Broadview®
www.broadview.com.cn

MATLAB
精品丛书

★ 算法理论、实现和应用紧密结合

★ 用实例说话

精通MATLAB科学计算

王正林 龚纯 何倩 编著

★ 系统、全面的科学计算内容

★ 丰富、详尽的MATLAB源程序

★ 全面、实用的随书附录



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

MATLAB
精品丛书

精通MATLAB科学计算

王正林 龚纯 何倩 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

前　言

科学计算是当今科学研究的三大基本手段之一，在发达国家甚至被作为衡量国家综合实力的一个重要方面。近年来，随着科技发展的加速，科学计算正日益得到关注，发展越来越快。

作为具有科学计算、符号运算和图形处理等多种功能的强有力实现工具，近年来MATLAB 科学计算软件包得到了业界的广泛认可，并且其应用领域已经拓展到了各个行业的很多学科，在各大公司、科研机构和高校里日益普及，得到广泛应用，其自身也因此得到了迅速发展，功能不断扩充，现已发展至 MATLAB R2007a。

MATLAB 作为一个功能强大的科学计算平台，几乎满足所有的计算需求，它已成为最为普遍的科学计算工具之一。

本书目的

目前，在欧美各国 MATLAB 的使用十分普及。在大学的数学、工程和科学系科，MATLAB 被用做许多课程的辅助教学手段，MATLAB 也成为大学生们必不可少的计算工具，甚至是一项必须掌握的基本技能；在科研机构和工业界，MATLAB 是高质量新产品研究、开发和分析的主要工具之一。

在我国，MATLAB 在各大专院校的应用日益普遍，许多专业已把 MATLAB 作为基本计算工具。在科研机构和工业界，MATLAB 正得到越来越广泛的应用。

为了更好地推动 MATLAB 科学计算在高校、各个学科和行业中的应用，使 MATLAB 真正成为不同专业的学生及科研、工程技术人员所普遍认可和广泛使用的科学计算工具，在充分借鉴以往类似书籍经验并尽力弥补其中不足的基础上，我们结合最新版本的 MATLAB 编撰了此书。

本书从实用角度出发，通过大量典型的应用实例，讲述如何使用 MATLAB 进行科学计算，重点、详细地讲述 MATLAB 在线性方程求解、矩阵运算与特征值计算、插值与曲线拟合、数值微分、数值积分、常微分方程求解、复数运算、概率统计、非线性方程求解、最优化处理，以及偏微分方程求解等领域中的应用。

作为一本使读者快速上手和熟练掌握 MATLAB 的书籍，全书紧密围绕精心设计的经典应用实例展开，而且这些实例通过了实际调试，以随书光盘的形式方便地提供给读者。

本书特色

本书主要讲述 MATLAB 在科学计算中的应用，具有以下特点。

1. 内容系统、全面

本书全面、详尽地讲述了 MATLAB 的科学计算功能，重点讲述了 MATLAB 在矩阵运算、线性/非线性方程、常微分/偏微分求解、数值微分、积分、插值与曲线拟合、复数运算、概率统计运算和最优化运算等领域的应用，内容广泛，覆盖了科学计算的主要方面。

2. 实例丰富、实用性强

本书结合了高校教学中不同层次的科学计算课程，精心挑选了大量的算法实例，通过实例帮助读者理解、领会和掌握算法，而且许多实例来自各大学教材及参考资料，适合各层次的读者。

3. 算法原理与 MATLAB 实现结合，侧重算法实现

本书避开了深奥的算法理论与复杂的算法细节，重点讲述算法的思想、原理，以及 MATLAB 的编程实现和实际应用，在编程过程中深化对算法思想和理论的理解，强化对算法原理的掌握。

4. 算法源代码丰富，编程参考价值高

本书作者精心编写和调试了大量科学计算算法的 MATLAB 源代码，通过学习这些程序，读者不仅可以更快、更透彻地理解和领会这些算法，而且能掌握 MATLAB 的使用，培养和提高实际计算的能力和技巧。

5. 附录内容丰富，查询方便

本书不仅以附录的形式提供了 MATLAB 程序设计入门，主要函数命令、算法的索引和注释，而且还以随书光盘的形式提供了算法源程序，便于读者查询。

本书导读

本书讲述的重点是 MATLAB 在科学计算中的应用，同时将 MATLAB 的使用方法和编程技巧渗透于其中。全书正文部分共分 12 章和 3 个附录，内容概要如下。

- 第 1 章“绪论”，介绍 MATLAB 的发展概况以及使用 MATLAB 进行科学计算的优势，讲述 MATLAB 的基本操作以及编程技巧。

  全面了解 MATLAB 进行科学计算的特点，熟悉 MATLAB 的基本操作，掌握 MATLAB 的编程技巧，是熟练运用 MATLAB 进行科学计算的基础。

- 第 2 章“线性方程组求解”，讲述用 MATLAB 实现线性方程组的常用解法，包括求逆法、分解法、迭代法、特殊解法等，以及非齐次线性方程组的解法。

  在自然科学和工程技术中，很多问题的解决可以归结为线性方程组求解，线性方程组求解是应用广泛的科学计算。利用 MATLAB，可以调用其自带的函数或通过编程来求解线性方程组。

- 第 3 章“数据插值与拟合”，讲述在 MATLAB 中进行数据插值的常用算法，包括拉格朗日插值法、艾特肯插值法、牛顿插值法、等距节点插值、高斯插值、埃尔米特插值和有理分式插值，以及进行数据拟合的常用算法，包括切比雪夫逼近、勒让德逼近、帕德逼近、傅里叶逼近、多项式曲线拟合和最小二乘法拟合等。

  数据插值与拟合是工程实践和科学实验中两种常用的科学计算方法。在 MATLAB 中，既可以利用其提供的插值和拟合函数，又可以利用编程来实现数据的插值和拟合。

- 第 4 章“矩阵特征值计算”，讲述 MATLAB 中进行矩阵特征值运算的常用解法，包括特征多项式法、幂法、收缩法、逆幂法和 QR 方法等。

  物理、力学和工程技术中很多常见的问题在数学上都归结为矩阵特征值问题。在 MATLAB 中，既可以利用其提供的函数，又可以利用编程来实现矩阵特征值的运算。

- 第 5 章“求导与微分计算”，讲述 MATLAB 中进行求导和微分的函数，以及编程实现常见的微分求取数值算法，包括中点、三点和五点公式法，样条函数法，辛普森数值微分法，以及理查森外推算法等。

  求导和数值微分在工程中有很多的应用，利用 MATLAB 提供的函数，可以方便地求出导数的表达式及数值，而利用 MATLAB 编程的数值算法则可以求取某一点处达到一定精度的数值导数。

- 第 6 章“积分计算”，讲述 MATLAB 中进行数值积分的函数，包括不定积分、定积分和样条函数积分的函数，以及数值积分算法，包括梯形法、辛普森法、牛顿-科茨法、高斯系列公式法、区间逐次分半法、龙贝格积分法和自适应法数值积分法等。

  数值积分是一种十分基础而且重要的运算，数值积分可用于计算解析定义的函数的积分，也可以计算以列表形式给出的函数的积分。利用 MATLAB 提供的函数或编程算法，可以方便地进行积分运算。

- 第 7 章“非线性方程求解”，讲述利用 MATLAB 中的函数以及通过编程实现的一些经典的数值方法来求解非线性方程。

  非线性方程是常见的一类方程，形式多样，利用 MATLAB 提供的函数，可以求解一些简单的非线性方程；通过 MATLAB 编程，可以解决一些较为复杂的非线性方程。

- 第 8 章“常微分方程求解”，讲述利用 MATLAB 中的求解函数、求解器，以及通过编程实现一些经典的数值方法来求解常微分方程。

  常微分方程作为微分方程的基本类型之一，在自然界与工程界有很广泛的应用。利用 MATLAB 的求解函数，可以求解常微分方程的解析解；通过求解器和编程，可以得到常微分方程的数值解。

- 第 9 章“偏微分方程求解”，讲述利用 MATLAB 中的偏微分方程工具箱（PDE

Toolbox）的函数进行偏微分方程求解，以及通过编程实现三大类偏微分方程的数值求解。

最优化计算在实际中有着广泛的应用。可以利用 MATLAB 提供的最优化计算工具箱中的函数进行最优化计算，也可以编程实现相应的最优化算法。

- 第 10 章“复数与复变函数计算”，讲述 MATLAB 中的复数基本运算，MATLAB 在复变函数中的应用，包括留数运算，泰勒级数，傅里叶变换，以及拉普拉斯变换等。

  复数，以及在其基础上发展起来的复变函数，解决了许多实变函数无法解决的问题，有着广泛的工程实际应用。MATLAB 不仅支持实数运算，而且还支持在运算和函数中使用复数或复数矩阵，利用 MATLAB，可方便地解决一些复数领域的问题。

- 第 11 章“概率统计计算”，讲述利用 MATLAB 中的统计工具箱（Statistics Toolbox）中相关函数进行统计量计算、参数估计、假设检验、方差分析和回归分析，以及绘制统计图等。

  概率统计在社会、经济、科学等各个领域有着广泛的应用。MATLAB 提供了功能强大的统计工具箱，包含 200 多个用于概率统计方面的功能函数，且其具有简单的接口操作。熟练掌握和运用这些函数，将会给概率统计工作带来极大的方便。

- 第 12 章“最优化计算”，讲述利用 MATLAB 实现几类常见的最优化计算的方法，包括无约束最优化方法和约束最优化方法，以及 MATLAB 最优化工具箱中的一些优化函数的使用。

  最优化计算在实际中有着广泛的应用，可以利用 MATLAB 提供的最优化计算工具箱中的函数进行最优化计算，也可以编程实现相应的最优化算法。

- 附录 A“MATLAB 程序设计入门”，讲述了 MATLAB 程序设计的基础知识，包括 MATLAB 概述、桌面环境，以及编程等基本概念和基本操作。

  一些简单的、通用的科学计算，可以调用 MATLAB 的自带函数及工具箱提供的函数来实现；而对于一些复杂的、专门的科学计算，可能需要通过编写 MATLAB 程序来实现，可见，熟练掌握 MATLAB 程序设计，是高效使用 MATLAB 进行科学计算的基本要求。

  对于 MATLAB 程序设计的初学者，建议认真学习本附录，而对于熟悉 MATLAB 编程的读者，则可以有选择地学习本附录。

- 附录 B“本书所编写的算法程序索引”，给出了本书精心编写的科学计算的算法程序的注释和索引，这些算法程序按章节顺序列出，非常便于查询。

  本书的精华部分，也就是书中编写的这些算法程序，代码正确、注释丰富、可读性强，读者既可以直接调用这些算法进行实际应用，又可以通过研究这些算法程序来加深对算法的理解和提高编程技巧；而且这些程序代码有序地保存在随书光盘中，读者可方便地进行调用、调试和使用。

- 附录 C“MATLAB 科学计算常用函数注释”，分类列出了 MATLAB 科学计算中常用到的函数命令及功能注释，包括线性代数、曲线拟合工具箱、样条工具箱、偏微

分方程工具箱、统计工具箱和最优化工具箱中的函数的名称及注释。

上述函数的索引和注释以及 MATLAB Help，对于学习 MATLAB，熟练使用 MATLAB 进行科学计算提供有力的帮助。

光盘说明

本书附带光盘中包括了全书主要实例对应的 MATLAB 的 M 文件。所有代码按照章节存放在各个文件夹下，如“第 1 章”文件夹下存放了本书第 1 章主要的算法程序代码或实例代码，依此类推。在每一个文件夹下的 M 文件，其名称和书中的实例编号一一对应，如 Ex0101.m 文件对应于例 1-1 的实例，依此类推。

对于算法的程序代码，在光盘中存为同函数名的 M 文件，例如，richason.m 表示里查森迭代法求解线性方程组的程序，依此类推。

读者可以通过运行光盘提供的代码文件，亲自体验本书主要实例的运行效果。由于所有代码都是在 MATLAB 7（R14）下编写并调试通过，因此，使用本光盘中实例前，读者需要安装 MATLAB 7（R14），并将包含待运行.m 文件的文件夹添加到 MATLAB 路径或设置为 MATLAB 当前目录。如读者需要运行 ex0101.m，那么就需要将包含此 M 文件的“第 1 章”文件夹添加到 MATLAB 路径，或者将其设置为 MATLAB 当前目录，然后通过命令窗口调用文件名，或者在 M-Editor 窗口打开并运行代码文件等方式来运行此 M 文件。

本书读者

本书既可以作为 MATLAB 教学用书，又可作为高等数学、线性代数、计算方法、复变函数、概率统计、数学规划、偏微分方程解法以及动态仿真等课程的教学辅导书，还可以作为物理、化学、机械、计算机、控制等领域的科研人员以及工程计算人员学习和使用 MATLAB 的参考书。

致谢

在本书编写过程中，得到了朱沭红老师的大力支持，在此对她表示衷心的感谢！对 IBM 公司的肖静小姐，清华大学的刘玉泉、张锴，中国科学院的傅玉等给予我们持续的鼓励和支持表示感谢，同时对各位钻研 MATLAB 的网友给予的启发和帮助表示感谢。

由于时间仓促，作者水平和经验有限，书中错漏之处在所难免，敬请读者指正，我们的电子邮箱是：wa_2003@126.com。

作　者

2007 年 3 月于北京

内 容 简 介

本书结合高校数学课程教学和工程科学计算应用的需要，从实用角度出发，通过大量的算法实现和典型应用实例，详尽系统地讲述 MATLAB 在线性方程组求解、插值与曲线拟合、矩阵特征值计算、求导与微分计算、积分计算、非线性方程求解、常微分方程求解、偏微分方程求解、复数与复变函数计算、概率统计计算，以及最优化计算等领域中的应用。

本书既可以作为 MATLAB 教学用书，又可以作为高等数学、线性代数、计算方法、复变函数、概率统计、数学规划、偏微分方程解法，以及动态仿真等课程的教学辅导书，还可以作为物理、化学、计算机、机械、控制等领域的科研人员和工程计算人员学习和使用 MATLAB 的参考书。

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

图书在版编目（CIP）数据

精通 MATLAB 科学计算 / 王正林，龚纯，何倩编著. —北京：电子工业出版社，2007.7
（MATLAB 精品丛书）
ISBN 978-7-121-04627-8

Ⅰ. 精… Ⅱ. ① 王… ② 龚… ③ 何… Ⅲ. 计算机辅助计算－软件包，MATLAB－高等学校－教材
Ⅳ. TP391.75

中国版本图书馆 CIP 数据核字（2007）第 093447 号

责任编辑：朱沐红　许　艳
印　　刷：北京天宇星印刷厂
装　　订：涿州市桃园装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：29.5　字数：610 千字
印　　次：2007 年 7 月第 1 次印刷
印　　数：5000 册　定价：49.80 元（含光盘 1 张）

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

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

服务热线：（010）88258888。

目　录

第 1 章　绪论 …… 1

1.1　MATLAB 中的科学计算概述 …… 1

1.1.1　MATLAB 的发展概况 …… 1

1.1.2　MATLAB 进行科学计算的优势 …… 2

1.2　MATLAB 的基本操作 …… 3

1.2.1　M 文件操作及常用查询命令 …… 3

1.2.2　数据的输入输出 …… 4

1.2.3　绘制二维图形 …… 7

1.2.4　三维图形的输出 …… 12

1.2.5　基本数学函数 …… 15

1.2.6　向量和矩阵的基本运算 …… 19

1.3　MATLAB 编程的技巧 …… 23

1.3.1　嵌套计算 …… 23

1.3.2　循环结构 …… 25

1.3.3　循环和嵌套 …… 27

1.3.4　例外处理机制 …… 28

1.3.5　全局变量的使用 …… 29

1.3.6　通过 varargin 传递参数 …… 31

1.4　小结 …… 32

第 2 章　线性方程组求解 …… 33

2.1　求逆法 …… 33

2.2　分解法 …… 34

2.2.1　LU 分解法 …… 34

2.2.2　QR 分解法 …… 35

2.2.3　Cholesky 分解法 …… 36

2.2.4　其他分解法 …… 37

2.3　迭代法 …… 40

2.3.1　逐次逼近法 …… 40

2.3.2　里查森迭代法 …… 41

2.3.3　Jacobi 迭代法 …… 42

2.3.4　Gauss-Seidel 迭代法 …… 44

2.3.5　超松弛迭代法 …… 46

2.3.6　两步迭代法 …… 50

2.3.7　梯度法 …… 51

2.3.8　其他迭代法 …… 57

2.4　特殊解法 …… 58

2.4.1　三对角矩阵的追赶法 …… 58

2.4.2　快速求解法 …… 60

2.5　非奇次线性方程组的解法 …… 61

2.5.1　超定方程的解法 …… 61

2.5.2　有无穷组解的线性方程组的解法 …… 62

2.6　小结 …… 63

第 3 章　数据插值与拟合 …… 64

3.1　MATLAB 中的插值函数 …… 64

3.1.1　一元插值函数 …… 65

3.1.2　二元插值函数 …… 69

3.1.3　其他插值相关的函数 …… 70

3.2　拉格朗日插值法 …… 73

3.3　艾特肯插值法 …… 75

3.4　利用均差的牛顿插值法 …… 77

3.5　等距节点插值法 …… 79

3.5.1　利用差分的牛顿插值 …… 79

3.5.2　高斯插值 …… 83

3.6　埃尔米特插值法 …… 87

3.7　有理分式插值法 …… 89

3.8　函数逼近与曲线拟合 …… 93

3.8.1　切比雪夫逼近 …… 93

3.8.2　勒让德逼近 …… 95

3.8.3　帕德逼近 …… 97

3.8.4　傅里叶逼近 …… 99

3.8.5　多项式曲线拟合 …… 101

3.8.6　最小二乘法拟合 …… 102

3.9　小结 …… 103

第 4 章　矩阵特征值计算 …… 104

4.1　特征值与特征向量 …… 104

4.2 条件数与病态矩阵……104
4.3 相似变换……107
4.4 特征值求法……108
4.4.1 特征多项式法……109
4.4.2 幂法……110
4.4.3 瑞利商加速幂法……112
4.4.4 收缩法……114
4.4.5 逆幂法……115
4.4.6 位移逆幂法……117
4.4.7 QR 算法……119
4.5 舒尔分解和奇异值分解……125
4.6 功能强大的 eig 函数……126
4.7 矩阵指数……128
4.8 小结……129
第 5 章 求导与微分计算……130
5.1 MATLAB 中和微分有关的函数……130
5.2 求导数的其他方法……133
5.2.1 中点公式……133
5.2.2 三点公式法和五点公式法……134
5.2.3 样条函数法……137
5.2.4 辛普森数值微分法……138
5.2.5 理查森外推算法……142
5.3 小结……144
第 6 章 积分计算……145
6.1 MATLAB 中的不定积分函数……145
6.2 MATLAB 中的定积分函数……146
6.2.1 定积分计算函数……146
6.2.2 二重积分计算函数……146
6.2.3 三重积分计算……147
6.3 梯形法数值积分……148
6.4 辛普森法数值积分……149
6.5 牛顿-科茨法数值积分……152
6.6 高斯系列公式数值积分……154
6.6.1 高斯公式……154
6.6.2 高斯-拉道公式……156
6.6.3 高斯-洛巴托公式……158
6.7 区间逐次分半法数值积分……160
6.7.1 区间逐次分半梯形公式数值积分……160
6.7.2 区间逐次分半辛普森公式数值积分……162
6.7.3 区间逐次分半布尔公式数值积分……163
6.8 龙贝格积分法……165
6.9 自适应法求积分……167
6.10 样条函数求积分……169
6.11 简单的奇异积分……170
6.11.1 高斯-拉盖尔公式……170
6.11.2 高斯-埃尔米特公式……172
6.12 重积分的数值计算……173
6.12.1 梯形公式……174
6.12.2 辛普森公式……175
6.13 小结……177
第 7 章 非线性方程求解……178
7.1 MATLAB 中的非线性方程求根函数……178
7.1.1 fzero 函数……178
7.1.2 fsolve 函数……180
7.2 非线性方程求根的其他数值方法……181
7.2.1 二分法……181
7.2.2 黄金分割法……183
7.2.3 不动点迭代法……185
7.2.4 弦截法……189
7.2.5 史蒂芬森弦截法……190
7.2.6 抛物线法……192
7.2.7 牛顿法……194
7.2.8 两步迭代法……199
7.2.9 重根的迭代方法……201
7.3 非线性方程组的数值解法……203
7.3.1 不动点迭代法……203
7.3.2 牛顿法……204
7.3.3 牛顿下山法……207
7.3.4 拟牛顿法……208

7.4 小结……210
第 8 章 常微分方程求解……211
8.1 MATLAB 中的常微分方程求解函数……211
8.1.1 常微分方程符号解函数 dsolve……211
8.1.2 求解器 solver……213
8.2 欧拉法（Euler）……215
8.2.1 简单欧拉法……215
8.2.2 改进的欧拉法……217
8.3 龙格-库塔法（Runge-Kutta）……220
8.4 预估-校正法(predictor-corrector)……225
8.4.1 Adams-Bashforth-Moulton 方法……225
8.4.2 Hamming 方法……226
8.5 常微分方程求解综合实例……229
8.6 差分方程求解……231
8.6.1 用 filter 函数解差分方程……231
8.6.2 递推法解差分方程……233
8.6.3 利用 z 反变换求解……235
8.7 小结……236
第 9 章 偏微分方程求解……237
9.1 偏微分方程概述……237
9.2 椭圆偏微分方程……238
9.2.1 常规 Helmholtz 方程的数值解……238
9.2.2 满足牛顿边值条件的 Helmholtz 方程……242
9.3 抛物线偏微分方程……245
9.3.1 显式前向欧拉法……246
9.3.2 隐式后向欧拉法……249
9.3.3 Grank-Nicholson 方法……252
9.3.4 二维抛物线方程……255
9.4 双曲线偏微分方程……258
9.4.1 显式中心差分法……258
9.4.2 二维双曲线型方程……261
9.5 有限元法……264
9.6 偏微分方程求解工具（PDETOOL）的使用……271
9.6.1 可由 PDETOOL 求解的基本偏微分方程……271
9.6.2 PDETOOL 的使用说明……272
9.7 小结……277
第 10 章 复数与复变函数计算……278
10.1 复数的表示法……278
10.1.1 复数的一般表示……278
10.1.2 复数矩阵的表示……280
10.1.3 复数绘图……281
10.2 复数的基本运算……282
10.2.1 复数的结构操作函数……282
10.2.2 复数的基本数学运算……283
10.2.3 复数方程求根……285
10.3 留数的基本运算……286
10.3.1 留数基础……286
10.3.2 MATLAB 中留数的求取……286
10.4 泰勒级数……287
10.5 傅里叶变换……289
10.5.1 基本傅里叶变换……289
10.5.2 傅里叶反变换……292
10.6 拉普拉斯变换……294
10.6.1 基本拉氏变换……294
10.6.2 拉氏反变换……296
10.7 小结……297
第 11 章 概率统计计算……298
11.1 MATLAB 统计工具箱介绍……298
11.2 随机变量的数字特征……299
11.2.1 期望……299
11.2.2 方差、标准差、矩……302
11.2.3 协方差、相关系数……304
11.2.4 偏斜度和峰度……307
11.2.5 其他数字特征……308
11.3 随机变量的数字特征……309
11.3.1 概率密度函数（pdf）……309
11.3.2 累积分布函数（cdf）与逆累积分布函数……311

11.3.3 特殊分布的期望和方差 ……314
11.3.4 随机数生成器 ……316
11.4 参数估计 ……317
11.5 假设检验 ……320
11.5.1 单个总体 $N(\mu,\sigma^2)$ 均值 μ 的检验 ……321
11.5.2 两个正态总体均值差的检验（t 检验） ……322
11.5.3 基于成对数据的检验（t 检验） ……323
11.5.4 正态总体方差的假设检验 ……324
11.6 方差分析 ……326
11.6.1 单因素试验的方差分析 ……326
11.6.2 双因素试验的方差分析 ……328
11.7 回归分析 ……329
11.7.1 一元多项式回归 ……330
11.7.2 多元线性回归 ……332
11.7.3 非线性回归 ……333
11.7.4 逐步回归 ……336
11.8 统计图绘制 ……338
11.9 小结 ……342
第 12 章 最优化计算 ……343
12.1 无约束最优化 ……343
12.1.1 黄金搜索法 ……343
12.1.2 二次插值法 ……345
12.1.3 Nelder-Mead 算法 ……348
12.1.4 最速下降法 ……352
12.1.5 牛顿法 ……355
12.1.6 模拟退火法 ……357
12.1.7 遗传算法 ……360
12.2 约束最优化 ……364
12.2.1 拉格朗日乘子法 ……364
12.2.2 惩罚函数法 ……366
12.3 MATLAB 的内置最优化函数 ……368
12.3.1 最优化工具箱 ……368
12.3.2 无约束最优化函数 ……369
12.3.3 约束最优化函数 ……372
12.3.4 线性规划函数 ……375
12.4 最优化问题应用综合实例 ……377
12.4.1 无约束最优化问题综合实例 ……377
12.4.2 约束最优化问题综合实例 ……378
12.5 小结 ……382
附录 A MATLAB 程序设计入门 ……383
附录 B 本书所编写的算法程序索引 ……423
附录 C MATLAB 科学计算常用函数注释 ……428
参考文献 ……448

第 1 章 绪 论

MATLAB 语言可谓真正的 21 世纪的科学计算语言，在科学计算中有着独特的优势。了解 MATLAB 进行科学计算的特点，熟悉 MATLAB 的各种操作是高效利用 MATLAB 进行科学计算所必须的。

通过本章的学习，读者会对 MATLAB 的发展概况及特点有一个全面的认识，并能熟悉 MATLAB 的基本操作，掌握初步的 MATLAB 的编程技巧，为熟练运用 MATLAB 进行科学计算打下基础。

1.1 MATLAB 中的科学计算概述

1.1.1 MATLAB 的发展概况

MATLAB 是由美国 Mathworks 公司推出的用于数值计算和图形处理的科学计算软件。MATLAB 是英文 MATrix LABoratory（矩阵实验室）的缩写。它的第 1 版（DOS 1.0）版本发行于 1984 年，经过 20 余年的不断改进，现今已推出 7.x 版。

MATLAB 以商品形式出现的短短几年内，其良好的开放性和运行的可靠性使原先控制领域里的封闭式软件包（如英国的 UMIST、瑞典的 LUND 和 SIMNON、德国的 KEDDC）纷纷被淘汰，而以 MATLAB 为平台加以重建。20 世纪 90 年代时，MATLAB 已经成为国际控制界公认的标准计算软件。

为了完善 MATLAB 的功能，MathWork 公司又对 MATLAB 进行了改进，推出新版本。于 2001 年推出的 MATLAB 6.x 版本集中了日常数学处理中的各种功能，包括高效的数值计算、矩阵运算、信号处理和图形生成等功能。在 MATLAB 环境下，用户可以进行程序设计、数值计算、图形绘制、输入输出和文件管理等各项操作。

2005 年 MATLAB 7.x 版亮相，7.x 版在继承和发展其原有的数值计算和图形可视能力的同时，还有以下几个重要变化：

（1）新的开发环境，包括多文档管理、编辑器、工作空间浏览器等快捷工具；

（2）可在编辑器中部分执行 M 文件代码，并能以 HTML、Word 或 LaTex 发布 M 代码；

（3）可创建嵌套函数；

（4）可设条件断点，在表达式为真时停止运行；

（5）可以在计算和处理更大整型数据集时保持数据类型，能处理更大的单精度数据集；

（6）使用新的绘图界面，实现无 M 代码时交互式地创建和编辑图形；可以自动生成图形的 M 代码，并利用该代码重建图形；

（7）改进了图形标注；

（8）可对一组图形对象进行旋转、平移和缩放；

（9）增加压缩的 MAT 文件选项，节约磁盘空间；

（10）MAT 文件中的字符数据可以用于多种语言。

MATLAB 是一个功能十分强大的系统，是集数值计算、图形管理和程序开发为一体的科学计算环境。除此之外，MATLAB 还具有很强的功能扩展能力，与它的主系统一起，可以配备各种各样的工具箱，完成一些特定的任务，而且用户可以根据自己的工作任务，开发自己的工具箱。

在许多高校里，MATLAB 是攻读学位的大学生、硕士生、博士生必须掌握的基本工具。在国际学术界，MATLAB 已经被确认为准确、可靠的科学计算标准软件。在许多国际一流学术刊物上，都可以看到关于 MATLAB 的应用的论文。在设计研究单位和工业部门，MATLAB 被认为是进行高效研究与开发的首选软件工具。

1.1.2 MATLAB 进行科学计算的优势

MATLAB 科学计算环境，具有强大的计算绘图能力，提供大量的函数库、工具箱，几乎涵盖了所有的工程计算领域，被誉为“演算纸”式的工程计算工具。MATLAB 进行科学计算的优势主要可以概括为以下几点：

（1）MATLAB 集成了几乎所有的科学研究和工程计算要用的算法，非常便于进行科学计算，而且默认数据结构是双精度数组，能实现高精度的科学计算；

（2）MATLAB 将高性能的数值计算和可视化集成在一起，并提供大量的内置函数和开放的程序和数据接口，因而广泛地应用于科学计算、控制系统与信息处理等领域的分析、仿真和设计工作。

（3）MATLAB 包含各种能够进行常规运算的工具箱，如常用的矩阵代数运算、数组运算、方程求根、优化计算，以及函数求导积分符号运算等；同时还提供了编程计算的编程特性，通过编程解决一些复杂的工程问题。在 MATLAB 中可绘制二维、三维图形，使输出结果可视化。这些强大功能为科学计算带来了方便。

（4）MATLAB 语言是一种解释执行的脚本语言，简单易学，使用 MATLAB 软件进行科学计算，能够极大地加快科研人员进行研究开发的进度，减少在编写程序和开发算法方面所消耗的时间和经费支出，从而获得最大的效能。

1.2 MATLAB 的基本操作

MATLAB 是一个具有高度封装性的软件包，其内部的程序包都提供了输入输出接口，用户可以方便地进行学习和使用。本节将介绍一些基本的操作指令，这些指令是编写 MATLAB 程序的基础。

1.2.1 M 文件操作及常用查询命令

MATLAB 程序文件以.m 扩展名形式保存（M 文件）。一个 M 文件是一个封装的程序，提供输入输出接口，实现特定的功能。用户可以直接在命令窗口中运行这个程序：输入文件名（程序名）和相关参数即可；也可以在其他 M 文件中调用这个程序。

1. M 文件操作

要创建或者编辑一个 M 文件，只需在主菜单的左上角点击“File/New/M-file”或者“File/Open”选项即可。

如果用户需要执行的 M 文件不在当前的路径中，文件名将不会被 MATLAB 识别。此时需要将 M 文件所在的目录添加入 MATLAB 路径中。具体操作为：单击“File/Set Path”，选择“Add Folder”按钮，在备选框中选择文件所在位置，保存并关闭打开的窗口。

2. 常用查询命令介绍

MATLAB 提供的查询命令有 help 和 lookfor，可以方便地进行查询，获得帮助。

（1）help

help 命令用来查询已知命令，帮助用户了解特殊命令/函数的用法。

格式：help 命令名

如以下示例所示：

```
>> help disp
 DISP Display array.
 DISP(X) displays the array, without printing the array name.  In
...
```

（2）lookfor

该命令用来寻找未知的命令，找到所需的命令后，即可用 help 命令进一步找出其用法。

格式：lookfor 命令或字符串

如以下示例所示：

```
>> lookfor disp
DISP Display array.
```

```
DISPLAY Display array.
…
```

1.2.2 数据的输入输出

MATLAB 将所有输入数据存储在固定位置。这些数据包括用户直接从命令窗口输入的、由文件输入或从键盘输入的数据。MATLAB 会保存这些数据直到 clear 命令出现或者用户退出 MATLAB 时为止。

1. 通过 MATLAB 命令窗口输入/输出数据

MATLAB 的一个显著特征是它处理向量/矩阵的方式与处理标量的方式类似，具体如【例 1-1】所示。

【例 1-1】 MATLAB 命令窗口的输入/输出操作实例。从命令窗口输入以下矩阵：

$$\boldsymbol{A}=[1,2,3,4],\quad \boldsymbol{B}=\begin{bmatrix}1\\2\\3\end{bmatrix},\quad \boldsymbol{C}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$

解：在命令窗口中输入：

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

如果用户希望实时检查命令，则可在命令后直接按回车键，命令结果将紧接着显示在命令窗口中。反之，以分号“;”结束命令，在按回车键后将没有任何结果显示，如下所示：

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

2. 通过文件输入/输出数据

MATLAB 可以处理两类数据文件：

（1）二进制文件，扩展名为.mat。该文件能够保存多个变量的值，但在被 MATLAB 处理时不能被其他程序共享。

（2）ASCII 文件，扩展名为.dat。它仅能保存一个变量值，却能被多个程序共享。

【例 1-2】 .mat 文件读写的实例。

解：在命令窗口中输入：

```
>> save MyDatFile C B A %保存数据，默认保存类型为二进制文件，此时在当前工作目录下
```

```
                                %成 myfile.mat 文件
>> clear C B A %清空内存
>> C
??? Undefined function or variable 'C'. %此时已无 C 的记录
>> load MyDatFile C B A %从.mat 文件中读取变量
>> C
```

输出结果为：

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

如果需要保存的文件类型为 ASCII，则存取时须增加类型说明，如【例 1-3】所示。

【例 1-3】 ASCII 文件的读写实例。

解：在命令窗口中输入：

```
>> save MyASCIIFile.dat C   %保存 ASCII 文件为 MyASCIIFile.dat
>> load MyASCIIFile.dat     %读取文件
>> MyASCIIFile %显示文件
```

输出结果为：

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

值得注意的是，如果将字母等非数字形式保存为 ASCII 类型时，保存的是变量的 ASCII 码。

在 MATLAB 命令窗口中继续输入如下内容：

```
>> clear all
>> aa='ee';
>> save aaf.dat aa /ascii
>> clear aa
>> load aaf.dat
>> aaf
```

输出结果为：

```
aaf =
101   %ee 的 ASCII 码
```

另外，如果.dat 文件保存的是矩阵，必须保证其每行的列数相同。

3. 从键盘输入数据

MATLAB 中的 input 命令提示用户从键盘输入，并将输入的变量在命令窗口中显示出来，例如在命令窗口中输入：

```
>> a=input('Please enter a: ')
```

输出为：

```
Please enter a:
```

如果输入下列向量，并按回车键后，则输出 *a* 的结果，如下所示：

```
[1/3 1/7 1/9]
a =
0.3333    0.1429    0.1111
```

4．输入/输出的数据格式

注意，输出数据仅取小数点后的前 4 位，这是由 MATLAB 的数据格式选择决定的。用户可以通过命令 format 自行选择 MATLAB 的数据格式。

【例 1-4】 使用 format 选择数据格式实例。

解：接着在命令窗口中输入：

```
>> format rat %有理数格式
>> a
```

输出结果为：

```
a =
      1/3           1/7           1/9
```

在命令窗口中输入：

```
>> format long %14 位精度小数
>> a
```

输出结果为：

```
a =
   0.33333333333333   0.14285714285714   0.11111111111111
```

其他常见的 MATLAB 的数据格式如表 1-1 所示。

表 1-1　常用数据格式

格　式	说　明	举　例
format	短格式（默认显示格式），同 short	3.1416
format short	短格式（默认显示格式），只显示 5 位	3.1416
format long	长格式，双精度数 15 位，单精度数 7 位	3.14159265358979
format short e	短格式 e 方式（科学计数格式）	3.1416e+000
format long e	长格式 e 方式	3.141592653589793e+000
format short g	短格式 g 方式	3.1416
format long g	长格式 g 方式	3.14159265358979
format compact	压缩格式/紧凑格式	
format loose	自由格式/宽松格式	

MATLAB 可以用不同的格式显示数据，这些显示结果与内存中的真实数据并不完全吻合。

除此以外，还可以用 dip 命令和 fprintf 命令来显示结果。disp 命令显示变量或字符串

内容时，无“x=”或者“ans=”。例如，

继续在 MATLAB 命令窗口中输入：

```
>> disp(a)
```

输出结果为：

```
    0.3333    0.1429    0.1111
```

fprintf 命令能够以加后缀的方式显示不同的格式。fprintf 的格式说明符如表 1-2 所示。

表 1-2 fprintf 常用的类型说明符和特殊符

类型符	类型	字符	含义
%c	字符型	\n	换行
%s	字符串型	\t	水平制表（跳到下一个 tab 位置）
%d	十进制整型	\b	退格，将当前位置移到前一列
%f	浮点型	\r	回车
%e	十进制指数型	\f	换页，将当前位置移到下页开头
%x	十六进制整数型	%%	%
%bx	十六进制浮点型		

【例 1-5】 数据精度使用实例。使用 fprintf 输出不同精度的数据。

解： 在 MATLAB 命令窗口中输入：

```
>>fprintf('a(1) is %6.5f .\n a(2) is %5.2f', a(1), a(2));% 6.5f 表示显示 6
% 位，精确到小数点
```

输出结果为：

```
a(1) is 0.33333 .
a(2) is  0.14
```

1.2.3 绘制二维图形

MATLAB 有很强的图形处理功能，可以方便地实现数据的可视化。下面介绍二维图形的绘制。

1. 常用的图形输出命令 plot

plot 的调用格式有如下 3 种。

（1）plot(a, 's')

如果 ***a*** 是实矩阵，则以序列[1 2 3…]为横坐标，分别以各列为纵坐标画出曲线。字符串 s 设定曲线颜色和绘图方式，用颜色字符串的前 1~3 个字母指定颜色，如“yellow”以“y”表示。

如果 ***a*** 是复数矩阵，则分别以各列的实部和虚部为横纵坐标画出曲线。

表 1-3 给出了 MATLAB 中颜色和线型的选项。

表 1-3　颜色和线型的选项

字　母	颜　色	标　点	线　型
y	黄色	.	点线
m	粉红	○	圈线
r	亮蓝	×	×线
g	绿色	＋	＋字线
b	蓝色	—	实线
w	白色	*	星形线
k	黑色	—·(--)	点划线

【例 1-6】 二维图形绘制命令使用实例之一。使用 plot 函数绘制[1 2; 2 4; 3 2; 5 4; 6 4; 7 2; 8 5; 5 10]所表示的 8*2 实矩阵的曲线。

解：在 MATLAB 命令窗口中输入：

```
>> a=[1 2;2 4;3 2;5 4;6 4;7 2;8 5;5 10];
>> plot(a, '-r')
```

输出结果如图 1-1 所示。

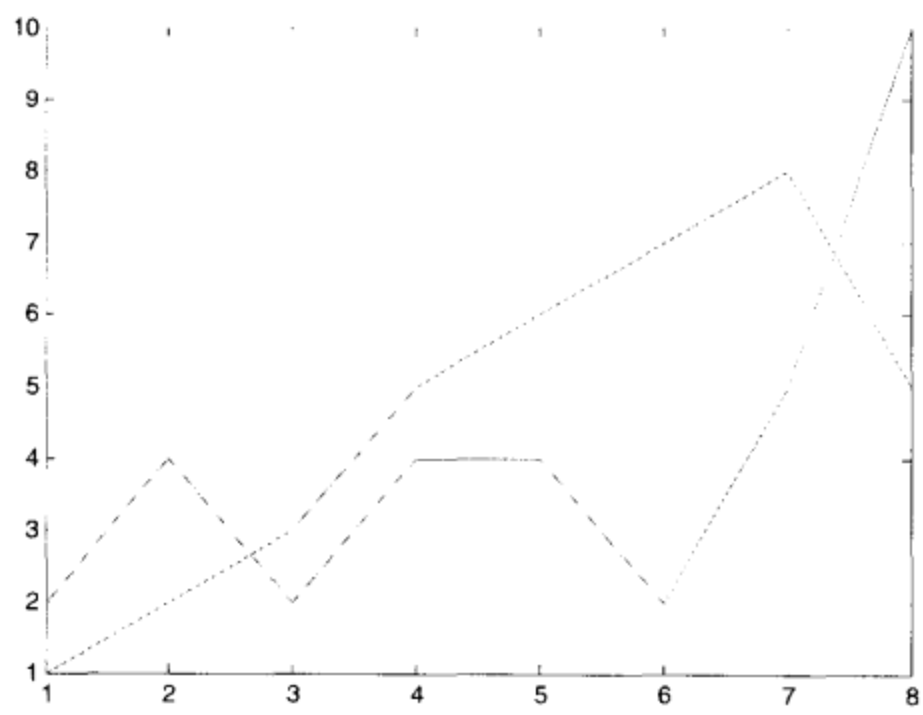

图 1-1　8*2 实矩阵的曲线

（2）plot(a,b,'s')

