数
sizes.NumDiscStates	离散状态数
sizes.NumOutputs	输出数
sizes.NumInputs	输入数
sizes.DirFeedthrough	直接馈入标志
sizes.NumSampleTimes	采样时间数

首先对结构的每个字段进行初始化,然后再调用 `simsizes` 函数:

```
sys = simsizes(sizes);
```

该命令将 `sizes` 结构的信息传递给向量 `sys`。

将 Level-1 S-函数各 `flag` 代码映射到 Level-2 S-函数的对应方法,就可以将 Level-1 M 文件 S-函数转化为 Level-2 M 文件 S-函数,可参照 `flag` 参数表(见表 8-16)。另外,还需要执行如下操作:

- 在 `Dwork` 向量中存储 Level-2 S-函数的状态信息,在 `PostPropagationSetup` 方法中初始化。
- 使用 `DialogPrm run-time` 对象属性访问 Level-2 S-函数的对话参数代替在调用语句中传递参数。
- 对于可变采样时间的 S-函数,在 `Outputs` 方法中更新 `NextTimeHit run-time` 对象属性,以设置 Level-2 S-函数的下一个采样时间。

2. Level-2 M 文件 S-函数

Level-2 M 文件 S-函数应用程序接口(API)允许用户使用 MATLAB 语言创建完全自定义的模块,支持包括矩阵信号和结构信号在内的任意数据类型的数据多输入多输出信号系统,这是 Level-1 M 文件 S-函数所不能实现的。在 `User-Defined Functions` 模块库中选择 Level-2 M-File S-functions 模块,就可以在 Simulink 模型中添加一个 Level-2 M 文件 S-函数模块。该文件由一系列回调方法组成,这些回调方法执行由 S-函数定义的初始化工作以及模块输出的计算。

为了促进任务的执行,Simulink 将一个 `run-time` 对象作为参数传递给回调方法。`run-time` 对象就像一个 S-函数模块的 M 代理,在仿真期间或模型更新时允许回调方法设置和访问模块的属性。

Level-2 M 文件 S-函数 API 定义了组成 Level-2 M 文件 S-函数回调方法的识别标记和通用目的。S-函数本身提供了这些回调方法的执行。执行时按顺序确定模块的属性(如端口、参数、状态)和行为(如模块输出与时间、输入、参数、状态之间的函数关系)。通过创建一个适当回调执行系列的 S-函数,可以定义一个满足特定应用需求的模块类型。

在 MATLAB 根目录下的 `toolbox\simulink\blocks` 文件夹中有一个 Level-2 M 文件 S-函数 API 模板文件 `msfuntmpl.m`。要创建一个 M 文件 S-函数,将模板中的代码复制到对应 Level-2 M 文件 S-函数模块的 M 文件中并对其进行编辑,编辑方法可以参考代码注释。模板文件 `msfuntmpl.m` 的内容如下(为简洁起见,删除了注释和空行):

```
function msfuntmpl(block)
setup(block);
function setup(block)
    block.NumInputPorts = 1;
    block.NumOutputPorts = 1;
    block.SetPreCompInpPortInfoToDynamic;
    block.SetPreCompOutPortInfoToDynamic;
    block.InputPort(1).DatatypeID = 0; % double
    block.InputPort(1).Complexity = 'Real';
```

```

block.OutputPort(1).DatatypeID = 0; % double
block.OutputPort(1).Complexity = 'Real';
block.NumDialogPrms = 3;
block.DialogPrmsTunable = {'Tunable','Nontunable','SimOnlyTunable'};
block.SampleTimes = [0 0];
block.SetAccelRunOnTLC(false);
block.RegBlockMethod('CheckParameters', @CheckPrms);
block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
block.RegBlockMethod('SetInputPortDataType', @SetInpPortDataType);
block.RegBlockMethod('SetOutputPortDataType', @SetOutPortDataType);
block.RegBlockMethod('SetInputPortComplexSignal', @SetInpPortComplexSig);
block.RegBlockMethod('SetOutputPortComplexSignal', @SetOutPortComplexSig);
block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('ProcessParameters', @ProcessPrms);
block.RegBlockMethod('InitializeConditions', @InitializeConditions);
block.RegBlockMethod('Start', @Start);
block.RegBlockMethod('Outputs', @Outputs);
block.RegBlockMethod('Update', @Update);
block.RegBlockMethod('Derivatives', @Derivatives);
block.RegBlockMethod('Projection', @Projection);
block.RegBlockMethod('SimStatusChange', @SimStatusChange);
block.RegBlockMethod('Terminate', @Terminate);
block.RegBlockMethod('WriteRTW', @WriteRTW);
function CheckPrms(block)
    a = block.DialogPrm(1).Data;
    if ~strcmp(class(a), 'double')
        error('Invalid parameter');
    end
function ProcessPrms(block)
    block.AutoUpdateRuntimePrms;
function SetInpPortFrameData(block, idx, fd)
    block.InputPort(idx).SamplingMode = fd;
    block.OutputPort(1).SamplingMode = fd;
function SetInpPortDims(block, idx, di)
    block.InputPort(idx).Dimensions = di;
    block.OutputPort(1).Dimensions = di;
function SetOutPortDims(block, idx, di)
    block.OutputPort(idx).Dimensions = di;