若 ***a***，***b*** 为同维矩阵，则分别以 ***a*** 的各列为横坐标，***b*** 的各列为纵坐标，按照 s 指定的线型，画出若干曲线，曲线条数为 ***a***，***b*** 的列数。

若 ***a*** 为列向量，***b*** 为行数与 ***a*** 相同的矩阵，则以 ***a*** 为横坐标，分别以 ***b*** 的各列为纵坐标画出曲线，曲线条数为 ***b*** 的列数。

【例 1-7】 二维图形绘制命令使用实例之二。使用 plot 函数绘制以[1 2; 3 4; 4 3; 5 4; 6 5; 5 6]为横坐标，以[1 10; 3 8; 4 7; 5 9; 6 8; 5 11]为纵坐标所表示的 6*2 实矩阵的曲线。

解：在 MATLAB 命令窗口中输入：

```
>> a=[1 2;3 4;4 3;5 4;6 5;5 6];
>> b=[1 10;3 8;4 7;5 9;6 8;5 11];
>> plot(a, b)
```

输出结果如图 1-2 所示。

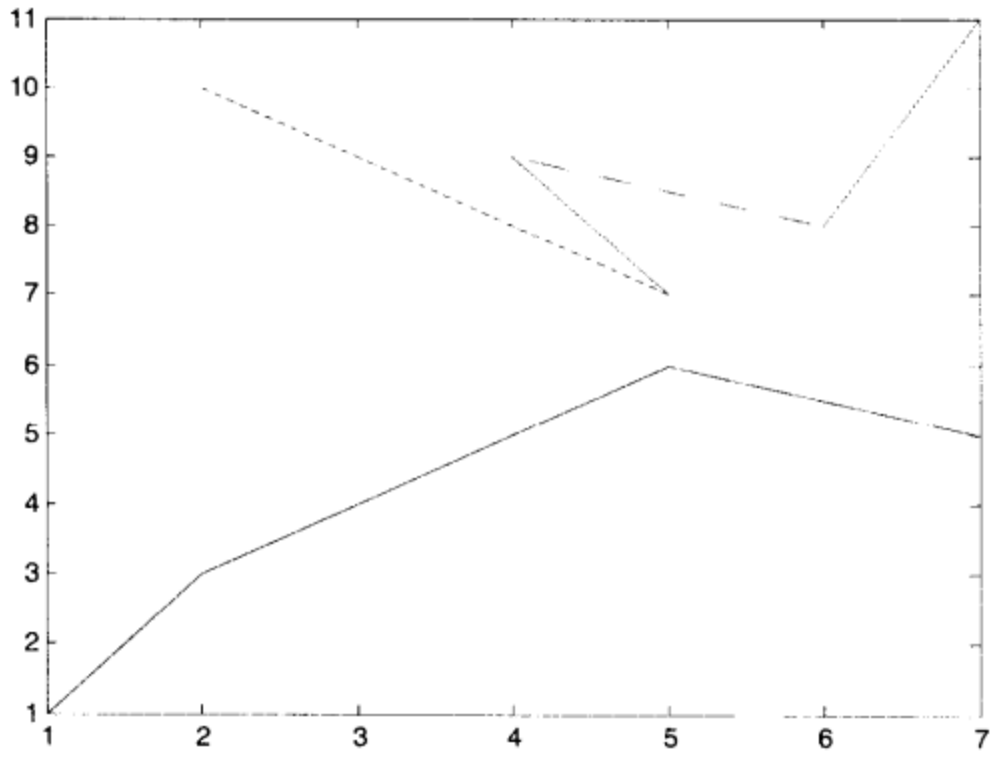

图 1-2 同维 6*2 矩阵的曲线

（3）plot(a1，b1，'s1'，a2，b2，'s2'，…)

此命令按照上述(2)中所述的规则依次作出：***a***1 与 ***b***1 构成的图，***a***2 与 ***b***2 构成的图……即在同一幅图中显示多重曲线。

【例 1-8】 一图中多条曲线绘制实例。在一幅图中作出 $y=x^2$ 和 $y=2x+3$ 的曲线。

解：在 MATLAB 命令窗口中输入：

```
>> x1=0:0.1:10;
>> y1=x1.^2;
>> x2=0:0.5:10;
>> y2=2*x2+3;
>> plot(x1, y1, x2, y2)
```

输出结果如图 1-3 所示。

多重线的另一种画法是利用 hold 命令。

在已经画好的图形上，若设置 hold on，MATLAB 将把新的 plot 命令产生的图形叠加在原来的图形上。

命令 hold off 将结束这个过程。对【例 1-8】采用 hold on 命令可以得到与图 1-3 相同的结果。

2. 图形标记

MATLAB 提供在图形上添加网格、标题、x 轴标记、y 轴标记等函数，表 1-4 给出了常用的函数及其功能。

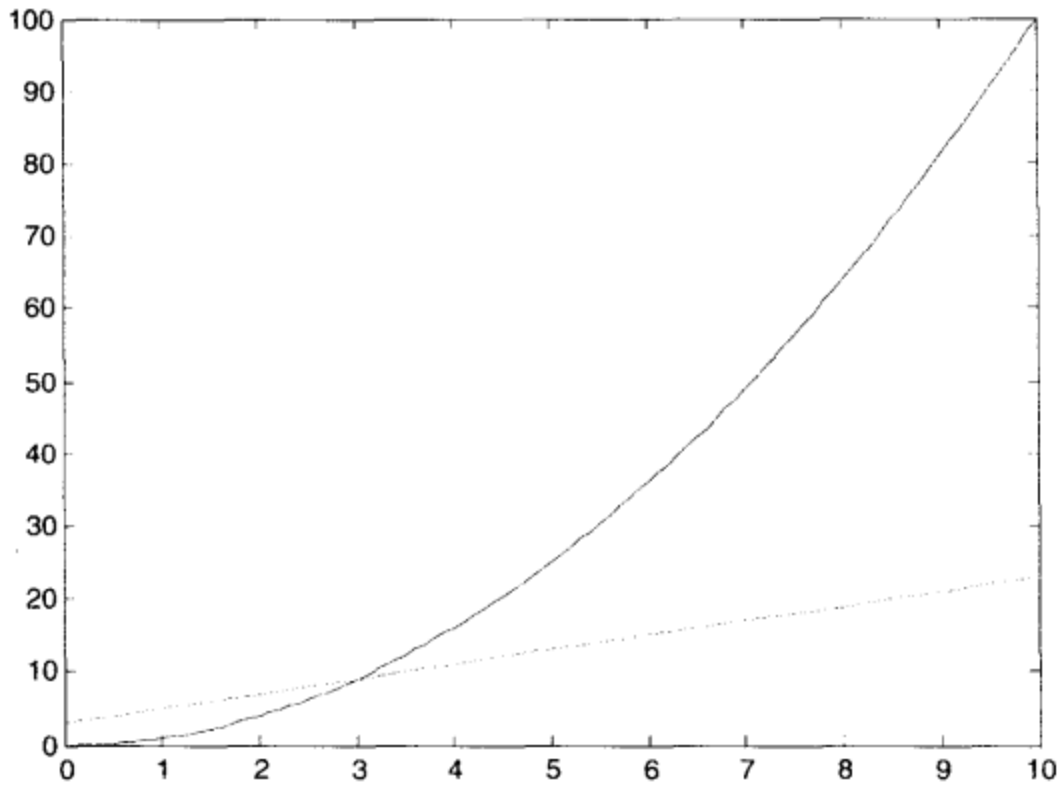

图 1-3　x 轴坐标(0,10)范围内的二次曲线和一次曲线

表 1-4　常用的图形标记函数

函 数 名	功　　能
title	为图形加标题
xlabel	为 x 轴加标记
ylabel	为 y 轴加标记
text	为图形加文本注释
gtext	用鼠标放置文本
grid	加网格线

下面举例说明上述操作，因涉及的操作较多，故用.m 文件来实现。

【例 1-9】 图形标记命令使用实例。在同一窗口作出 x^2 与 sin(x)的函数图，并添加网格线、标题，以及 x 轴和 y 轴标注。

解：编写 M 文件 ex0101.m

```
function ex0101()
x=0:pi/15:2*pi;
y1=0.5*x.^2;
y2=10*sin(x);
plot(x, y1, 'r.-', x, y2);          %画图，其中，y1 用点划线
grid                                %网格线
xlabel('variable x')                %x 坐标
ylabel('variable y')                %y 坐标
title('x^2 and sin(x)');            %标题
text(6.2, -5.5, 'sin(x)');          %在坐标 6.2 和-5.5 处标注 y2
text(6.2, 16, 'x^2');               %标注 y1
```

运行该函数，输出结果如图 1-4 所示。

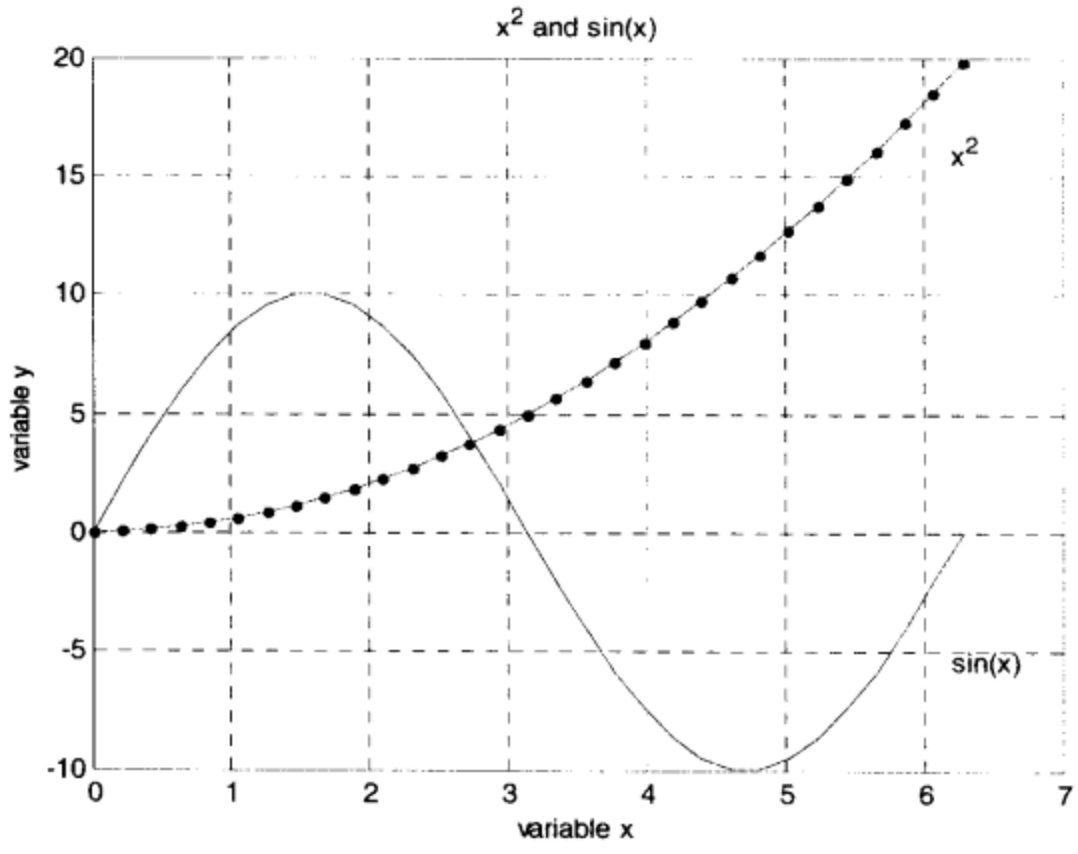

图 1-4 二次函数和正弦函数曲线

3. 拆分窗口：subplot 命令

用 subplot(*m*, *n*, *p*)命令，可以在同一个画面上建立几个坐标系，把一个画面分成 *m*×*n* 个图形区域，*p* 代表当前的区域号，每个区域可以对应一个独立的图。

【例 1-10】 拆分窗口绘图实例。在同一个窗口中作出如下的函数图：

$$y=e^x,\quad y=\log(x),\quad y=\sin(x),\quad y=\cos(x)$$

解：编写 MATLAB 函数文件 ex0102.m

```
function ex0102()
x=0.1:pi/15:2*pi
y1=exp(x);
y2=log(x);
y3=sin(x);
y4=cos(x);
subplot(2, 2, 1);                    %拆分为 2*2 个区域，分别在各个区域作图
plot(x, y1);
title('exp(x)');
subplot(2, 2, 2);
plot(x, y2);
title('ln(x)');
subplot(2, 2, 3);
plot(x, y3);
title('sin(x)');
subplot(2, 2, 4);
plot(x, y4);
title('cos(x)');
```

运行该函数，输出结果如图 1-5 所示。

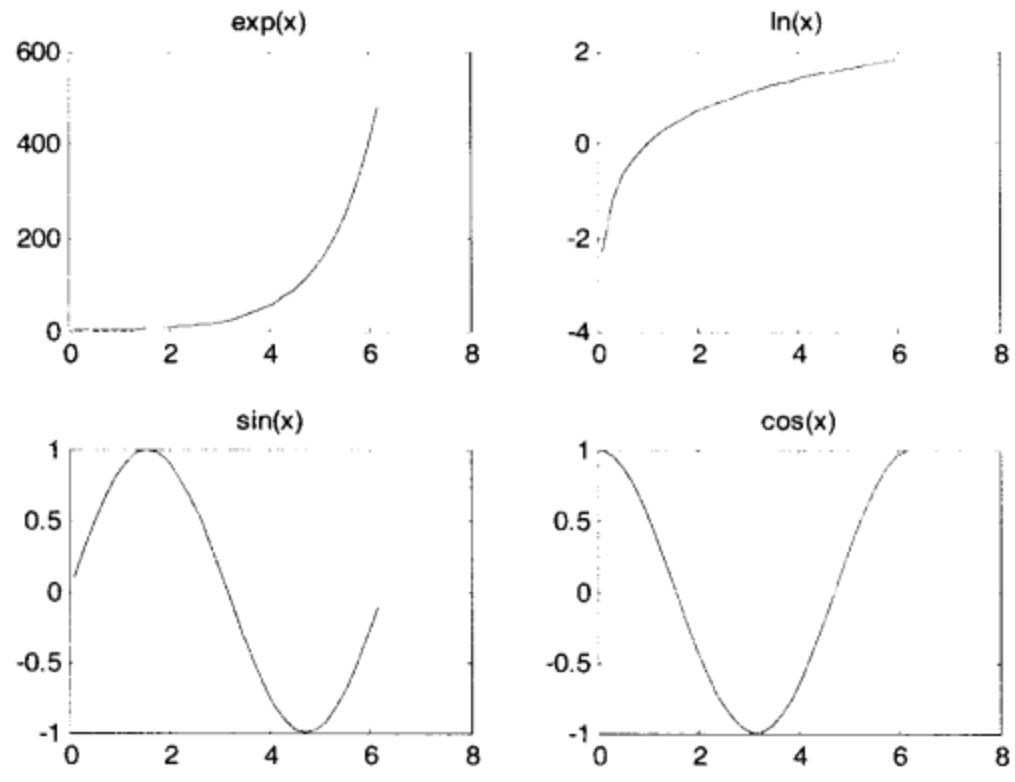

图 1-5　拆分窗口

4．其他常用命令

其他常用的二维图形绘制命令如表 1-5 所示。

表 1-5　常用的二维图形绘制命令

函　数	功　能
fill	绘制并填充二维图形
ginput	鼠标获取坐标，[*a*，*b*，butkey]=ginput，*a* 为 *x* 轴坐标，*b* 为 *y* 轴坐标，button 值代表鼠标被按下了哪个键。
box	控制图形的边框，box off 可关闭图形边框
semilogx	自变量的对数作为 *x* 轴
semilogy	因变量的对数作为 *y* 轴
loglog	两者均为对数
plotyy	双 *y* 轴的图形
axis	控制坐标轴的刻度
axis equal	输出图形的 *x*、*y* 轴等长
axis off	关闭坐标轴的显示，可产生不含坐标轴的图形
figure	打开新的绘图窗口进行绘图

1.2.4　三维图形的输出

为了显示三维图形，MATLAB 提供了各种各样的函数，例如，plot3()、mesh()和 contour()等。有一些函数可在三维空间中画线，而另一些可以画曲面与线格框架。

（1）plot3 命令

该命令将绘制二维图形的函数 plot 的特性扩展到三维空间。函数格式除了包括第三维的信息（比如 Z 方向）之外，与二维函数 plot 相同。

plot3 一般调用格式为 plot3(*x*1, *y*1, *z*1, *S*1, *x*2, *y*2, *z*2, *S*2,..., ***xn***, ***yn***, ***zn***, *Sn*)，这里 ***xn***，***yn*** 和 ***zn*** 是向量或矩阵，*Sn* 是可选的字符串，用来指定颜色、标记符号和/或线形。

当 *x*、*y*、*z* 均可表为 *t* 的显示函数时，利用 plot3 可以很容易作出三维图形。

然而，有的时候用户可能希望从不同角度观察这个三维图，MATLAB 提供了这一功能，用 view()函数可以确定一个观察点，又称为视点，MATLAB 可以作出在该视点观察原三维图的新视图。

【例 1-11】 三维图形绘制实例。用 plot3 作出如下函数的三维视图。

$$\begin{cases} x = e^{-0.1t}\cos(t) \\ y = e^{-0.1t}\sin(t) \\ z = t \end{cases}$$

解： 编写 MATLAB 函数文件 ex0103.m。

```
function ex0103()
t = 0:pi/50:6*pi;
xt = exp(-0.1*t).*cos(t); yt = exp(-0.1*t).*sin(t);
%拆分窗口为 2*2
subplot(2,2,1), plot3(xt, yt, t), grid on
title('(xt, yt, t)的三维图 a');
subplot(2,2,2), plot3(xt, yt, t), grid on, view([0 0 1]);
title('在(0,0,1)观察 a 的图 a1');
subplot(2,2,3), plot3(t, xt, yt), grid on
title('(t, xt, yt)的三维图 b');
subplot(2,2,4), plot3(t, xt, yt), grid on, view([0 -4 0])
title('在(0, -4 ,0)观察 b 的图 b1');
```

运行该函数，输出结果如图 1-6 所示。

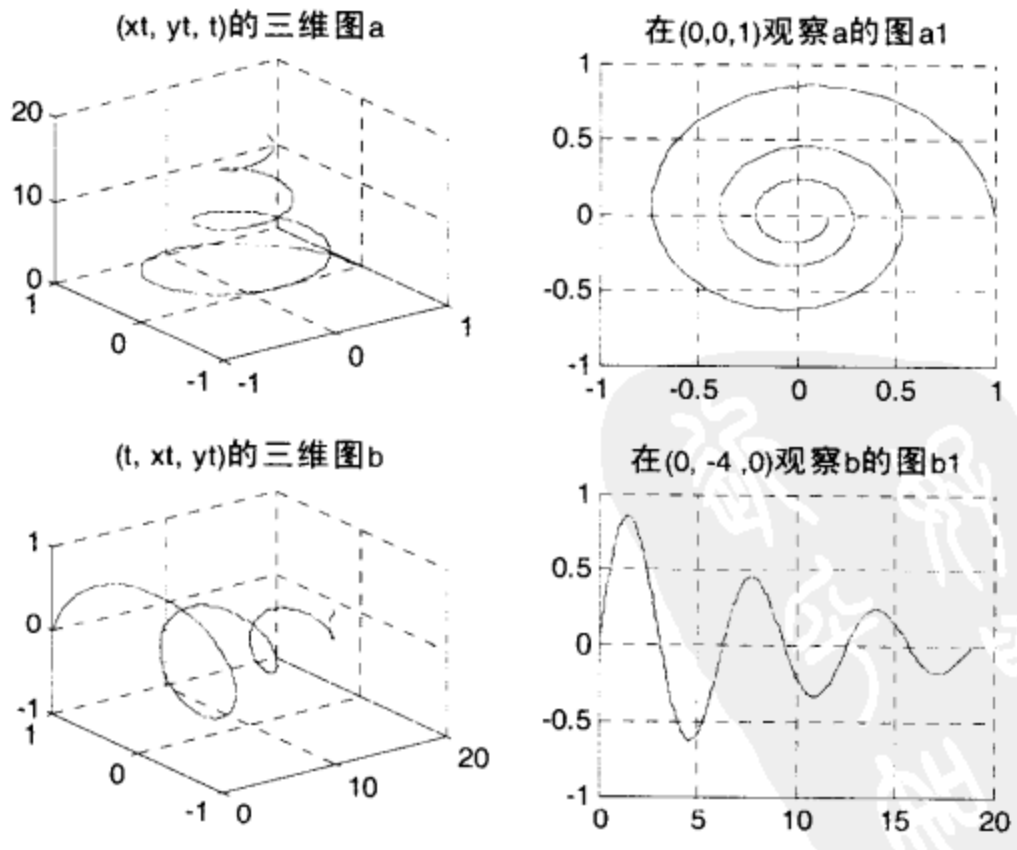

图 1-6 不同视点的三维图

图 1-6 所示为在不同视点下的三维图形。其中图 a1 为在视点(0,0,1)下观察 a 的视图，相当于 a 在 X-Y 平面的投影。图 b1 为 b 在视点(0,-4,0)下观察 b 的视图，相当于 b 的在 Y-Z 平面的投影。

（2）mesh()和 contour()命令

形如 $z=f(x,y)$ 的函数的三维图形是空间中的一个曲面。为了在 MATLAB 里画出这个曲面，z 的值需存放在一个矩阵中。

x、y 与 z 这三个量的联系是：

$$\begin{cases} z(i,:)=f(x,y(i)) \\ z(:,j)=f(x(j),y) \end{cases}$$

为了便于 z 的计算，MATLAB 提供了函数 meshgrid()来处理 x 和 y。

meshgrid()的具体调用形式为：

```
[X,Y]= meshgrid (x,y)
```

它将 x 向量作为 X 的一个行向量，并复制该行，最后得一各行相同的方阵 X，将 y 作为 Y 的一个列向量，并复制该列，最后得一个各列相同的 Y 矩阵。将 X、Y 代入 f（x，y）便可求解出 z 矩阵，再用下面的命令格式作图：

mesh(X, Y, Z)：为数据点绘制网格线，X、Y、Z 分别为三维空间的坐标位置；

contour(X, Y, Z)：二维等高线，X、Y、Z 分别为三维空间的坐标位置；

contour3(X, Y, Z, n)：三维等高线，X、Y、Z 分别为三维空间的坐标位置，n 是等高线的条数。

下面通过实例讲述上述命令的应用。

【例 1-12】 三维曲面图形绘制实例。作出 $z=x^2+y^2$，其中 $x\in[-2,2]$ $y\in[-2,2]$。

解： 编写 MATLAB 文件 ex0104.m。

```
function ex0104()
x = -2:1:2; y = -2:1:2;
[X, Y] = meshgrid(x, y)                    %各行都等于 x 的 X 矩阵
                                           %和各列都等于 y 的 Y 矩阵
Z = X.^2 + Y.^2;                           %求解 Z 矩阵
subplot(2,2,1), mesh(X,Y,Z), grid on
title('网格图 a');
subplot(2,2,2), mesh(X,Y,Z),view([0,30]), grid on
title('在方位角 0° 和视角 30° 处观察图 a');
subplot(2,2,3), contour(X,Y,Z)
title('二维等高线');
subplot(2,2,4), contour3(X,Y,Z,10)
title('三维等高线');
```

运行该函数，输出结果如图 1-7 所示。

而且在 MATLAB 窗口中输出如下结果：

```
X =
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
```

```
     -2     -1      0      1      2
     -2     -1      0      1      2
Y =
     -2     -2     -2     -2     -2
     -1     -1     -1     -1     -1
      0      0      0      0      0
      1      1      1      1      1
      2      2      2      2      2
```

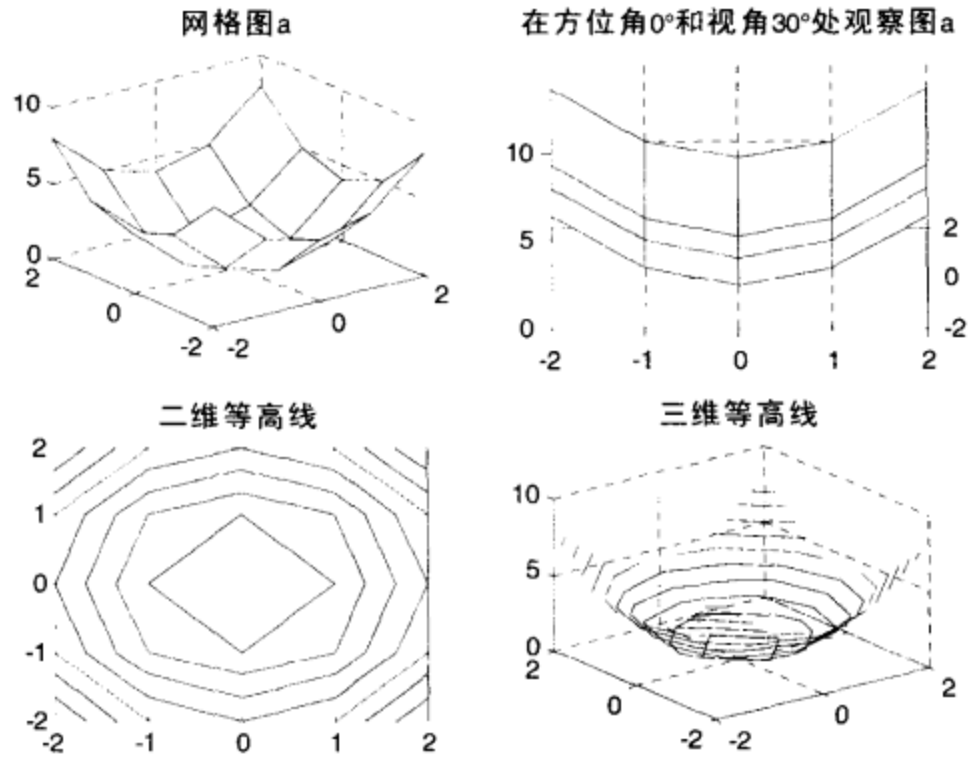

图 1-7 网格线和等高线

1.2.5 基本数学函数

MATLAB 集成了几乎所有的科学研究和工程计算要用的算法，提供了大量的数学函数。下面讲述 MATLAB 的基本数学函数。

1. MATLAB 中的内建数学函数

MATLAB 提供许多内建数学函数，如三角函数、对数函数等，可以方便用户直接调用进行计算。调用格式为：

```
r=MathFun(x)
```

其中 r 为函数计算结果，MathFun 为函数名，x 为输入值。

一个函数可以被当做另一个函数的引数，例如：X=sin(asin(x))。

表 1-6、表 1-7 和表 1-8 给出了 MATLAB 中的常见的函数。

表 1-6 常见三角函数

函数符号	名 称	函数符号	名 称
sin(x)	正弦函数	asin(x)	反正弦函数
cos(x)	余弦函数	acos(x)	反余弦函数
tan(x)	正切函数	atan(x)	反正切函数

续表

函数符号	名　称	函数符号	名　称
cot(*x*)	余切函数	acot(*x*)	反余切函数
sec(*x*)	正割函数	asec(*x*)	反正割函数
csc(*x*)	余割函数	acsc(*x*)	反余割函数
sinh(*x*)	双曲正弦函数	coth(*x*)	双曲余切函数
cosh(*x*)	双曲余弦函数	cech(*x*)	双曲正割函数
tanh(*x*)	双曲正切函数	ctanh(*x*)	双曲余割函数

表 1-7　常见基本数学函数

函数符号	功能描述	函数符号	功能描述
abs(*x*)	绝对值	sum(*x*)	求和
max(*x*)	最大值	min(*x*)	最小值
sqrt(*x*)	开平方	exp(*x*)	以 e 为底的指数
log(*x*)	自然对数	log10(*x*)	以 10 为底的对数
sign(*x*)	符号函数	mod(*x*,*y*)	两整数相除的余数
conj(*x*)	求复数的共轭	imag(*x*)	取复数的虚部
real(*x*)	取复数的实部		

表 1-8　常见取整函数

函数符号	功能描述
round(*x*)	四舍五入到 *x* 最近的整数
fix(*x*)	与 0 最接近的整数，即对正数取小于它的最大整数，对负数取大于它的最大整数
ceil(*x*)	比 *x* 大的最小的整数
floor(*x*)	比 *x* 小的最大整数

在数学运算和编写程序中还涉及若干具有特殊意义的变量，MATLAB 按照数学上的习惯定义这些变量，用户可以直接使用它们，而无需重新定义，具体说明见表 1-9。

表 1-9　MATLAB 中的特殊变量及其含义

符　号	含　义	符　号	含　义
i, j	$\sqrt{-1}$	pi	π
eps	计算机的最小正数	realmax realmin	最大/最小正数
break	退出循环	Inf inf	无穷大
end	循环结束语句	NaN	非数
nargin	输入参数个数	nargout	输出参数个数
varargin	变长度输入宗量	varargout	变长度输出宗量

2. 函数的构造和求解

MATLAB 有多种定义函数的方法，常用的方法有 3 种，下面分别对其进行介绍：

（1）用 function 构造函数

此种方法构造函数，需以.m 文件保存，函数名与文件名应一致，用户可以像调用

MATLAB 自带的函数一样调用该函数完成特定功能。

首先要通过 File 菜单下新建一个 M-file，然后在 M-file 里编辑函数即可。构造函数的格式为：

```
[输出实参表]=函数名(输入实参数)
注释部分
函数体语句
return 语句(可以有可以没有)
```

【例 1-13】 使用 function 构造函数及求解实例。利用 function 构造函数，创建并保存函数 $z=\frac{x+y}{2}$，求解其分别在点[1,4]、[2,5]和[3,6]处的函数值。

解：编写函数文件 ex0105.m：

```
function z=ex0105(x, y)
z=(x+y)/2;
```

求解函数在给定点处的值，在 MATLAB 命令窗口中输入：

```
>> a=[1 2 3];
>> b=[4 5 6];
>> ex0105(a, b)
```

输出结果为：

```
ans =
2.5000    3.5000    4.5000
```

也可以用 feval 求解，如下所示。

在 MATLAB 命令窗口中输入：

```
>> feval('ex0105', a, b)
```

输出结果为：

```
ans =
2.5000    3.5000    4.5000
```

（2）用 inline 构造函数

可以在命令窗中直接用 inline()函数定义函数。

【例 1-14】 使用 inline 构造函数及求解实例。利用 inline 构造函数，使用函数 inline()，创建并保存函数 $z=\frac{x+y}{2}$，求解其分别在点[1,4]、[2,5]和[3,6]处的函数值。

解：紧接上例，在 MATLAB 命令窗口输入以下语句，创建函数 myfun，

```
>> myfun=inline('(x+y)/2', 'x', 'y');
```

直接输入函数名及参数，求解其在给定点的值，在 MATLAB 命令窗口输入：

```
>> myfun(a, b)
```

输出结果为：

```
ans =
    2.5000    3.5000    4.5000
```

用 feval 求解，在 MATLAB 命令窗口输入：

```
>> feval(myfun, a, b)
```

输出结果为：

```
ans =
2.5000    3.5000    4.5000
```

（3）用 syms 构造符号函数

符号函数通常用 syms 或 sym 进行构造。

【例 1-15】 使用 syms 构造符号函数及求解实例。利用 syms 构造符号函数（$x+y$）/2 并求解。

解：在 MATLAB 命令窗口输入：

```
>> syms x y;
>> myfun=(x+y)/2
```

输出结果为：

```
myfun =
1/2*x+1/2*y
```

syms 构造的符号函数只能用 subs 进行求解，具体过程如下。

继续在 MATLAB 命令窗口输入：

```
>> subs(myfun,{x,y},{a,b})
```

输出结果为：

```
ans =
    2.5000    3.5000    4.5000
```

3. 多项式的求解和算术运算

MATLAB 中求解多项式的函数为 polyval()，其调用格式为：

```
y=polyval(a,x0)
```

其中 a 为多项式的按降序排列的系数向量，y 对应于 x0 处的函数值。

【例 1-16】 多项式的表示和计算实例。创建多项式 x^4+3x^2-x+2，并计算 $x=1$ 时的值。

解：在 MATLAB 命令窗口输入：

```
>> a=[1 0 3 -1 2];  %多项式为 x^4+3*x^2-x+2
>> polyval(a, 1)    %得到多项式在 x=1 处的值
```

输出结果为：

```
ans =
     5
```

因为多项式的乘法可表示为：

$$(a_N x^N + \ldots + a_1 x + a_0)(b_M x^M + \ldots + b_1 x + b_0) = c_{N+M} x^{N+M} + \ldots + c_1 x + c_0$$

其中，

$$c_k = \sum_{m=\max(0,k-M,k-N)}^{\min(k,N,M)} a_{k-m}b_m \text{，}(k=0, 1, 2,\ldots, N+M)$$

故两个多项式的乘法可以通过处理两者的系数向量得到。MATLAB 提供的多项式卷积函数 conv 也可进行多项式乘法运算，它的调用格式为：

```
conv(A,B)
```

其中，*A*、*B* 分别为多项式的系数行向量。

【例 1-17】 多项式乘法实例。计算多项式 x^4+3x^2-x+2 与 $6x^3+5x^2+4x+1$ 的乘积。

解：在 MATLAB 命令窗口输入：

```
>> a=[1 0 3 -1 2];
>> b=[6 5 4 1];
>> conv(a, b)
```

输出结果为：

```
ans =
     6     5    22    10    19     9     7     2
```

1.2.6 向量和矩阵的基本运算

向量和矩阵的运算是 MATLAB 的基础，向量又可以看做特殊的矩阵，本小节主要介绍向量和矩阵的基本运算。

1. 向量的基本运算

（1）向量创建

生成向量的最直接的方法就是在命令窗口中直接输入。

格式的要求是：

向量元素需要用方括号“[]”括起来，元素之间可以用空格、逗号或分号分隔。

其中，空格和逗号分隔生成行向量，用分号分隔生成列向量。

【例 1-18】 向量创建实例。创建一维向量 *x*=[1,2,3]。

解：在 MATLAB 命令窗口中输入：

```
>> x=[1 2 3]
```

输出结果为：

```
x=
     1     2     3
```

上面的操作创建了一个行向量。为得到一个列向量，可以使用转置，或者直接创建。

继续在 MATLAB 命令窗口中输入：

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

输出结果为：

```
xT=
     1
     2
     3
```

（2）向量运算中的常用函数

表 1-10 给出了向量运算中的常用函数。

表 1-10　向量运算中常用函数

函 数 名	调用形式	函数意义
linspace	*x*=linspace(*a*,*b*,*n*)	产生线性分布在 *a*、*b* 之间且包含 *a*、*b* 的长度为 *n* 的行向量
zeros	*x*=zeros(*n*,*m*)	产生 *n* 行 *m* 列的零矩阵
length	len=length(*x*)	返回向量的长度或矩阵的最大维数
size	size(*x*)	返回向量的行数和列数
+	*x*+*y*	同维向量相加
*	c**x*	向量和标量相乘
dot	dot(*x*, *y*)	向量点积
cross	cross(*x*, *y*)	向量叉积

2. 矩阵的基本运算

MATLAB 中，基本的数据单元不是整数、浮点数等，而是没有维数限制的矩阵。这是 MATLAB 作为一种矩阵运算语言的主要特点。向量可看做 1*n 或 n*1 的矩阵，因此，本部分讨论的所有对矩阵的操作同样适用于向量。

（1）矩阵的标识

所谓矩阵的标识指的是对矩阵中特定元素的提取，其主要操作形式为：

```
A(vr,vc)
```

其中 *vr* 为行号，*vc* 为列号，如有一个冒号“:”表示取全部行或列。

以下举例说明：

A(1, 3)　　表示取第 1 行，第 3 列元素；

A(1:3, [3 5 7])　　表示取第 1、2、3 行和第 3、5、7 列位置上的元素；

A(:, 3)　　表示取全部行的第 3 列元素。

（2）矩阵的创建

矩阵的创建方法主要有直接法和由其他矩阵构造这两种方法。

1）直接创建

直接创建就是直接输入矩阵元素。具体符号的意义与向量的创建相同。下面举例进行说明。

【例 1-19】　直接创建矩阵实例。直接创建矩阵 $\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$。

解：在 MATLAB 命令窗口中输入：

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

输出结果为：

```
A =
     1     2     3
     4     5     6
```

另外，MATLAB 提供了构造特殊矩阵的函数，包括 ones，zeros，rand 等。具体说明见表 1-11。

表 1-11 特殊矩阵的创建

ones(*n*)	创建 *n* 阶矩阵
ones(*m*,*n*)	创建 *m* 行 *n* 列的矩阵
ones(*m*, *n*, *p*)	创建 *m* 行 *n* 列 *p* 页的三维矩阵
eye(*n*)	创建 *n* 阶单位矩阵
zeros(*m*,*n*,*p*)	创建全为零的矩阵
rand(*m*,*n*,*p*)	创建均匀分布的矩阵.
randn(*m*,*n*,*p*)	创建正态分布的矩阵
pascal(*n*)	创建 *n* 阶帕斯卡矩阵
magic(*n*)	创建 *n* 阶魔方矩阵

2）由其他矩阵构造

由其他矩阵创建指的是通过矩阵的一些特殊运算来得到所要求的矩阵。【例 1-20】给出了 4 种此类方法。

【例 1-20】 矩阵基本运算实例。由已知矩阵通过取转置、取其中的几行或几列、两矩阵合并，以及取对角方法构造新矩阵。

解： 在 MATLAB 命令窗口中输入：

```
>> A=[1 2 3 ;4 5 6];
>> B=A'                                    %取转置
B =
     1     4
     2     5
     3     6
>> A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16];
>> B=A(1:3, 2:4)                           %A 的 1 到 3 行和 2 到 4 列构成 B
B =
     2     3     4
     6     7     8
    10    11    12

>> A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16];
>> B=A(1:3, 2:4)
B =
     2     3     4
     6     7     8
    10    11    12
>> C=A(1:3, 1:3)
C =
     1     2     3
     5     6     7
     9    10    11
```

```
>>D=[B, C]                                    %由两个以上的矩阵合并构成新矩阵.按列扩
                                             %展，行数不变，列数增加
D =
     2     3     4     1     2     3
     6     7     8     5     6     7
    10    11    12     9    10    11
>> D=[B;C]                                    %按行扩展，列数不变，行数增加
D =
     2     3     4
     6     7     8
    10    11    12
     1     2     3
     5     6     7
     9    10    11

>> diag(D)                                    %由矩阵的对角元素构成列向量
ans =
     2
     7
    12
```

另外，矩阵中的数不仅可以是整数、浮点数，还可以是复数。

3. 矩阵运算和矩阵操作

常用的矩阵运算主要包括矩阵的加、减、乘、除以及旋转翻折等，主要操作符号表达式及其含义如表 1-12 所示。

表 1-12　矩阵运算主要操作符号表达式及其含义

操 作 符	含　义	操 作 符	含　义
A+B	矩阵相加	***A - B***	矩阵相减
A * B	矩阵乘法	***A.* B***	矩阵点乘
A / B	矩阵右除	***A./ B***	矩阵右点除
A \ B	矩阵左除	***A \. B***	矩阵左点除
A ^n	矩阵乘方	***A.^n***	矩阵元素乘方
rot90(***A***)	矩阵旋转 90°	fliplr(***A***)	矩阵左右翻折
flipud(***A***)	矩阵上下翻折	reshape(***A***)	保持元素不变改变矩阵的维数