```

```

    block.InputPort(1).Dimensions = di;
function SetInpPortDataType(block, idx, dt)
    block.InputPort(idx).DataTypeID = dt;
    block.OutputPort(1).DataTypeID = dt;
function SetOutPortDataType(block, idx, dt)
    block.OutputPort(idx).DataTypeID = dt;
    block.InputPort(1).DataTypeID = dt;
function SetInpPortComplexSig(block, idx, c)
    block.InputPort(idx).Complexity = c;
    block.OutputPort(1).Complexity = c;
function SetOutPortComplexSig(block, idx, c)
    block.OutputPort(idx).Complexity = c;
    block.InputPort(1).Complexity = c;
function DoPostPropSetup(block)
    block.NumDworks = 1;
    block.Dwork(1).Name = 'x1';
    block.Dwork(1).Dimensions = 1;
    block.Dwork(1).DatatypeID = 0; % double
    block.Dwork(1).Complexity = 'Real'; % real
    block.Dwork(1).UsedAsDiscState = true;
    block.AutoRegRuntimePrms;
function InitializeConditions(block)
function Start(block)
    block.Dwork(1).Data = 0;
function WriteRTW(block)
    block.WriteRTWParam('matrix', 'M', [1 2; 3 4]);
    block.WriteRTWParam('string', 'Mode', 'Auto');
function Outputs(block)
    block.OutputPort(1).Data = block.Dwork(1).Data + block.InputPort(1).Data;
function Update(block)
    block.Dwork(1).Data = block.InputPort(1).Data;
function Derivatives(block)
function Projection(block)
function SimStatusChange(block, s)
    if s == 0
        disp('Pause has been called');
    elseif s == 1
        disp('Continue has been called');
    end
function Terminate(block)

```

在 User-Defined Functions 模块库中选择 Level-2 M-File S-functions 模块, 添加到当前模型中。打开该模块的参数设置对话框, 在 M-file name 字段中输入已经编写好的 Level-2 M 文件 S-函数名, 就可以在模型内创建一个 S-函数实例。

要由一个包含 Level-2 M 文件 S-函数的模型生成代码, 需要提供一个相应的 TLC 文件。而模型在加速模式运行时不需要 TLC 文件, 这是因为 Simulink 加速器以解释方式运行 Level-2 M 文件 S-函数。

Level-2 M 文件 S-函数 API 指定了 M 文件 S-函数必须执行和可以选择忽略的一系列回调方法。Level-2 M 文件 S-函数 API 定义的方法通常对应由 C MEX 文件 S-函数 API 定义的方法, 并且二者名称相似。表 8-18 给出了 Level-2 M 文件 S-函数回调方法及其对应的 C MEX 文件 S-函数回调方法。