此外，MATLAB 还提供了一些常用的矩阵操作函数，如表 1-13 所示。

表 1-13　矩阵操作函数及其动能

norm(***A***)	向量或矩阵的范数	rank(***A***)	矩阵的秩
det(***A***)	矩阵的行列式	trace(***A***)	矩阵的迹
inv(***A***)	方阵的逆矩阵	eig(***A***)	特征值及特征向量
size(***A***)	矩阵的阶数	cond(***A***)	矩阵的条件数
lu(***A***)	矩阵的 LU 分解	qr(***A***)	矩阵的 QR 分解

下面还将列出矩阵运算的几点注意事项：

（1）两个矩阵组合成一个新矩阵时，若按照列扩展，则需要两者具有同样的行数，同

样，按照行扩展时，需两者具有同样的列数；

（2）只能对非奇异方阵作矩阵求逆运算；

（3）MATLAB 函数 pinv(*A*)可用来求矩阵 A 的左/右伪逆，返回一个与 A^T 同维的矩阵 X，且满足：$AXA = A, XAX = X$；

（4）采用 pinv(*A*)作用于非满秩矩阵时，其结果不是 A 的左/右逆。

1.3 MATLAB 编程的技巧

MATLAB 是一种科学计算语言，但同时也具有和 C、FORTRAN 等高级语言相类似的语言特征，能方便地实现程序控制。利用 MATLAB 的程序控制功能，可以将有关 MATLAB 命令编成程序存储在一个文件中（M 文件），然后运行该文件，MATLAB 就会自动依次执行文件中的命令，直到全部命令执行完毕。

编程时，首先要考虑到变量初值或者变量类型改变时，程序的应对能力，即程序的鲁棒性，因此出色的程序要具有较好的例外处理机制。此外，还要考虑程序的执行效率。

一些编程技巧能提高程序的执行效率，因此，掌握一些 MATLAB 的编程技巧也是非常必要的，下面对常用的 MATLAB 编程技巧进行介绍。

1.3.1 嵌套计算

一个程序的执行速度取决于它所调用的子程序个数以及采用的算法。通常希望子程序越少越好，算法效率越高越好。

嵌套计算是一种具有较小时间复杂度的算法，如【例 1-21】。

【例 1-21】 嵌套计算与直接求值的比较实例。考虑下面两个多项式，其中式（**）为式（*）的嵌套表达式：

$$p(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad (*)$$

$$p1(x) = ((a_3x + a_2)x + a_1)x + a_0 \qquad (**)$$

易知式（*）和式（**）所需要的乘法数分别为：6 次和 3 次。显然后者具有更高的执行效率。

下面用具体程序进行说明，创建函数 ex0109()。

```
function ex0109()
N = 100000;
a = [1:N];
x = 1;
tic                                    %初始化时钟
p1 = sum(a.*x.^[N-1:-1:0]);            %按照(*)求值
p1,  toc                               %时钟停止，获得执行时间
tic,  p2=a(1);
for i = 2:N                            %嵌套计算(**)
    p2 = p2*x + a(i);
end
```

```
p2, toc
tic, p3 = polyval(a, x), toc          %用 MATLAB 自带函数求值
```

运行函数，输出结果为：

```
p1 =
  5.0001e+009
Elapsed time is 0.035852 seconds.
p2 =
  5.0001e+009
Elapsed time is 0.005543 seconds.
p3 =
  5.0001e+009
Elapsed time is 0.088367 seconds.
```

可见，嵌套算法耗时最少。而用 polyval 求值执行速度最慢。

采用嵌套算法不仅能够提高执行效率，而且此方法具有更强的解决问题的能力，下面的实例可以说明这个问题。

【例 1-22】 嵌套计算与非嵌套计算的比较实例。求 possion 分布的有限项和，形如：

$$S(M)=\sum_{n=0}^{M}\frac{\lambda^n}{n!}e^{-\lambda}$$

由概率论知识可知，当 M 很大时，上式的值趋向于 1。

分别用式（*）和式（**）来求解，创建函数 ex0110()。

```
function ex0110()
r = 80;
M = 160;
p = exp(-r);
S1 = 0;
for k = 1:M
    p=p*r/k;                          %嵌套,式(**)
    S1=S1+p;
end
S1
S2= 0;
for k = 1:M
    p = r^k/factorial(k);             %非嵌套,式(*)
    S2 = S2 + p;
end
S2*exp(-r);
S2
```

运行函数，输出结果为：

```
S1 =
    1.0000
S2 =
  5.5406e+034
```

由结果可知，嵌套方法的结果非常接近真实值，而非嵌套的方法根本无法得到正确的结果。

1.3.2 循环结构

循环结构可按照给定的条件，重复执行指定的语句。MATLAB 用于实现循环结构的语句有 for 语句和 while 语句。

1. for 语句

for 语句的格式为：

```
for 循环变量=表达式 1:表达式 2:表达式 3
循环体语句
end
```

其中“表达式 1”的值为循环变量的初值，“表达式 2”的值为步长，“表达式 3”的值为循环变量的终值。步长为 1 时，“表达式 2”可以省略。

for 循环执行固定次数的循环，当循环变量超过终值时，循环结束。

【例 1-23】 for 循环语句使用实例。设计百分制分数分类器，输出为 1 表示及格，0 表示不及格，其余值表示分数不合规则。

解：编写 MATLAB 程序文件，创建函数 ex0106，如下所示：

```
function ex0106()
a= [1 50 80 30 45 90 100];
af=zeros(1, length(a));                    %预先分配空间
for n = 1:length(a)
    if(a(n)>100)                           %大于 100 分
        af(n) = 2;
       fprintf('\n\a bad data??\n');
    break;
    elseif a(n) >= 60,  af(n) = 1;         %小于 100 分，大于 60 分
    elseif a(n) >= 0,  af(n) = 0;          %小于 60 分，大于 0 分
    else                                   %小于 0 分
       af(n) = -1;
       fprintf('\n\a bad data??\n');
    break;
end
end
af
```

运行函数，输出结果为：

```
af =
     0     0     1     0     0     1     1
```

注意

for 循环不能通过在循环内重新对循环变量赋值来终止，【例 1-24】用来说明这个问题。

【例 1-24】 for 循环的循环变量赋值使用实例。找出矩阵 a 中的最小元素，并计算循环次数。

解：创建函数 ex0107，如下所示：

```
function ex0107()
a=[9 2 4 3 5 6 8 0];
min=inf;
time=0;
for n=1:8
    if a(n)<min
       min=a(n);
    end
    n=11;                          %改变循环变量不影响循环执行次数
    time=time+1;
end
time
min
```

运行函数，输出结果为：

```
time =
     8
min =
     0
```

由此结果可知，矩阵 a 中的最小元素为 0，循环次数为 8，在循环中对循环变量赋值并不改变循环次数。

2. while 语句

while 语句的格式为：

```
while （条件）
循环体语句
end
```

while 循环执行的次数不固定，只要循环条件满足，循环会一直进行下去。因此在使用 while 循环时，要特别注意避免出现死循环。

【例 1-25】 while 循环语句使用实例。用 while 循环找出矩阵 ***a*** 中的最小元素。

解：创建函数 ex0108，如下所示：

```
function ex0108()
n = 1;
a=[9 2 4 3 5 6 8 0];
min=inf;
while  n <=8                       %循环结束条件
  if a(n)<min
     min=a(n);
  end
  n=n+1;                           %指向数组下一个位置
end
min
```

上述程序输出结果如下：

```
min =
     0
```

对于循环的使用需要注意以下几点：

（1）尽量避免使用循环。在 MATLAB 编程中，采用循环会降低程序的执行速度，应尽量避免使用，可以用其他方式，如向量运算等代替。

（2）为了得到最大的速度，在 for 循环被执行之前，应预先分配数组。否则，在 for 循环内每执行一次命令，对数组重新分配一次内存，这样会降低 MATLAB 的执行效率。

（3）优先考虑内联（inline）函数，矩阵运算应该尽量采用 MATLAB 7.0 的内联函数，因为内联函数是由更底层的编程语言 C 语言构造的，其执行速度显然快于使用循环的矩阵运算。

（4）应用 MEX 技术。尽管采用了很多措施，但执行速度仍然很慢，比如说耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 FORTRAN 语言。按照 MEX 技术要求的格式编写相应部分的程序，然后通过编译连接，形成在 MATLAB 7.0 中可以直接调用的动态链接库（DLL）文件，这样可以显著地加快运算速度。有关 MEX 技术及其应用的详细内容可参考相关书籍。

1.3.3 循环和嵌套

程序的嵌套是指程序执行过程中调用自身的过程，又称为递归调用。接下来，将比较在解决同一个问题时，循环和嵌套的性能差异。

【例 1-26】 循环和嵌套的性能比较实例。计算以下阶乘，

$$n!=n(n-1)(n-2)...2\cdot1$$

解： 用循环和嵌套两种方法分别编程实现。

采用嵌套的方法具有较好的可读性，然而当 n 值较大时，却可能因为不断嵌套，消耗太多的内存，导致运行出错。因此，嵌套算法的时间消耗会随着 n 值的增大急剧增加。

循环方法实现的程序为 ex0111(n)，其中利用了计时函数来计算阶乘的时间。

```
function ex0111(n)
% for 循环计算阶乘
m = 1;
tic                          %计时开始
for k = 2:n
    m = m*k;
end
toc                          %计时停止
```

利用该函数，计算 100 的阶乘，输出的计时结果为：

```
ex0111(100)
Elapsed time is 0.000014 seconds.
```

嵌套方法实现的程序为 ex0112(n)，其中也利用了计时函数来计算阶乘的时间，如下所示。

```
function m=ex0112(n)
%用嵌套方法计算阶乘
```

```
if n <= 1
    m = 1;
else
    m = n*ex0112(n-1);
end
```

利用该函数，计算 100 的阶乘，输出的计时结果为：

```
>> tic, ex0112(100), toc
Elapsed time is 0.034706 seconds.
```

可见，自身嵌套消耗较多的时间，且随着 n 的增大，其效率急剧下降。

1.3.4 例外处理机制

优秀的程序员能够指导用户如何使用他编写的程序，而且在用户使用不当时，能够给出错误提示信息，并引导用户正确使用函数。前面编写的一些程序都没有相应的错误处理机制，如函数 ex0111，其输入应大于零，当输入小于零时会得到错误的结果。【例 1-27】给出了这样的实例。

【例 1-27】 例外处理机制使用实例。用户输入错误的函数参数时的例外处理，给函数 ex0111 输入错误的参数，如下所示，

```
>> ex0111(-1)
Elapsed time is 0.000017 seconds.
m =
     1
```

显然错误的结果是由于用户不清楚函数参数的取值范围导致的。这种错误由于并不影响程序的运行，因而很难被发现。所以程序员在编程时，应当考虑到可能发生的此类错误，并给出处理错误的机制和错误提示信息。

对上例，如果用户输入了错误的参数，可以采用下面语句终止程序，并提示出错，

```
if n<0
    error('input must be positive, stopped');
end
```

此时，再执行上面的命令的结果如下：

```
>> ex0111(-1)
??? Error using ==> ex0111
input must be positive, stopped
```

这种错误多数都是由于越界造成的，尤其在使用矩阵时，要注意引用矩阵位置不要超过它的边界。

另外，如果用户输入的函数参量超过设定的最大个数，或者类型不合要求也会出现这种错误。

但一般而言，对于输入参量小于设定个数的情形，MATLAB 内置程序一般会对未赋值参量作默认处理。典型的例子是 plot(Y, X)函数，plot(Y)默认的 X 坐标是[0，1，2…]序列。

程序员在编写程序的时候也应当注意处理这种情况。采用 nargin 函数可以判断输入参量的个数，从而设定未被指定的输入参数的值或者直接报错。

【例 1-28】 nargin 函数应用实例。利用 nargin 函数，实现两多项式的相加，并具有一定的报错功能。

解： 编写函数 ex0113。

```
function p=ex0113(a, b)
%求多项式 a，b 的和 p
if nargin==1                    %输入参数个数为 1，另一个默认参数为全 0 向量
   b=zeros(4, 1);
elseif nargin==0
   error('empty input');        %输入参数个数为 0，报错
end
a=a(:).' ;b=b(:).' ;
na=length(a) ; nb=length(b) ;
p=[zeros(1, nb-na) a]+[zeros(1, na-nb) b];  %多项式相加
```

在 MATLAB 命令窗口中输入正确的参数，调用该函数，输出正确的结果，如下所示：

```
>> a=[1 2 3 4];
>> ex0113(a)
ans =
    1    2    3    4
```

在 MATLAB 命令窗口中输入错误的参数，调用该函数，输出报错结果，如下所示：

```
>> ex0113()
??? Error using ==> ex0113
empty input
```

可见，用户输入参数不合要求时，会作默认处理或报错，使得程序具有更强的适应性。

1.3.5 全局变量的使用

全局变量是指在不同的工作空间以及基本的工作空间中可以共享的变量。用户只需在主程序或者任何子程序中声明一个或多个全局变量，则函数和主程序中都可以直接引用它们，采用如下格式生成全局变量。

```
global v1 v2…vn
```

表达式中各变量之间用空格隔开。

使用全局变量时要注意以下几点：

（1）它可以在主程序和函数之间不需要经过输入或输出变量直接传递数据。但要注意在函数调用中使用它们时，调用结束后全局变量在工作区中仍然存在；

（2）两个或多个函数也可以共有一个全局变量，只要同时在这些函数中用 global 语句加以定义即可；

（3）使用全局变量时必须十分小心，最好把全局变量名取得长一些或全部用大写，以免与函数中的局部变量重名。如果重名，容易出现致命错误。所以，使用全局变量不是一

个好的编程方法；

（4）一旦变量被声明为全局的，则在任何声明它的地方都可以对它进行修改。这在一定程度上破坏了子程序的独立性。如果全局变量被多个子程序修改，则用户很难知道全局变量的确切值，这使得程序的可读性大大下降。

下面用实例进行说明全局变量的用法。

【例 1-29】 全局变量使用实例。本例用以说明全局变量的声明及函数传递。

解：建立子程序 ex0114.m 和主程序 ex0115.m，同时在子程序 ex0114.m 以及主程序 ex0115.m 中定义全局变量 *D*，具体程序如下：

```
function x = ex0114(t, D)
global D                              %声明全局变量
t(find(t == 0)) = eps;                %防止分母出现 0 项
x = sin(pi*t/D)./(pi*t/D);
%主函数
function ex0115()
global D
D = 2;
b1 = -2;
b2 = 2;
t = b1 +[0:100]/100*(b2 - b1);
%通过全局变量传递参数
plot(t, ex0114(t))
```

本程度运行结果如图 1-8 所示。

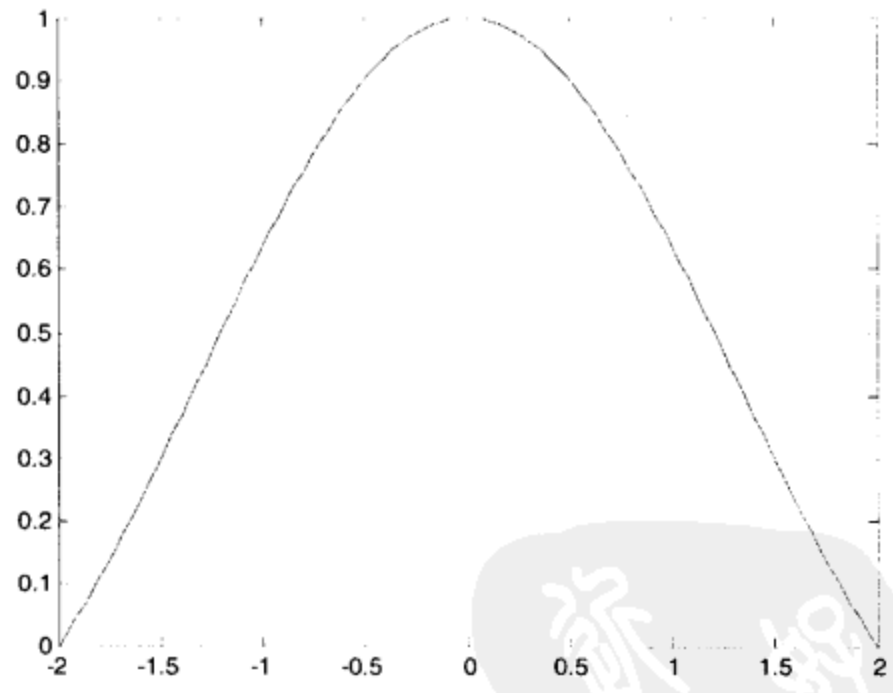

图 1-8 【例 1-29】输出结果

如果在子程序 ex0114 中修改全局变量的值，则变量声明时即对其进行赋值。例如将 ex0114 中的声明语句改为：global *D*=1，则可得图 1-9 所示结果。

显然，ex0114 中对全局变量的赋值覆盖了 ex0115 中的赋值，但这种修改在主程序中却不能得到反映。如果主程序和子程序不是同一个程序员编写，几乎很难找到结果与程序不符合的原因。不过，在使用全局变量时，MATLAB 会给出如下的警告信息，提示用户注意

全局变量是否在其他地方被修改：

Warning: The value of local variables may have been changed to match the globals. Future versions of MATLAB will require that you declare a variable to be global before you use that variable.

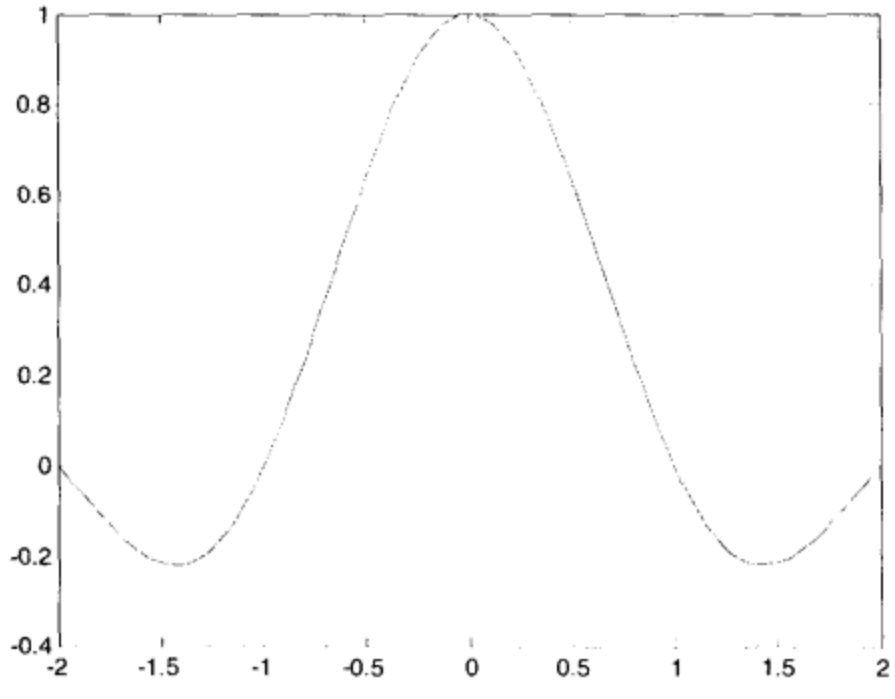

图 1-9 【例 1-29】中全局变量在子程序中被修改的输出结果

1.3.6 通过 varargin 传递参数

在编写函数时 varargin 只能作为函数的最后一个参量，主要传递函数中调用的子函数的可选项的参数，其大小也随着输入参量的变化而发生改变。

【例 1-30】 通过 varargin 传递参数的实例。该实例实现在不同输入参数时对函数 ex0114 的作图，其中通过 varargin 传递可选参数。

解：分别编写主程序 ex0116.m 和作图程序 ex0117.m。

```
function ex0116()
%主程序。通过 varargin 传递可选参数
D = 1; b1 = -2; b2 = 2;
t = b1+[0:100]/100*(b2 - b1);
bounds = [b1 b2];
subplot(1, 3, 1), ex0117('ex0114')          %作出 x 的函数图，bouds 为默认值[-1,1]
axis([b1 b2 -0.4 1.2])
subplot(1, 3, 2), ex0117('ex0114', bounds)  %输入两个参数，作[-2, 2]上 x 函数图
axis([b1 b2 -0.4 1.2])
subplot(1, 3, 3), ex0117('ex0114', bounds, D)   %可选项输入为 1
axis([b1 b2 -0.4 1.2])

function ex0117(ftn, bounds, varargin)
%子程序。varagin 为可选变量，输入时可以不考虑
if nargin < 2
    bounds = [-1 1];                            %bounds 默认为[-1,1]
end
```

```
b1 = bounds(1); b2 = bounds(2);
t = b1 + [0:100]/100*(b2 - b1);
x = feval(ftn, t, varargin{:});
plot(t, x)
```

运行主程序，输出结果如图 1-10 所示。

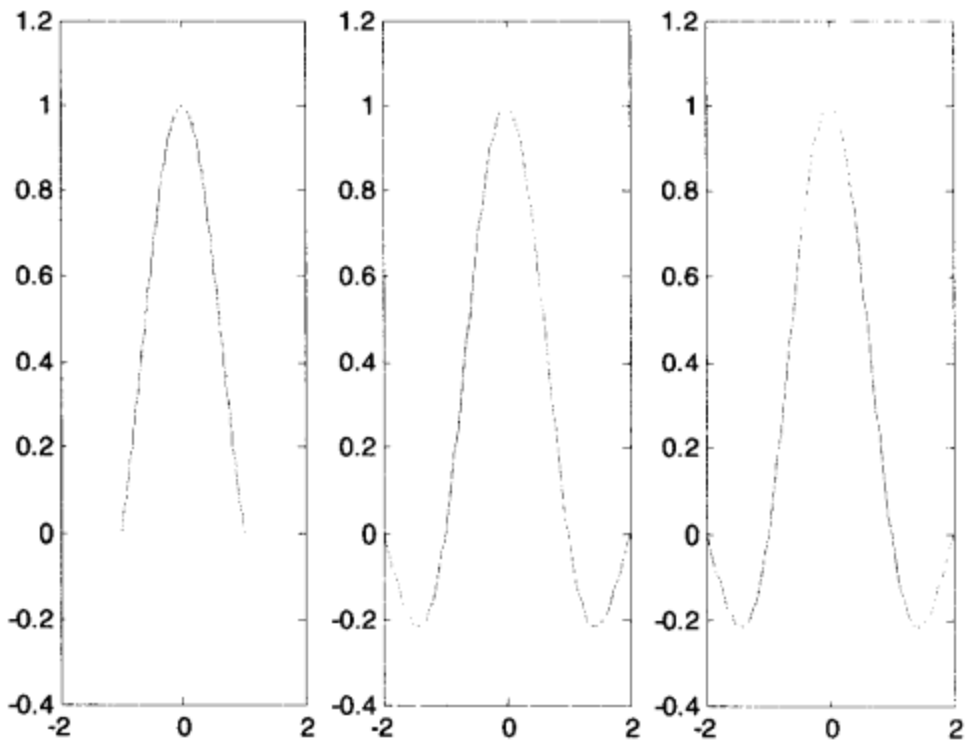

图 1-10　varargin 函数传递参数

对于一个具有多输入参量的程序，传统的方法是针对所有可能出现的输入情况进行处理。上例对应了 3 种输入参量情况，即输入参量个数分别为 1、2、3 的情况。普通处理方法是利用 if-elseif-else 结构对其进行处理。当变量很多时，该方法过于烦琐。

由于 varargin 本身可以根据输入参数个数的变化自动调节大小，因此，在调用具有多个可选参数的子程序的过程中使用时，可以大大简化程序。

1.4　小结

本章围绕数值计算介绍了 MATLAB 7 的基础知识，包括 MATLAB 的基本操作和编程技巧，这些都是后面内容的基础。

MATLAB 拥有众多的内置函数，学习 MATLAB 时，读者不要试图完全记住或者掌握它们，需要学会使用 help、lookfor 等命令查找所需的命令或函数。

在编写 MATLAB 程序，尤其是大型、复杂的 MATLAB 程序时，要多从用户角度考虑，力求让程序的例外处理机制完美，具有更好的可读性；但同时也要考虑算法的执行效率，找到这两方面的一个较好的平衡。

第 2 章

线性方程组求解

在自然科学和工程技术中，很多问题可以归结为求解线性方程组。采用 MATLAB，不仅可以利用其提供的相关函数直接解决一些简单的线性方程组，而且可以通过简洁的编程来解决一些复杂的线性方程组。

通过本章的介绍，读者既能应用 MATLAB 中相应的函数求解线性方程组，又能通过编程，灵活使用迭代法和其他的特殊解法来求解线性方程组。

2.1 求逆法

MATLAB 中求解线性方程最直接的方法是矩阵求逆法，它适用于系数矩阵的数据无规律且系数矩阵的阶数比较小的情况。如果系数矩阵的阶数太大的话，系数矩阵求逆就需要花费很长时间。

在 MATLAB 中用求逆法求解，线性方程可以直接用左除法“\”求解，也可以用求逆函数 inv()求解。

【例 2-1】 左除法和求逆法求解线性方程组实例。用左除法和求逆法求解如下线性方程组，

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 1 \\ -x_1 + 3x_2 + 7x_3 = 4 \\ 9x_1 + 3x_3 = 7 \end{cases}$$

解：用左除法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[1 2 3; -1 3 7; 9 0 3]
>> b=[1 4 7]';
>> x=A\b
```

输出计算结果为：

```
>> x =
    0.3333
   -1.6667
    1.3333
```

用逆矩阵法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[1 2 3; -1 3 7; 9 0 3];
>> b=[1 4 7]';
>> x=inv(A)*b
```

输出计算结果为：

```
>> x =
    0.3333
   -1.6667
    1.3333
```

由计算结果可知，方程组的解为 $[x_1, x_2, x_3] = [0.3333, -1.6667, 1.3333]$ 。

2.2 分解法

矩阵分解法是指根据一定的原理用某种算法将系数矩阵分解成若干个矩阵的代数运算，常用的分解是乘积分解，而分解后的新矩阵一般是某种特殊矩阵。常用的分解有 LU 分解、QR 分解、Cholesky 分解、Schur 分解、Hessenberg 分解和奇异分解等。

2.2.1 LU 分解法

矩阵的 LU 分解也叫三角分解，还有的书称之为 Doolittle 分解，它是将矩阵分解为一个单位下三角矩阵与上三角矩阵的乘积。只要矩阵非奇异，这种分解总是可以进行的。

MATLAB 提供了 lu 函数来求矩阵的 LU 分解：

[*L*,*U*]=lu(*A*)：产生一个上三角矩阵 *U* 和一个变换形式的下三角阵 *L*（进行了行交换），使得 *X*=*LU*；

[*L*,*U*,*P*]=lu(*A*)：产生一个上三角矩阵 *U* 和一个下三角阵 *L* 及置换矩阵 *P*，使得 *PA*=*LU*；分解以后，方程组 $Ax = b$ 的解可写为 *x*=*U*\(*L**b*)或 *x*=*U*\(*L**P***b*)。

【例 2-2】 LU 分解法求解线性方程组实例。用 LU 分解法求解以下线性方程组，

$$\begin{cases} 1.5x_1 + 3x_2 - 0.8x_3 + 4x_4 = 4 \\ 2x_1 + 9x_3 + 10x_4 = 0 \\ -7x_1 + 4.8x_2 - 0.6x_3 + x_4 = 1 \\ 14x_1 + 12.3x_2 - 4x_3 + 5x_4 = -2 \end{cases}$$ 。

解：用 LU 分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[1.5 3 -0.8 4; 2 0 9 10; -7 4.8 -0.6 1; 14 12.3 -4 5];
```

```
>> b=[4 0 1 -2]';
>> [L,U]=lu(A);
>>  x=U\(L\b)
```

输出计算结果为：

```
>> x =
    -0.3721
    -0.8291
    -1.5336
     1.4547
```

由计算结果可知，方程组的解为$[x_1,x_2,x_3,x_4]=[-0.3721,-0.8291,-1.5336,1.4547]$。

2.2.2 QR 分解法

矩阵的 QR 分解就是把矩阵分解为一个正交矩阵和一个上三角矩阵的乘积。MATLAB 中对矩阵进行 QR 分解的函数调用格式如下：

[*Q*,*R*]=qr(*X*)：产生一个正交矩阵 *Q* 和一个上三角阵 *R*，使得 *X*=*QR*；

[*Q*,*R*,*E*]=qr(*X*)：产生一个正交矩阵 *Q*、一个上三角阵 *R* 以及一个置换矩阵 *E*，使得 *XE*=*QR*；

实现分解以后，方程组的解可写为 *x*=*R*\(*Q**b*)或 *x*=*E*(*R*\(*Q**b*))。

【例 2-3】 QR 分解法求解线性方程组实例。用 QR 分解法求解以下线性方程组，

$$\begin{cases} x_1+0.5x_2+0.3333x_3+0.25x_4=1 \\ 0.5x_1+0.3333x_2+0.25x_3+0.2x_4=2 \\ 0.3333x_1+0.25x_2+0.2x_3+0.1667x_4=2 \\ 0.25x_1+0.2x_2+0.1667x_3+0.1429x_4=1 \end{cases}。$$

解：用 QR 分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[   1.0000    0.5000    0.3333    0.2500;
         0.5000    0.3333    0.2500    0.2000;
         0.3333    0.2500    0.2000    0.1667;
         0.2500    0.2000    0.1667    0.1429 ];
>> b=[1 2 2 1]';
>> [Q,R]=qr(A)
>> x=R\(Q\b)
```

输出 *Q*、*R* 的计算结果为：

```
Q =
   -0.8381    0.5226   -0.1540   -0.0263
   -0.4191   -0.4417    0.7278    0.3157
   -0.2794   -0.5288   -0.1395   -0.7892
   -0.2095   -0.5021   -0.6536    0.5261
R =
   -1.1932   -0.6705   -0.4749   -0.3698
         0    -0.1185   -0.1257   -0.1175
         0         0    -0.0062   -0.0096
         0         0         0    0.0002
```

输出解的计算结果为：

```
x =
  1.0e+003 *
    0.1160
   -1.4400
    3.6000
   -2.3800
```

由计算结果可知，方程组的解为$[x_1,x_2,x_3,x_4]=[116,-1440,3600,-2380]$。

2.2.3 Cholesky 分解法

当系数矩阵 $\boldsymbol{A}$ 正定且对称时，它能分解为以下的形式：$\boldsymbol{A}=\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}$。其中 $\boldsymbol{R}$ 为上三角矩阵，$\boldsymbol{R}^{\mathrm{T}}$ 为 $\boldsymbol{R}$ 的转置，是下三角矩阵，这种分解称为 Cholesky 分解。

MATLAB 中与 Cholesky 分解有关的函数如下：

（1）$\boldsymbol{R}$=chol($\boldsymbol{X}$)：对 $\boldsymbol{X}$ 进行 Cholesky 分解，使得 $\boldsymbol{X}=\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}$；

（2）[$\boldsymbol{R}$, p]=chol($\boldsymbol{X}$)：对 $\boldsymbol{X}$ 进行 Cholesky 分解，使得 $\boldsymbol{X}=\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}$。

如果 $\boldsymbol{X}$ 对称正定，则 p=0；否则，p 为一正整数。如果 $\boldsymbol{X}$ 满秩，$\boldsymbol{R}$ 是为 p–1 阶的上三角矩阵，且有 $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}=\boldsymbol{X}(1:(p-1),1:(p-1))$；实现分解后，方程组的解写成 $\boldsymbol{x}=\boldsymbol{R}\backslash(\boldsymbol{R}^{\mathrm{T}}\backslash\boldsymbol{b})$的形式。

【例 2-4】 Cholesky 分解法求解线性方程组实例。用 Cholesky 分解法求解以下线性方程组，

$$\begin{cases}9x_1-36x_2+30x_3=1\\-36x_1+192x_2-180x_3=1\\30x_1-180x_2+180x_3=1\end{cases}。$$

解：用 Cholesky 分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[9 -36 30;-36 192 -180;30 -180 180];
>> b=ones(3,1);
>> R=chol(A)
>> x=R\(R'\b)
```

输出 $\boldsymbol{R}$ 的计算结果为：

```
R =
    3.0000  -12.0000   10.0000
         0     6.9282   -8.6603
         0          0    2.2361
```

输出解的计算结果为：

```
x =
    1.8333
    1.0833
    0.7833
```

由计算结果可知，方程组的解为$[x_1,x_2,x_3]=[\ 1.8333,1.0833,0.7833]$。

说明

在 MATLAB 7 中，当 $\boldsymbol{A}$ 正定但不对称时，用 $\boldsymbol{R}$=chol($\boldsymbol{A}$)也能得出一个矩阵，

MATLAB 并不报错，但是只要验证一下 $\boldsymbol{R}^{\mathrm{T}}*\boldsymbol{R}$ 就会看出 $\boldsymbol{R}^{\mathrm{T}}*\boldsymbol{R}$ 并不等于 $\boldsymbol{A}$，因此用公式 $\boldsymbol{x}=\boldsymbol{R}\backslash(\boldsymbol{R}^{\mathrm{T}}\backslash\boldsymbol{b})$就会得出错误的结果。

2.2.4 其他分解法

MATLAB 中还提供了矩阵的奇异值分解、Hessenberg 分解和 Schur 分解等函数，可用来求解线性方程组，如表 2-1 所示。

表 2-1 其他的矩阵分解函数

函数及语法	说　明
[*U*,*S*,*V*] =svd(*A*)	奇异值分解，使得 $\boldsymbol{A}=\boldsymbol{U}*\boldsymbol{S}*\boldsymbol{V}^{\mathrm{T}}$
[*P*,*H*] =hess(*A*)	Hessenberg 分解，使得 $\boldsymbol{A}=\boldsymbol{P}*\boldsymbol{H}*\boldsymbol{P}^{\mathrm{T}}$，其中 $\boldsymbol{P}$ 为酉矩阵
[*U*,*T*] =schur(*A*)	Schur 分解，使得 $\boldsymbol{A}=\boldsymbol{U}*\boldsymbol{T}*\boldsymbol{U}^{\mathrm{T}}$，其中 $\boldsymbol{U}$ 为正交矩阵，$\boldsymbol{T}$ 为 Schur 矩阵

【例 2-5】 奇异值分解法求解线性方程组实例。用奇异值分解法求解以下线性方程组，

$$\begin{cases} x_1+0.5x_2+0.3333x_3+0.25x_4+0.2x_5=1 \\ 0.5x_1+0.3333x_2+0.25x_3+0.2x_4+0.1667x_5=0 \\ 0.3333x_1+0.25x_2+0.2x_3+0.1667x_4+0.1429x_5=1 \\ 0.25x_1+0.2x_2+0.1667x_3+0.1429x_4+0.125x_5=0 \\ 0.2x_1+0.1667x_2+0.1429x_3+0.125x_4+0.1111x_5=1 \end{cases}。$$

解：用奇异值分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>  A=[  1.0000    0.5000    0.3333    0.2500    0.2000;
         0.5000    0.3333    0.2500    0.2000    0.1667;
         0.3333    0.2500    0.2000    0.1667    0.1429;
         0.2500    0.2000    0.1667    0.1429    0.1250;
         0.2000    0.1667    0.1429    0.1250    0.1111];
>> [U,S,V]=svd(A)
>> b=[1 0 1 0 1]';
>> x=V*inv(S)*U'*b
```

输出分解的 $\boldsymbol{U}$、$\boldsymbol{S}$、$\boldsymbol{V}$ 的计算结果为：

```
U =
   -0.7679    0.6019   -0.2142    0.0472    0.0062
   -0.4458   -0.2759    0.7241   -0.4327   -0.1167
   -0.3216   -0.4249    0.1205    0.6674    0.5062
   -0.2534   -0.4439   -0.3096    0.2330   -0.7672
   -0.2098   -0.4290   -0.5652   -0.5576    0.3762
S =
    1.5671         0         0         0         0
         0    0.2085         0         0         0
         0         0    0.0114         0         0
         0         0         0    0.0003         0
         0         0         0         0    0.0000
V =
   -0.7679    0.6019   -0.2142    0.0472    0.0062
   -0.4458   -0.2759    0.7241   -0.4327   -0.1167
```

```
   -0.3216   -0.4249    0.1205    0.6674    0.5062
   -0.2534   -0.4439   -0.3096    0.2330   -0.7672
   -0.2098   -0.4290   -0.5652   -0.5576    0.3762
```

输出的计算结果为：

```
x =
    1.0e+005 *
    0.0170
   -0.3180
    1.3713
   -2.0720
    1.0143
```

由计算结果可知，方程组的解为，

$$[x_1,x_2,x_3,x_4,x_5]=10^5*[0.0170,-0.3180,1.3713,-2.0720,1.0143]。$$

说明

- 奇异值分解很有用，将系数矩阵进行奇异值分解以后，A 的奇异值都在 S 的对角线上，这样就可以粗略估计 A 的条件数，看看 A 是否是病态矩阵，以此决定相应的求解方法；
- 如果系数矩阵对称，那么奇异值分解后的 U 和 V 都是正交矩阵，而 S 是对角矩阵，这样求解方程就十分方便。

【例 2-6】 Hessenberg 分解法求解线性方程组实例。用 Hessenberg 分解法求解以下线性方程组，

$$\begin{cases} x_1+x_2+x_3+x_4=1 \\ x_1+2x_2+3x_3+4x_4=4 \\ x_1+3x_2+6x_3+10x_4=7 \\ x_1+4x_2+10x_3+20x_4=-2 \end{cases}。$$

解：用 Hessenberg 分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [  1     1     1     1;
          1     2     3     4;
          1     3     6    10;
          1     4    10    20;];
>> [P,H]=hess(A)
>> b=[1 4 7 -2]';
>> x=P*inv(H)*P'*b
```

输出分解的 P、H 的计算结果为：

```
P =
    0.7754   -0.6247   -0.0925         0
   -0.6092   -0.7015   -0.3698         0
    0.1662    0.3431   -0.9245         0
         0         0         0    1.0000
H =
    0.2147   -0.2654         0         0
   -0.2654    1.0844    1.0802         0
```

```
         0      1.0802      7.7009    -10.8167
         0           0    -10.8167     20.0000
```

输出的计算结果为：

```
x =
   10.0000
  -33.0000
   36.0000
  -12.0000
```

由计算结果可知，方程组的解为，$[x_1,x_2,x_3,x_4]=[10,-33,36,-12]$。

说明

- Hessenberg 矩阵指的是第一子对角线以下的元素为 0 的矩阵。
- Hessenberg 分解法分解后产生正交矩阵和 Hessenberg 矩阵，正交矩阵的逆就是其转置；如果系数矩阵对称的话，产生的 Hessenberg 矩阵就是三对角矩阵，这种矩阵的求逆速度也很快，因此用这种方法求解线性方程组也很快。

【例 2-7】 Schur 分解法求解线性方程组实例。Schur 分解法求解以下线性方程组，

$$\begin{cases} x_1+x_2+x_3+x_4=1 \\ x_1+2x_2+3x_3+4x_4=4 \\ x_1+3x_2+6x_3+10x_4=7 \\ x_1+4x_2+10x_3+20x_4=-2 \end{cases}。$$

解：用 Schur 分解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[     1     1     1     1;
           1     2     3     4;
           1     3     6    10;
           1     4    10    20;];
>> b=[1 4 7 -2]';
>>[U,T]=schur(A)
>>x=U*inv(T)*U'*b
```

输出分解的 ***U***、***T*** 的计算结果为：

```
U =
    0.3087   -0.7873    0.5304    0.0602
   -0.7231    0.1632    0.6403    0.2012
    0.5946    0.5321    0.3918    0.4581
   -0.1684   -0.2654   -0.3939    0.8638
T =
    0.0380         0         0         0
         0    0.4538         0         0
         0         0    2.2034         0
         0         0         0   26.3047
```