表 8-18 Level-2 M 文件 S-函数回调方法及其对应的
C MEX 文件 S-函数回调方法

Level-2 M 文件方法	C MEX 文件方法
setup method	mdlInitializeSizes
CheckParameters	mdlCheckParameters
Derivatives	mdlDerivatives
Disable	mdlDisable
Enable	mdlEnable
InitializeCondition	mdlInitializeConditions
Outputs	mdlOutputs
PostPropagationSetup	mdlSetWorkWidths
ProcessParameters	mdlProcessParameters
Projection	mdlProjection
SetInputPortComplexSignal	mdlSetInputPortComplexSignal
SetInputPortDataType	mdlSetInputPortDataType
SetInputPortDimensions	mdlSetInputPortDimensionInfo
SetInputPortSampleTime	mdlSetInputPortSampleTime
SetInputPortSamplingMode	mdlSetInputPortFrameData
SetOutputPortComplexSignal	mdlSetOutputPortComplexSignal
SetOutputPortDataType	mdlSetOutputPortDataType
SetOutputPortDimensions	mdlSetOutputPortDimensionInfo
SetOutputPortSampleTime	mdlSetOutputPortSampleTime
SimStatusChange	mdlSimStatusChange
Start	mdlStart
Terminate	mdlTerminate
Update	mdlUpdate
WriteRTW	mdlRTW
ZeroCrossings	mdlZeroCrossings

Level-2 M 文件 S-函数的 `setup` 方法的作用是初始化模型中相应 Level-2 M-File S-Function 模块的实例。在这个方面，主函数类似于 C MEX S-函数中的 `mdlInitializeSizes` 回调方法。主函数执行的 `setup` 任务包括：

- 设置模块的输入端个数和输出端个数。
- 设置端口属性，如维数、数据类型、复杂性和采样时间。
- 设置参数数目并检查这些参数的有效性。
- 使用 M 文件中其他局部函数句柄寄存可变模块方法，用 S-函数模块的 run-time 对象 `RegBlockMethod` 方法来传递。

在 MATLAB 根目录下的 `toolbox\simulink\blocks` 文件夹中的 `msfcn_unit_delay.m` 文件的 `setup` 方法，对有一个输入端、一个输出端和一个对话参数的 S-函数进行了初始化。它通过使用 `SetPreCompInPortInfoToDynamic` 和 `SetPreCompOutPortInfoToDynamic` 的 run-time 对象方法以及直接参数设置对编译属性进行了初始化，该方法记录了四个 S-函数方法：

```
function setup(block)

    block.NumDialogPrms = 1;

    %% Register number of input and output ports
    block.NumInputPorts = 1;
    block.NumOutputPorts = 1;

    %% Setup functional port properties to dynamically
    %% inherited.
    block.SetPreCompInPortInfoToDynamic;
    block.SetPreCompOutPortInfoToDynamic;

    block.InputPort(1).Dimensions = 1;
    block.InputPort(1).DirectFeedthrough = false;

    block.OutputPort(1).Dimensions = 1;

    %% Set block sample time to inherited
    block.SampleTimes = [0.1 0];

    %% Register methods
    block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
    block.RegBlockMethod('InitializeConditions', @InitConditions);
    block.RegBlockMethod('Outputs', @Output);
    block.RegBlockMethod('Update', @Update);

%endfunction
```

当 Simulink 调用一个 Level-2 M 文件 S-函数的回调方法时, Simulink 将传递一个 Simulink.MSFcnRunTimeBlock 类的实例到该方法并作为该方法的参量。这个实例就是 S-函数模块的 run-time 对象, 它将服务于 Level-2 M 文件 S-函数回调方法, 就像 SimStruct 结构服务于 C MEX 文件 S-函数回调方法一样。这使得正被调用的方法能够提供和获取模块的大量元素信息, 如端口、参数、状态和工作向量, 这是通过获取或设置模块 run-time 对象的属性或调用模块 run-time 对象方法来完成的。