输出的计算结果为：

```
x =
    10.0000
   -33.0000
```

```
   36.0000
  -12.0000
```

由计算结果可知，方程组的解为$[x_1,x_2,x_3,x_4]=[10,-33,36,-12]$。

说明

- Schur 矩阵是上三角阵，且其对角元素为被分解矩阵的特征值。
- Schur 分解法在分解后产生正交矩阵和 Schur 矩阵，如果系数矩阵对称的话，产生的 Schur 矩阵就是对角阵，显然对角阵的逆矩阵非常容易得到，因此用这种方法求解线性方程组也很快。

2.3 迭代法

在实际应用中（例如在运筹学、图论等领域中），往往会出现这样一种情况：系数矩阵阶数很高（例如 10^5），但系数矩阵含零元素相对较多，如果用前面的几种分解法去求解的话，非零元素反而增多了，这就有点得不偿失了。

本节介绍另一类常用的求解方法——迭代法。迭代法是将求一组解转换为求一个近似解序列的过程，并用最终的近似解来逼近真实解。迭代法需要考虑以下 3 个重要的问题。

（1）迭代的初始值

考虑初始值的选取是否有范围限制，不同的初始值对最终的迭代结果是否有影响。

（2）迭代算法

考虑怎样由当前的迭代结果得出下次迭代的初始值；由于不同的算法会带来不同的误差，所以应考虑算法使误差放大还是减少。

（3）迭代的收敛性

考虑迭代是否收敛，收敛的过程是快还是慢。

以上问题的具体分析请参考数值分析的教材，下面讲述 MATLAB 中常见到的几种迭代法。

2.3.1 逐次逼近法

对于 n 阶线性方程组，$\boldsymbol{Ax=b}$ 的系数矩阵 $\boldsymbol{A}$（假设 $\boldsymbol{A}$ 为非奇异）进行如下分解

$$\boldsymbol{A=Q-C}$$

其中 $\boldsymbol{Q}$ 为非奇异。

则方程组可变换为

$$\boldsymbol{x=Bx+r}$$

其中 $\boldsymbol{B=Q^{-1}C}$，$\boldsymbol{r=Q^{-1}b}$，那么迭代过程可以写成

$$\boldsymbol{x_{k+1}=Bx_k+r}$$

这种迭代过程称为逐次逼近法。取不同的 $\boldsymbol{Q}$ 和 $\boldsymbol{C}$，就得到不同的迭代算法。通常情况下，逐次逼近法收敛的充分条件是迭代矩阵的谱半径小于 1。

2.3.2 里查森迭代法

里查森迭代法可以说是最简单的迭代法了，它采用的迭代公式如下：

$$\boldsymbol{x}_{k+1}=(\boldsymbol{I}-\boldsymbol{A})\boldsymbol{x}_k+\boldsymbol{b}$$

在 MATLAB 中编程实现的里查森迭代法函数为：richason。

功能：用里查森迭代法求线性方程组 $\boldsymbol{A}x=\boldsymbol{b}$ 的解。

调用格式：[x,n]=richason($\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{x}$0, eps, M)

其中，$\boldsymbol{A}$ 为线性方程组的系数矩阵；

$\boldsymbol{b}$ 为线性方程组中的常数向量；

$\boldsymbol{x}$0 为迭代初始向量；

eps 为解的精度控制（此参数可选）；

M 为迭代步数控制（此参数可选）；

$\boldsymbol{x}$ 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

理查森迭代法的 MATLAB 程序代码如下：

```
function [x,n]=richason(A,b,x0,eps,M)
%采用里查森迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if(nargin == 3)
    eps = 1.0e-6;                                    %eps 表示迭代精度
     M = 200;                                        %M 表示迭代步数的限制值
elseif(nargin == 4)
     M = 200;
end
I =eye(size(A));
x1=x0;
x=(I-A)*x0+b;
n=1;
%迭代过程
while(norm(x-x1)>eps)
    x1=x;
    x=(I-A)*x1+b;
    n = n + 1;                                       %n 为最终求出解时的迭代步数
    if(n>=M)
        disp('Warning:迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例 2-8】 理查森迭代法求解线性方程组实例。用理查森迭代法求解以下线性方程组，

$$\begin{cases} 1.0170x_1 - 0.0092x_2 - 0.0095x_3 = 1 \\ -0.0092x_1 + 0.9903x_2 + 0.0136x_3 = 0 \\ -0.0095x_1 + 0.0136x_2 + 0.9898x_3 = 1 \end{cases}。$$

解：用理查森迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A =[ 1.0170   -0.0092    0.0095;
       -0.0092    0.9903    0.0136;
        0.0095    0.0136    0.9898];
>> b=[1 0 1]';
>> x0 = [0 0 0];
>> [x,n]=richason(A,b,x0)
```

输出的计算结果为：

```
x =
    0.9738
   -0.0047
    1.0010
```

输出的迭代次数为：

```
n =
     5
```

可见，经过 5 步迭代，理查森迭代法求出了方程组的解为：

$$[x_1, x_2, x_3] = [0.9738, -0.0047, 1.0010]。$$

对上述迭代计算结果进行验证，在 MATLAB 命令窗口中输入如下程序：

```
>> A*x
```

输出结果为：

```
ans =
    1.0000
    0.0000
    1.0000
```

经过验证，也确定了解的正确性。一般情况下，理查森迭代法难以收敛，因为要使 $I-A$ 的谱半径小于 1，通常要求 A 是严格对角占优的矩阵。

2.3.3 Jacobi 迭代法

如果系数矩阵的主对角元全不为 0，在上节 A 的分解中取

$$Q=D$$

$$C=D-A$$

其中 D 是由 A 的主对角元素组成的对角阵，则有 $B=I-D^{-1}A$，$r=D^{-1}b$，迭代公式为：

$$x_{k+1}=(I-D^{-1}A)x_k+D^{-1}b$$

这种迭代方法称为 Jacobi 迭代法。

在 MATLAB 中编程实现的 Jacobi 迭代法函数为：jacobi。

功能：用 Jacobi 迭代法求线性方程组 $Ax=b$ 的解

调用格式：[x,n]= jacobi (A, b, x0, eps, varargin)

其中，A 为线性方程组的系数矩阵；

b 为线性方程组中的常数向量；

x0 为迭代初始向量；

eps 为解的精度控制（此参数可选）；

varargin 为迭代步数控制（此参数可选）；

x 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

Jacobi 迭代法的 MATLAB 程序代码如下：

```
function [x,n]=jacobi(A,b,x0,eps,varargin)
%采用 Jacobi 迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：varargin
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n

if nargin==3
    eps= 1.0e-6;
    M  = 200;
elseif nargin<3
    error
    return
elseif nargin ==5
    M  = varargin{1};
end

D=diag(diag(A));      %求 A 的对角矩阵
L=-tril(A,-1);        %求 A 的下三角阵
U=-triu(A,1);         %求 A 的上三角阵
B=D\(L+U);
f=D\b;
x=B*x0+f;
n=1;                  %迭代次数
%迭代过程
while norm(x-x0)>=eps
    x0=x;
    x =B*x0+f;
    n=n+1;
    if(n>=M)
        disp('Warning:迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例 2-9】 Jacobi 迭代法求解线性方程组实例。用 Jacobi 迭代法求解以下线性方程组，

$$\begin{cases} 0.9889x_1 - 0.0005x_2 - 0.0002x_3 = 1 \\ -0.0046x_1 + 0.9946x_2 + 0.0077x_3 = 0 \\ -0.0002x_1 + 0.0092x_2 + 0.9941x_3 = 1 \end{cases}。$$

解：用 Jacobi 迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>A = [ 0.9889   -0.0005   -0.0002;
        -0.0046    0.9946    0.0077;
        -0.0002    0.0092    0.9941];
>> b=[1 0 1]';
>> x0 = ones(3,1);
>> [x,n]=jacobi(A,b,x0)
```

输出的计算结果为：

```
x =
1.0114
-0.0031
1.0062
```

输出的迭代次数为：

```
n =
4
```

可见，经过 4 步迭代，Jacobi 法求出了方程组的解为：

$$[x_1, x_2, x_3] = [1.0114, -0.0031, 1.0062]。$$

说明

Jacobi 迭代法对任意初始向量 $x0$ 收敛的充要条件为 B 的谱半径小于 1，即 B 的所有特征值的绝对值都小于 1。

2.3.4 Gauss-Seidel 迭代法

仍然设系数矩阵 A 的主对角元全不为零，如果对 A 做以下分解：

$$A=(D-L)-U$$

其中 D 的意义同 Jacobi 迭代法，L 为下三角矩阵，U 为上三角矩阵，它的迭代公式为：

$$x_{k+1}=(D-L)^{-1}Ux_k+(D-\mathrm{L})^{-1}b$$

在 MATLAB 中编程实现的 Gauss-Seidel 迭代法函数为：gauseidel。

功能：用 Gauss-Seidel 迭代法求线性方程组 $Ax=b$ 的解。

调用格式：[x,n]= gauseidel (A, b, $x0$, eps, M)。

其中，A 为线性方程组的系数矩阵；

b 为线性方程组中的常数向量；

$x0$ 为迭代初始向量；

eps 为解的精度控制（此参数可选）；

M 为迭代步数控制（此参数可选）；

x 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

Gauss-Seidel 的 MATLAB 程序代码如下：

```
function [x,n]=gauseidel(A,b,x0,eps,M)
%采用 Gauss-Seidel 迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n

if nargin==3
    eps= 1.0e-6;
    M  = 200;
elseif nargin == 4
    M  = 200;
elseif nargin<3
    error
    return;
end
D=diag(diag(A));     %求 A 的对角矩阵
L=-tril(A,-1);       %求 A 的下三角阵
U=-triu(A,1);        %求 A 的上三角阵
G=(D-L)\U;
f=(D-L)\b;
x=G*x0+f;
n=1;                 %迭代次数
  %迭代过程
while norm(x-x0)>=eps
    x0=x;
    x=G*x0+f;
    n=n+1;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例 2-10】 Gauss-Seidel 迭代法求解线性方程组实例。用 Gauss-Seidel 迭代法求解以下线性方程组，

$$\begin{cases} 1.4449x_1 + 0.7948x_2 + 0.8801x_3 = 1 \\ 0.6946x_1 + 1.9568x_2 + 0.1730x_3 = 0 \\ 0.6213x_1 + 0.5226x_2 + 1.9797x_3 = 1 \end{cases}。$$

解： 用 Gauss-Seidel 迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [1.4449    0.7948    0.8801;
        0.6946    1.9568    0.1730;
```

```
          0.6213    0.5226    1.9797];
>> b=[1 0 1]';
>> x0=zeros(3,1);
>> [x,n]=gauseidel(A,b,x0)
```

输出的计算结果为：

```
x =
    0.5929
   -0.2443
    0.3835
```

输出的迭代次数为：

```
n =
    11
```

可见，经过 11 步迭代，Gauss-Seidel 迭代法求出方程组的解为：

$[x_1, x_2, x_3] = [0.5929, -0.2443, 0.3835]$。

2.3.5 超松弛迭代法

如果对 $\boldsymbol{A}$ 作以下分解：

$$\boldsymbol{A}=(\boldsymbol{D}-\omega\boldsymbol{L})-((1-\omega)\boldsymbol{D}+\omega\boldsymbol{U})$$

其中 $\boldsymbol{D}$、$\boldsymbol{L}$、$\boldsymbol{U}$ 的意义与 Gauss-Seidel 迭代法相同。

ω是一个事先选好的常数，称为松弛因子，当$\omega>1$ 时叫超松弛法，也叫 SOR 迭代法；当$\omega<1$ 时叫低松弛法。其迭代公式为：

$$\boldsymbol{x}_{k+1}=(\boldsymbol{D}-\omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D}+\omega\boldsymbol{U}]\boldsymbol{x}_k+\omega(\boldsymbol{D}-\omega\boldsymbol{L})^{-1}\boldsymbol{b}$$

关于超松弛迭代法的收敛性有如下结论：

若系数矩阵 $\boldsymbol{A}$ 对称正定，当 $0<\omega<2$，SOR 迭代法收敛。

在 MATLAB 中编程实现的超松弛迭代法函数为：SOR。

功能：用超松弛迭代法求线性方程组 $\boldsymbol{Ax=b}$ 的解。

调用格式：[x, n]=SOR ($\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{x}$0, w, eps, M)

其中，$\boldsymbol{A}$ 为线性方程组的系数矩阵；

$\boldsymbol{b}$ 为线性方程组中的常数向量；

$\boldsymbol{x}$0 为迭代初始向量；

w 为松弛因子；

eps 为解的精度控制（此参数可选）；

M 为迭代步数控制（此参数可选）；

$\boldsymbol{x}$ 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

超松弛迭代法的 MATLAB 程序代码如下：

```
function [x,n]=SOR(A,b,x0,w,eps,M)
%采用超松弛迭代法求线性方程组 Ax=b 的解
```

```
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n

if nargin==4
    eps= 1.0e-6;
    M  = 200;
elseif nargin<4
    error
    return
elseif nargin ==5
    M  = 200;
end
if(w<=0 || w>=2)      %收敛条件要求
    error;
    return;
end
D=diag(diag(A));     %求 A 的对角矩阵
L=-tril(A,-1);       %求 A 的下三角阵
U=-triu(A,1);        %求 A 的上三角阵
B=inv(D-L*w)*((1-w)*D+w*U);
f=w*inv((D-L*w))*b;
x=B*x0+f;
n=1;                 %迭代次数
  %迭代过程
while norm(x-x0)>=eps
    x0=x;
    x =B*x0+f;
    n=n+1;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例 2-11】 超松弛迭代法求解线性方程组实例。用超松弛迭代法求解以下线性方程组，

$$\begin{cases} 4x_1 + 3x_2 = 24 \\ 3x_1 + 4x_2 - x_3 = 30 \\ -x_2 + 4x_3 = -24 \end{cases}。$$

解：用超松弛迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>A=[ 4 3 0; 3 4 -1; 0 -1 4];
>> b=[ 24 30 -24]';
>> x0=[0 0 0]';
>> [x,n]=SOR(A,b,x0,1.25)
```

输出的计算结果为：

```
x =
```

```
    3.0000
    4.0000
   -5.0000
```

输出的迭代次数为：

```
n =
    14
```

可见，经过 14 步迭代，SOR 迭代法求出了方程组的解为 $[x_1, x_2, x_3]=[3,4,-5]$。

超松弛迭代法还有一种改进形式，叫做对称逐次超松弛迭代法(SSOR)，它采用的是两步迭代公式：

$$(\boldsymbol{D}-\omega\boldsymbol{L})\,\boldsymbol{x}_{k+1/2}=\omega(\boldsymbol{U}\boldsymbol{x}_k+\boldsymbol{b})+(1-\omega)\boldsymbol{D}\boldsymbol{x}_k$$

$$(\boldsymbol{D}-\omega\boldsymbol{U})\,\boldsymbol{x}_{k+1}=\omega(\boldsymbol{L}\boldsymbol{x}_k+\boldsymbol{b})+(1-\omega)\boldsymbol{D}\boldsymbol{x}_{k+1/2}$$

在 MATLAB 中编程实现的对称逐次超松弛迭代法函数为：SSOR。

功能：对称逐次超松弛迭代法求线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的解。

调用格式：[*x*,*n*]=S SOR (*A*, *b*, *x*0, *w*, eps, *M*)。

其中，*A* 为线性方程组的系数矩阵；

b 为线性方程组中的常数向量；

*x*0 为迭代初始向量；

w 为松弛因子；

eps 为解的精度控制（此参数可选）；

M 为迭代步数控制（此参数可选）；

x 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

SSOR 算法用 MATLAB 实现如下：

```
function [x,n]=SSOR(A,b,x0,w,eps,M)
%采用对称逐次超松弛迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n

if nargin==4
    eps= 1.0e-6;
    M  = 200;
elseif nargin<4
    error
    return
elseif nargin ==5
    M  = 200;
end
if(w<=0 || w>=2)
```

```
    error;
    return;
end
D=diag(diag(A));          %求 A 的对角矩阵
L=-tril(A,-1);            %求 A 的下三角阵
U=-triu(A,1);             %求 A 的上三角阵
B1=inv(D-L*w)*((1-w)*D+w*U);        %第一步的迭代矩阵
B2=inv(D-U*w)*((1-w)*D+w*L);        %第二步的迭代矩阵
f1=w*inv((D-L*w))*b;
f2=w*inv((D-U*w))*b;
x12=B1*x0+f1;                       %第一步的迭代得到的中间值
x  =B2*x12+f2;
n=1;                    %迭代次数
  %迭代过程
while norm(x-x0)>=eps
    x0=x;
    x12=B1*x0+f1;
    x  =B2*x12+f2;
    n=n+1;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例 2-12】 对称逐次超松弛迭代法求解线性方程组实例。用 SSOR 迭代法求解以下线性方程组，

$$\begin{cases} 4x_1 + 3x_2 = 24 \\ 3x_1 + 4x_2 - x_3 = 30 \\ -x_2 + 4x_3 = -24 \end{cases}。$$

解：用 SSOR 迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>A=[ 4 3 0; 3 4 -1; 0 -1 4];
>> b=[ 24 30 -24]';
>> x0=[0 0 0]';
>> [x,n]=SSOR(A,b,x0,1.25)
```

输出的计算结果为：

```
x =
    3.0000
    4.0000
   -5.0000
```

输出的迭代次数为：

```
n =
    34
```

可见，经过 34 步迭代，SSOR 迭代法求出方程组的解为 $[x_1, x_2, x_3] = [3,4,-5]$。

上面的两个例子采用的是同一个线性方程组，且采用的松弛系数都是 1.25，从求解所需的步骤来看，SOR 迭代法比 SSOR 迭代法快一些。

2.3.6 两步迭代法

两步迭代法的迭代公式如下：

$$(D-L)x_{k+1/2}=Ux_k+b$$

$$(D-U)x_{k+1}=Lx_{k+1/2}+b$$

其中 D、L、U 的定义和前面一样，即 D 为矩阵 A 分解的对角矩阵，L 为矩阵 A 分解的下三角阵，U 为矩阵 A 分解的上三角阵。

在 MATLAB 中编程实现的两步迭代法函数为：twostep。

功能：用两步迭代法求线性方程组 $Ax=b$ 的解。

调用格式：[x,n]= twostep (A, b, x0, eps, varargin)。

其中，A 为线性方程组的系数矩阵；

b 为线性方程组中的常数向量；

x0 为迭代初始向量；

eps 为解的精度控制（此参数可选）；

varargin 为迭代步数控制（此参数可选）；

x 为线性方程组的解；

n 为求出所需精度的解实际的迭代步数。

两步迭代法用 MATLAB 实现如下：

```
function [x,n]=twostep(A,b,x0,eps,varargin)
%采用两步迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：varargin
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n

if nargin==3
    eps= 1.0e-6;
    M  = 200;
elseif nargin<3
    error
    return
elseif nargin ==5
    M  = varargin{1};
end
D=diag(diag(A));     %求 A 的对角矩阵
L=-tril(A,-1);       %求 A 的下三角阵
U=-triu(A,1);        %求 A 的上三角阵
B1=(D-L)\U;
B2=(D-U)\L;
f1=(D-L)\b;
f2=(D-U)\b;
```

```
x12=B1*x0+f1;
x  =B2*x12+f2;
n=1;                    %迭代次数
  %迭代过程
while norm(x-x0)>=eps
    x0 =x;
    x12=B1*x0+f1;
    x  =B2*x12+f2;
    n=n+1;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

【例2-13】 两步迭代法求解线性方程组实例。采用两步迭代法求解以下线性方程组，

$$\begin{cases} 4x_1 + 3x_2 = 24 \\ 3x_1 + 4x_2 - x_3 = 30 \\ -x_2 + 4x_3 = -24。 \end{cases}$$

解：用两步迭代法求解，在MATLAB命令窗口中输入求解程序：

```
>> A=[ 4 3 0; 3 4 -1; 0 -1 4];
>> b=[ 24 30 -24]';
>> x0=[0 0 0]';
>> [x,n]=twostep(A,b,x0)
```

输出的计算结果为：

```
x =
    3.0000
    4.0000
   -5.0000
```

输出的迭代次数为：

```
n =
    30
```

可见，经过30步迭代，两步迭代法求出了方程组的解为$[x_1,x_2,x_3]=[3,4,-5]$。

从上面两个例子可以看出，与两步迭代法比，SSOR迭代法稍快一些。

2.3.7 梯度法

梯度法相对于前面的几种对系数矩阵的简单分解而构造的迭代法而言，它不须考虑收敛的问题，而且求解的速度也相对较快。对于系数矩阵对称正定的方程组，梯度法有最速下降法、共轭梯度法和预处理的共轭梯度法这三种常见的求法，下面分别进行讲述。

1. 最速下降法

最速下降法的基本思路是将求线性方程组：

$$Ax=b$$

的解转化为求二次泛函

$$\Phi(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} - 2\boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$$

的极小值问题。

通常的做法为先任意给定一个初始向量，然后确定一个搜索的方向和搜索步长，如此循环直到找到极小值。

最速下降法采取的搜索方向是当前点的负梯度方向，每次的搜索步长都使得泛函取极小值。

在 MATLAB 中编程实现的最速下降法函数为：fastdown。

功能：用最速下降法求线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的解。

调用格式：[$\boldsymbol{x}$,n]= fastdown ($\boldsymbol{A}$, b, $\boldsymbol{x}$0, eps)。

其中，$\boldsymbol{A}$ 为线性方程组的系数矩阵；

$\boldsymbol{b}$ 为线性方程组中的常数向量；

$\boldsymbol{x}$0 为初始向量；

eps 为解的精度控制（此参数可选）；

$\boldsymbol{x}$ 为线性方程组的解；

n 为求出所需精度的解实际的搜索步数。

最速下降法用 MATLAB 实现如下：

```
function [x,n]=fastdown(A,b,x0,eps)
%采用最速下降法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%初始向量：x0
%解的精度控制：eps
%线性方程组的解：x
%求出所需精度的解实际的搜索步数：n

if(nargin == 3)
    eps = 1.0e-6;
end
r  = b-A*x0;
d  = dot(r,r)/dot(A*r,r);
x  = x0+d*r;
n=1;
%迭代过程
while(norm(x-x0)>eps)
    x0 = x;
    r  = b-A*x0;
    d  = dot(r,r)/dot(A*r,r);
    x  = x0+d*r;
    n  = n + 1;
end
```

【例 2-14】 最速下降法求解线性方程组实例。采用最速下降法求解以下线性方程组，且分别取迭代初始值[1,1,1,1]和[2,10,-1,2]，分析迭代初值对迭代步数的影响。

$$\begin{cases} x_1+0.5x_2+0.3333x_3+0.25x_4=1 \\ 0.5x_1+0.3333x_2+0.25x_3+0.2x_4=0 \\ 0.3333x_1+0.25x_2+0.2x_3+0.1667x_4=1 \\ 0.25x_1+0.2x_2+0.1667x_3+0.1429x_4=0 \end{cases}$$

解：用最速下降法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A =[ 1.0000    0.5000    0.3333    0.2500;
        0.5000    0.3333    0.2500    0.2000;
        0.3333    0.2500    0.2000    0.1667;
        0.2500    0.2000    0.1667    0.1429;];
>> b=[ 0.5 1 2 0]';
>> x0=[1 1 1 1]';   %迭代初值
>> [x,n]=fastdown(A,b,x0)
```

输出的计算结果为：

```
x =
  1.0e+004 *
    0.0368
   -0.4260
    1.0380
   -0.6790
```

输出的迭代次数为：

```
n =
     105178
```

可见，当迭代初始值为[1,1,1,1]时，经过 105178 步迭代，最速下降法求出了方程组的解为$[x_1,x_2,x_3,x_4]$=[368, -4260, 10380, -6790]。

如果改变迭代的初始值，那么迭代步数会怎样变化呢？

在 MATLAB 命令窗口中接着输入：

```
>> x0=[2 10 -1 2]';  %新的迭代初值
>> [x,n]=fastdown(A,b,x0)
```

输出的计算结果为：

```
x =
  1.0e+004 *
    0.0368
   -0.4260
    1.0380
   -0.6790
```

输出的迭代次数为：

```
n =
       41379
```

可见，当迭代初始值为[2,10,-1,2]时，经过 41379 步迭代，最速下降法求出了方程组的解为$[x_1,x_2,x_3,x_4]$=[368,-4260,10380,-6790]。

从迭代结果可以看出，所需步数大大降低了，说明对最速下降法而言，初始值的选取

很重要，选得好的话，能大大降低求解所需的迭代步数。

2. 共轭梯度法

共轭梯度法是从整体来寻找最佳的搜索方向的。它的思路是第一步仍然取负梯度方向作为搜索方向，对于接下来的各步，在过当前点由负梯度向量和上一步的搜索向量组成的平面内寻找最佳的搜索方向，因此它比最速下降法快。

在 MATLAB 中编程实现的共轭梯度法函数为：conjgrad。

功能：共轭梯度法求线性方程组 $\boldsymbol{Ax=b}$ 的解。

调用格式：[$\boldsymbol{x}$,n]= conjgrad ($\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{x}$0)。

其中，$\boldsymbol{A}$ 为线性方程组的系数矩阵；

$\boldsymbol{b}$ 为线性方程组中的常数向量；

$\boldsymbol{x}$0 为初始向量；

$\boldsymbol{x}$ 为线性方程组的解；

n 为求出所需精度的解实际的搜索步数。

共轭梯度法用 MATLAB 实现如下：

```
function [x,n]=conjgrad(A,b,x0)
%采用共轭梯度法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%初始向量：x0
%线性方程组的解：x
%求出所需精度的解实际的搜索步数：n

if(nargin == 3)
    eps = 1.0e-6;
end
r1  = b-A*x0;
p1  = r1;
d   = dot(r1,r1)/dot(p1,A*p1);
x   = x0+d*p1;
r2  = r1-d*A*p1;
f   = dot(r2,r2)/dot(r1,r1);
p2  = r2+f*p1;
n   = 1;
for(i=1:(rank(A)-1))
    x0 = x;
    p1 = p2;
    r1 = r2;
    d  = dot(r1,r1)/dot(p1,A*p1);
    x  = x0+d*p1;
    r2 = r1-d*A*p1;
    f  = dot(r2,r2)/dot(r1,r1);
    p2 = r2+f*p1;
    n  = n + 1;
end
d  = dot(r2,r2)/dot(p2,A*p2);
x  = x+d*p2;
n  = n + 1;
```

【例 2-15】 共轭梯度法求解线性方程组实例。采用共轭梯度法求解以下线性方程组，

$$\begin{cases} 25x_1 - 300x_2 + 1050x_3 - 1400x_4 + 630x_5 = 5 \\ -300x_1 + 4800x_2 - 18900x_3 + 26880x_4 - 12600x_5 = 3 \\ 1050x_1 - 18900x_2 + 79380x_3 - 117600x_4 + 56700x_5 = -1 \\ -1400x_1 + 26880x_2 - 117600x_3 + 179200x_4 - 88200x_5 = 0 \\ 630x_1 - 12600x_2 + 56700x_3 - 88200x_4 + 44100x_5 = -2 \end{cases}$$

解：用共轭梯度法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A= [25       -300       1050      -1400        630;
       -300       4800     -18900      26880     -12600;
       1050     -18900      79380    -117600      56700;
      -1400      26880    -117600     179200     -88200;
        630     -12600      56700     -88200      44100;];
>> b=[ 5 3 -1 0 -2]';
>> x0=[0 0 0 0 0]';
>> [x,n]=conjgrad(A,b,x0)
```

输出的计算结果为：

```
x =
  5.7667
  2.9167
  1.9310
  1.4333
  1.1349
```

输出的迭代次数为：

```
n =
     6
```

可见，当迭代初始值为[0, 0, 0, 0, 0]时，经过 6 步迭代，共轭梯度法求出了方程组的解为$[x_1, x_2, x_3, x_4, x_5]$=[5.7667, 2.9167, 1.9310, 1.4333, 1.1349] 。

输入下面的表达式以验证结果，在 MATLAB 命令窗口中输入：

```
>> A*x
```

输出的计算结果为：

```
ans =
    5.0000
    3.0000
   -1.0000
   -0.0000
   -2.0000
```

共轭梯度法求解的速度非常快，通常比最速下降法提高了好几个数量级。

3. 预处理的共轭梯度法

当 ***Ax=b*** 为病态方程组时，共轭梯度法会收敛得很慢。预处理技术是在用共轭梯度法求解之前对系数矩阵做一些变换后再求解的技术。

在方程组两边都乘以一个非奇异矩阵 $\boldsymbol{F}$，令

$$\boldsymbol{A}=\boldsymbol{F}\boldsymbol{A}\boldsymbol{F}^T$$

$$\boldsymbol{x}=\boldsymbol{F}^{-T}\boldsymbol{x}$$

$$\boldsymbol{b}=\boldsymbol{F}\boldsymbol{b}$$

对方程 $\boldsymbol{A}\,\boldsymbol{x}=\boldsymbol{b}$ 用共轭梯度法求解，则 $\boldsymbol{F}^T\boldsymbol{x}$ 为原方程的解。理想情况下是取 $\boldsymbol{F}=\boldsymbol{A}^{-1}$，但对病态矩阵，求逆太耗费时间，实用的取法步骤为：

步骤 1，取 $\boldsymbol{F}^2=\mathrm{diag}(1/a_{11},\ 1/a_{22}\cdots\cdots 1/a_{nn})$；

步骤 2，取系数矩阵 $\boldsymbol{A}$ 的三条对角线构成的矩阵，再对其进行 Cholesky 分解，将 $\boldsymbol{F}$ 取为分解后的上三角矩阵。

另外 $\boldsymbol{F}$ 还有一些其他的取法，具体可参考数值计算的书籍。

MATLAB 中有一个对应的处理函数 pcg()，它的用法如下：

```
[x,flag] = pcg(A,b,tol,maxit,M1,M2,x0,p1,p2,…) ;
[x,flag,relres] = pcg(A,b,tol,maxit,M1,M2,x0,p1,p2,…);
[x,flag,relres,iter] = pcg(A,b,tol,maxit,M1,M2,x0,p1,p2,…);
[x,flag,relres,iter,resvec] = pcg(A,b,tol,maxit,M1,M2,x0,p1,p2,…);
```

其中，tol 为迭代的精度，

maxit 为迭代的最大步数，

$\boldsymbol{M}1$ 和 $\boldsymbol{M}2$ 就是预处理矩阵；

flag 的取值为：0 表示在指定迭代次数之内按要求精度收敛；1 表示在指定迭代次数内不收敛；2 表示 $\boldsymbol{M}$ 为坏条件的预处理因子；3 表示两次连续迭代完全相同；4 表示标量参数太小或太大；

relres 为相对误差：norm($\boldsymbol{b}$-$\boldsymbol{A}$*$\boldsymbol{x}$)/norm($\boldsymbol{b}$)；

iter 为计算 x 的迭代次数；

resvec 为每次迭代的残差：norm($\boldsymbol{b}$-$\boldsymbol{A}$*$\boldsymbol{x}$0)。

【例 2-16】 预处理的共轭梯度法求解线性方程组实例。采用预处理的共轭梯度法求解以下线性方程组，

$$\begin{cases}25x_1-300x_2+1050x_3-1400x_4+630x_5=5\\-300x_1+4800x_2-18900x_3+26880x_4-12600x_5=3\\1050x_1-18900x_2+79380x_3-117600x_4+56700x_5=-1\\-1400x_1+26880x_2-117600x_3+179200x_4-88200x_5=0\\630x_1-12600x_2+56700x_3-88200x_4+44100x_5=-2\end{cases}$$

解：用预处理的共轭梯度法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A= [25       -300       1050      -1400        630;
       -300       4800     -18900      26880     -12600;
       1050     -18900      79380    -117600      56700;
      -1400      26880    -117600     179200     -88200;
       630      -12600      56700     -88200      44100;];
>> b=[ 5 3 -1 0 -2]';
>> x0=[0 0 0 0 0]';
```

```
>> M=pascal(5);  %预处理矩阵
>> [x,flag,re,it]=pcg(A,b,1.e-8, 1000, M,M,x0)
%输出的预处理矩阵
M =
     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70
```

输出的计算结果为：

```
x =
     5.7667
     2.9167
     1.9310
     1.4333
     1.1349
```

输出的 flag 取值为 0，表示在指定迭代次数之内按要求精度收敛

```
    flag =
0
```

输出的相对误差：

```
re =
      5.7305e-012
```

输出的计算 x 的迭代次数：

```
    it =
      10
```

可见，当迭代初始值为[0,0,0,0,0]时，经过 10 步迭代，预处理的共轭梯度法求出了方程组的解为$[x_1, x_2, x_3, x_4, x_5]$=[5.7667, 2.9167, 1.9310, 1.4333, 1.1349]。

2.3.8 其他迭代法

MATLAB 还有一些迭代法，其用法和参数与预处理共轭梯度法基本上一样，如表 2-2 所示。

表 2-2 其他的线性方程组迭代法求解函数

函数及语法	说 明
x = symmlq(A,b)	线性方程组的 LQ 解法
x = bicg(A,b)	线性方程组的双共轭梯度解法
x = bicgstab(A,b)	线性方程组的稳定双共轭梯度解法
x = lsqr(A,b)	线性方程组的共轭梯度的 LSQR 解法
x = gmres(A,b)	线性方程组的广义最小残差解法
x = minres(A,b)	线性方程组的最小残差解法
x = qmr(A,b)	线性方程组的准最小残差解法

【例 2-17】 最小残差法求解线性方程组实例。采用最小残差法求解以下线性方程组，

$$\begin{cases} 3x_2 + 8x_3 = 1 \\ x_1 + 4x_2 + 7x_3 = 1 \\ -2x_1 + 6x_2 + 9x_3 = 1 \end{cases}$$

解：用最小残差解法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>A=[ 0 3 8;1 4 7; -2 6 9];
>> b= [ 1 1 1]';
>> [x,flag,re, it]=minres(A,b)
```

输出的计算结果为：

```
x =
 0.0754
 0.0807
 0.0826
```

输出的 flag 取值为 1，表示在指定迭代次数之内不收敛：

```
flag =
     1
```

输出的相对误差：

```
re =
    0.0519
```

输出的计算 x 的迭代次数

```
it =
     3
```

由计算结果可知，经过迭代 3 步，flag=1，表明默认情况下不收敛，因此需要选择其他的解法来求解。

2.4 特殊解法

2.4.1 三对角矩阵的追赶法

对于系数矩阵为对角占优的三对角矩阵，有一种特殊的解法叫追赶法。

追赶法首先对系数矩阵做如下三角分解：

$$A=LU$$

其中 L 为下三角的两对角矩阵，U 为上三角的两对角矩阵，且主对角线元素全为 1。

求解时先求解 $Ly=b$ 的解 y，再求解 $Ux=y$ 的解 x。

在 MATLAB 中编程实现的追赶法函数为：followup。

功能：用追赶法求线性方程组 $Ax=b$ 的解。

调用格式：x= followup (A,b)。

其中，A 为线性方程组的系数矩阵；

b 为线性方程组中的常数向量；

x 为线性方程组的解；

追赶法用 MATLAB 实现如下：

```
function x=followup(A,b)
%采用追赶法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x

n = rank(A);
for(i=1:n)
    if(A(i,i)==0)
        disp('Error: 对角有元素为 0! ');
        return;
    end
end;
d = ones(n,1);
a = ones(n-1,1);
c = ones(n-1);
for(i=1:n-1)
    a(i,1)=A(i+1,i);
    c(i,1)=A(i,i+1);
    d(i,1)=A(i,i);
end
d(n,1) = A(n,n);
%求解 Ly=b 的解 y，解保存在 b 中，
for(i=2:n)
    d(i,1)=d(i,1) - (a(i-1,1)/d(i-1,1))*c(i-1,1);
    b(i,1)=b(i,1) - (a(i-1,1)/d(i-1,1))*b(i-1,1);
end
%求解 Ux=y 的解 x，
x(n,1) = b(n,1)/d(n,1);
for(i=(n-1):-1:1)
    x(i,1) = (b(i,1)-c(i,1)*x(i+1,1))/d(i,1);
end
```

【例 2-18】 追赶法求解线性方程组实例。采用追赶法求解以下线性方程组，

$$\begin{cases} 2.5x_1 + x_2 = 1 \\ x_1 + 1.5x_2 + x_3 = 1 \\ x_2 + 0.5x_3 + x_4 = 1 \\ x_3 + 0.5x_4 + x_5 = 1 \\ x_4 + 1.5x_5 + x_6 = 1。 \\ x_5 + 2.5x_6 = 1 \end{cases}$$

解： 用追赶法求解，在 MATLAB 命令窗口中输入求解程序：

```
>>A = [2.5000    1.0000         0         0         0         0;
       1.0000    1.5000    1.0000         0         0         0;
       0         1.0000    0.5000    1.0000         0         0;
       0         0         1.0000    0.5000    1.0000         0;
```

```
        0          0          0          1.0000    1.5000    1.0000;
        0          0          0           0         1.0000    2.5000];
>> b=ones(6,1)
>> x=followup(A,b)
```

输出的计算结果为：

```
x =
    0.4615
   -0.1538
    0.7692
    0.7692
   -0.1538
    0.4615
```

可见，追赶法求出了方程组的解为：

$[x_1, x_2, x_3, x_4, x_5, x_6] = [0.4615, -0.1538, 0.7692, 0.7692, -0.1538, 0.4615]$ 。

最后可以输入下面的表达式以验证结果的正确性，在 MATLAB 命令窗口中输入：

```
>> A*x
```

输出的计算结果为：

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

可见，追赶法求解的结果是正确的。

2.4.2 快速求解法

MATLAB 中有一个通用的求解线性方程组的函数，它的用法如下：

x = linsolve(A, b, options)

其意义为快速求解方程 $Ax=b$，其中 A 的结构由 options 决定，内容如表 2-3 所示。

表 2-3 options 取值表

options	说　明
LT	下三角阵
UT	上三角阵
UHESS	上三角 Hessenberg 矩阵
SYM	实对称阵
POSDEF	正定矩阵
RECT	一般矩阵
TRANSA	指出求解的方程是 $Ax=b$ 还是 $A'x=b$，A'指 A 的共轭转置

【例 2-19】 快速求解法求解线性方程组实例。采用快速解法求解以下线性方程组，

$$\begin{cases} 4x_1 - x_2 + 2x_3 = 1 \\ -x_1 + 5x_2 = -1 \\ 2x_1 + 6x_3 = 2。 \end{cases}$$

解：用快速求解法，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[4 -1 2;-1 5 0;2 0 6];
>> b=[1 -1 2]';
>> optt.SYM=true;
>> optt.POSDEF=true;
>> optt.TRANSA=false;
>> x=linsolve(A,b,optt)
```

输出的计算结果为：

```
x =
    0.0426
   -0.1915
    0.3191
```

可见，快速求解法求出了方程组的解为$[x_1, x_2, x_3] = [0.0426, -0.1915, 0.3191]$。

2.5 非奇次线性方程组的解法

2.5.1 超定方程的解法

超定方程一般是没有解的，这里所说的解是某种意义上误差最小的解。MATLAB 对超定方程有几种解法，由于不同方法对误差最小的定义不同，由此得出的解也不同：

（1）用伪逆法来求解，MATLAB 中的求解函数为：

$$\boldsymbol{x} = \text{pinv}(\boldsymbol{A})*\boldsymbol{b}$$

（2）用左除法来求解，MATLAB 中的求解函数为：

$$\boldsymbol{x} = \boldsymbol{A}\backslash\boldsymbol{b}$$

（3）用最小二乘法来求解，MATLAB 中的求解函数为：

$$\boldsymbol{x} = \text{lsqnonneg}(\boldsymbol{A}，\ \boldsymbol{b})$$

【例 2-20】 超定方程求解实例。采用伪逆法、左除法和最小二乘法求解以下超定方程组，

$$\begin{cases} x_1 - 2x_2 + 3x_3 = 1 \\ 4x_1 + x_2 = 0 \\ 7x_1 + x_2 + 6x_3 = 1 \\ 9x_1 + 5x_2 + 8x_3 = 0 \end{cases}$$

解：在 MATLAB 命令窗口中输入求解程序：

```
>> A=[1 -2 3;4 1 0;7 1 6;9 5 8];
>> b=[1 0 1 0]';
>> x=pinv(A)*b   %用伪逆法求解
```

输出的计算结果为：

```
x =
    0.0903
   -0.3248
    0.1048
>> x=A\b    %用左除法求解
```

输出的计算结果为：

```
x =
    0.0903
   -0.3248
    0.1048
>> x=lsqnonneg(A,b) %用最小二乘法求解
```

输出的计算结果为：

```
x =
         0
         0
    0.0826
```

从上述结果可以看出，采用不同的解法，得到的结果不完全相同。

2.5.2 有无穷组解的线性方程组的解法

对于此类方程的解法，可分为以下 3 个步骤：

步骤 1，求 $\boldsymbol{Ax}=\boldsymbol{b}$ 的一个特解；

步骤 2，求 $\boldsymbol{Ax}=0$ 的通解；

步骤 3，将特解与通解组合成最终的解。

【例 2-21】 有无穷组解的线性方程组求解实例。求解以下线性方程组，

$$\begin{cases} 4x_1+8x_2-6x_3+2x_4=1 \\ x_1+3x_2+9x_3+5x_4=2 \\ 5x_1+11x_2+3x_3+7x_4=3 \\ 3x_1+5x_2-15x_3-3x_4=-1 \end{cases}$$

解：在 MATLAB 命令窗口中输入求解程序：

```
>> A=[4  8 -6 2;1  3 9  5;5  11 3 7;3  5  -15 -3];
>> b=[1 2 3 -1]';
>> D=[A b]
>> S=rref(D) %采用增广矩阵法求解
```

输出的计算结果为：

```
S =
    1.0000         0   -22.5000   -8.5000   -3.2500
         0    1.0000    10.5000    4.5000    1.7500
         0         0         0         0         0
         0         0         0         0         0
```

上面例子是采用增广矩阵法求解的，从输出的 S 结果可以看出，此方程的一个特解为 x_0= [-3.25　1.75　0　0]^T，其基础解系有两个基向量：

$$\xi_1 = [-22.5 \quad 10.5 \quad 1 \quad 0]^T, \quad \xi_2 = [-8.5 \quad 4.5 \quad 0 \quad 1]^T$$

则最终的解为 $x = x_0 + k_1 \xi_1 + k_2 \xi_2$。

2.6 小结

本章详细讲述了 MATLAB 中求解线性方程组的常用解法，包括求逆法、分解法、迭代法、特殊解法，以及非奇次线性方程组的解法。

每种求解方法也各有其优缺点，适用于不同的方程组。选择线性方程组解法的主要原则可总结如下：

（1）首先判断系数矩阵是否方为阵，是方阵就转下面的步骤，否则此方程要么无解，要么有无穷解，此时用非奇次线性方程组的解法求解；

（2）观察系数矩阵的特征，是否为对称矩阵或正定矩阵，它们适合用矩阵分解法求解；

（3）稀疏矩阵和病态矩阵可考虑用迭代法求解；

（4）特殊解法适合特殊方程。

第 3 章

数据插值与拟合

在工程实践和科学实验中，常常需要从一组试验观测数据之中找到自变量与因变量之间的函数关系，一般可用一个近似函数来表示。函数的产生办法因观测数据和要求不同而异，数据插值与拟合是两种常用的方法。采用 MATLAB，既可以利用其提供的插值和拟合函数，又可以利用编程来实现数据的插值和拟合。

通过本章，读者不仅能够掌握 MATLAB 中相应的插值和拟合函数，而且能灵活编程来解决数据插值和拟合问题。

3.1 MATLAB 中的插值函数

函数插值来源于函数的以下问题：只知道函数在某区间有定义且已得到区间内一些离散点的值，希望能用简单的表达式近似给出函数在此区间上的整体描述，并能与已知离散点上的值相等。

插值法按插值函数的形式主要分为以下几类：

- 代数多项式插值；
- 三角多项式插值；
- 有理分式插值；

代数多项式插值是最常用的插值方式，其内容也是最丰富的，它又可分为以下几种插值方式：

（1）非等距节点插值，包括拉格朗日插值、艾肯特插值和利用均差的牛顿插值；

（2）等距节点插值，包括利用差分的牛顿插值和高斯插值等；

（3）在插值中增加了导数插值的埃尔米特插值；

（4）分段插值，包括分段线性插值、分段埃尔米特插值和样条函数插值；

（5）反插值。

按被插值函数的变量个数还可把插值法分为一元插值和多元插值。下面首先介绍MATLAB提供的插值方法，然后再介绍数值方法中常用的插值方法。

3.1.1 一元插值函数

MATLAB中的一元插值函数为interp1()，它的功能是一维数据插值（表格查找）。该命令对数据点之间进行计算内插值，它找出一元函数 $f(x)$ 在中间点的数值，其中函数 $f(x)$ 由所给数据决定。

一元插值函数interp1()的几种调用格式如表3-1所示。

表3-1 一维插值函数interp1的语法格式

语法形式	说 明
y = interp1(**x**, **Y**, x_i)	由已知点集（**x**,**Y**）插值计算 x_i 上的函数值 y
y = interp1(**Y**, x_i)	相当于 **x**=1:length(**Y**)的 interp1(**x**,**Y**, x_i)
y = interp1(**x**,**Y**, x_i,method)	用指定插值方法计算插值点 x_i 上的函数值 y
y = interp1(**x**,**Y**, x_i,method,'extrap')	对 x_i 中超出已知点集的插值点用指定方法计算函数值 y
y = interp1(**x**,**Y**, x_i,method,extrapval)	用指定方法插值 x_i 上的函数值 y，x_i 中超出已知点集处函数值取 extrapval
pp = interp1(**x**,**Y**,method,'pp')	用指定方法进行插值，但返回结果为分段多项式

MATLAB中一维插值有多种算法，由interp1函数中的method参数指定。MATLAB中一维插值的各种算法如表3-2所示。

表3-2 一维插值算法（method）

method	方法描述
'nearest'	最邻近插值：插值点处函数值取与插值点最邻近的已知点上的函数值
'linear'	分段线性插值：插值点处函数值由连接其最邻近的两侧点的线性函数预测，MATLAB 中 interp1 的默认方法
'spline'	样条插值：默认为三次样条插值。可用 spline 函数代替
'pchip'	三次 Hermite 多项式插值。可用 pchip 函数代替
'cubic'	同 pchip，三次 Hermite 多项式插值
'v5cubic'	MATLAB 5 中的三次多项式插值，不能用于外插 当已知点不是等间距时，转换为 spline 方法

下面详细介绍表3-2中的几种算法。

1. linear（分段线性插值）

它的算法是在每个小区间 $[x_i, x_{i+1}]$ 上采用简单的线性插值。在区间 $[x_i, x_{i+1}]$ 上的子插值多项式为：

$$F_i = \frac{x - x_{i+1}}{x_i - x_{i+1}} f(x_i) + \frac{x - x_i}{x_{i+1} - x_i} f(x_{i+1})$$

由此整个区间$[x_1, x_n]$上的插值函数为：

$$F(x) = \sum_{i=1}^{n} F_i l_i(x)$$

其中$l_i(x)$的定义如下：

$$l_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & x \in [x_{i-1}, x_i] (i = 0\text{略去}) \\ \dfrac{x - x_{i+1}}{x_i - x_{i+1}}, & x \in [x_i, x_{i+1}] (i = n\text{略去}) \\ 0, & x \notin [x_{i-1}, x_{i+1}] \end{cases}$$

分段线性插值方法则在速度和误差之间取得了比较好的均衡，其插值函数具有连续性，但在已知数据点处的斜率一般都会改变，因此不是光滑的。分段线性插值方法是 MATLAB 一维插值的默认方法。

2. spline（样条插值）

样条函数是用分段低次多项式去逼近函数。样条函数可以给出光滑的插值曲线，只要在插值区间端点提供某些导数信息，样条插值可以适应不同的光滑需求。三次样条是使用最广泛的样条插值，它在每个子区间$[x_i, x_{i+1}]$上都是有二阶连续导数的三次多项式，即：

$$p(x) = \begin{cases} p_1(x), & x_1 \leqslant x \leqslant x_2 \\ p_2(x), & x_2 \leqslant x \leqslant x_3 \\ \cdots & \cdots \\ p_n(x), & x_n \leqslant x \leqslant x_{n+1} \end{cases}$$

其中每段$p_i(x)$都是三次多项式。

对于给定的离散的测量数据x, y（称为断点），要寻找一个三项多项式$y = p(x)$，以逼近每对数据(x_i, y_i)点间的曲线。过两点(x_i, y_i)和(x_{i+1}, y_{i+1})只能确定一条直线，而通过一点的三次多项式曲线有无穷多条。为使通过中间断点的三次多项式曲线具有唯一性，要增加以下的连续条件和边界条件（因为三次多项式有 4 个系数）：

（1）三次多项式在点(x_i, y_i)处有：$p_i(x_i) = p_{i+1}(x_i)$；

（2）三次多项式在点(x_i, y_i)处有：$p_i'(x_i) = p_{i+1}'(x_i)$；

（3）三次多项式在点(x_i, y_i)处有：$p_i''(x_i) = p_{i+1}''(x_i)$；

（4）边界条件：$f''(x_1) = f''(x_n) = 0$；

表 3-2 中的各种方法中：

（1）nearest 方法速度最快，占用内存最小，但一般来说误差最大，插值结果最不光滑；

（2）spline 三次样条插值法是所有插值方法中运行耗时最长的，其插值函数以及插值函数的一阶、二阶导函数都连续，因此是最光滑的插值方法，占用内存上比 cubic 方法小，

但当已知数据点不均匀分布时可能出现异常结果。

（3）cubic 三次多项式插值法中插值函数及其一阶导函数都是连续的，因此其插值结果也比较光滑，运算速度比 spline 方法略快，但占用内存最多。在实际的使用中，应根据实际需求和运算条件选择合适的算法。

另外，已知数据点不等间距分布时，interp1q 函数比 interp1 函数执行速度快，因为前者不检查已知数据点是否等间距，不过 interp1q 函数要求 x 必须单调递增。

【例 3-1】 线性插值函数应用实例。采用 interp1 对 sin 函数进行分段线性一维插值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=0:2*pi;
>> y=sin(x);
>> xx=0:0.5:2*pi;
>> yy=interp1(x,y,xx);
>> plot(x,y,'s',xx,yy)
```

程序运行后，输出如图 3-1 所示的结果。

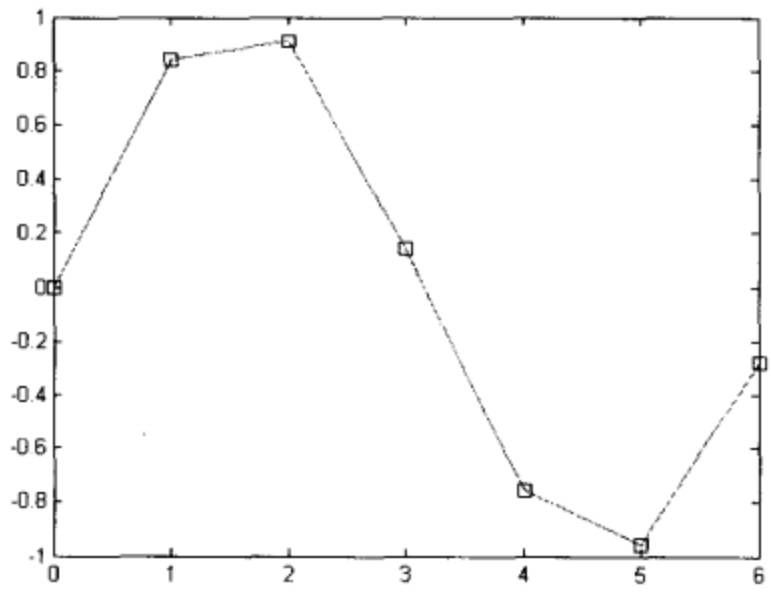

图 3-1 sin 函数一维插值结果

【例 3-1】中用默认的方法（分段线性的 linear 方法）对已知的 7 个 sin 函数的数据点进行插值，用 plot 画出插值结果。从图 3-1 可以看出分段线性插值就是用联结两个邻近的已知点的线性函数插值计算该区间内插值点上的函数值。

【例 3-2】 一维插值方法使用实例。用其他一维插值方法对以下 7 个离散数据点（1, 3.5）、（2, 2.1）、（3, 1.3）、（4, 0.8）、（5, 2.9）、（6, 4.2）、（7, 5.7）进行一维插值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=[1 2 3 4 5 6 7];
>> y=[3.5 2.1 1.3  0.8 2.9 4.2 5.7];
>> xx=1:0.5:7;
>> y1=interp1(x,y,xx,'nearest');
>> y2=interp1(x,y,xx,'spline');
>> y3=interp1(x,y,xx,'cubic');
>> plot(x,y,'o', xx,y1,'-',xx,y2,'-.',xx,y3,':')
```

程序运行后，输出如图 3-2 所示的结果。

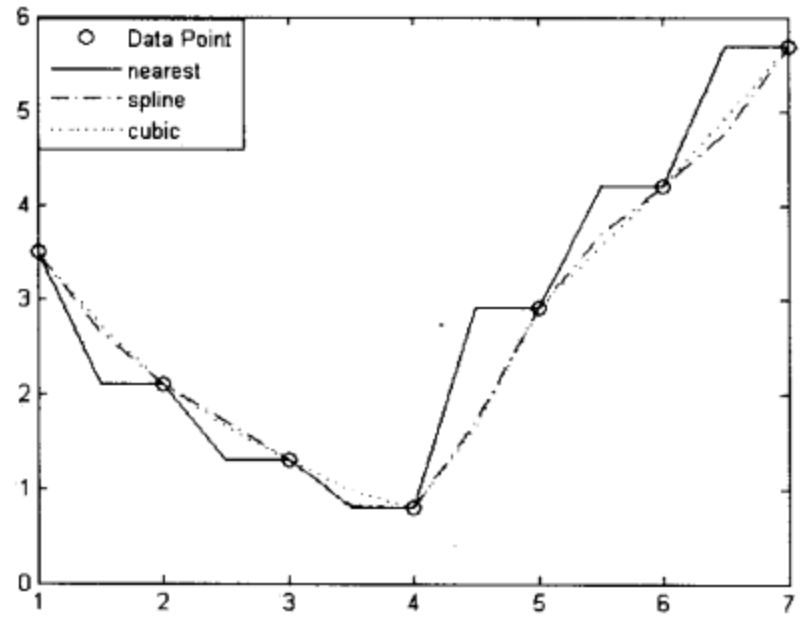

图 3-2　采用 nearest、spline 和 cubic 对离散数据插值图

上面的两个例子都是对已知数据点集内部的点进行的插值运算，这称作内插。内插可以根据已知数据点的分布，构建能够代表分布特性的函数关系，比较准确地估测插值点上的函数值。当插值点落在已知数据集外部时的插值称为外插，interp1 函数中可以通过添加'extrap'参数，指明插值算法也用于外插运算。

【例 3-3】　一维外插值函数应用实例。采用 interp1 对 sin 函数进行一维外插值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=0:2*pi;
>> y=sin(x);
>> xx=0:0.5:2*pi;
>> y1=interp1(x,y,xx,'nearest','extrap');
>> y2=interp1(x,y,xx,'linear','extrap');
>> y3=interp1(x,y,xx,'spline','extrap');
>> y4=interp1(x,y,xx,'cubic','extrap');
>> plot(x,y,'s', xx,y1,'-',xx,y2,'-.',xx,y3,':',xx,y4,'--')
```

程序运行后，输出如图 3-3 所示的结果。

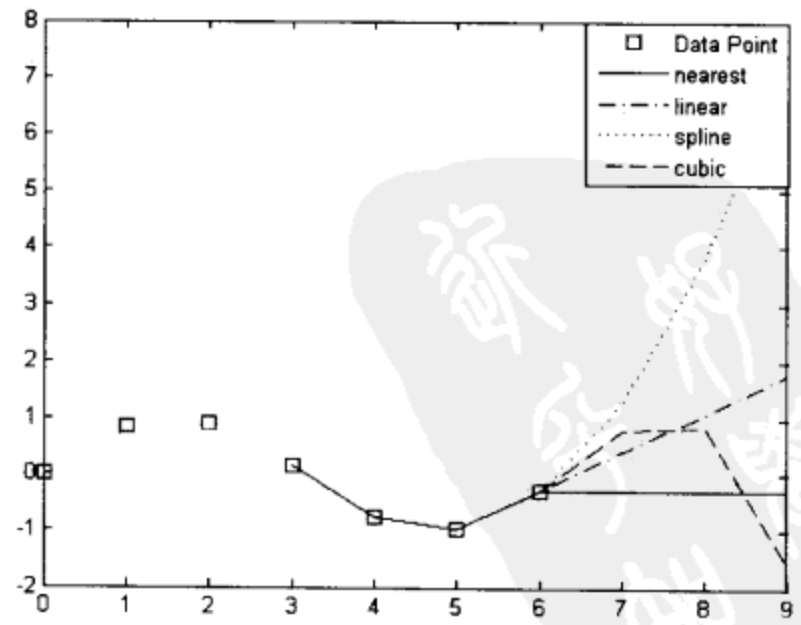

图 3-3　对 sin 函数采用 nearest、linear、spline 和 cubic 进行外插值图



说明

不管用那种插值方法，当插值点位于已知数据集合外时，插值运算对该处函数值的估计都很可能与实际函数值有很大的偏差。

3.1.2　二元插值函数

MATLAB 中提供了 interp2 函数进行二维插值，其用法类似于一维插值函数 interp1，不过它可选的插值方法只有三种：'nearest'、'linear'和'cubic'。其中第二种是默认的插值方法。

在使用 interp2 的过程中要注意以下几点：

- 在二维插值中已知数据点集（*X*, *Y*）必须是栅格格式，一般用 meshgrid 函数产生；
- interp2 函数要求（*X*, *Y*）必须是严格单调的，即单调递增或者单调递减；
- interp2 函数输入中，若已知点集（*X*, *Y*）在平面上分布不是等间距时，interp2 函数首先通过一定的变换将其转换为等间距的；
- 当输入点集（*X*, *Y*）已经是等间距分布的话，可以在 method 参数前面加星号（*），即如'*cubic'这样输入参数，这样可以提高插值速度。

【例 3-4】　二元插值函数应用实例。采用 nearest、linear 和 cubic 二维插值，绘制三维表面图。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> [x,y] = meshgrid(-4:4);                    %产生已知数据栅格点
>> z = peaks(x,y);                            %计算已知点上的函数值
>> [x1,y1] = meshgrid(-4:0.25:4);             %产生更精细的插值栅格点
>> z1 = interp2(x,y,z,x1,y1,'nearest');
>> surf(x1,y1,z1)                             %画基于最邻近法插值的三维表面图
>> z2 = interp2(x,y,z,x1,y1,'linear');
>> surf(x1,y1,z2)                             %画基于二维分段线性插值的三维表面图
>> z3 = interp2(x,y,z,x1,y1,'cubic');
>> surf(x1,y1,z3)                             %画基于二维三次多项式插值的三维表面图
```

程序运行后，依次输出如图 3-4、图 3-5 和图 3-6 所示的结果。

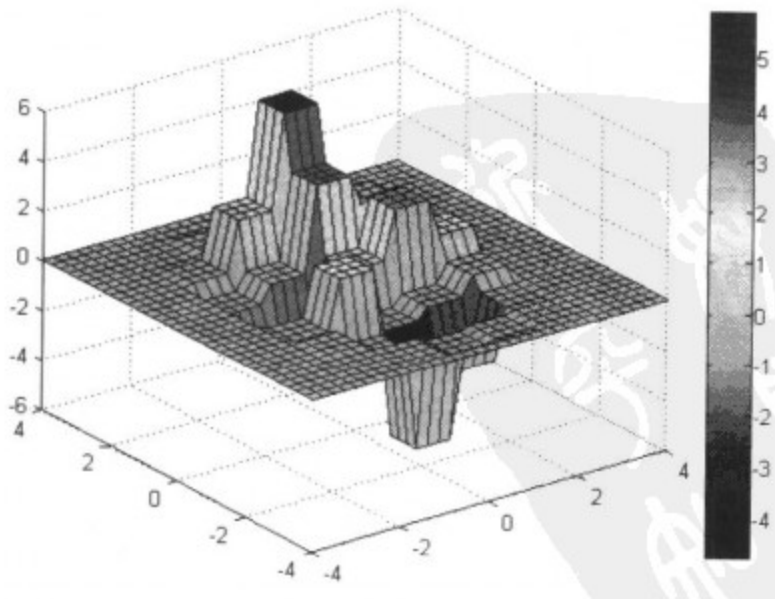

图 3-4　采用 nearest 二维插值得到的三维表面图

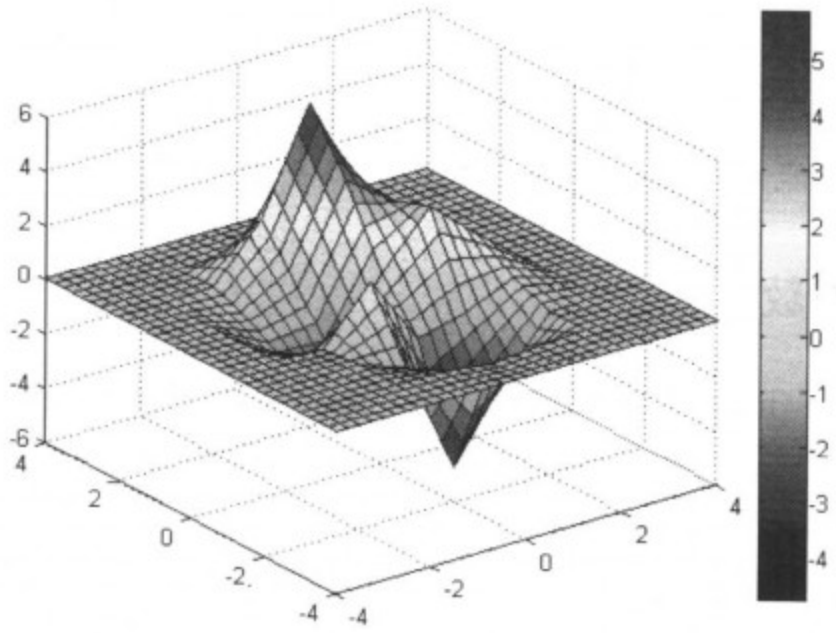

图 3-5　采用 linear 二维插值得到的三维表面图

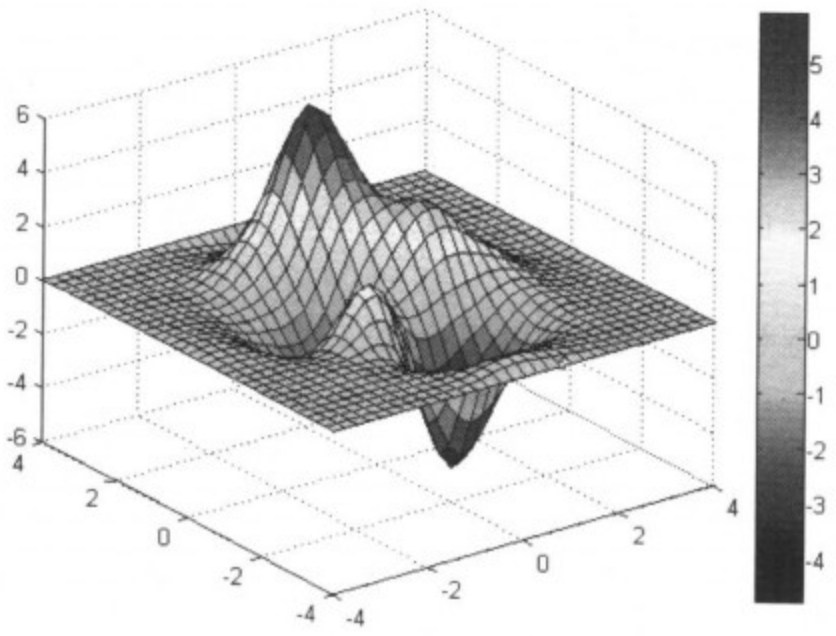

图 3-6　采用 cubic 二维插值得到的三维表面图

3.1.3　其他插值相关的函数

当数据点呈现周期分布时，显然用上面的几种插值算法都不太合适。这时可以用 interpft 函数进行插值。此函数用快速 Fourier 算法作一维插值，其函数定义如下：

$$y = \text{interpft}(x, n)$$

它返回周期函数在重采样的 n 个等距的点的插值。要注意的是 n 必须大于 x 的长度。

【例 3-5】　interpft 插值函数应用实例。采用 interpft 函数对正弦函数插值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=0:2*pi;
>> y=sin(x);
>> z=interpft(y,15);
>> xx=linspace(0,2*pi, 15);
>> plot(x,y,'-o',xx,z,':o')
```

程序运行后，输出如图 3-7 所示的结果。

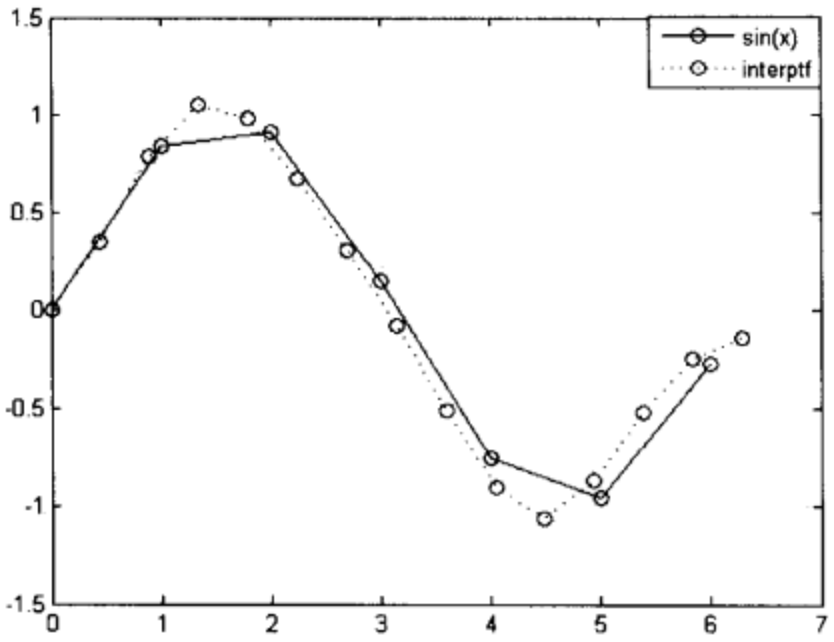

图 3-7 interpft 函数对周期函数的插值图

样条插值可以用 spline 函数，三次多项式插值可以用 pchip 函数。这两个函数的语法格式如表 3-3 所示。

表 3-3 spline 函数和 pchip 函数语法格式

函数语法	说 明
y_i=spline（x, y, x_i）	相当于 y_i=interp1（x,y, x_i,'spline'）
pp=spline（x, y） y_i =ppval（pp, x_i）	返回分段样条插值函数 以 pp 为插值函数计算 x_i 上的函数插值结果
y_i =pchip（x, y, x_i）	相当于 y_i=interp1（x, y, x_i,'cubic'）
pp=pchip（x, y） y_i =ppval（pp, x_i）	返回分段三次 Hermite 多项式插值函数 以 pp 为插值函数计算 x_i 上的函数插值结果

【例 3-6】 样条插值和多项式插值应用实例。使用 spline 函数和 pchip 函数对下表中的点进行细密插值并作图比较。

-3	-2	-1	0	1	2	3
-1	-1	-1	0	1	1	1

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x = -3:3;
>> y = [-1 -1 -1 0 1 1 1];          %产生已知点
>> t = -3:.01:3;                    %产生细密插值点的横坐标
>> p = pchip(x,y,t);                %用'cubic'方法对已知点进行细密插值
>> ppol=spline(x,y)                 %先对已知点进行分段样条插值
ppol =
      form: 'pp'
    breaks: [-3 -2 -1 0 1 2 3]
     coefs: [6x4 double]
    pieces: 6
     order: 4
       dim: 1
```

```
>> s=ppval(ppol,t);                    %再用插值后的样条函数求插值点的 y 值
>> plot(x,y,'o',t,p,'-',t,s,'-.')
%圆圈代表已知点构成的曲线
%短横线代表用'cubic'方法进行细密插值后的曲线
%点划线代表先分段插值在细密插值后的曲线
```

程序运行后，输出如图 3-8 所示的结果。

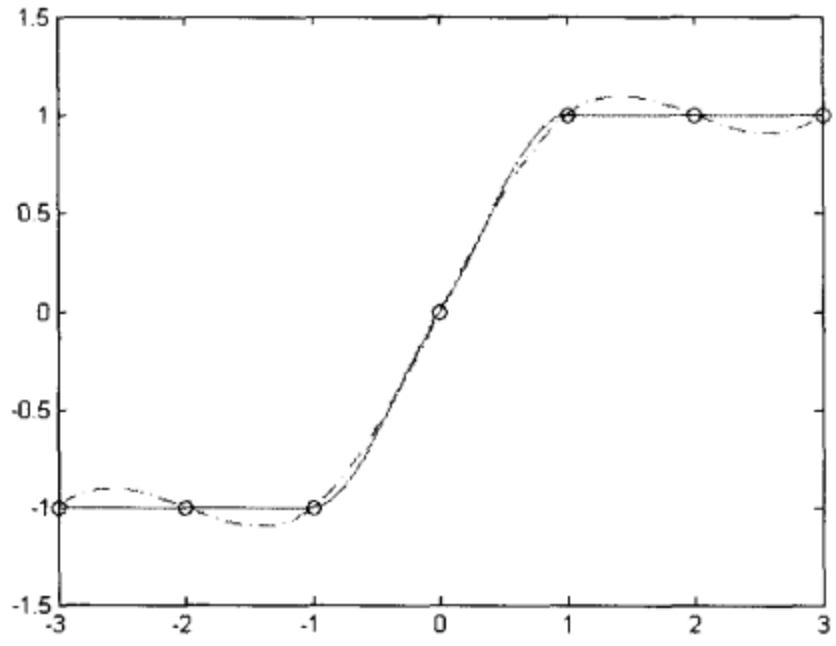

图 3-8　spline 函数和 pchip 函数插值图

MATLAB 还提供了三维甚至 n 维的插值函数，为了方便产生插值点，MATLAB 还有几个函数用来产生二维或高维空间的栅格点。它们的具体用法如表 3-4 所示。

表 3-4　与插值有关的其他函数

语法形式	说　明
VI = interp3(*X*, *Y*, *Z*, V, *X*1, *Y*1, *Z*1, method)	按照指定的插值方法找出由参量 *X*, *Y*, *Z* 决定的三元函数 V=V(*X*, *Y*, *Z*)在点（*X*1, *Y*1, *Z*1）的值
VI = interpn(*X*1, *X*2,...,V, *Y*1, *Y*2,...,method)	按照指定的插值找出由参量 *X*1, *X*2,....决定的 n 元函数 V=V(*X*1, *X*2,....)在点（*Y*1, *Y*2,....）的值
ZI = griddata(*x*, *y*, *z*, *XI*, *YI*)	按已知向量 *x*, *y*, *z* 拟合二元曲面 $z=f(x,y)$，并返回曲面 *z* 在点（*XI*, *YI*）处的插值
[*X*,*Y*] = meshgrid(*x*, *y*)	产生二（三）维空间的栅格点
[*X*1, *X*2, *X*3,...] = ndgrid(*x*1, *x*2, *x*3,...)	产生 n 维空间的栅格点

【例 3-7】 高维插值函数应用实例。已知曲面为 $z=\sin x+\sin y$，其中 $-3\leqslant x\leqslant 3$，$-3\leqslant y\leqslant 3$。用 griddata 函数求曲面在 $(x,y)=(-0.5,1)$ 处的插值，并和理论值比较。

解：在 MATLAB 命令窗口中输入以下命令：

```
>>[X,Y]=meshgrid(-3:3);
>> z=sin(X)+sin(Y);
>> z1=griddata(X,Y,z,-0.5,1)
z1 =
    0.4207
```

用 griddata 函数求得曲面在 $(x,y)=(-0.5,1)$ 处的插值为 0.4207，理论值为 sin(–0.5) +

sin1=0.3620，有一点误差，这时可以试试 griddata 函数的其他插值算法：

```
>> griddata(X,Y,z,-0.5,1,'linear')
ans =
    0.4207
>> griddata(X,Y,z,-0.5,1,'cubic')
ans =
    0.3563
>> griddata(X,Y,z,-0.5,1,'nearest')
ans =
     0
>> griddata(X,Y,z,-0.5,1,'v4')
ans =
    0.3629
```

从上面的结果可以看出函数 griddata 默认采用 linear 算法，本题中 v4 算法得出的结果最好。

高维插值中已知点栅格和插值点栅格都要用 ndgrid 函数产生，但没有一维插值和二维插值那么常用。

3.2 拉格朗日插值法

拉格朗日插值法是基于基函数的插值方法，插值多项式可表示为：

$$L(x)=\sum_{i=0}^{n} y_i l_i(x)$$

其中 $l_i(x)$ 称为 i 次基函数：

$$l_i(x)=\frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

则一般离散数据 (x_i, y_i) 的拉格朗日插值多项式如下：

$$L(x)=\sum_{i=0}^{n} y_i \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

很容易验证它满足插值条件。

在 MATLAB 中编程实现的拉格朗日插值法函数为：Language。

功能：求已知数据点的拉格朗日插值多项式。

调用格式：f = Language (*x*,*y*)或 f = Language (*x*,*y*,*x*0)

其中，*x* 为已知数据点的 *x* 坐标向量；

y 为已知数据点的 *y* 坐标向量；

*x*0 为插值点的 *x* 坐标；

f 为求得的拉格朗日插值多项式或在 *x*0 处的插值

在 MATLAB 中实现拉格朗日插值的代码如下：

```
function f = Language(x,y,x0)
%求已知数据点的拉格朗日插值多项式
```

```
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的 拉格朗日插值多项式或在 x0 处的插值：f

syms t;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                                          %检错

f = 0.0;
for(i = 1:n)
    l = y(i);
    for(j = 1:i-1)
        l = l*(t-x(j))/(x(i)-x(j));
    end;
    for(j = i+1:n)
        l = l*(t-x(j))/(x(i)-x(j));          %计算拉格朗日基函数
    end;

    f = f + l;                               %计算拉格朗日插值函数
    simplify(f);                             %化简

    if(i==n)
        if(nargin == 3)
            f = subs(f,'t',x0);              %计算插值点的函数值
        else
            f = collect(f);                  %将插值多项式展开
            f = vpa(f,6);                    %将插值多项式的系数化成 6 位精度的小数
        end
    end
end
```

【例 3-8】 拉格朗日插值法应用实例。根据下表所列的数据点求出其拉格朗日插值多项式，并计算当 $x=1.6$ 时的 y 值。

x	1	1.2	1.8	2.5	4
y	0.8415	0.9320	0.9738	0.5985	–0.7568

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=[1 1.2 1.8 2.5 4];
>> y=[0.8415 0.9320 0.9738 0.5985 -0.7568];
>> f=Language(x,y)
f =
    .328112e-1*t^4-.204917*t^3-.145485e-1*t^2+1.05427*t-.261189e-1
>> f=Language(x,y,1.6)
f =
    0.9992
```

表格中的数据点是按 $y=\sin x$ 给出的，sin(1.6)=0.9996，从插值函数在 x=1.6 的值看出，插值函数的精度是比较高的。

3.3 艾特肯插值法

拉格朗日插值法简单且易于建立，但它有一个缺点，那就是一旦增加插值节点的个数，原有的多项式的计算结果并不能加以利用，插值多项式必须重新建立，而且其形式不易简化，计算比较复杂。

艾特肯插值法是通过递推来建立插值多项式的，它的基本思想是 $k+1$ 次插值多项式可由两个 k 次插值多项式通过线性插值得到。

令

$$p_{0,1,\cdots,k}(x)=\frac{1}{x_n-x_k}\begin{vmatrix} p_{0,1,\cdots,k}(x) & x_k-x \\ p_{0,1,\cdots,k-1,n}(x) & x_n-x \end{vmatrix}$$

利用上式，可得到

$$p_{0,1}(x)=\frac{1}{x_1-x_0}\begin{vmatrix} p_0(x) & x_0-x \\ p_1(x) & x_1-x \end{vmatrix}$$

$$p_{0,2}(x)=\frac{1}{x_2-x_0}\begin{vmatrix} p_0(x) & x_0-x \\ p_2(x) & x_2-x \end{vmatrix}$$

递推下去，可得到表 3-5 所示的计算表格。

表 3-5 艾特肯插值多项式计算表格

已知数据	构造过程
x_0 $p_0(x)=f(x_0)$	
x_1 $p_1(x)=f(x_1)$	$p_{0,1}(x)$
x_2 $p_2(x)=f(x_2)$	$p_{0,2}(x)$ $p_{0,1,2}(x)$
x_3 $p_3(x)=f(x_3)$	$p_{0,3}(x)$ $p_{0,1,3}(x)$ $p_{0,1,2,3}(x)$
···	···
x_n $p_n(x)=f(x_n)$	$p_{0,n}(x)$ $p_{0,1,n}(x)$ $p_{0,1,2,n}(x)$ ··· $p_{0,1,2,\cdots,n}(x)$

说明

表右边每一项的计算公式如下，设第 i 行，第 j 列的多项式为 A(i, j)，其中 $i=1,2,\cdots,n$ ， $j=1,2,\cdots,n$ 。

例如，

$$A(1,1)=p_{0,1}(x)\ ,\quad A(3,2)=p_{0,1,3}(x)$$

则，

$$A(i,j)=A(i,j-1)\frac{x-x_{j-1}}{x_i-x_{j-1}}-A(j-1,j-1)\frac{x-x_i}{x_i-x_{j-1}}$$

最右下角的 $p_{0,1,\cdots,n-1,n}(x)$ 就是最终的插值多项式。

在 MATLAB 中编程实现的艾特肯插值法函数为：Atken。

功能：求已知数据点的艾特肯插值多项式。

调用格式：f = Atken (x, y)或f = Atken (x, y, $x0$)

其中，x 为已知数据点的 x 坐标向量；

y 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的艾特肯插值多项式或是在 $x0$ 处的插值。

在 MATLAB 实现艾特肯插值的代码如下：

```
function f = Atken(x,y,x0)
%求已知数据点的艾特肯插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的艾特肯插值多项式或在 x0 处的插值：f

syms t;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                                          %检错

y1(1:n) = t;                                 %符号函数数组要赋初值
for(i=1:n-1)
    for(j=i+1:n)
       y1(j) = y(j)*(t-x(i))/(x(j)-x(i))+y(i)*(t-x(j))/(x(i)-x(j));
    end
    y = y1;
    simplify(y1);
end

if(nargin == 3)
    f = subs(y1(n),'t',x0);                  %计算插值点的函数值
else
    simplify(y1(n));                         %化简
    f = collect(y1(n));                      %将插值多项式展开
    f = vpa(f,6);                            %将插值多项式的系数化成 6 位精度的小数
end
```

【例 3-9】 艾特肯插值法应用实例。根据下表所示的数据点求出其艾特肯插值多项式，并计算当 x=1.6 时的 y 值。

x	1	1.2	1.8	2.5	4
y	0.8415	0.9320	0.9738	0.5985	−0.7568

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=[1 1.2 1.8 2.5 4];
>> y=[0.8415 0.9320 0.9738 0.5985 -0.7568];
>> f=Atken(x,y)
```

```
f =
    .328112e-1*t^4-.204917*t^3-.145485e-1*t^2+1.05427*t-.261189e-1
>> f=Atken(x,y,1.6)
f =
    0.9992
```