8.5.5 应用实例

本小节将通过一个实例说明编写 S-函数实现系统仿真的方法。本小节所要仿真的系统是一个实用电源模块, 该系统的等效电路图如图 8-46 所示。

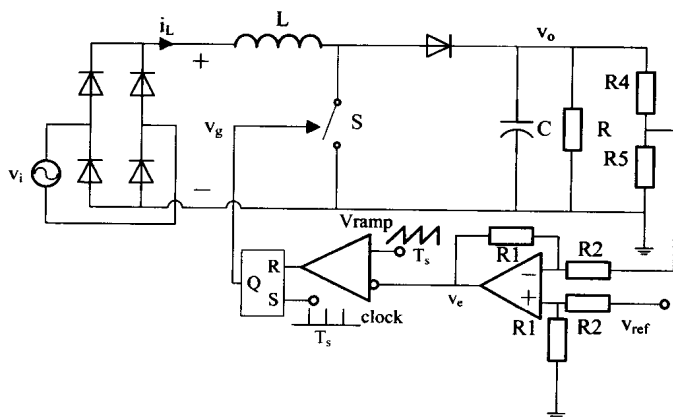


图 8-46 系统电路图

该电路由主电路和控制电路构成, 图 8-46 的上半部分为主电路, 下半部分为控制电路。由于该主电路中存在开关和二极管, 所以根据开关和二极管的通断情况, 电路有三种工作模式, 如图 8-47 所示。

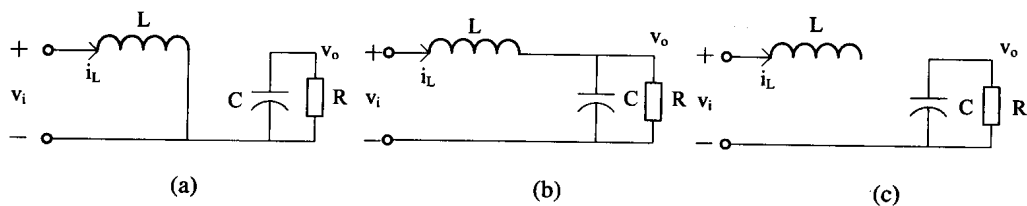


图 8-47 电路的三种工作模式

设 i_L 、 v_o 为状态变量, 对应电路三种工作模式下的状态方程如下:

$$\begin{cases} \dot{i}_L = \frac{v_i}{L} \\ \dot{v}_o = -\frac{v_o}{RC} \end{cases} \quad \begin{cases} \dot{i}_L = \frac{v_i}{L} - \frac{v_o}{L} \\ \dot{v}_o = \frac{i_L}{C} - \frac{v_o}{RC} \end{cases} \quad \begin{cases} \dot{i}_L = 0 \\ \dot{v}_o = -\frac{v_o}{RC} \end{cases}$$

由于电路存在多种工作模式, 而使用模块库中的 Switch 模块来搭建系统较为困难, 甚至容易出错。但是可以根据开关和状态来判断电路工作在哪一个模式, 因此, 以开关为输

入变量, 以 i_L 、 v_o 为状态变量, 以 i_L 、 v_o 为输出变量, 以 v_i 、 L 、 R 、 C 、 f 作为参数, 可以编写主电路的 S-函数并保存为 boostpfc.m。该 S-函数的代码如下(考虑到寄生电阻, 因此含有 r_l 和 r_c 两项参数, 不需要时可以将其设为零; 考虑到电感电流初值和电容电压初值, 因此含有 iL 和 vC 两项参数, 不需要时亦可将其设为零):

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag,Vi,L,C,R,iL,vC,rL,rC,f)
```