表格中的数据点是按 $y=\sin x$ 给出的，sin(1.6)=0.9996，从插值函数在 $x=1.6$ 时的值看出，插值函数的精度是比较高的。

3.4 利用均差的牛顿插值法

函数 f 的零阶均差定义为 $f[x_0]=f(x_0)$，一阶均差定义为：

$$f[x_0,x_m]=\frac{f(x_m)-f(x_0)}{x_m-x_0}$$

一般地，函数 f 的 k 阶均差定义为：

$$f[x_0,x_1,\cdots,x_{k-1},x_m]=\frac{f[x_0,x_1,\cdots,x_{k-2},x_m]-f[x_0,x_1,\cdots,x_{k-1}]}{x_m-x_{k-1}}$$

利用均差的牛顿插值多项式为：

$$\begin{aligned}N(x)=&f(x_0)+f[x_0,x_1](x-x_0)+\\&f[x_0,x_1,x_2](x-x_0)(x-x_1)+\cdots+\\&f[x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})\end{aligned}$$

系数的计算过程如表 3-6 所示。

表 3-6 均差计算表格

	一阶均差	二阶均差	三阶均差	…	n 阶均差
$f(x_0)$					
$f(x_1)$	$f[x_0,x_1]$				
$f(x_2)$	$f[x_0,x_2]$	$f[x_0,x_1,x_2]$			
$f(x_3)$	$f[x_0,x_3]$	$f[x_0,x_1,x_3]$	$f[x_0,x_1,x_2,x_3]$		
…	…	…	…		
$f(x_n)$	$f[x_0,x_n]$	$f[x_0,x_1,x_n]$	$f[x_0,x_1,x_2,x_n]$	…	$f[x_0,x_1,\cdots,x_n]$

说明

表 3-6 中每一项均差的计算公式如下，设第 i 行，第 j 列的均差的值为 $A(i,j)$，其中 $i=1,2,\cdots,n$，$j=1,2,\cdots,n$。

例如 $A(1,1)=f[x_0,x_1]$，$A(3,2)=f[x_0,x_1,x_3]$

则，

$$A(i,j)=\frac{A(i,j-1)-A(j-1,j-1)}{x_i-x_{j-1}}$$

表 3-6 中对角线上的元素就是插值多项式中对应的系数。

在 MATLAB 中编程实现的均差形式的牛顿插值法函数为：Newton。

功能：求已知数据点的均差形式的牛顿插值多项式。

调用格式：f = Newton (x, y)或f = Newton (x,y,$x0$)

其中，x 为已知数据点的 x 坐标向量；

y 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的均差形式的牛顿插值多项式或在 $x0$ 处的插值。

在 MATLAB 中实现利用均差的牛顿插值的代码如下：

```
function f = Newton(x,y,x0)
%求已知数据点的均差形式的牛顿插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的 均差形式的牛顿插值多项式或在 x0 处的插值：f

syms t;

if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
    return;
end
f = y(1);
y1 = 0;
l  = 1;

for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = (y(j)-y(i))/(x(j)-x(i));
    end
    c(i) = y1(i+1);
    l = l*(t-x(i));
    f = f + c(i)*l;
    simplify(f);
    y = y1;

    if(i==n-1)
        if(nargin == 3)
            f = subs(f,'t',x0);
        else
            f = collect(f);                    %将插值多项式展开
            f = vpa(f, 6);
        end
    end
end
```

【例 3-10】 利用均差的牛顿插值法应用实例。根据下表所列的数据点求出其均差形式的牛顿插值多项式，并计算当 x=1.6 时的 y 值。

x	1	1.2	1.8	2.5	4
y	1	1.44	3.24	6.25	16

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=[1 1.2 1.8 2.5 4];
>> y=[1 1.44 3.24 6.25 16];
>> f=Newton(x,y)
f =
    .182711e-14-.482154e-14*t+1.00000*t^2-.169177e-14*t^3+.211471e-15*t^4
>> f=Newton(x,y,2.0)
f =
    4
```

表格中的数据点是按 $y=x^2$ 给出的，从插值函数在 $x=2.0$ 的值看出，插值函数有很高的精度。

3.5 等距节点插值法

在实际应用中，如果节点之间是等距的话，那么利用差分可得到以下一些很方便的插值方法。

3.5.1 利用差分的牛顿插值

差分分为前向差分、后向差分和中心差分三种，它们的记法及定义如下：

$$\Delta^n f(x_i)=\Delta^{n-1}f(x_{i+1})-\Delta^{n-1}f(x_i)$$

$$\nabla^n f(x_i)=\nabla^{n-1}f(x_i)-\nabla^{n-1}f(x_{i-1})$$

$$\delta^n f(x_i)=\delta^{n-1}f(x_{i+\frac{1}{2}})-\delta^{n-1}f(x_{i-\frac{1}{2}})$$

其中，Δ 代表前向差分；

Δ 代表后向差分；

δ 代表中心差分。

假设 $\Delta^0 f(x_i)=\nabla^0 f(x_i)=\delta^0 f(x_i)=f(x_i)$。为了方便计算，可构造如表 3-7 所示的差分表（$f_i=f(x_i)$）。

表 3-7 差分计算表格

	$\Delta(\nabla,\delta)$	$\Delta^2(\nabla^2,\delta^2)$	$\Delta^3(\nabla^3,\delta^3)$	...
$f(x_0)$				
$f(x_1)$	$\Delta f_0(\nabla f_1,\delta f_{1/2})$			
$f(x_2)$	$\Delta f_1(\nabla f_2,\delta f_{3/2})$	$\Delta^2 f_0(\nabla^2 f_2,\delta^2 f_1)$		
$f(x_3)$	$\Delta f_2(\nabla f_3,\delta f_{5/2})$	$\Delta^2 f_1(\nabla^2 f_3,\delta^2 f_2)$	$\Delta^3 f_0(\nabla^3 f_3,\delta^3 f_{3/2})$	
...	...	...	...	...

说明

表中每一项差分的计算公式如下，设第 i 行，第 j 列的均差的值为 $A(i,j)$，其中 $i=1,2,\cdots,n$，$j=1,2,\cdots,n$。

例如，

$$A(1,1)=\Delta f_0(\nabla f_1,\delta f_{1/2})\,,\quad A(3,2)=\Delta^2 f_1(\nabla^2 f_3,\delta^2 f_2)$$

则，

$$A(i,j)=A(i,j-1)-A(i-1,j-1)\,。$$

根据使用差分格式的不同，差分形式的牛顿差值又可细分为以下两种具体的差分形式。

1．前向牛顿插值

前向牛顿插值多项式可表示如下：

$$\begin{aligned}N(x)&=N(x_0+th)\\&=f(x_0)+\binom{t}{1}\Delta f(x_0)+\binom{t}{2}\Delta^2 f(x_0)+\cdots+\binom{t}{n}\Delta^n f(x_0)\end{aligned}$$

其中 h 叫做步长，$h=x_1-x_0$，且 t 的取值范围为 $0\leqslant t\leqslant n$。

在 MATLAB 中编程实现的前向牛顿差分插值法函数为：Newtonforward。

功能：求已知数据点的前向牛顿差分插值多项式。

调用格式：f = Newtonforward ($\boldsymbol{x}$, $\boldsymbol{y}$)或 f = Newtonforward ($\boldsymbol{x}$, $\boldsymbol{y}$, $x0$)。

其中，$\boldsymbol{x}$ 为已知数据点的 x 坐标向量；

$\boldsymbol{y}$ 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的前向牛顿差分插值多项式或在 $x0$ 处的插值。

在 MATLAB 实现前向牛顿差分插值的代码如下：

```
function f = Newtonforward(x,y,x0)
%求已知数据点的前向牛顿差分插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的前向牛顿差分插值多项式或在 x0 处的插值：f

syms t;
if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
    return;
end
f = y(1);
y1 = 0;
xx =linspace(x(1),x(n),(x(2)-x(1)));
```

```
if(xx ~= x)
    disp('节点之间不是等距的! ');
    return;
end
for(i=1:n-1)
    for(j=1:n-i)
       y1(j) = y(j+1)-y(j);
    end
    c(i) = y1(1);
    l = t;
    for(k=1:i-1)
       l = l*(t-k);
    end;

    f = f + c(i)*l/factorial(i);
    simplify(f);
    y = y1;

    if(i==n-1)
       if(nargin == 3)
          f = subs(f,'t',(x0-x(1))/(x(2)-x(1)));
       else
          f = collect(f);
          f = vpa(f, 6);
       end
    end
end
```

2. 后向牛顿插值

后向牛顿插值多项式可表示如下：

$$N(x)=N(x_0+th)$$
$$=f(x_n)+\binom{t}{1}\nabla f(x_n)+\binom{t+1}{2}\nabla^2 f(x_n)+\cdots+\binom{t+n-1}{n}\nabla^n f(x_n)$$

h 仍然是步长，$h=x_1-x_0$，但 t 的取值范围为 $-n\leqslant t\leqslant 0$。

在 MATLAB 中编程实现的后向牛顿差分插值法函数为：Newtonback。

功能：求已知数据点的后向牛顿差分插值多项式。

调用格式：f = Newtonback ($\boldsymbol{x}$, $\boldsymbol{y}$)或 f = Newtonback ($\boldsymbol{x}$, $\boldsymbol{y}$, $x0$)。

其中，$\boldsymbol{x}$ 为已知数据点的 x 坐标向量；

$\boldsymbol{y}$ 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的后向牛顿差分插值多项式或在 $x0$ 处的插值。

在 MATLAB 中实现后向牛顿差分插值的代码如下：

```
function f = Newtonback(x,y,x0)
%求已知数据点的后向牛顿差分插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的后向牛顿差分插值多项式或在 x0 处的插值：f
```

```
syms t;

if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等! ');
    return;
end

f = y(n);
y1 = 0;

xx =linspace(x(1),x(n),(x(2)-x(1)));
if(xx ~= x)
    disp('节点之间不是等距的! ');
    return;
end

for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = y(j)-y(j-1);
    end
    c(i) = y1(n);
    l = t;
    for(k=1:i-1)
        l = l*(t+k);
    end;
    f = f + c(i)*l/factorial(i);
    simplify(f);
    y = y1;

    if(i==n-1)
        if(nargin == 3)
            f = subs(f,'t',(x(n)-x0)/(x(2)-x(1)));
        else
            f = collect(f);
            f = vpa(f, 6);
        end
    end
end
```

【例 3-11】 利用差分的牛顿插值法应用实例。根据下表所列的数据点求出其差分形式的牛顿插值多项式，并计算当 $x=1.55$ 的 y 值。

x	1	1.2	1.4	1.6	1.8
y	0.8415	0.9320	0.9854	0.9996	0.9738

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[0.8415  0.9320  0.9854  0.9996  0.9738];
>> f=Newtonforward(x,y)
```

```
f =
    .287954*t-.275409e-1*t^2+.278262e-2*t^3-.161141e-3*t^4
>> f=Newtonback(x,y)
f =
    .159941*t-.961895e-2*t^2+.204368e-3*t^3-.161141e-3*t^4
>> f= Newtonforward(x,y,1.55)
f =
    0.9998
>> f= Newtonback(x,y,1.55)
f =
    0.9998
```

表格中的数据点是按 $y=\sin x$ 给出的，而 sin(1.55)=0.9998，从插值函数在 x=1.55 处的值看出，牛顿差分插值函数是很准确的。

3.5.2 高斯插值

高斯插值是从中间的节点开始的，它也有前向和后向之分。

1. 前向高斯插值

对于节点个数为偶数的情况（$n=2m$），设节点编号为 x_i，x_{i+1}，x_{i-1}，…，x_{i+m}，x_{i-m}，前向高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i+\frac{1}{2}})+\binom{t}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i+\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m} f(x_i)$$

对于节点个数为奇数的情况（$n=2m+1$），设节点编号为 x_i，x_{i+1}，x_{i-1}，…，x_{i+m}，x_{i-m}，x_{i+m+1}，高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i+\frac{1}{2}})+\binom{t}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i+\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m} f(x_i)+\binom{t+m}{2m+1}\delta^{2m+1} f(x_{i+\frac{1}{2}})$$

2. 后向高斯插值

对于节点个数为偶数的情况（$n=2m$），设节点编号为 x_i，x_{i-1}，x_{i+1}，…，x_{i-m}，x_{i+m}，注意节点编号顺序，后向高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i-\frac{1}{2}})+\binom{t+1}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i-\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i-\frac{1}{2}})+\binom{t+m}{2m}\delta^{2m} f(x_i)$$

对于节点个数为奇数的情况（$n=2m+1$），设节点编号为 x_i，x_{i-1}，x_{i+1}，…，x_{i-m}，x_{i+m}，x_{i-m-1}，高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i-\frac{1}{2}})+\binom{t+1}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i-\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1}f(x_{i-\frac{1}{2}})+\binom{t+m}{2m}\delta^{2m}f(x_i)+\binom{t+m}{2m+1}\delta^{2m+1}f(x_{i-\frac{1}{2}})$$

实际应用的比较多的是由高斯公式衍生出来的斯特林公式和贝塞尔公式。

（1）用于奇数个节点插值的斯特林公式

斯特林公式是对前向高斯插值和后向高斯插值取平均值得到的：

$$S(x)=f(x_i)+\frac{1}{2}\binom{t}{1}(\delta f(x_{i+\frac{1}{2}})+\delta f(x_{i-\frac{1}{2}}))+\frac{1}{2}[\binom{t}{2}+\binom{t+1}{2}]\delta^2 f(x_i)+$$
$$\frac{1}{2}\binom{t+1}{3}(\delta^3 f(x_{i+\frac{1}{2}})+\delta^3 f(x_{i-\frac{1}{2}}))+\cdots+$$
$$\frac{1}{2}[\binom{t+m-1}{2m-1}+\binom{t+m}{2m-1}]\delta^{2m-1}f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m}f(x_i)$$

这个公式只适合奇数个节点插值。

（2）用于偶数个节点插值的贝塞尔公式

贝塞尔公式是高斯公式的另一种平均：

$$B(x)=\frac{1}{2}(f(x_i)+f(x_{i+1}))+\frac{1}{2}[\binom{t}{1}+\binom{t-1}{1}]\delta f(x_{i+\frac{1}{2}})+$$
$$\frac{1}{2}\binom{t}{2}(\delta^2 f(x_i)+\delta^2 f(x_{i+1}))+$$
$$\frac{1}{2}[\binom{t+1}{3}+\binom{t}{3}]\delta^3 f(x_{i+\frac{1}{2}})+\cdots+$$
$$\frac{1}{2}[\binom{t+m}{2m+1}+\binom{t+m-1}{2m+1}]\delta^{2m+1}f(x_{i+\frac{1}{2}})$$

这个公式只适合偶数个节点插值。

在 MATLAB 中编程实现的高斯插值法函数为：Gauss。

功能：求已知数据点的高斯插值多项式。

调用格式：f = Gauss (x, y)或f = Gauss (x, y, $x0$)

其中，x 为已知数据点的 x 坐标向量；

y 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的高斯插值多项式或在 $x0$ 处的插值

在 MATLAB 中实现高斯插值的代码如下：

```
function f = Gauss(x,y,x0)
%求已知数据点的高斯插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
```

```
%插值点的 x 坐标：x0
%求得的高斯插值多项式或在 x0 处的插值：f

if(length(x) == length(y))
    n = length(x);
    end
else
    disp('x 和 y 的维数不相等！');
    return;
end

xx =linspace(x(1),x(n),(x(2)-x(1)));
if(xx ~= x)
    disp('节点之间不是等距的！');
    return;
end

if( mod(n,2) ==1)                          %n 为奇数时采用斯特林插值
    if(nargin == 2)
        f = GStirling(x,y,n);              %只须求出插值多项式
    else if(nargin == 3)
        f = GStirling(x,y,n,x0);           %求出在 x0 处的值
        end
    end
else                                       %n 为偶数时采用贝塞尔插值
    if(nargin == 2)
        f = GBessel(x,y,n);
    else if(nargin == 3)
        f = GBessel(x,y,n,x0);
        end
    end
end

function f = GStirling(x,y,n,x0)           %斯特林插值
syms t;
nn = (n+1)/2;
f = y(nn);

for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = y(j)-y(j-1);
    end
    if(mod(i,2)==1)
        c(i) = (y1((i+n)/2)+y1((i+n+2)/2))/2;
    else
        c(i) = y1((i+n+1)/2)/2;
    end


    if(mod(i,2)==1)
        l = t+(i-1)/2;
        for(k=1:i-1)
            l = l*(t+(i-1)/2-k);
        end
    else
```

```
        l_1 = t+i/2-1;
        l_2 = t+i/2;
        for(k=1:i-1)
            l_1 = l_1*(t+i/2-1-k);
            l_2 = l_2*(t+i/2-k);
        end
        l = l_1 + l_2;
    end

    l = l/factorial(i);
    f = f + c(i)*l;
    simplify(f);
    f = vpa(f, 6);
    y = y1;

    if(i==n-1)
        if(nargin == 3)
            f = subs(f,'t',(x0-x(nn))/(x(2)-x(1)));
        end
    end
end

function f = GBessel(x,y,n,x0)                          %贝塞尔插值
syms t;
nn = n/2;
f = (y(nn)+y(nn+1))/2;

for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = y(j)-y(j-1);
    end
    if(mod(i,2)==1)
        c(i) = y1((i+n+1)/2)/2;
    else
        c(i) = (y1((i+n)/2)+y1((i+n+2)/2))/2;
    end


    if(mod(i,2)==0)
        l = t+i/2-1;
        for(k=1:i-1)
            l = l*(t+i/2-1-k);
        end
    else
        l_1 = t+(i-1)/2;
        l_2 = t+(i-1)/2-1;
        for(k=1:i-1)
            l_1 = l_1*(t+(i-1)/2-k);
            l_2 = l_2*(t+(i-1)/2-1-k);
        end
        l = l_1 + l_2;
    end

    l = l/factorial(i);
    f = f + c(i)*l;
    simplify(f);
```

```
        f = vpa(f, 6);
        y = y1;

        if(i==n-1)
            if(nargin == 4)
                f = subs(f,'t',(x0-x(nn))/(x(2)-x(1)));
            end
        end
    end
```

【例 3-12】 高斯插值法应用实例。根据下表所列的数据点求出其高斯插值多项式，并计算当 $x=1.5$ 时的 y 值。

x	1	1.2	1.4	1.6	1.8
y	0	0.2630	0.4854	0.6781	0.8480

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;;
>> y=[0  0.2630  0.4854  0.6781  0.8480];
>> f=Gauss(x,y)
f =
    .485427+.207519*t-.743684e-2*t*(t-1.)-.743684e-2*t*(t+1.)+.149349e-2
*t*(t+1.)*(t-1.)-.805703e-4*t*(t+1.)*(t-1.)*(t-2.)-.805703e-4*t*(t+1.)*(t+2.
)*(t-1.)
>> f=Newtonback(x,y)
f =
    .159941*t-.961895e-2*t^2+.204368e-3*t^3-.161141e-3*t^4
>>f=Gauss(x,y,1.5)
f =
    0.5849
```

由于 x 和 y 是奇数维的，高斯函数将调用斯特林公式插值。表格中的数据点是按 $y=\log_2 x$ 给出的，而 $\log_2 1.5=0.5850$，从插值函数在 $x=1.5$ 处的值看出，插值结果与理论值符合得很好。

3.6 埃尔米特插值法

埃尔米特插值法满足在节点上等于给定函数值，而且在节点上的导数值也等于给定的导数值。对于有高阶导数的情况，埃尔米特插值多项比较复杂，在实际应用中，常常遇到的是函数值与一阶导数值给定的情况。在此种情况下，n 个节点 x_1，x_2，…，x_n 的埃尔米特插值多项式的表达式如下：

$$H(x)=\sum_{i=1}^{n} h_i[(x_i-x)(2a_i y_i-y_i')+y_i]$$

其中 $y_i=y(x_i)$，$y_i'=y'(x_i)$

$$h_i=\prod_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)^2,\quad a_i=\sum_{\substack{j=1\\ j\neq i}}^{n}\frac{1}{x_i-x_j}$$

在 MATLAB 中编程实现的埃尔米特插值法函数为：Hermite。

功能：求已知数据点的埃尔米特插值多项式。

调用格式：f = Hermite (x, y, y_1)或 f = Hermite (x, y, y_1,$x0$)。

其中，x 为已知数据点的 x 坐标向量；

y 为已知数据点的 y 坐标向量；

y_1 为已知数据点的导数向量；

$x0$ 为插值点的 x 坐标；

f 为求得的埃尔米特插值多项式或在 $x0$ 处的插值。

在 MATLAB 中实现埃尔米特插值的代码如下：

```
function f = Hermite(x,y,y_1,x0)
%求已知数据点的埃尔米特插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%已知数据点的导数向量：y_1
%插值点的 x 坐标：x0
%求得的埃尔米特插值多项式或在 x0 处的插值：f

syms t;
f = 0.0;

if(length(x) == length(y))
    if(length(y) == length(y_1))
        n = length(x);
    else
        disp('y 和 y 的导数的维数不相等！');
        return;
    end
else
    disp('x 和 y 的维数不相等！');
    return;
end

for i=1:n
    h = 1.0;
    a = 0.0;
    for j=1:n
        if( j ~= i)
            h = h*(t-x(j))^2/((x(i)-x(j))^2);
            a = a + 1/(x(i)-x(j));
        end
    end

    f = f + h*((x(i)-t)*(2*a*y(i)-y_1(i))+y(i));

    if(i==n)
        if(nargin == 4)
            f = subs(f,'t',x0);
        else
            f = vpa(f,6);
        end
```

```
    end
end
```

【例 3-13】 埃尔米特插值法应用实例。根据下表所列的数据点求出其埃尔米特插值多项式，并计算当 $x=1.44$ 时的 y 值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416
y'	0.5000	0.4564	0.4226	0.3953	0.3727

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;;
>> y=[1  1.0954  1.1832  1.2649 1.3416];
>> y_1=[0.5  0.4564 0.4226 0.3953 0.3727];
>> f=Hermite(x,y,y_1)
f = 678.168*(t-1.20000)^2*(t-1.40000)^2*(t-1.60000)^2*(t-1.80000)^2*(-20.3333+
21.3333*t)+10850.7*(t-1.)^2*(t-1.40000)^2*(t-1.60000)^2*(t-1.80000)^2*(-10.4066+
9.58511*t)+24414.1*(t-1.)^2*(t-1.20000)^2*(t-1.60000)^2*(t-1.80000)^2*(.591580+
.422600*t)+10850.7*(t-1.)^2*(t-1.20000)^2*(t-1.40000)^2*(t-1.80000)^2*(17.4979-
10.1456*t)+678.168*(t-1.)^2*(t-1.20000)^2*(t-1.40000)^2*(t-1.60000)^2*(50.9823-
27.5781*t)
>> f=Hermite(x,y,y_1,1.44)
f =
    1.2000
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，而 $\sqrt{1.44}=1.22$，从插值函数在 x=1.44 处的值看出，插值结果偏差也不大，如果增加插值点的个数，精度会更高。

3.7 有理分式插值法

所谓有理分式插值，就是寻求如下形式的有理分式函数：

$$R_{m.n}=\frac{a_0+a_1x+\cdots+a_mx^m}{b_0+b_1x+\cdots+b_nx^n}$$

满足 $R_{m.n}(x_i)=f(x_i)$。

在已知 $f(x_i)$ 的情况下，可以通过以下两种算法来得到 $R_{m.n}$。

1. 倒差商-连分式算法

定义例差商如下：

$$f[x_0,x_m]^{-1}=\frac{x_m-x_0}{f(x_m)-f(x_0)}$$

一般情况下，k 阶倒差商定义为：

$$f[x_0,x_1,\cdots,x_{k-1},x_m]^{-1}=\frac{x_m-x_{k-1}}{f[x_0,x_1,\cdots,x_{k-2},x_m]-f[x_0,x_1,\cdots,x_{k-1}]}$$

显然，倒差商就是均差的倒数。仿照均差表，可构造如表 3-8 所示的倒差商表。

表 3-8　倒差商计算表格

	一阶均差	二阶均差	三阶均差	…	n 阶均差
$f(x_0)$					
$f(x_1)$	$f[x_0,x_1]^{-1}$				
$f(x_2)$	$f[x_0,x_2]^{-1}$	$f[x_0,x_1,x_2]^{-1}$			
$f(x_3)$	$f[x_0,x_3]^{-1}$	$f[x_0,x_1,x_3]^{-1}$	$f[x_0,x_1,x_2,x_3]^{-1}$		
…	…	…	…		
$f(x_n)$	$f[x_0,x_n]^{-1}$	$f[x_0,x_1,x_n]^{-1}$	$f[x_0,x_1,x_2,x_n]^{-1}$	…	$f[x_0,x_1,\cdots,x_n]^{-1}$

则最终的插值公式可表示为：

$$R_{m.n}=f(x_0)+\cfrac{x-x_0}{f[x_0,x_1]^{-1}+\cfrac{x-x_1}{f[x_0,x_1,x_2]^{-1}+\cfrac{x-x_2}{\ddots\cfrac{x-x_{n-1}}{f[x_0,x_1,x_2,\cdots,x_n]^{-1}}}}}$$

注意

对给定的数据(x_i,y_i)组，并不一定存在对应的有理分式插值函数。

在 MATLAB 中编程实现的有理分式形式的倒差商插值法函数为：DCS。

功能：用倒差商算法求已知数据点的有理分式形式的插值分式。

调用格式：f = DCS (x,y)或f = DCS (x,y,$x0$)

其中，x 为已知数据点的 x 坐标向量；

y 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的有理分式形式的插值分式或在 $x0$ 处的插值。

在 MATLAB 中实现有理分式插值（倒差商算法）的代码如下：

```
function f = DCS(x,y,x0)
%用倒差商算法求已知数据点的有理分式形式的插值分式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的有理分式形式的插值分式或在 x0 处的插值：f

syms t;
if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
    return;
end

c(1) = y(1);
```

```
for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = (x(j)-x(i))/(y(j)-y(i));
    end
    c(i+1) = y1(i+1);
    y = y1;
end

f = c(n);
for(i=1:n-1)
    f = c(n-i) + (t-x(n-i))/f;
    f = vpa(f,6);
    if(i==n-1)
        if(nargin == 3)
            f = subs(f,'t',x0);
        else
            f = vpa(f,6);
        end
    end
end;
```

【例 3-14】 有理分式插值法应用实例之一。根据下表所列的数据点，用有理分式的倒差商形式进行插值，并计算当 $x=1.44$ 的 y 值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;;
>> y=[1  1.0954  1.1832  1.2649  1.3416];
>> f=DCS(x,y)
f =
    1.+(t-1.)/(2.09545+(t-1.20000)/(2.27866+(t-1.40000)/(1.83429+.383649*t)))
>> f=DCS(x,y,1.44)
f =
    1.2000
```

表格中的数据点也是按 $y=\sqrt{x}$ 给出的，从插值结果看，偏差也不大，如果增加插值点的个数，精度会更高。

2. Neville 迭代算法

由 Neville 迭代算法得到 $R_{m.n}$ 的迭代公式如下：

$$N_{i,1}(x)=f(x_i),\ N_{i,0}(x)=0$$

$$N_{i,k}(x)=N_{i,k-1}(x)+\frac{N_{i,k-1}(x)-N_{i-1,k-1}(x)}{\dfrac{x-x_{i-k}}{x-x_i}\left[1-\dfrac{N_{i,k-1}(x)-N_{i-1,k-1}(x)}{N_{i,k-1}(x)-N_{i-1,k-2}(x)}\right]}$$

$$1\leqslant k<i,\ i=1,2,\cdots,n$$

将算法列成表格，如下表所示。

i	$N_{i,1}$	$N_{i,2}$	$N_{i,3}$	···
1	$f(x_1)$			
2	$f(x_2)$	$N_{2,2}$		
3	$f(x_3)$	$N_{3,2}$	$N_{3,3}$	
4	$f(x_4)$			
···	···	···	···	

最终的 $N_{i,i-1}$ 就是所要的插值分式。

注意

对给定的数据 (x_i, y_i) 组，用 Neville 算法不一定能得到有理分式插值函数，在迭代过程中可能出现分母为零的情况。

在 MATLAB 中编程实现的有理分式形式的 Neville 插值法函数为：Neville。

功能：用 Neville 算法求已知数据点的有理分式形式的插值分式。

调用格式：f = Neville ($\boldsymbol{x}$, $\boldsymbol{y}$)或 f = Neville ($\boldsymbol{x}$, $\boldsymbol{y}$, $x0$)。

其中，$\boldsymbol{x}$ 为已知数据点的 x 坐标向量；

$\boldsymbol{y}$ 为已知数据点的 y 坐标向量；

$x0$ 为插值点的 x 坐标；

f 为求得的有理分式形式的插值分式或是在 $x0$ 处的插值

在 MATLAB 中实现有理分式插值（Neville 算法）的代码如下：

```
function f = Neville(x,y,x0)
%用 Neville 算法求已知数据点的有理分式形式的插值分式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的有理分式形式的插值分式或在 x0 处的插值：f

syms t;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end

y1(1:n) = t;
for(i=1:n-1)
    for(j=i+1:n)
        if(j==2)
            y1(j) = y(j)+(y(j)-y(j-1))/((t-x(j-i))/(t-x(j)))*
                                        (1-(y(j)-y(j- 1))/y(j));
        else
            y1(j) = y(j)+(y(j)-y(j-1))/((t-x(j-i))/(t-x(j)))*
                                        (1-(y(j)-y(j- 1))/(y(j)-y(j-2)));
        end
    end
```

```
    y = y1;
    if(i==n-1)
        if(nargin == 3)
            f = subs(y(n-1),'t',x0);
        else
            f = vpa(y(n-1),6);
        end
    end
end
```

【例 3-15】 有理分式插值法应用实例之二。根据下面的数据点，用有理分式的 Neville 形式进行插值，并计算当 $x=1.44$ 的 y 值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;;
>> y=[1  1.0954  1.1832  1.2649  1.3416];
>> f= Neville(x,y);                          %表达式过长，在此不再列出
>> f= Neville(x,y,1.44)
```

输出结果为：

```
f =
    1.0030
```

表格中的数据点也是按 $y=\sqrt{x}$ 给出的，从插值结果看，偏差比较大，一般来说迭代算法的精度很难保证。

3.8 函数逼近与曲线拟合

在区间 $[a,b]$ 上已知一连续函数 $f(x)$，如果 $f(x)$ 的表达式太过复杂不利于用计算机来进行计算，自然而然地想到用一简单函数去近似 $f(x)$，这就是函数逼近问题。

如果 $f(x)$ 的表达式未知，只知道描述 $f(x)$ 的一条曲线，这就是曲线拟合问题。

和插值问题不同的是，逼近与拟合并不要求逼近函数在已知点上的值一定得等于原函数的函数值，而是按照某种标准使得两者的差值达到最小。

因为任何连续函数都可以泰勒展开，通过取展开项的前几项就构成了泰勒逼近，泰勒逼近十分简单，但它收敛慢，精度也不高，下面介绍几种比较实用的逼近方法。

3.8.1 切比雪夫逼近

当一个连续函数定义在区间 $[-1,1]$ 上时，它可以展开成切比雪夫级数。即：

$$f(x)=\sum_{n=0}^{\infty} f_n T_n(x)$$

其中 $T_n(x)$ 为 n 次切比雪夫多项式，具体表达式可通过递推得出：

$$T_0(x)=1,\ T_1(x)=x$$
$$T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)$$

它们之间满足如下的正交关系：

$$\int_{-1}^{1}\frac{T_n(x)T_m(x)\mathrm{d}x}{\sqrt{1-x^2}}=\begin{cases}0, & n\neq m\\ \dfrac{\pi}{2}, & n=m\neq 0\\ \pi, & n=m=0\end{cases}$$

在实际应用中，可根据所需的精度来截取有限项数。切比雪夫级数中的系数由下式决定：

$$f_0=\frac{1}{\pi}\int_{-1}^{1}\frac{f(x)}{\sqrt{1-x^2}}\mathrm{d}x$$
$$f_n=\frac{2}{\pi}\int_{-1}^{1}\frac{T_n(x)f(x)}{\sqrt{1-x^2}}\mathrm{d}x$$

在 MATLAB 中编程实现的切比雪夫逼近法函数为：Chebyshev。

功能：用切比雪夫多项式逼近已知函数。

调用格式：f = Chebyshev (y,k)或 f = Chebyshev (y,k,$x0$)

其中，y 为已知函数；

k 为逼近已知函数所需项数；

$x0$ 是逼近点的 x 坐标；；

f 是求得的切比雪夫逼近多项式或是在 $x0$ 处的逼近值。

在 MATLAB 中实现切比雪夫逼近的代码如下：

```
function f = Chebyshev(y,k,x0)
%用切比雪夫多项式逼近已知函数
%已知函数：y
%逼近已知函数所需项数：k
%逼近点的 x 坐标：x0
%求得的切比雪夫逼近多项式或在 x0 处的逼近值：f

syms t;
T(1:k+1) = t;
T(1) = 1;
T(2) = t;
c(1:k+1) = 0.0;

c(1)=int(subs(y,findsym(sym(y)),sym('t'))*T(1)/sqrt(1-t^2),t,-1,1)/pi;
c(2)=2*int(subs(y,findsym(sym(y)),sym('t'))*T(2)/sqrt(1-t^2),t,-1,1)/pi;
f = c(1)+c(2)*t;

for i=3:k+1
    T(i) = 2*t*T(i-1)-T(i-2);
    c(i) = 2*int(subs(y,findsym(sym(y)),sym('t'))*T(i)/sqrt(1-t^2),t,-1,1)/2;
    f = f + c(i)*T(i);
    f = vpa(f,6);
```

```
    if(i==k+1)
        if(nargin == 3)
            f = subs(f,'t',x0);
        else
            f = vpa(f,6);
        end
    end
end
```

【例 3-16】 切比雪夫逼近应用实例。用切比雪夫公式（取 6 项）逼近函数 $\frac{1}{2-x}$，并求当 $x=0.5$ 时的函数值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> f=Chebyshev('1/(2-x)',6)
f =
.455803+.277012*t+.243094*t^2+.647768e-1*t*(2.*t^2-1.)+.173569e-1*t*(2.*
t*(2.*t^2-1.)-1.*t)+.501051e-2*t*(2.*t*(2.*t*(2.*t^2-1.)-1.*t)-2.*t^2+1.)+.1
34256e-2*t*(2.*t*(2.*t*(2.*t*(2.*t^2-1.)-1.*t)-2.*t^2+1.)-2.*t*(2.*t^2-1.)+t
)
>> f= f=Chebyshev('1/(2-x)',6,0.5)
f =
    0.6293
```

事实上函数的准确值为 $1/(2-0.5)=0.6667$，逼近结果比较接近。

3.8.2 勒让德逼近

勒让德逼近也要求被逼近函数定义在区间[-1,1]上，勒让德多项式也可通过递推来定义：

$$P_0(x)=1,\ P_1(x)=x$$
$$(n+1)P_{n+1}(x)=(2n+1)xP_n(x)-nP_{n-1}(x)$$

它们之间满足如下的正交关系：

$$\int_{-1}^{1}P_n(x)P_m(x)\mathrm{d}x=\begin{cases}0, & n\neq m\\ \dfrac{1}{2n+1}, & n=m\end{cases}$$

勒让德级数中的系数由下式决定：

$$f_n=\frac{2n+1}{2}\int_{-1}^{1}P_n(x)f(x)\mathrm{d}x$$

在 MATLAB 中编程实现的勒让德逼近法函数为：Legendre。

功能：用勒让德多项式逼近已知函数。

调用格式：f = Legendre (**y**,***k***)或 f = Legendre (**y**,***k***,*x*0)

其中，***y*** 为已知函数；

k 为逼近已知函数所需项数；

*x*0 为逼近点的 *x* 坐标；；

f为求得的勒让德逼近多项式或在$x0$处的逼近值。

在 MATLAB 中实现勒让德逼近的代码如下：

```
function f = Legendre(y,k,x0)
%用勒让德多项式逼近已知函数
%已知函数：y
%逼近已知函数所需项数：k
%逼近点的 x 坐标：x0
%求得的勒让德逼近多项式或在 x0 处的逼近值：f

syms t;
P(1:k+1) = t;
P(1) = 1;
P(2) = t;
c(1:k+1) = 0.0;

c(1)=int(subs(y,findsym(sym(y)),sym('t'))*P(1),t,-1,1)/2;
c(2)=int(subs(y,findsym(sym(y)),sym('t'))*P(2),t,-1,1)/2;
f = c(1)+c(2)*t;

for i=3:k+1
    P(i) = ((2*i-3)*P(i-1)*t-(i-2)*P(i-2))/(i-1);
    c(i) = int(subs(y,findsym(sym(y)),t)*P(i),t,-1,1)/2;
    f = f + c(i)*P(i);

    if(i==k+1)
        if(nargin == 3)
            f = subs(f,'t',x0);
        else
            f = vpa(f,6);
        end
    end
end
```

【例 3-17】 勒让德逼近应用实例。**用**勒让德公式（取 6 项）逼近函数$\frac{1}{2-x}$，并求当$x=0.5$时的函数值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> f=Legendre('1/(2-x)',6)
f =
.539128+.955158e-1*t+.305351e-1*t^2+.154825e-2*(7.50000*t^2-2.50000)*t+.
275682e-3*(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)*t+.565953e-4*(2.25000*
(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)*t-10.1250*t^2+3.37500)*t+.116697e
-4*(2.20000*(2.25000*(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)*t-10.1250*t
^2+3.37500)*t-2.93333*(7.50000*t^2-2.50000)*t+5.86667*t)*t
>> f=Legendre('1/(2-x)',6,0.5)
f =
    0.5935
```

从逼近结果看，函数的准确值为$1/(2-0.5)=0.6667$，与上例相比可看出勒让德级数的收敛速度比切比雪夫级数慢。

3.8.3 帕德逼近

帕德逼近是一种有理分式逼近。逼近公式如下：

$$f(x) \approx \frac{\sum_{k=0}^{L} p_k x^k}{1+\sum_{k=1}^{M} q_k x^k}$$

大量实验表明，当 $L+M$ 为常数时，取 $L=M$，帕德逼近精确度最好，而且速度最快。此时分子与分母中的系数可通过以下方法求解。

首先，求解线性方程组 $\boldsymbol{Aq}=\boldsymbol{b}$，得到 $(q_1,q_2,\cdots,q_n)$ 的值。其中

$$\boldsymbol{A}=\begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ a_2 & a_3 & \cdots & a_{n+1} \\ \vdots & \vdots & & \vdots \\ a_n & a_{n+1} & \cdots & a_{2n-1} \end{bmatrix}, \boldsymbol{q}=\begin{bmatrix} q_n \\ q_{n-1} \\ \vdots \\ q_1 \end{bmatrix}, \boldsymbol{b}=\begin{bmatrix} -a_{n+1} \\ -a_{n+2} \\ \vdots \\ -a_{2n} \end{bmatrix}$$

$$a_0=f(0),\ a_n=\frac{1}{n!}\frac{\mathrm{d}^n f(0)}{\mathrm{d}x^n}$$

然后通过下式求出 $(p_0,p_1,\cdots,p_n)$ 的值。

$$p_0=a_0,\ q_0=1,\ p_n=\sum_{i=0}^{n} q_i a_{n-i}$$

注意，函数的帕德逼近不一定存在。

在 MATLAB 中编程实现的帕德逼近法函数为：Pade。

功能：用帕德形式的有理分式逼近已知函数。

调用格式：f = Pade (y, n)或 f = Pade (y, n, $x0$)

其中，y 为已知函数；

n 为帕德有理分式的分母多项式的最高次数；

$x0$ 为逼近点的 x 坐标；；

f 为求得的帕德有理分式或在 $x0$ 处的逼近值。

在 MATLAB 中实现函数的 Pade 逼近的代码如下：

```
function f = Pade(y,n,x0)
%用帕德形式的有理分式逼近已知函数
%已知函数：y
%帕德有理分式的分母多项式的最高次数：n
%逼近点的 x 坐标：x0
%求得的帕德有理分式或在 x0 处的逼近值：f

syms t;
A = zeros(n,n);
q = zeros(n,1);
p = zeros(n+1,1);
b = zeros(n,1);
```

```
yy = 0;
a(1:2*n) = 0.0;

for(i=1:2*n)
    yy = diff(sym(y),findsym(sym(y)),n);
    a(i) = subs(sym(yy), findsym(sym(yy)), 0.0)/factorial(i);
end;

for(i=1:n)
    for(j=1:n)
        A(i,j)=a(i+j-1);
    end;
    b(i,1) = -a(n+i);
end;

q = A\b;
p(1) = subs(sym(y),findsym(sym(y)),0.0);
for(i=1:n)
    p(i+1) = a(n)+q(i)*subs(sym(y),findsym(sym(y)),0.0);
    for(j=2:i-1)
        p(i+1)=p(i+1)+q(j)*a(i-j);
    end
end

f_1 = 0;
f_2 = 1;
for(i=1:n+1)
    f_1 = f_1 + p(i)*(t^(i-1));
end
for(i=1:n)
    f_2 = f_2 + q(i)*(t^i);
end

if(nargin == 3)
    f = f_1/f_2;
    f = subs(f,'t',x0);
else
    f = f_1/f_2;
    f = vpa(f,6);
end
```

【例 3-18】 帕德逼近应用实例。用勒让德公式(取 4 项)逼近函数 $\frac{1}{1-x}$，并求当 $x=0.5$ 时的函数值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> f=Pade('1/(1-x)',4)
f =
(1.+1.00060*t+.988095*t^2+.821429*t^3+2.92857*t^4)/(1.+.595238e-3*t-.119
048e-1*t^2+.107143*t^3-.500000*t^4)
>> f=Pade('1/(1-x)',4,0.5)
f =
    2.0757
```

从逼近结果看，函数的准确值为 $1/(1-0.5)=2$，而用 4 次有理分式的帕德逼近已经达

到了很高的精度了。

3.8.4 傅里叶逼近

当被逼近函数为周期函数时，用代数多项式来逼近效率不高，而且误差也较大，这时用三角多项式来逼近是较好的选择。

三角多项式逼近也就是傅里叶逼近，任一周期函数都可以展开为傅里叶级数，通过选取有限的展开项数，就可以达到所需精度的逼近效果。

下面介绍连续周期函数和离散周期函数的傅里叶逼近的具体做法。

1. 连续周期函数的傅里叶逼近

对于连续周期函数，只要计算出其傅里叶展开系数即可。

在 MATLAB 中编程实现的连续周期函数的傅里叶逼近法函数为：FZZ。

功能：用傅里叶级数逼近已知的连续周期函数。

调用格式：[*A*0, *A*, *B*] = FZZ (func, *T*, *n*)。

其中，func 为已知函数；

T 为已知函数的周期；

n 为展开级数的项数；

*A*0 为展开后的常数项；

A 为展开后的余弦项系数；

B 为展开后的正弦项系数。

在 MATLAB 中实现连续周期函数的傅里叶逼近的代码如下：

```
function [A0,A,B]=FZZ(func,T, n)
%用傅里叶级数逼近已知的连续周期函数
%已知函数：func
%已知函数的周期：T
%展开项数：n
%展开后的常数项：A0
%展开后的余弦项系数：A
%展开后的正弦项系数：B

syms t;
func = subs(sym(func), findsym(sym(func)),sym('t'));
A0=int(sym(func),t,-T/2,T/2)/T;
for(k=1:n)
    A(k)=int(func*cos(2*pi*k*t/T), t, -T/2,T/2)*2/T;
    A(k)=vpa(A(k),4);
    B(k)=int(func*sin(2*pi*k*t/T), t, -T/2,T/2)*2/T;
    B(k)=vpa(B(k),4);
end
```

【例 3-19】 傅里叶逼近应用实例。用傅里叶级数（取 5 项）逼近函数 x，输出系数值。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> [A0,A,B]=FZZ('x',2*pi,5)
A0 =
     0
A =
     [ 0., 0., 0., 0., 0.]
B =
     [    -2.,    -1., -.6667, -.5000, -.4000]
```

结果表明，如果取 5 项逼近函数 $y=x$，有下面的式子：

$$x \approx -2\sin x - \sin 2x - 0.6667\sin 3x - 0.5\sin 4x - 0.4\sin 5x \text{。}$$

2．离散周期数据的傅里叶逼近

对于离散周期的数据拟合，只要计算出其离散傅里叶展开系数即可。其展开公式为：

$$y = \sum_{k=0}^{n-1} c_i e^{ikx}$$

而，

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n e^{-ikn\frac{2\pi}{N}}$$

在 MATLAB 中编程实现的离散周期数据点傅里叶逼近法函数为：DFF。

功能：离散周期数据点的傅里叶逼近。

调用格式：c = DFF (f, N)。

其中，f 为已知离散数据点；

N 为离散数据点的个数；

c 为离散傅里叶逼近系数。

在 MATLAB 中实现离散周期函数的傅里叶逼近的代码如下：

```
function c = DFF(f,N)
%离散周期数据点的傅里叶逼近
%已知离散数据点：f
%离散数据点的个数：N
%逼近系数：c

c(1:N)=0;
for(m=1:N)
    for(n=1:N)
        c(m)=c(m)+f(n)*exp(-i*m*n*2*pi/N);
    end
    c(m)=c(m)/N;
end
```

【例 3-20】 离散傅里叶逼近应用实例。对下列数据点进行离散傅里叶变换。

N	1	2	3	4	5	6
y	0.8415	0.9093	0.1411	-0.7568	-0.9589	-0.2794

解：在 MATLAB 命令窗口中输入以下命令：

```
>> y=[0.8415  0.9093  0.1411  -0.7568  -0.9589  -0.2794];
```

```
>> c = DFT(y,6)
```

输出结果为：

```
c =
  Columns 1 through 3
  -0.0926 - 0.5003i  -0.0260 - 0.0194i  -0.0251 + 0.0000i
  Columns 4 through 6
  -0.0260 + 0.0194i  -0.0926 + 0.5003i  -0.0172 - 0.0000i
```

经过离散傅里叶变换，y 就变成了 c 了，仔细观察向量 c，可发现它的分量中有两对共轭复数，这个特点有利于数据的存储。

3.8.5 多项式曲线拟合

对给定的试验数据点 $(x_i, y_i)(i=1,2,\cdots,N)$，可构造 m 次多项式：

$$P(x)=a_0+a_1x+\cdots+a_mx^m \qquad (m<N)$$

由曲线拟合的定义，应该使得下式取极小值：

$$\sum_{i=1}^{N}[\sum_{j=0}^{m}a_jx_i^j-y_i]^2$$

通过简单的运算可得出系数是下面线性方程组的解：

$$\begin{bmatrix} c_0 & c_1 & \cdots & c_m \\ c_1 & c_2 & \cdots & c_{m+1} \\ \vdots & \vdots & \cdots & \vdots \\ c_m & c_{m+1} & \cdots & c_{2m} \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}=\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix}$$

其中，

$$\begin{cases} c_k=\sum_{i=1}^{N}x_i^k,(k=0,1,\cdots,2m) \\ b_k=\sum_{i=1}^{N}y_ix_i^k,(k=0,1,\cdots,m) \end{cases}$$

在 MATLAB 中编程实现的多项式曲线拟合函数为：multifit。

功能：离散试验数据点的多项式曲线拟合。

调用格式：***A*** = multifit (***X***,***Y***,*m*)。

其中，***X*** 为试验数据点的 x 坐标向量；

Y 为试验数据点的 y 坐标向量；

m 为拟合多项式的次数；

A 为拟合多项式的系数向量。

在 MATLAB 中实现多项式曲线拟合的代码如下：

```
function A=multifit(X,Y,m)
%离散试验数据点的多项式曲线拟合
%试验数据点的 x 坐标向量：X
```

```
%试验数据点的 y 坐标向量：Y
%拟合多项式的次数：m
%拟合多项式的系数向量：A

N=length(X);
M=length(Y);
if(N ~= M)
    disp('数据点坐标不匹配！');
    return;
end

c(1:(2*m+1))=0;
b(1:(m+1))=0;

for j=1:(2*m+1)          %求出 c 和 b
    for k=1:N
        c(j)=c(j)+X(k)^(j-1);
        if(j<(m+2))
            b(j)=b(j)+Y(k)*X(k)^(j-1);
        end
    end
end

C(1,:)=c(1:(m+1));
for s=2:(m+1)
    C(s,:)=c(s:(m+s));
end

A=b'\C;                  %直接求解法求出拟合系数
```

【例 3-21】 多项式曲线拟合应用实例。用二次多项式拟合下表中所列的数据点。

x	1	2	3
y	2	5	10

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x = 1:3;
>> y = [1  5      10];
>> A = multifit(x,y,2)
```

输出结果为：

```
A =
    0.1282    0.3235    0.8718
```

即拟合的多项式为：

$$p = 0.1282 + 0.3235x + 0.8718x^2 。$$

3.8.6 最小二乘法拟合

最小二乘法拟合在科学实验的统计方法中经常使用。它的具体操作过程是从一组实验数据 (x_i, y_i) 中拟合出函数关系 $y = f(x)$，拟合的标准是使（ $f(x_i) - y_i$ ）的平方取极小值。

在 MATLAB 中可以使用 polyfit 函数对数据进行最小二乘拟合，它的基本调用格式为：

$$P = \text{polyfit}(X, Y, N)$$

表示用 N 次多项式拟合数据点 $[x_i, y_i]$。

【例 3-22】　最小二乘拟合应用实例。用四次多项式拟合下表中所列的数据点。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x = 1:0.2:1.8;
>> y = [1  1.0954     1.18321.2649  1.3416];
>> p = polyfit(x,y,4)
```

输出结果为：

```
P =
      -0.0104    0.0854   -0.3121    0.9086    0.3285
```

即拟合的多项式为：$p = -0.0104 + 0.0854x - 0.3121x^2 + 0.9086x^3 + 0.3285x^4$。

3.9　小结

本章首先讲述了 MATLAB 中插值和拟合函数，然后详细讲述了 MATLAB 中的插值和拟合的常用方法，包括拉格朗日插值法、艾特肯插值法、牛顿插值法、等距节点插值、埃尔米特插值和有理分式插值，以及一些常用的函数逼近和曲线拟合的数值方法。

插值和拟合是数据处理的一种重要手段，是来源于实际，又应用于实际的两种重要的计算方法。利用 MATLAB 实现起来也比较容易，因此应用起来也就比较方便。

第 4 章

矩阵特征值计算

物理、力学和工程技术中的很多问题在数学上都归结为求解矩阵特征值问题。当矩阵的阶数较小时，可以通过求特征多项式的根来得到特征值，即利用 MATLAB 中的函数来实现；当矩阵阶数比较大的时候，就需要用到一些数值方法，利用 MATLAB 编程也可方便实现。

通过本章，读者能够熟练掌握 MATLAB 中的特征值来计算相关函数，而且能通过编程实现多种矩阵特征值运算的算法。

4.1 特征值与特征向量

设 $\boldsymbol{A}$ 是 n 阶矩阵，$\boldsymbol{x}$ 是非零列向量，如果有数 λ 存在，满足 $\boldsymbol{Ax}=\lambda\boldsymbol{x}$，那么称 $\boldsymbol{x}$ 是矩阵 $\boldsymbol{A}$ 关于特征值 λ 的特征向量。

将 $\boldsymbol{Ax}=\lambda\boldsymbol{x}$ 改写为 $(\boldsymbol{A}-\lambda\boldsymbol{I})\boldsymbol{x}=0$，则称：

$$f(\lambda)=\left|\boldsymbol{A}-\lambda\boldsymbol{I}\right|=\lambda^n+a_1\lambda^{n-1}+\cdots+a_{n-1}\lambda+a_n$$

为矩阵 $\boldsymbol{A}$ 的特征多项式。

当 λ 是 $\boldsymbol{A}$ 的一个特征根时，它必然是 $f(\lambda)=0$ 的一个根。

4.2 条件数与病态矩阵

若矩阵 $\boldsymbol{A}$ 为非奇异，则称 $\|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ 为矩阵 $\boldsymbol{A}$ 的条件数，其中 $\|\cdot\|$ 为 $\boldsymbol{A}$ 的某种范数。

由于矩阵范数的定义不同，因而其条件数也不同，但是由于矩阵范数的等价性，故在

不同范数下的条件数也是等价的。

矩阵条件数的大小是衡量矩阵“坏”或“好”的标志。条件数大的矩阵称为病态矩阵或坏矩阵。一般说来，若矩阵的按模最大特征值与按模最小特征值之比值较大时，矩阵就会呈病态。特别地，当 A 的行列式值很小时，矩阵总是病态的。

设 X 为向量，常用的几种向量范数定义如下：

欧几里德范数，即 $\|X\|_2=\sqrt{\sum|x_k|^2}$；

∞-范数，即 $\|X\|=\max(abs(X))$；

1-范数，即 $\|X\|_1=\sum|x_k|$；

p-范数，$\|X\|_p=\sqrt[p]{\sum|x_k|^p}$。

对应于向量范数，矩阵 A 的范数定义如下：

欧几里德范数等于 A 的最大奇异值；

列范数等于 A 的列向量的 1-范数的最大值；

行范数等于 A 的行向量的 1-范数的最大值；

Frobenius 范数，即 $\|A\|_F=\sqrt{\sum\sum a_{ij}^2}$。

MATLAB 中提供了 norm 函数来求矩阵的范数，其具体用法如表 4-1 所示。

表 4-1 norm 函数

格　式	说　明
n = norm(A)	求 A 的欧几里德范数
n = norm(A,1)	求 A 的列范数
n = norm(A,2)	求 A 的欧几里德范数
n = norm(A,inf)	求 A 的行范数
n = norm(A, 'fro')	求 A 的 Frobenius 范数

不同的范数有相应不同的条件数，MATLAB 中提供了以下几个求矩阵条件数的函数。

（1）cond：求矩阵的条件数。

基本用法与功能如下。

c = cond(X)：求 X 的 2-范数的条件数，即 X 的最大奇异值和最小奇异值的商；

c = cond(X,p)：求 p-范数的条件数，p 的值可以是 1、2、inf 或者'fro'。

（2）condest：求矩阵的条件数估计值。

基本用法与功能如下。

c = condest (A)：求矩阵 A 的 1-范数的条件数的下界估值；

[c,v] = condest (A)：v 为向量，满足 $\|Av\|=\dfrac{\|A\|\cdot\|v\|}{c}$；

[c,v] = condest (A,t)：求上面的 c 和 v，同时显示出关于计算的步骤信息。如果 t=1，则计算的每步都显示出来；如果 t=–1，则给出商 c/rcond(A)。

（3）rcond：判断矩阵的病态程度。

基本用法与功能如下。

c = rcond($\boldsymbol{A}$)：对于病态矩阵 $\boldsymbol{A}$ 来说，给出一个接近于 0 的数；对于好条件矩阵 $\boldsymbol{A}$，则给出一个接近于 1 的数。

（4）condeig：求矩阵的特征值的条件数。

基本用法与功能如下。

c = condeig($\boldsymbol{A}$)：返回矩阵 $\boldsymbol{A}$ 的特征值的条件数；

[$\boldsymbol{V}$, $\boldsymbol{D}$, c] = condeig($\boldsymbol{A}$)：$\boldsymbol{D}$ 为 $\boldsymbol{A}$ 的特征值对角阵，$\boldsymbol{V}$ 为 $\boldsymbol{A}$ 的特征向量。

【例 4-1】 矩阵范数求取实例。求矩阵 $\boldsymbol{A}$ 的各种范数。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 3 & 8 \\ -5 & 2 & 9 \\ 0 & 1 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A= [1 3 8; -5 2 9; 0 1 4];
>> n1= norm(A)
>> n2= norm(A,1)                %列范数
>> n3= norm(A,inf)              %行范数
>> n4= norm(A,'fro')            %Frobenius 范数
```

输出计算结果为：

```
n1 =
   13.5336

n2 =
    21

n3 =
    16

n4 =
   14.1774
```

【例 4-2】 矩阵条件数求取实例。求矩阵 $\boldsymbol{A}$ 的各种条件数。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 3 & 8 \\ -5 & 2 & 9 \\ 0 & 1 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A= [1 3 8; -5 2 9; 0 1 4];
>> c1= cond(A)
>> c2= cond(A,1)                %1-范数的条件数
>> c3= cond(A,inf)              %行范数的条件数
>> c4= cond(A,'fro')            %Frobenius 范数的条件数
>> c5= rcond(A)
```

输出计算结果为：

```
c1 =
```

```
   40.5924

c2 =
   85.1053

c3 =
   61.4737

c4 =
   42.6696

c5 =
    0.0118
```

从 A 的条件数可以看出，A 是一个病态矩阵。

4.3 相似变换

相似矩阵有着相同的特征值，在 MATLAB 中，函数 balance 能够求出相似变换矩阵，它的基本用法与功能如下。

[T, B] = balance(A)：求相似变换矩阵 T 和平衡矩阵 B，满足 $B = T^{-1}AT$，且 T 为对角阵；

B = balance(A)：求平衡矩阵 B。

【例 4-3】 矩阵相似变换实例。利用函数 balance，计算矩阵 $\begin{bmatrix} 1 & 3 & 0 \\ 3 & 9 & 4 \\ -8 & 6 & 2 \end{bmatrix}$ 的平衡矩阵和相似变换矩阵。

解：在 MATLAB 命令窗口中输入：

```
>> J=[1 3 0;3 9 4;-8 6 2;];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T =
    0.5000         0         0
         0    1.0000         0
         0         0    2.0000
B =
    1.0000    6.0000         0
    1.5000    9.0000    8.0000
   -2.0000    3.0000    2.0000
>> B=balance(J)
B =
    1.0000    6.0000         0
    1.5000    9.0000    8.0000
   -2.0000    3.0000    2.0000
```

对于稀疏方阵（零元素较多），T 的形式可能会不同。

继续在 MATLAB 命令窗口中输入：

```
>> J=[1 2 0;2 4 0;8 3 9];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T =
     0     0     1
     0     1     0
     1     0     0
B =
     9     3     8
     0     4     2
     0     2     1
```

上面的 ***T*** 矩阵为副对角线矩阵，是因为 ***J*** 的第三列有两个 0，在求 ***B*** 之前把 ***J*** 做了行变换，如果想得到对角型的矩阵 ***T***，可采用 noperm 选项。

继续在 MATLAB 命令窗口中输入：

```
>> [T,B]=balance(J,'noperm')
```

输出计算结果为：

```
T =
    0.2500         0         0
         0    0.2500         0
         0         0    1.0000
B =
    1.0000    2.0000         0
    2.0000    4.0000         0
    2.0000    0.7500    9.0000
```

这样得出的 ***T*** 矩阵就为对角阵了。如果矩阵 ***A*** 为对角阵，则得出的 ***T*** 矩阵为单位矩阵。

继续在 MATLAB 命令窗口中输入：

```
>> J=[1 2 3;2 4 6;3 6 9];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T =
     1     0     0
     0     1     0
     0     0     1
B =
     1     2     3
     2     4     6
     3     6     9
```

求平衡矩阵的目的是调整矩阵各个元素之间的大小关系，降低其病态程度。

一般来说，改用平衡矩阵 ***B*** 来求特征值，会比直接用 ***A*** 来求特征值的精度要高。

4.4 特征值求法

特征值的求法有多种，有些方法只能求出一个特征值，如幂法；而有些方法能求出所

有的特征值，如收缩法。有的只能求出实特征值，有的能求出复特征值，如特征多项式法。有的能附带求出特征向量，而有的只能求出特征值。下面由简单到复杂依次介绍各种求矩阵特征值的算法。

4.4.1 特征多项式法

此方法的思想十分简单，根据特征值的性质，它应该是特征多项式的根，因此可以通过求特征多项式的根来求特征值。

在 MATLAB 中编程实现的特征多项式法的函数为：Chapoly。

功能：通过求矩阵特征多项式的根来求其特征值。

调用格式：l = Chapoly (A)

其中，A 为已知矩阵；

l 为求得的矩阵特征值。

特征多项式法的 MATLAB 程序代码如下：

```
function l = Chapoly(A)
%特征多项式法求矩阵特征值
%已知矩阵：A
%求得的矩阵特征值：l

syms t;
N = size(A);
n = N(1,1);

y = det(A-t*eye(n,n));
l = solve(y);

l = vpa(l, 5);          %结果取五位精度
```

【例 4-4】 特征多项式法求取特征值实例。采用特征多项式法，求矩阵 A 的特征值。

$$A=\begin{bmatrix}11 & 5 & 8\\ -5 & 6 & 10\\ -2 & 3 & 4\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A= [11 5 8; -5 6 10; -2 3 4];
>> l= Chapoly(A)
```

输出计算结果为：

```
l =
         -.5721
 10.786-6.4035*i
 10.786+6.4035*i
```

由计算结果可知，矩阵 A 的特征值为–0.5721，10.786–6.4035*i 和 10.786+6.4035*i。

在实际使用时，会发现用特征多项式法求矩阵的特征值速度很慢，因此几乎不使用这种方法。

4.4.2 幂法

幂法是用来求矩阵的主特征值和主特征向量的迭代法。主特征值指的是按模最大的特征值，它对应的特征向量称为主特征向量。幂法的迭代过程为：

给定迭代初始值 $x^0 \neq 0$和误差限$\varepsilon>0$；

$$y^{k+1} = Ax^k$$

$$m^{k+1} = \max(y^{k+1})$$

$$x^{k+1} = \frac{y^{k+1}}{m^{k+1}}$$

如果$\left|(m^{k+1} - m^k)\right| < \varepsilon$，结束。

其中，$\max(y^{k+1})$ 表示向量 y^{k+1} 按模最大的分量。幂法有如下的收敛性质：

$$\lim_{k\to\infty} x^k = \boldsymbol{a}_1 \Big/ \max(\boldsymbol{a}_1), \quad \lim_{k\to\infty} m^k = \lambda_1$$

其中 λ_1 为主特征值，$\boldsymbol{a}_1$ 为对应的主特征向量。

在 MATLAB 中编程实现的幂法的函数为：pmethod。

功能：幂法求矩阵的主特征值及主特征向量。

调用格式：[*l*, *v*, *s*]=pmethod (*A*,*x*0,eps)

其中，*A* 为已知矩阵；

*x*0 为迭代初始向量；

eps 为迭代的精度；

l 为求得的矩阵主特征值；

v 为求得的矩阵主特征向量；

s 为迭代步数。

幂法的 MATLAB 程序代码如下：

```
function [l,v,s] =pmethod(A,x0,eps)
%幂法求矩阵的主特征值及主特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵主特征向量：v
%迭代步数：s

if nargin==2
    eps = 1.0e-6;
end

v = x0;                    %v 为主特征向量
```

```
M = 5000;                    %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
    y = A*v;
    m = max(y);              %m 为按模最大的分量
    v = y/m;

    if(abs(m - l)<eps)
        l = m;               %到所需精度，退出，l 为主特征值
        s = k;               %s 为迭代步数
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
            l = m;
            s = M;
        else
            l = m;
        end
    end
end
```

注意

- 当矩阵为病态矩阵时，幂法的收敛速度非常慢；
- 初始迭代向量不能取零向量。

【例 4-5】 幂法求取特征值实例。采用幂法求矩阵 A 的主特征值和主特征向量。

$$A=\begin{bmatrix}1 & 5 & 2\\ 6 & -1 & 7\\ 1 & 3 & 1\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A  = [1 5 2; 6 -1 7; 1 3 1];
>> x0 = [ 1 1 1]';
>> [l,v,s]= pmethod(A,x0)
```

输出计算结果为：

```
l =
    8.0407

v =
    0.8661
    1.0000
    0.5491
s =
    84
```

由计算结果可知，经过 84 步迭代，求出了矩阵 A 的主特征值为 8.0407，对应的特征向量为$[0.8661, 1.0000, 0.5491]^T$。

4.4.3 瑞利商加速幂法

如果矩阵为对称矩阵，可以用瑞利法来加快幂法的收敛速度。对于非零向量 $\boldsymbol{x}$，它的瑞利商定义如下：

$$\boldsymbol{R}(\boldsymbol{x})=\frac{(\boldsymbol{A}\boldsymbol{x},\boldsymbol{x})}{(\boldsymbol{x},\boldsymbol{x})}$$

瑞利商加速幂法的迭代过程如下：

给定迭代初始值 $x^0\neq 0$ 和误差限 $\varepsilon>0$；

$$\boldsymbol{y}^{k+1}=\boldsymbol{A}\boldsymbol{x}^k$$

$$m^{k+1}=\frac{(\boldsymbol{A}\boldsymbol{x}^k,\boldsymbol{x}^k)}{(\boldsymbol{x}^k,\boldsymbol{x}^k)}$$

$$\boldsymbol{x}^{k+1}=\frac{\boldsymbol{y}^{k+1}}{m^{k+1}}$$

如果 $\left|(m^{k+1}-m^k)\right|<\varepsilon$，结束。

当然，如果矩阵不对称，也可以用瑞利商加速幂法来求其特征值，但是加速的效果可能不是很明显。

在 MATLAB 中编程实现的瑞利商加速幂法的函数为：rpmethod。

功能：瑞利商加速幂法求对称矩阵的主特征值及主特征向量。

调用格式：[l, v, s]=rpmethod (*A*, x0, eps)。

其中，*A* 为已知矩阵；

*x*0 为迭代初始向量；

eps 为迭代的精度；

l 为求得的矩阵主特征值；

v 为求得的矩阵主特征向量；

s 为迭代步数。

瑞利商加速幂法的 MATLAB 程序代码如下：

```
function [l,v,s]=rpmethod(A,x0,eps)
%瑞利商加速幂法求对称矩阵的主特征值及主特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵主特征向量：v
%迭代步数：s

if nargin==2
    eps = 1.0e-6;
end

v = x0;                    %v 为主特征向量
```

```
M = 500;                    %迭代步数限制
m = 0;
l = 0;

for(k=1:M)
    y = A*v;
    m = (y'*v)/(v'*v);      %m 为瑞利商
    if(rank(m)>1)
        m=(y*v')/(v*v');
    end
    v = y/m;

    if(abs(m - l)<eps)
        l = m;              %到所需精度，退出，l 为主特征值
        s = k;              %s 为迭代步数
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
             l = m;
             s = M;
        else
            l = m;
        end
    end
end
```

【例 4-6】 瑞利商加速幂法求取特征值实例。采用瑞利商加速幂法求矩阵 $\boldsymbol{A}$ 的主特征值和主特征向量。

$$\boldsymbol{A}=\begin{bmatrix}1 & 5 & 2\\ 6 & -1 & 7\\ 1 & 3 & 1\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A  = [1 5 2; 6 -1 7; 1 3 1];
>> x0 = [ 1 1 1]';
>> [l,v,s]= rpmethod(A,x0)
```

输出计算结果为：

```
l =
    8.0407
v =
    1.2239
    1.4131
    0.7759
s =
    74
```

由计算结果可知，经过 74 步迭代，求出了矩阵 $\boldsymbol{A}$ 的主特征值为 8.0407，对应的特征向量为$[0.8661, 1.0000, 0.5491]^T$。

和幂法相比，同一个矩阵，瑞利商加速幂法少用 10 步得到了主特征值。

4.4.4 收缩法

收缩法可用来求矩阵所有的特征值。它是基于幂法的求特征值的方法。

设矩阵 $\boldsymbol{A}$ 的 n 个特征值按模从大到小排序为：

$$|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$$

其相应的 n 个线性无关特征向量为 $\boldsymbol{a_1}$，$\boldsymbol{a_2}$，…，$\boldsymbol{a_n}$。在计算 $\boldsymbol{A}$ 的最大特征值 λ_1 及相应特征向量 $\boldsymbol{a}_1$ 后，可以通过收缩方法，继续用乘幂法计算 λ_2 及其相应的特征向量 $\boldsymbol{a}_2$。具体操作过程如下：

（1）用幂法求出 $\boldsymbol{A}$ 的主特征值 λ_1 和主特征向量 $\boldsymbol{a}_1$；

（2）令

$$\boldsymbol{B}=\boldsymbol{A}-\boldsymbol{a}_1\boldsymbol{A}_1^T=\begin{pmatrix}0 & 0 & \cdots & 0\\ a_{21}-a_{21}a_{11} & a_{22}-a_{21}a_{12} & \cdots & a_{2n}-a_{21}a_{1n}\\ \vdots & \vdots & & \vdots\\ a_{n1}-a_{n1}a_{11} & a_{n2}-a_{n1}a_{12} & \cdots & a_{nn}-a_{n1}a_{1n}\end{pmatrix}$$

其中 $\boldsymbol{A}_1$ 代表矩阵 $\boldsymbol{A}$ 的第一行组成的向量，a_{n1} 代表主特征向量 $\boldsymbol{a}_1$ 的第 n 个分量；

（3）去掉 $\boldsymbol{A}_1$ 的第 1 行和第 1 列：

$$\boldsymbol{B}_1=\begin{pmatrix}a_{22}-a_{21}a_{12} & a_{23}-a_{21}a_{13} & \cdots & a_{2n}-a_{21}a_{1n}\\ a_{32}-a_{31}a_{12} & a_{33}-a_{31}a_{13} & \cdots & a_{3n}-a_{31}a_{1n}\\ \vdots & \vdots & & \vdots\\ a_{n2}-a_{n1}a_{12} & a_{n3}-a_{n1}a_{13} & \cdots & a_{nn}-a_{n1}a_{1n}\end{pmatrix}$$

则 $\boldsymbol{B}_1$ 与有与 $\boldsymbol{A}$ 除 λ_1 外的相同的 $n-1$ 个特征值 $|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$，可以用幂法计算 λ_2 及其对应的特征向量 $\boldsymbol{a}_2$，如此经过 n 次收缩，就可把 $\boldsymbol{A}$ 的所有特征值求出来。

在 MATLAB 中编程实现的收缩法的函数为：spmethod。

功能：收缩法求矩阵全部特征值。

调用格式：*T*=spmethod (*A*, eps)。

其中，A 是已知矩阵；

eps 是精度；

T 是对角阵，对角元素为矩阵 *A* 的特征值。

收缩法的 MATLAB 程序代码如下：

```
function T=spmethod(A,eps)
%收缩法求矩阵全部特征值
%已知矩阵：A
%精度：eps
%求得的矩阵特征值：T

if nargin==1
    eps = 1.0e-6;
end
```

```
N = size(A);
n = N(1,1);
T = zeros(n,n);
B = A;
B1= B;

for(i=1:n)
    s  = size(B);
    r  = s(1,1);                                    %r 为 B 的阶
    [T(i,i),u] = pmethod(B,ones(r,1),eps);          %用幂法求解主特征值和主特征向量
    u = u/u(1,1);                                   %将主特征向量规一化
    B1 = B - u*B(1,:);
    s1  = size(B1);
    r1  = s(1,1);
    B  = B1(2:r1,2:r1);                             %B 为收缩后的矩阵
end
```

【例 4-7】 收缩法求取特征值实例。采用收缩法求矩阵 $\boldsymbol{A}$ 的所有特征值。

$$\boldsymbol{A}=\begin{bmatrix}3 & 2 & 1\\ 4 & 3 & 7\\ 5 & 8 & 6\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [3 2 1; 4 3 7; 5 8 6];
>> T = spmethod(A)
```

输出计算结果为：

```
T =
   13.3751         0         0
         0   -3.1534         0
         0         0    1.7782
```

所以用收缩法得到矩阵 $\boldsymbol{A}$ 的三个特征值为 13.3751，−3.1534，1.7782。

注意

在用以上几种方法求矩阵的特征值时，如果矩阵为病态矩阵或者矩阵有复特征值，就会出现错误的结果。

4.4.5 逆幂法

逆幂法是用来求矩阵按模的最小特征值及其对应的特征向量的迭代法。逆幂法的迭代公式如下。

给定迭代初始值 $\boldsymbol{x}^0\neq 0$和误差限$\varepsilon>0$

$$\boldsymbol{A}\boldsymbol{y}^{k+1}=\boldsymbol{x}^k$$

$$m^{k+1}=\max(\boldsymbol{y}^{k+1})$$

$$\boldsymbol{x}^{k+1}=\frac{\boldsymbol{y}^{k+1}}{m^{k+1}}$$

如果$\left|(m^{k+1}-m^k)\right|<\varepsilon$，则结束，

其中，$\max(\boldsymbol{y}^{k+1})$表示向量$\boldsymbol{y}^{k+1}$按模最大的分量。

逆幂法有如下的收敛性质：

$$\lim_{k\to\infty}\boldsymbol{x}^k=\boldsymbol{a}_n\Big/\max(\boldsymbol{a}_n),\quad \lim_{k\to\infty}m^k=\frac{1}{\lambda_n}$$

其中λ_n为矩阵按模的最小特征值，$\boldsymbol{a}_n$为λ_n对应的特征向量。

在 MATLAB 中编程实现的逆幂法的函数为：ipmethod。

功能：收缩法求矩阵全部特征值。

调用格式：[*l*, *v*]=ipmethod (*A*, *x*0, eps)。

其中，*A* 为已知矩阵；

*x*0 为迭代初始向量；

eps 为迭代精度；

l 为求得的矩阵按模的最小特征值；

v 为矩阵特征值 *l* 对应的特征向量。

逆幂法的 MATLAB 程序代码如下：

```
function [l,v]=ipmethod(A,x0,eps)
%逆幂法求矩阵按模的最小特征值及其特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵特征值 l 对应的特征向量：v

if nargin==2
    eps = 1.0e-6;
end

v = x0;                          %v 按模的最小特征值对应的特征向量
M = 500;                         %迭代步数限制
m = 0;
l = 0;

for(k=1:M)
    y = A\v;
    m = max(y);                  %m 为按模最大的分量
    v = y/m;

    if(abs(m - l)<eps)
        l = 1/m;                 %到所需精度，退出，l 为按模的最小特征值
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
            l = 1/m;
        else
            l = m;
```

```
        end
    end
end
```

【例 4-8】 逆幂法求取特征值实例。采用逆幂法求矩阵 $\boldsymbol{A}$ 的特征值。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 5 & 1 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [1 2 -2; -4 3 0; 5 1 4];
>> x0 = [1 1 1]';
>> [l,v] = ipmethod(A,x0)
```

输出计算结果为：

```
l =
    4.1129

v =
   -0.1946
    0.6967
    1.0000
```

所以得到 $\boldsymbol{A}$ 的按模最小的特征值为 4.1129，其对应的特征向量为[-0.1946，0.6967，1.0000]^T。

4.4.6 位移逆幂法

位移逆幂法是用来求矩阵离某个特定的常数最近的特征值及其对应的特征向量的迭代法。位移幂法的迭代公式如下。

给定迭代初始值 $\boldsymbol{x}^0\neq 0$和误差$\varepsilon>0$，常数μ，

$$(\boldsymbol{A}-\mu\boldsymbol{I})\boldsymbol{y}^{k+1}=\boldsymbol{x}^k$$

$$m^{k+1}=\max(\boldsymbol{y}^{k+1})$$

$$\boldsymbol{x}^{k+1}=\frac{\boldsymbol{y}^{k+1}}{m^{k+1}}$$

如果$|(m^{k+1}-m^k)<|\varepsilon$，结束

其中，$\max(\boldsymbol{y}^{k+1})$ 表示向量 $\boldsymbol{y}^{k+1}$ 按模最大的分量。

位移逆幂法有如下的收敛性质：

$$\lim_{k\to\infty}\boldsymbol{x}^k=\frac{\boldsymbol{a}_j}{\max(\boldsymbol{a}_j)},\quad \lim_{k\to\infty}m^k=\frac{1}{\lambda_j-\mu}$$

其中 λ_j 为离 μ 最近的特征值，$\boldsymbol{a}_j$ 为 λ_j 对应的特征向量。

在 MATLAB 中编程实现的位移逆幂法的函数为：dimethod。

功能：位移逆幂法求矩阵离某个常数最近的特征值及其对应的特征向量。

调用格式：[l, $\boldsymbol{v}$]=dimethod ($\boldsymbol{A}$, $\boldsymbol{x}$0, u, eps)

其中，*A* 为已知矩阵；

*x*0 为迭代初始向量；

u 为常数；

eps 为迭代精度；

l 为求得的矩阵离 u 最近的特征值；

v 为矩阵特征值 l 对应的特征向量

位移逆幂法的 MATLAB 程序代码如下：

```
function [l,v]=dimethod(A,x0,u,eps)
%位移逆幂法求矩阵离某个常数最近的特征值及其对应的特征向量
%已知矩阵：A
%迭代初始向量：x0
%常数：u
%迭代的精度：eps
%求得的矩阵离 u 最近的特征值：l
%求得的矩阵的特征值 l 对应的特征向量：v

if nargin==3
    eps = 1.0e-6;
end

N = size(A);
n = N(1,1);              %n 为 A 的阶
v = x0;
M = 5000;                %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
    y = (A-u*eye(n,n))\v;
    m = max(y);          %m 为按模最大的分量
    v = y/m;

    if(abs(m - l)<eps)
        l = 1/m + u;         %到所需精度，退出
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
            l = 1/m + u;
        else
            l = m;
        end
    end
end
```

【例 4-9】 位移逆幂法求取特征值实例。采用位移逆幂法求矩阵 *A* 的离 4 最近的特征值。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 5 & 1 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [1 2 -2; -4 3 0; 5 1 4];
>> x0 = [1 1 1]';
>> [l,v] = dimethod(A,x0,4)
```

输出计算结果为：

```
l =
    4.1145
v =
   -0.1943
    0.6974
    1.0000
```

所以矩阵 $\boldsymbol{A}$ 离 4 最近的特征值为 4.1145。

注意

如果所有的特征值离给定的数太远，结果将不会收敛。

4.4.7 QR 算法

QR 基本算法是求矩阵特征值的最有效和应用最广泛的一种方法。它的算法的基本依据是以下的两个定理：

(1)设 $\boldsymbol{A}$ 是 n 阶矩阵，其 n 个特征值为 $\lambda_1,\lambda_2,\cdots,\lambda_n$，那么存在一个酉矩阵 $\boldsymbol{U}$，使得 $\boldsymbol{U}^H\boldsymbol{A}\boldsymbol{U}$ 是以 $\lambda_1,\lambda_2,\cdots,\lambda_n$ 为对角元的上三角矩阵；

(2) 设 $\boldsymbol{A}$ 是 n 阶实矩阵，那么，存在一个正交矩阵 $\boldsymbol{Q}$，使得 $\boldsymbol{Q}^H\boldsymbol{A}\boldsymbol{Q}$ 为一个准上三角矩阵，它的每一个对角元是 $\boldsymbol{A}$ 的一个特征值，对角元上的二阶块矩阵的两个特征值是 $\boldsymbol{A}$ 的一对共轭复特征值。

1. QR 基本算法

QR 基本算法的过程如下：

给定循环步数 M，$\boldsymbol{A}^1=\boldsymbol{A}$，

$k=1,2,\cdots M$，计算：

$$\boldsymbol{A}^k=\boldsymbol{Q}^k\boldsymbol{R}^k$$

$$\boldsymbol{A}^{k+1}=\boldsymbol{R}^k\boldsymbol{Q}^k$$

QR 基本算法有如下的收敛性质：

如果 $\boldsymbol{A}$ 的特征值满足 $|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$，则 QR 基本算法产生的矩阵序列 $\{\boldsymbol{A}^k\}$ 基本收敛到上三角矩阵（特别地，当 $\boldsymbol{A}$ 为对称阵时，收敛到对角阵），对角元素收敛到 $\boldsymbol{A}$ 的特征值。

在 MATLAB 中编程实现的 QR 基本算法的函数为：qrtz。

功能：QR 基本算法求矩阵全部特征值。

调用格式：T = qrtz (A, M)。

其中，A 为已知矩阵；

M 为迭代步数；

l 为矩阵 A 的全部特征值

QR 基本算法的 MATLAB 程序代码如下：