```
switch flag,
```

```
case 0,
```

```
    [sys,x0,str,ts]=mdlInitializeSizes(iL,vC);
```

```
case 1,
```

```
    sys=mdlDerivatives(t,x,u,Vi,L,C,R,rL,rC,f);
```

```
case 2,
```

```
    sys=mdlUpdate(t,x,u);
```

```
case 3,
```

```
    sys=mdlOutputs(t,x,u);
```

```
case 9,
```

```
    sys=mdlTerminate(t,x,u);
```

```
otherwise
```

```
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end
```

```
function [sys,x0,str,ts]=mdlInitializeSizes(iL,vC)
```

```
sizes = simsizes;
```

```
sizes.NumContStates = 2;
```

```
sizes.NumDiscStates = 0;
```

```
sizes.NumOutputs = 2;
```

```
sizes.NumInputs = 1;
```

```
sizes.DirFeedthrough = 0;
```

```
sizes.NumSampleTimes = 1; % at least one sample time is needed
```

```
sys = simsizes(sizes);
```

```
x0 = [iL,vC];
```

```
str = [];
```

```
ts = [0 0];
```

```
function sys=mdlDerivatives(t,x,u,Vi,L,C,R,rl,rc,f)
```

```
dx=[0,0];
```

```
%% x(1)=iL    x(2)=vC
```

```
if u==1
```

```
dx(1)=Vi*abs(sin(2*pi*f*t))/L-x(1)*r/L;
```

```
dx(2)=-x(2)/(R+rc)/C;
```

```
end
```

```
if u==0 && x(1)>0
```

```
dx(1)=(Vi*abs(sin(2*pi*f*t))-x(2)*R/(R+rc)-x(1)*(rl+R*rc/(R+rc)))/L;
```

```
dx(2)=(x(1)*R-x(2))/C/(R+rc);
```

```
end
```

```
if u==0 && x(1)<1e-6
```

```
dx(1)=0;
```

```
dx(2)=-x(2)/(R+rc)/C;
```

```
end
```

```
sys = dx;
```

```
function sys=mdlUpdate(t,x,u)
```

```
sys = [];
```

```
function sys=mdlOutputs(t,x,u)
```

```
sys = [x];
```

```
function sys=mdlTerminate(t,x,u)
```

```
sys = [];
```

创建一个模型文件，系统模块的连接如图 8-48 所示。其中 S-Function 模块和 Repeating Sequence 模块的参数设置如图 8-49 所示。

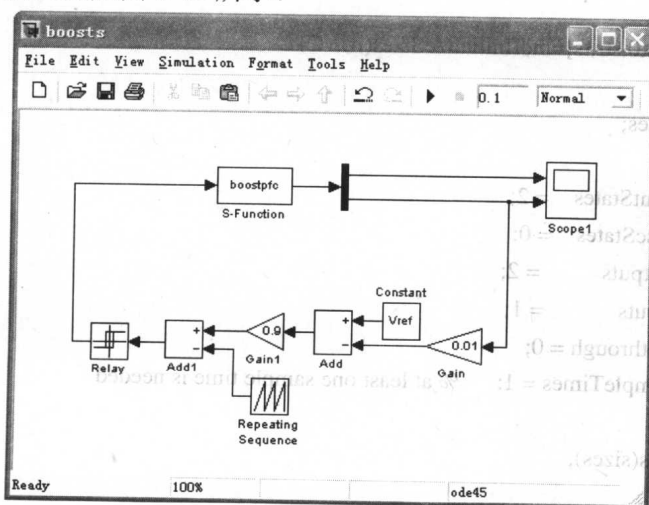


图 8-48 系统模块的连接

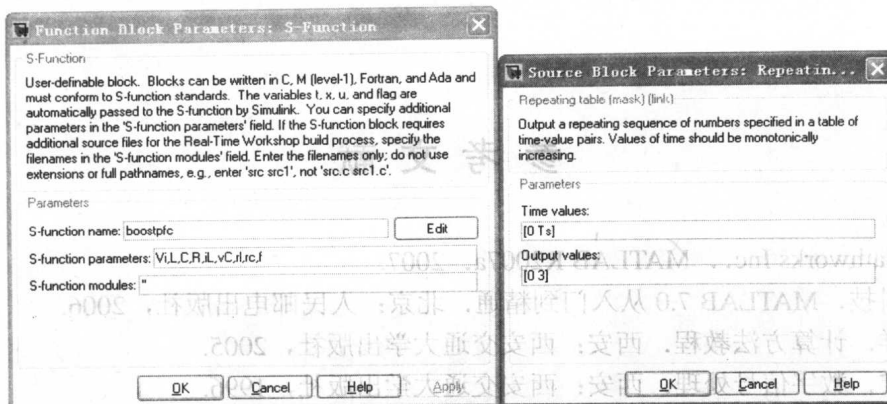


图 8-49 S-Function 模块和 Repeating Sequence 模块的参数设置

系统搭建完毕后，可以通过命令方式对参数变量赋值，在命令窗输入：

```
>> Vi=156;
>> Vref=2.8;
>> R=300;
>> L=130e-6;
>> C=470e-6;
>> f=50;
>> Ts=20e-6;
>> iL=0;
>> vC=150;
>> rl=0;
>> rc=0;
```

赋值完毕后运行仿真，将最大步长设为 $T_s/10$ ，终止时间设为 0.05，仿真得到的电压电流波形如图 8-50 所示。

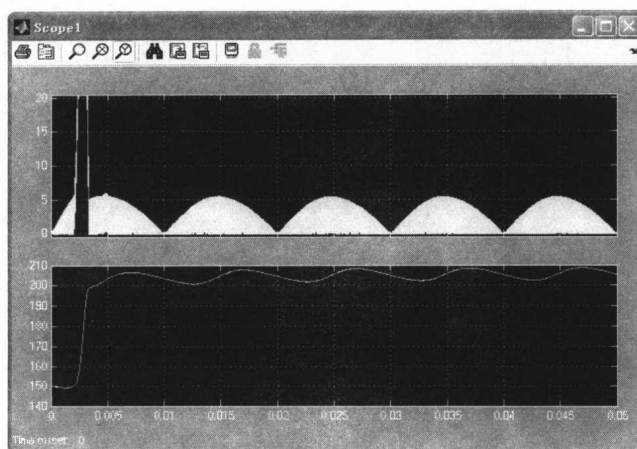


图 8-50 仿真得到的电压电流波形

可以看出，电感电流是规则的整流全波，电容电压是有纹波的直流流量。

参 考 文 献

- [1] The Mathworks Inc.. MATLAB R2007a. 2007.
- [2] 求是科技. MATLAB 7.0 从入门到精通. 北京: 人民邮电出版社, 2006.
- [3] 凌永祥. 计算方法教程. 西安: 西安交通大学出版社, 2005.
- [4] 郑南宁. 数字信号处理. 西安: 西安交通大学出版社, 1996.
- [5] 刘慧颖. MATLAB R2006a 基础教程. 北京: 清华大学出版社, 2005.
- [6] 蒲俊, 吉家锋, 伊良忠. MATLAB6.0 数学手册. 上海: 浦东电子出版社, 2002.
- [7] 陈怀琛. MATLAB 及其在理工课程中的应用指南. 西安: 西安电子科技大学出版社, 1999.
- [8] 洪乃刚. 电力电子和电力拖动控制系统的 MATLAB 仿真. 北京: 机械工业出版社, 2005.

封面

书名

版权

前言

目录

第1章 MATLAB概述

1.1 MATLAB R2007a简介

1.1.1 MATLAB的新版本特性

1.1.2 MATLAB的新产品概况

1.2 桌面工具与开发环境

1.2.1 主菜单

1.2.2 工具栏

1.2.3 当前路径

1.2.4 工作区间

1.2.5 命令窗

1.2.6 历史命令记录

1.2.7 Strat菜单

1.3 编辑 / 调试器

1.3.1 M文件的创建

1.3.2 M文件的运行和调试

1.3.3 M文件的结果发布

1.4 帮助系统

1.4.1 命令窗查询帮助

1.4.2 帮助浏览器

第2章 矩阵与数组

2.1 创建矩阵

2.1.1 创建矩阵和数值序列

2.1.2 创建特殊矩阵

2.1.3 合并矩阵

2.2 索引

2.2.1 线性索引

2.2.2 访问单个元素

2.2.3 访问多个元素

2.3 获取矩阵信息

2.3.1 矩阵的阶数与维数

2.3.2 矩阵元素的数据类型

2.3.3 矩阵的数据结构

2.4 基本操作和运算

2.4.1 矩阵的扩大和缩小

2.4.2 改变矩阵的形状

2.4.3 矩阵的算术运算

2.4.4 矩阵的关系运算和逻辑运算

2.5 空矩阵、标量和向量

- 2.5.1 空矩阵
- 2.5.2 标量
- 2.5.3 向量
- 2.6 多维数组
 - 2.6.1 多维数组的创建
 - 2.6.2 多维数组的索引
 - 2.6.3 改变多维数组的形状
 - 2.6.4 多维数组的运算
- 第3章 数据类型
 - 3.1 数值类型
 - 3.1.1 整数
 - 3.1.2 浮点数
 - 3.1.3 复数
 - 3.1.4 无穷与非数
 - 3.1.5 判断数据类型
 - 3.1.6 数据显示形式
 - 3.2 逻辑类型
 - 3.2.1 创建逻辑数组
 - 3.2.2 逻辑数组的用途
 - 3.2.3 判断逻辑类型
 - 3.3 字符串
 - 3.3.1 创建字符数组
 - 3.3.2 字符串单元数组
 - 3.3.3 字符串的操作
 - 3.3.4 字符串类型与数值类型之间的转化
 - 3.4 日期与时间
 - 3.4.1 日期的表现形式
 - 3.4.2 日期表现形式之间的转化
 - 3.4.3 当前日期与时间
 - 3.5 结构
 - 3.5.1 创建结构数组
 - 3.5.2 结构数组的操作
 - 3.6 单元数组
 - 3.6.1 创建单元数组
 - 3.6.2 单元数组的操作
 - 3.7 函数句柄
 - 3.7.1 创建和调用函数句柄
 - 3.7.2 利用句柄调用函数
 - 3.8 MATLAB类
 - 3.9 Java类
- 第4章 数学运算基础
 - 4.1 矩阵与线性代数
 - 4.1.1 矩阵分析

- 4.1.2 求解线性方程组
- 4.1.3 逆矩阵与伪逆矩阵
- 4.1.4 矩阵的分解
- 4.1.5 矩阵的非线性运算
- 4.1.6 特征值与特征向量
- 4.1.7 奇异值分解
- 4.2 多项式与插值
 - 4.2.1 多项式
 - 4.2.2 插值
- 4.3 快速傅里叶变换
 - 4.3.1 快速傅里叶变换的概念
 - 4.3.2 快速傅里叶变换的应用
- 4.4 函数的函数
 - 4.4.1 函数的表示方法
 - 4.4.2 函数的最小值与零点
 - 4.4.3 数值积分
 - 4.4.4 嵌套函数与匿名函数
- 4.5 求解微分方程
 - 4.5.1 常微分方程初值问题
 - 4.5.2 延迟微分方程初值问题
 - 4.5.3 常微分方程边值问题
 - 4.5.4 求解偏微分方程
- 4.6 稀疏矩阵
 - 4.6.1 创建稀疏矩阵
 - 4.6.2 稀疏矩阵的查看
 - 4.6.3 稀疏矩阵的操作

第5章 M文件程序设计基础

- 5.1 M文件介绍
 - 5.1.1 脚本和函数
 - 5.1.2 P代码文件
 - 5.1.3 变量类型
 - 5.1.4 关键字和特殊值
 - 5.1.5 符号参考
- 5.2 程序流程控制
 - 5.2.1 条件控制语句
 - 5.2.2 循环控制语句
 - 5.2.3 错误控制语句
 - 5.2.4 程序终止语句
- 5.3 数据输入 / 输出
 - 5.3.1 打开文件
 - 5.3.2 读写操作
 - 5.3.3 关闭文件
 - 5.3.4 更多文件I/O函数

5.4 程序调试与优化

5.4.1 程序的调试

5.4.2 程序的优化

第6章 符号计算功能

6.1 符号对象的创建与使用

6.1.1 创建符号变量和表达式

6.1.2 创建符号数学函数

6.2 数学计算功能

6.2.1 符号微积分

6.2.2 函数的极限

6.2.3 级数求和

6.2.4 泰勒级数展开

6.3 表达式的化简和替换

6.3.1 符号表达式的化简

6.3.2 符号表达式的替换

6.4 线性代数

6.4.1 基本代数运算

6.4.2 线性代数运算

6.4.3 特征值

6.4.4 约当标准型

6.4.5 奇异值分解

6.4.6 特征值轨迹

6.5 求解符号方程

6.5.1 求解代数方程

6.5.2 求解代数方程组

6.5.3 求解常微分方程

6.6 简易符号绘图函数

6.6.1 二维基本绘图

6.6.2 二维极坐标绘图

6.6.3 三维曲线绘图

6.6.4 三维网格绘图

6.6.5 三维表面绘图

6.6.6 等高线绘图

6.7 调用Maple函数

6.7.1 maple函数

6.7.2 mfun函数

6.7.3 sym函数

6.8 积分变换

6.8.1 傅里叶变换

6.8.2 拉普拉斯变换

6.8.3 Z变换

第7章 基本绘图功能

7.1 图形窗口

7.1.1 图形窗口的创建与设置

7.1.2 图形窗口的工具栏

7.1.3 图形窗口的主菜单

7.2 绘制二维图形

7.2.1 基本绘图函数

7.2.2 图形处理函数

7.3 绘制三维图形

7.3.1 三维曲线图

7.3.2 三维网格图

7.3.3 三维曲面图

7.4 绘制特殊图形

7.4.1 条形图与区域图

7.4.2 饼形图

7.4.3 直方图

7.4.4 离散数据图

7.4.5 方向和速度向量图

7.4.6 等高线图

第8章 Simulink仿真环境

8.1 Simulink基础

8.1.1 Simulink的启动

8.1.2 Simulink工作环境

8.1.3 Simulink模块的基本操作

8.1.4 Simulink仿真步骤

8.1.5 Simulink求解算法

8.2 Simulink的模块库

8.2.1 Commonly Used Blocks模块库

8.2.2 Continuous模块库

8.2.3 Discontinuities模块库

8.2.4 Discrete模块库

8.2.5 Logic and Bit Operations模块库

8.2.6 Lookup Tables模块库

8.2.7 Math Operations模块库

8.2.8 Model Verification模块库

8.2.9 Model-Wide Utilities模块库

8.2.10 Ports & Subsystems模块库

8.2.11 Signal Attributes模块库

8.2.12 Signal Routing模块库

8.2.13 Sinks模块库

8.2.14 Sources模块库

8.2.15 User-Defined Functions模块库

8.3 子系统及封装技术

8.3.1 创建子系统

8.3.2 封装子系统

8.3.3 自定义模块库

8.4 仿真运行与分析

8.4.1 仿真的运行控制

8.4.2 仿真数据的输入和输出

8.4.3 错误诊断

8.4.4 改善仿真性能和精度

8.4.5 使用命令运行仿真

8.4.6 观察输出轨迹

8.4.7 线性化模型

8.4.8 寻找稳态工作点

8.5 S-函数的设计与应用

8.5.1 S-函数的概念

8.5.2 S-函数的使用

8.5.3 S-函数的工作方式

8.5.4 编写S-函数

8.5.5 应用实例

参考文献