```
function l = qrtz(A,M)
%QR 基本算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l

for(i=1:M)                          %M 为迭代步数
    [q,r]=qr(A);
    A = r*q;
    l = diag(A);
end
```

【例 4-10】 QR 算法求取特征值实例。采用 QR 算法法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}1 & 5 & 6\\ 4 & 7 & 0\\ 8 & 11 & 4\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [1 5 6; 4 7 0; 8 11 4];
>> l = qrtz(A,20)
```

输出计算结果为：

```
l =
   13.4695
   -3.8565
    2.3869
```

继续在 MATLAB 命令窗口中输入：

```
>> l = qrtz(A,500)
```

输出计算结果为：

```
l =
   13.4695
   -3.8566
    2.3871
```

因此得到矩阵 A 的三个特征值为 13.4695，–3.8565，2.3869。

从上面可以看出，QR 算法的收敛速度很快，而且用其他方法可能不收敛的矩阵却可以用 QR 算法求出其特征值。如果矩阵为病态的话，则用 QR 算法也可求出其特征值，但需要的迭代步骤较多。

【例 4-11】 QR 算法求取病态矩阵特征值实例。采用 QR 算法法求病态矩阵 A 的所有特征值。

$$\boldsymbol{A}=\begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 3 & 1 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [1 2 -2; -4 3 0; 3 1 4];
>> [l,v] = pmethod(A,x0)
```

输出计算结果为：

```
（迭代步数太多，收敛速度太慢！）
l =
    3.6464
v =
    0.0542
   -1.2547
    1.0000
```

继续在 MATLAB 命令窗口中输入：

```
>> l= qrtz(A,3000)
```

输出计算结果为：

```
l =
    1.3669
    2.5185
    4.1146
```

继续在 MATLAB 命令窗口中输入：

```
>> l = qrtz(A,5000)
```

输出计算结果为：

```
l =
    2.7055
    1.1801
    4.1145
```

用特征多项式法计算可得出 $\boldsymbol{A}$ 的特征值为：[4.1145，1.9428+3.6386*i，1.9428–3.6386*i]^T。上面的计算表明用幂法不收敛，QR 算法是不能求复特征值的，如果矩阵为病态矩阵的话，QR 算法需要比较长的迭代才能得出实特征根。

注意

如果不满足$|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$的条件，QR 基本算法可能不收敛。

2. 海森伯格阵的 QR 算法

当 $\boldsymbol{A}$ 为一般矩阵时，QR 基本算法的计算量很大，在实际使用时，通常采用的做法是先将矩阵 $\boldsymbol{A}$ 正交相似变换为上海森伯格矩阵，然后再实施 QR 基本算法。

给定循环步数 M，$\boldsymbol{A}^1=\text{Hessenberg}(\boldsymbol{A})$，

$k=1,2,\cdots M$，计算：

$$\boldsymbol{A}^k=\boldsymbol{Q}^k\boldsymbol{R}^k$$

$$\boldsymbol{A}^{k+1}=\boldsymbol{R}^k\boldsymbol{Q}^k$$

这样每一步迭代过程中的$\boldsymbol{Q}^k$和$\boldsymbol{A}^k$都是上海森伯格矩阵。所谓上海森伯格矩阵是指一个n阶矩阵$\boldsymbol{A}$，如果当$i>j+1$时，$a_{ij}=0$，称$\boldsymbol{A}$为上海森伯格矩阵。

在 MATLAB 中编程实现的海森伯格 QR 算法的函数为：hessqrtz。

功能：海森伯格 QR 算法求矩阵全部特征值。

调用格式：T = hessqrtz ($\boldsymbol{A}$, M)。

其中，$\boldsymbol{A}$为已知矩阵；

M为迭代步数；

l为矩阵$\boldsymbol{A}$的全部特征值。

海森伯格矩阵的 QR 算法的 MATLAB 程序代码如下：

```
function l = hessqrtz(A,M)
%海森伯格 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l

A = hess(A);                %将 A 化为上海森伯格矩阵
for(i=1:M)                  %M 为迭代步数
    [q,r]=qr(A);
    A = r*q;
    l = diag(A);
end
```

【例 4-12】 海森伯格矩阵的 QR 算法求取特征值实例。采用海森伯格矩阵的 QR 算法求矩阵$\boldsymbol{A}$的所有特征值。

$$\boldsymbol{A}=\begin{bmatrix}6 & 3 & 2\\ 4 & 3 & 8\\ 7 & 9 & 5\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [6 3 2; 4 3 8; 7 9 5];
>> l = hessqrtz(A,20)
```

输出计算结果为：

```
l =
   15.4121
   -4.4150
    3.0029
```

所以得到矩阵$\boldsymbol{A}$的三个特征值为 15.4121，−4.4150，3.0029。而正确结果计算如下。

继续在 MATLAB 命令窗口中输入：

```
>> l = Chapoly(A)
```

输出计算结果为：

```
l =
      3.
   15.412
   -4.4120
```

由此可见海森伯格矩阵的 QR 算法收敛速度也是很快的。

3. 位移 QR 算法

为了加快 QR 算法的收敛速度，产生了所谓的位移 QR 算法，它类似于位移逆幂法。其算法的迭代过程如下。

给定循环步数 M，$\boldsymbol{A}^1 = Hessenberg(A)$，

$k = 1,2,\cdots M$，选择 μ^k，然后计算：

$$\boldsymbol{A}^k - \mu^k \boldsymbol{I} = \boldsymbol{Q}^k \boldsymbol{R}^k$$

$$\boldsymbol{A}^{k+1} = \boldsymbol{R}^k \boldsymbol{Q}^k + \mu^k \boldsymbol{I}$$

一般 μ^k 的选择有以下两种考虑方法：

（1）选 $\mu^k = a_{nn}^k$，即瑞利商位移；

（2）迭代过程中，如果子矩阵 $\begin{bmatrix} a_{n-1,n-1}^k & a_{n-1,n}^k \\ a_{n,n-1}^k & a_{nn}^k \end{bmatrix}$ 的两个特征值为实数时，选最接近 a_{nn}^k 的那个作为 μ^k，即威尔金森位移。

在 MATLAB 中编程实现的瑞利商位移的 QR 算法的函数为：rqrtz。

功能：瑞利商位移的 QR 算法求矩阵全部特征值。

调用格式：T = rqrtz ($\boldsymbol{A}$, M)

其中，$\boldsymbol{A}$ 为已知矩阵；

M 为迭代步数；

l 为矩阵 $\boldsymbol{A}$ 的全部特征值。

瑞利商位移的 QR 算法的 MATLAB 程序如下：

```
function l = rqrtz(A,M)
%瑞利商位移的 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l

A = hess(A);
for(i=1:M)
    N = size(A);
    n = N(1,1);
    u = A(n,n);
    [q,r]=qr(A-u*eye(n,n));
    A = r*q+u*eye(n,n);
    l = diag(A);
end
```

在 MATLAB 中编程实现的威尔金森位移的 QR 算法的函数为：wilkqrtz。

功能：威尔金森位移的 QR 算法求矩阵全部特征值。

调用格式：T = wilkqrtz (A, M)

其中，A 为已知矩阵；

M 为迭代步数；

l 为矩阵 A 的全部特征值

威尔金森位移的 QR 算法的 MATLAB 程序如下：

```
function l = wilkqrtz(A,M)
%威尔金森位移的 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l

A = hess(A);
for(i=1:M)
    N = size(A);
    n = N(1,1);
    A1 = A((n-1):n,(n-1):n);
    t = Chapoly(A1);
    if(imag(t(1,1)) == 0 && imag(t(2,1)) == 0)  %两个特征值是否
                                                %为实数
        if(abs(t(1,1)-A(n,n))<abs(t(1,1)-A(n,n)))
            u = t(1,1);
        else
            u = t(2,1);          %选最接近 A(n,n)的那个作为 u
        end
    else
        u = A(n,n);              %如果两个特征值有一个为复数,取 A(n,n)作为 u
    end

    [q,r]=qr(A-u*eye(n,n));
    A = r*q+u*eye(n,n);
    l = diag(A);
end
```

【例 4-13】 位移 QR 算法求取特征值实例。采用位移 QR 算法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}6 & 3 & 2\\ 4 & 3 & 8\\ 7 & 9 & 5\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [6 3 2; 4 3 8; 7 9 5];
>> l = rqrtz(A,13)
```

输出计算结果为：

```
l =
   15.4121
```

```
    3.0000
   -4.4121
>> l = wilkqrtz(A,13)
```

输出计算结果为：

```
l =
   15.4121
    3.0000
   -4.4121
```

所以得到矩阵 A 的三个特征值为 15.4121，3.0000，–4.4121。位移 QR 算法比 QR 算法的收敛速度更快。

4.5 舒尔分解和奇异值分解

舒尔分解可以直接给出矩阵的特征值。而对于对称矩阵，奇异值分解能给出特征值的绝对值。这两种分解的函数的用法在第 2 章中已介绍过，此处不再赘述。

【例 4-14】 舒尔分解法求取特征值实例。采用舒尔分解法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-4 & 3 & 10\\ 9 & 1 & 8\\ 7 & 6 & 2\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [-4 3 10; 9 1 8; 7 6 2];
>> [S,U] = schur(A)
```

输出计算结果为：

```
S =
   -0.4422   -0.8713   -0.2130
   -0.6719    0.4791   -0.5647
   -0.5941    0.1066    0.7973
U =
   13.9957    3.5156   -3.8192
         0   -8.6568   -4.6962
         0         0   -6.3388
```

则 A 的三个特征值为 13.9957，–8.6568，–6.3388。

【例 4-15】 奇异分解法求取特征值实例。采用奇异分解法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-4 & 9 & 10\\ 9 & 1 & 8\\ 10 & 8 & 2\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [-4 9 10; 9 1 8; 10 8 2];
>> [S,U] = svd(A)
```

输出计算结果为：

```
S =
    0.5249   -0.8505    0.0327
    0.5782    0.3282   -0.7470
    0.6246    0.4110    0.6641
U =
   17.8115         0         0
         0   12.3053         0
         0         0    6.5062
```

则 A 的三个特征值的绝对值为 17.8115，12.3053，6.5062。事实上 A 的特征值为 17.8115，−12.3053，−6.5062。

4.6 功能强大的 eig 函数

MATLAB 提供了一个求矩阵特征值的函数 eig，它综合了以上几种求特征值的方法，而且能求出复特征值。它的用法如表 4-2 所示。

表 4-2 eig 函数的用法

格　式	说　明
L = eig(*A*)	求 *A* 的特征值
[V, L] = eig(*A*)	求 *A* 的特征值和特征向量
[V, L] = eig(*A*,'nobalance')	如果 *A* 中某项非常小，采用此法更精确
[V, L] = eig(*A*,*B*)	计算广义特征向量阵和广义特征值阵，使 AV=BVD
[V, L] = eigs(*A*,k,s)	计算 *k* 个由 *s* 指定的 *A* 的特征向量和特征值

表 4-2 中第 5 式中 *s* 可取的值如下：

'lm'：求具有最大幅值的 *k* 个特征向量和特征值；

'sm'：求具有最小幅值的 *k* 个特征向量和特征值；

'lr'：求具有最大实部的 *k* 个特征向量和特征值；

'sr'：求具有最小实部的 *k* 个特征向量和特征值；

'be'：求谱的两端的 *k* 个特征向量和特征值。

【例 4-16】 MATLAB 中的 eig 函数求取特征值实例。采用 eig 函数求一般矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-1 & 7 & 6\\ 0 & 1 & 5\\ 3 & -8 & 2\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [-1 7 6; 0 1 5; 3 -8 2];
>> [V, L] = eig(A)
```

输出计算结果为：

```
V =
  -0.9339          -0.7759                -0.7759
  -0.3490          -0.3779 - 0.2594i      -0.3779 + 0.2594i
  -0.0780           0.3191 - 0.2934i       0.3191 + 0.2934i
L =
   2.1177               0                      0
        0          -0.0589 + 4.6093i            0
        0               0                      -0.0589 - 4.6093i
```

由计算结果可知，矩阵 A 的三个特征值为 2.1177，–0.0589 + 4.6093i，–0.0589–4.6093i。

【例 4-17】 MATLAB 中的 eig 函数求取病态矩阵特征值实例。采用 eig 函数求病态矩阵 A 的所有特征值。

$$A=\begin{bmatrix} 1 & 10^{-18} & 2 \\ 2.0\times10^{-18} & 6 & 3 \\ 5 & 2.0\times10^{-19} & 4 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[1 1.0e-18  2 ; 2.0e-18 6 3;5  2.0e-19  4];
>> [V, L] = eig(A)
```

输出计算结果为：

```
V =
   -0.6767    0.0000    0.0000
   -0.2900    1.0000   -1.0000
    0.6767    0.0000    0.0000
L =
   -1.0000         0         0
         0    6.0000         0
         0         0    6.0000
```

继续在 MATLAB 命令窗口中输入：

```
>> [V,L]=eig(A,'nobalance')
```

输出计算结果为：

```
V =
  -1.0000           0.0000 + 0.0000i   0.0000 - 0.0000i
  -0.4286          -0.0000 - 1.0000i  -0.0000 + 1.0000i
   1.0000           0.0000 + 0.0000i   0.0000 - 0.0000i
L =
  -1.0000                0                      0
        0          6.0000 + 0.0000i             0
        0                0                      6.0000 - 0.0000i
```

所以矩阵 A 的三个特征值为–1.0000，6.0000 + 0.0000i，6.0000–0.0000i。

从上面的例子可以看出，'nobalance'选项可提高结果的精度。

【例 4-18】 MATLAB 中的 eig 函数求取任意个数特征值实例。采用用 eigs 函数求矩阵 A 的任意个数特征值。

$$A=\begin{bmatrix} 1 & 5 & 2 \\ -7 & 6 & 3 \\ 5 & 9 & 4 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[1 5  2 ; -7 6 3; 5  9  4];
>> [V,L]=eigs(A,2,'lm')
```

输出计算结果为：

```
V =
    0.3830    0.2623
    0.0854   -0.2811
    0.9198    0.9231

L =
    6.9173         0
         0    2.6804
```

继续在 MATLAB 命令窗口中输入：

```
>> [V,L]=eigs(A,2,'sr')
```

输出计算结果为：

```
V =
   -0.1612    0.2623
    0.3553   -0.2811
   -0.9207    0.9231

L =
    1.4023         0
         0    2.6804
```

从结果可以看出，矩阵 A 有两个特征值为 1.4023，2.6804。

4.7 矩阵指数

矩阵级数 $e^{At}=\sum_{n=0}^{\infty}\frac{(At)^n}{n!}$ 称为矩阵指数。

如果按定义来求矩阵指数，显然运算量非常大。利用下面的两条性质很快可以算出一些矩阵的矩阵指数。

（1）如果矩阵 A 为对角阵 $diag(\lambda_1,\lambda_2,\cdots,\lambda_n)$，则它的矩阵指数为 $diag(e^{\lambda_1},e^{\lambda_2},\cdots,e^{\lambda_n})$；

（2）$\exp(P^{-1}AP)=P^{-1}\exp(A)P$；

因此要求一个矩阵的矩阵指数，可以先求其特征值，再用第二条性质来求其矩阵指数。

【例 4-19】 矩阵指数求取实例。求矩阵 $A=\begin{bmatrix} 3 & 2 & 6 \\ 1 & 0 & 5 \\ -4 & 8 & 9 \end{bmatrix}$ 的矩阵指数。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> A=[3 2 6;1 0 5; -4 8 9];
>> [V,L]=eig(A)
V =
    0.6767   -0.9129    0.2283
    0.3917   -0.3651    0.8380
    0.6234   -0.1826   -0.4956
L =
    9.6847         0         0
         0    5.0000         0
         0         0   -2.6847
>> syms t;
>> B=V*exp(L*t)*inv(V);
>> B=vpa(B,6)
```

由于 MATLAB 输出结果很长，将其整理如下：

$$e^{At}=\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

其中：

M_{11}=–0.575869*exp(9.68466*t)–0.975788+1.66667*exp(5*t)–0.0907977*exp(–2.68466*t);

M_{12}=0.851988*exp(9.68466*t)–0.02844–1.11111*exp(5*t)+0.259123*exp(–2.68466*t);

M_{13}=1.17537*exp(9.68466*t)–0.02104–1.11111*exp(5*t)–0.0642577*exp(–2.68466*t);

M_{21}=–0.333333*exp(9.68466*t)–2.65813+0.666669*exp(5*t)–0.333333*exp(–2.68466*t);

M_{22}=0.493161*exp(9.68466*t)+2.12226–0.444445*exp(5*t)+0.951284*exp(–2.68466*t);

M_{23}=0.680345*exp(9.68466*t)+2.31059–0.444446*exp(5*t)–0.235901*exp(–2.68466*t);

M_{31}=–0.530470*exp(9.68466*t)+0.168500+0.333333*exp(5*t)+0.197137*exp(–2.68466*t);

M_{32}=0.784821*exp(9.68466*t)–0.197920–0.222222*exp(5*t)–0.562599*exp(–2.68466*t);

M_{33}=1.08271*exp(9.68466*t)–1.14647–0.222222*exp(5*t)+0.139514*exp(–2.68466*t);

这样就轻松地得到了矩阵 ***A*** 的矩阵指数。

4.8 小结

本章详细讲述了 MATLAB 中进行矩阵特征值运算的常用解法，包括特征多项式法、幂法、收缩法、逆幂法和 QR 方法等。

进行矩阵特征值计算，用幂法求最大模特征值特别适用于稀疏矩阵，它的计算简单，如果用逆幂法结合位移技巧则可以加快收敛速度；QR 方法可以求全部的特征值，它对于中小型的稠密矩阵的特征值计算非常有效，特别对于对称矩阵，QR 算法简直可以说是完美的算法；此外，一些古典的方法如雅克比方法，不但过程十分复杂，而且与现代的一些方法相比无明显的优越性，因此前面没有介绍。

第 5 章

求导与微分计算

求导数和微分是数值计算的基本内容，很多和方程有关的数值计算，如插值与拟合、方程求根和微分方程求解都涉及导数、微分的计算。

MATLAB 提供了专门求导数的符号函数 diff，能够帮助解决求取导数和微分的问题。而在工程应用中，通常没有必要求出函数的导数的表达式，工程师们感兴趣的是在某一点处能达到一定精度的数值导数，利用 MATLAB 也能方便地解决这类问题。

5.1 MATLAB 中和微分有关的函数

MATLAB 的符号运算工具箱包含了微积分运算、化简和代换、解方程等几个方面的工具。符号运算工具箱中与导数直接有关的函数是 diff，此外 MATLAB 中还提供了求多元函数的梯度、曲面法线以及求多变量函数的雅克比矩阵等一些函数。

1. diff 函数

函数 diff 既能够求一元函数的导数，还能求多元数组的差分，而且它还能够求符号函数对某个变量的偏导数及符号函数的全微分，它的具体用法如表 5-1 所示。

表 5-1 diff 函数

格　式	说　明
f = diff(fun)	求函数 fun 的一阶导数
f = diff(fun,'x')	求函数 fun 关于变量 x 的一阶导数
f = diff(fun, 'x',n)	求函数 fun 关于变量 x 的 n 阶导数
d = diff(X)	计算向量 $\boldsymbol{X}$ 的向前差分，$d(i)=\boldsymbol{X}(i+1)-\boldsymbol{X}(i)$，$i=1,2,\cdots,n-1$

续表

格　式	说　明
d = diff(*X*, *n*)	计算向量 *X* 的 *n* 阶向前差分。例如，diff(*X*,2)=diff(diff(*X*))
d = diff(*A*, *n*, dim)	计算矩阵 *A* 的 *n* 阶差分，dim=1 时(默认状态)，按列计算差分；dim=2，按行计算差分
d = diff(*S*)	求符号函数 *S* 的全微分
d = diff(*S*, 'x')	求符号函数 *S* 关于变量 *x* 的偏导数
d = diff(*S*,*n*)	将符号函数 *S* 微分 *n* 次
d = diff(*S*, 'x',*n*)	求符号函数 *S* 关于变量 *x* 的 *n* 阶偏导数

【例 5-1】 一元求导函数应用实例。采用 diff 函数求导。

解： 在 MATLAB 命令窗口中输入下列命令，并得出结果如下：

```
>> df = diff('sin(x)');
df =
    cos(x)
>>  df = diff('sqrt(x)')
df =
    1/2/x^(1/2)
>> df = diff('log(x)')
df =
    1/x
>> df = diff('log(x*y)','x')
df =
    1/x
>> df = diff('sin(x*y)','y')
df =
    cos(x*y)*x
>> df = diff('sin(x*y)','x',3)
df =
    -cos(x*y)*y^3
```

从上面的例子中，可以看到函数 diff 的各种用法，功能十分强大。

2. gradient 函数和 surfnorm 函数

这两个函数适用于多元函数，曲面对应于一个二元函数，曲面的法线垂直于梯度的方向。它们的具体用法见表 5-2 所示。

表 5-2　gradient 函数和 surfnorm 函数

格　式	说　明
[gx, gy] = gradient(*F*)	求函数 *F* 的数值梯度，dx=dy=*1*
[gx, gy] = gradient(*F*, *H*)	求函数 *F* 的数值梯度，dx=dy=*H*
[gx, gy] = gradient(*F*, *Hx*, *Hy*)	求函数 *F* 的数值梯度，dx= *Hx*, dy=*Hy*
[gx, gy, . . .] = gradient(*F*, . . .)	用法和上面的类似，只不过变元为 *n* 个，在 *F* 后可指定每个变元的间距
[nx, ny, nz] = surfnorm(*X*, *Y*, Z)	给出(*X*,*Y*,*Z*)所表示曲面的法线

【例 5-2】 多元函数梯度计算实例。下面是某曲面上的一些离散点，求在每点处曲面沿 *x* 和 *y* 方向的数值梯度和法矢量。

x	1	1.2	1.4	2.3	5
y	0	-0.6	3	4	2
z	-1	7	7.2	9	1.4

解：在 MATLAB 命令窗口中输入下列命令：

```
>> F=[ 1 1.2 1.4 2.3 5;0 -0.6 3 4  2;-1 7 7.2 9 1.4];
>> [gx,gy]=gradient(F)  %计算梯度
```

输出结果为：

```
gx =
    0.2000    0.2000    0.5500    1.8000    2.7000
   -0.6000    1.5000    2.3000   -0.5000   -2.0000
    8.0000    4.1000    1.0000   -2.9000   -7.6000

gy =
   -1.0000   -1.8000    1.6000    1.7000   -3.0000
   -1.0000    2.9000    2.9000    3.3500   -1.8000
   -1.0000    7.6000    4.2000    5.0000   -0.6000
>> n=surfnorm(F)  %计算法向量
n =
   -0.1400   -0.0304   -0.4661   -0.8739   -0.6404
    0.8858   -0.4393   -0.5999    0.1416    0.8619
   -0.9930   -0.3153   -0.1761    0.3960    0.9955
```

3. jacobian 函数

多元函数的雅克比矩阵是一个非常重要的概念，多元函数的积分、矩阵微积分及坐标转换等都要用到它。在 MATLAB 中用 jacobian 函数求雅克比矩阵，jacobian 函数的用法如下：

```
j=jacobian(F,V)
```

它计算向量 $\boldsymbol{F}$ 对向量 $\boldsymbol{V}$ 的雅可比矩阵。

【例 5-3】 雅可比矩阵求取实例。计算函数向量 $\boldsymbol{f}=[x^2 \quad \sin(xyz) \quad e^t]^T$ 关于自变量 x，y，z，t 的雅克比矩阵。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> syms x y z t;
>> f = [x^2 sin(x*y*z)  exp(t)];
>> v = [ x y z t];
>> j = jacobian(f,v)
```

输出结果为：

```
j =
[           2*x,             0,             0,          0]
[ cos(x*y*z)*y*z, cos(x*y*z)*x*z, cos(x*y*z)*x*y,     0]
[             0,             0,             0,     exp(t)]
```

上例中 $\boldsymbol{f}$ 是函数向量，它有三个分量，每个分量都是 x，y,，z，t 的函数，因此它对自变量的雅克比矩阵是 3 行 4 列的。

5.2 求导数的其他方法

借助插值理论与数值积分，可以得到以下的几种求导数的方法。

5 2.1 中点公式

此方法的思想十分简单，由导数的定义知：

$$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x-h)}{2h}$$

在实际应用中，可以用有限的很小的 h 值代入右边的式子求得导数的近似值。表面上看，似乎步长越小，结果越精确，其实不然。当 h 值十分小时，$f(x+h)$ 的值与 $f(x-h)$ 的值很接近，在计算机上直接相减会造成有效数字的损失，因此 h 值也不能太小。

一般情况下，如果 $f(x)$ 足够光滑，h 取 0.1 就够了。

在 MATLAB 中编程实现的中点公式函数为：MidPoint。

功能：中点公式求取导数。

调用格式：df = MidPoint(func, $x0$, h)。

其中，func 为函数名；

$x0$ 为求导点；

h 为离散步长；

df 为导数值。

中点公式的 MATLAB 程序代码如下：

```
function df=MidPoint(func,x0,h)
%采用中点公式，求取函数 func 在 x0 处的导数
%函数名：func
%求导点：x0
%离散步长 ：h
%导数值： df

if nargin == 2
    h = 0.1;                                          %h 默认值为 0.1
else if (nargin == 3 && h == 0.0)
        disp('h 不能为 0! ');
        return;
    end
end

y1 = subs(sym(func), findsym(sym(func)),x0+h);
y2 = subs(sym(func), findsym(sym(func)),x0-h);
df = (y1-y2)/(2*h);                                   %中点公式
```

【例 5-4】 中点公式法求导数应用实例。采用中点公式法求函数 $f=\sqrt{x}$ 在 $x=4$ 处的导数。

解：在 MATLAB 命令窗口中输入：

```
>>df=MidPoint('sqrt(x)',4)
```

输出计算结果为：

```
df =
    0.2500
```

采用中点公式法求函数 $f=\sqrt{x}$ 在 $x=4$ 处的导数为 0.25，而导数的精确值也是 0.25。

5.2.2 三点公式法和五点公式法

三点公式是由等距节点插值公式得来的，其思想是先将函数用等距节点公式进行插值，然后对插值多项式求导数，再取有限的项数。

三点公式有以下三种形式，分别由等距牛顿前插、等距牛顿后插和斯特林公式得出。

（1）等距牛顿前插得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(-3y_0+4y_1-y_2)$$

（2）等距牛顿后插得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(3y_0-4y_{-1}+3y_{-2})$$

（3）斯特林公式得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(y_1-y_{-1})$$

上述式子中：

$$y_0=f(x_0),\quad y_1=f(x_0+h)$$

$$y_2=f(x_0+2h),\quad y_{-1}=f(x_0-h),\quad y_{-2}=f(x_0-2h)$$

五点公式法和三点公式法的得出过程是一样的，只不过取的项数比三点公式法多，如下所示。

$$f'(x_0)\approx\frac{1}{12h}(-25y_0+48y_1-36y_2+16y_3-3y_4)$$

$$f'(x_0)\approx\frac{1}{12h}(-3y_{-1}-10y_0+18y_1-16y_2+y_3)$$

$$f'(x_0)\approx\frac{1}{12h}(y_{-2}-8y_{-1}+8y_1-y_2)$$

$$f'(x_0)\approx\frac{1}{12h}(3y_1+10y_0-18y_{-1}+16y_{-2}-y_{-3})$$

$$f'(x_0)\approx\frac{1}{12h}(25y_0-48y_{-1}+36y_{-2}-16y_{-3}+3y_{-4})$$

在 MATLAB 中编程实现的三点公式法函数为：ThreePoint。

功能：三点公式法求函数导数。

调用格式：d*f* = ThreePoint(func, *x*0, type, *h*)

其中，func 为函数名；

*x*0 为求导点；

type 为三点公式的三种形式；

h 为离散步长；

d*f* 为导数值。

三点公式法的 MATLAB 程序代码如下：

```
function df=ThreePoint(func,x0,type,h)
%采用三点公式，求取函数 func 在 x0 处的导数
%函数名： func
%求导点： x0
%公式的形式：type（取值为 1，2，3）
%离散步长 ： h
%导数值： df

if nargin == 3
    h = 0.1;
else if (nargin == 4 && h == 0.0)
        disp('h 不能为 0! ');
        return;
    end
end

y0 = subs(sym(func), findsym(sym(func)),x0);
y1 = subs(sym(func), findsym(sym(func)),x0+h);
y2 = subs(sym(func), findsym(sym(func)),x0+2*h);
y_1 = subs(sym(func), findsym(sym(func)),x0-h);
y_2 = subs(sym(func), findsym(sym(func)),x0-2*h);

switch type
    case 1,
        df = (-3*y0+4*y1-y2)/(2*h);      %用第一个公式求导数

    case 2,
        df = (3*y0-4*y_1+y_2)/(2*h);     %用第二个公式求导数

    case 3,
        df = (y1-y_1)/(2*h);             %用第三个公式求导数

end
```

在 MATLAB 中编程实现的五点公式法函数为：FivePoint。

功能：五点公式法求函数导数。

调用格式：d*f* =FivePoint(func, *x*0, type, *h*)。

其中，func 为函数名；

*x*0 为求导点；

type 为五点公式的五种形式；

h 为离散步长；

d*f* 为导数值。

五点公式法的 MATLAB 程序代码如下：

```
function df=FivePoint(func,x0,type,h)
%采用五点公式，求取函数 func 在 x0 处的导数
%函数名： func
%求导点： x0
%公式的形式：type（取值为 1，2，3，4，5）
%离散步长 ： h
%导数值： df

if nargin == 3
    h = 0.1;
else if (nargin == 4 && h == 0.0)
        disp('h 不能为 0! ');
        return;
    end
end

y0 = subs(sym(func), findsym(sym(func)),x0);
y1 = subs(sym(func), findsym(sym(func)),x0+h);
y2 = subs(sym(func), findsym(sym(func)),x0+2*h);
y3 = subs(sym(func), findsym(sym(func)),x0+3*h);
y4 = subs(sym(func), findsym(sym(func)),x0+4*h);
y_1 = subs(sym(func), findsym(sym(func)),x0-h);
y_2 = subs(sym(func), findsym(sym(func)),x0-2*h);
y_3 = subs(sym(func), findsym(sym(func)),x0-3*h);
y_4 = subs(sym(func), findsym(sym(func)),x0-4*h);

switch type
    case 1,
        df = (-25*y0+48*y1-36*y2+16*y3-3*y4)/(12*h);    %用第一个公式求导数

    case 2,
        df = (-3*y_1-10*y0+18*y1-16*y2+y3)/(12*h);      %用第二个公式求导数

    case 3,
        df = (y_2-8*y_1+8*y1-y2)/(12*h);                %用第三个公式求导数

    case 4,
        df = (3*y1+10*y0-18*y_1+16*y_2-y_3)/(12*h);     %用第四个公式求导数

    case 5,
        df = (25*y0-48*y_1+36*y_2-16*y_3+3*y_4)/(12*h); %用第五个公式求导数
end
```

【例 5-5】 三点公式法求导数应用实例。采用三点公式法求函数 $f=\sin(x)$ 在 $x=2$ 处的导数。

解： 在 MATLAB 命令窗口中输入：

```
>> df1=ThreePoint('sin(x)',2,1);
>> df2=ThreePoint('sin(x)',2,2);
>> df3=ThreePoint('sin(x)',2,3);
```

用三种方法得到的计算结果为：

```
df1 =
   -0.4178

df2 =
   -0.4173

df3 =
   -0.4155
```

而函数 $f=\sin(x)$ 在 $x=2$ 处的导数为 $\cos(2)=-0.4161$，从上面三种方法得出的结果来看，都是十分接近的，其中以第三种方法的近似效果最佳。

【例 5-6】 五点公式法求导数应用实例。采用五点公式法求函数 $f=\sin(x)$ 在 $x=2$ 处的导数。

解：在 MATLAB 命令窗口中输入：

```
>> df1=FivePoint('sin(x)',2,1);
>> df2=FivePoint('sin(x)',2,2);
>> df3=FivePoint('sin(x)',2,3);
>> df4=FivePoint('sin(x)',2,4);
>> df5=FivePoint('sin(x)',2,5);
```

用五种方法得到的计算结果为：

```
df1 =
   -0.4161
df2 =
   -0.4161
df3 =
   -0.4161
df4 =
   -0.4161
df5 =
   -0.4161
```

而函数 $f=\sin(x)$ 在 $x=2$ 处的导数为 $\cos(2)=-0.4161$，从上面的结果来看，五点公式法的精度是很高的。

5.2.3 样条函数法

MATLAB 中的样条工具箱中提供了求样条函数的导数的函数，与求导有关的样条函数的操作如表 5-3 所示。

表 5-3 样条函数操作

格 式	说 明
Y = csape(x, y, conds, valconds)	构造各种边界条件下的三次插值样条函数
FY = fnder(Y)	对样条函数进行微分

表 5-3 中，conds 可取的值如下。

'complete'：给定端点的斜率，斜率大小在 valconds 中给出；

'not-a-knot'：两个端点存在三节连续导数；

'periodic'：给定周期特性；

'second'：给定端点的二阶导数，大小在 valconds 中给出；

'variational'和'vadahonal'：给定端点的二阶导数，且大小为 0。

【例 5-7】 样条函数法求导数应用实例。采用样条函数求下表所列数据点中前 6 个点处的数值导数。

x	0	0.5	1	1.5	2	2.5	3
y	0	0.25	1	2.25	4	6.25	9

解：在 MATLAB 命令窗口中输入：

```
>> x=0:0.5:3;
>> y=[0 0.25 1 2.25 4 6.25 9];
>> Y = csape(x,y,'second', [2, 2])
```

输出计算结果为：

```
Y =
      form: 'pp'
    breaks: [0 0.5000 1 1.5000 2 2.5000 3]
     coefs: [6x4 double]
    pieces: 6
     order: 4
       dim: 1
>> df = fnder(Y)
df =
      form: 'pp'
    breaks: [0 0.5000 1 1.5000 2 2.5000 3]
     coefs: [6x3 double]
    pieces: 6
     order: 3
       dim: 1
>> df.coefs
ans =
         0    2.0000         0
         0    2.0000    1.0000
         0    2.0000    2.0000
         0    2.0000    3.0000
         0    2.0000    4.0000
         0    2.0000    5.0000
```

最终结果保存在 df.coefs 中，其中最右边一列为一阶导数，中间一列为二阶导数，最左边一列为三阶导数。

题目中的数据点来自函数 $y=x^2$，理论上它在 $x=0, 0.5, 1, 1.5, 2, 2.5$ 处的一阶导数依次为 0，1，2，3，4，5，二阶导数都为 2，三阶导数全为 0。

从例子的结果可以看出，一阶导数、二阶导数和三阶导数的数值和理论值一样。

5.2.4 辛普森数值微分法

辛普森数值微分是用来求等距节点在节点处的导数的，辛普森数值微分公式如下：

$$\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 4 \end{bmatrix} \begin{bmatrix} f'(x_1) \\ f'(x_2) \\ f'(x_3) \\ \vdots \\ f'(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 3(y_2 - y_0)/h - f'(x_0) \\ 3(y_3 - y_1)/h \\ 3(y_4 - y_2)/h \\ \vdots \\ 3(y_n - y_{n-2})/h - f'(x_n) \end{bmatrix}$$

其中，$y_n = f(x_n)$，$x_n = x_0 + nh$。

如果端点导数值$-f'(x_0)$和$-f'(x_n)$未知，则将它们用中点微分公式近似，这时的辛普森数值微分公式为：

$$\begin{bmatrix} 2 & 0 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 0 & 2 \end{bmatrix} \begin{bmatrix} f'(x_1) \\ f'(x_2) \\ f'(x_3) \\ \vdots \\ f'(x_{n-1}) \end{bmatrix} = \begin{bmatrix} (y_2 - y_0)/h \\ 3(y_3 - y_1)/h \\ 3(y_4 - y_2)/h \\ \\ (y_n - y_{n-2})/h \end{bmatrix}$$

在 MATLAB 中编程实现的辛普森数值法（用于已知函数表达式）函数为：CISimpson。

功能：辛普森数值法求已知函数在某点的导数值。

调用格式：df = CISimpson (func, x0, n, h)。

其中，func 为函数名；

x0 为求导点；

n 为将已知函数离散的数据点数；

h 为离散步长；

df 为导数值。

辛普森数值微分法（适用于已知函数表达式）的 MATLAB 程序代码如下：

```
function df=CISimpson(func,x0,n,h)
%辛普森数值法求已知函数 func 在 x0 点的导数值
%函数名: func
%求导点: x0
%将已知函数离散的数据点数: n
%离散步长: h
%导数值: df

if nargin == 2                          %以下是参数的判断过程
    h = 0.1;
    n = 5;
else
    if (nargin == 3 )
        if (n<5)
            disp('n 不能小于 5!');
            return;
        else
            h = 0.1;
        end
    else (nargin == 4 && h == 0.0)
```

```
        disp('h不能为0! ');
        return;
    end
end

for(i=1:n)                              %这个循环计算节点的函数值
     if (mod(n,2) == 0)
        y(i)= subs(sym(func), findsym(sym(func)),x0+(i-n/2)*h);
    else
        y(i)= subs(sym(func), findsym(sym(func)),x0+(i-(n+1)/2)*h);
     end
end

f(1)=(y(3)-y(1))/(2*h);
f(2)=(y(n)-y(n-2))/(2*h);               %这两行用中心微分法给出端点的导数

b(1:n-2,1) = zeros(n-2,1);
b(1,1)=3*(y(3)-y(1))/h-f(1);
b(n-2,1)=3*(y(n)-y(n-2))/h-f(2);
for(i=2:(n-3))
    b(i,1) = 3*(y(i+2)-y(i))/h;
end                                     %这一块是辛普森公式的右边的列向量

for(i=1:n-2)
    for(j=1:n-2)
        if( (i == j+1) || (j == i+1))
            A(i,j)= 1;
        else if( i == j)
            A(i,j) = 4;
            end
        end
    end
end                                     %这一块是系数矩阵

[Q,R]=qr(A);
DF = R\(Q\b);                           %用 QR 分解法求解

if( mod(n,2) == 0)
    df = DF(n/2);
else
    df = DF((n+1)/2);
end                                     %这里是求出 x0 处的导数值
```

在 MATLAB 中编程实现的辛普森数值法（用于离散数据点）函数为：DISimpson。

功能：辛普森数值法求离散数据点在某点的导数值。

调用格式：df = DISimpson (*X*, *Y*, n, p)。

其中，*X* 为离散数据的 *x* 坐标向量；

Y 为离散数据的 *y* 坐标向量；

n 为数据点的个数；

p 为求第 *p* 个点处的导数；

d*f* 为导数值。

辛普森数值微分法（适用于数据向量）的另一个 MATLAB 程序代码如下：

```
function df = DISimpson(X,Y,n,p)
%辛普森数值法求 n 个数据点在第 p 个点处的导数
%离散数据的 x 坐标向量：X
%离散数据的 y 坐标向量：Y
%数据点的个数：n
%要求导数的点：p
%导数值： df

if n < 5
    disp('n 不能小于 5!');
    return;
end                                          %n 为数据点个数，p 为要求导数值的数据点编号

if p == 0
    disp('p 不能等于 0!');
    return;
end

h = X(2)-X(1);
xx =linspace(X(1),X(n),h);
if(xx ~= X)
    disp('节点之间不是等距的! ');
    return;
end                                          %判断是否是等距节点

f(1)=(Y(3)-Y(1))/(2*h);
f(2)=(Y(n)-Y(n-2))/(2*h);                    %这两行用中心微分法给出端点的导数
b(1,1)=3*(Y(3)-Y(1))/h-f(1);
b(n-2,1)=3*(Y(n)-Y(n-2))/h-f(2);
for(i=2:n-3)
    b(i,1) = 3*(Y(i+2)-Y(i))/h;
end                                          %这一块是辛普森公式的右边的列向量

for(i=1:n-2)
    for(j=1:n-2)
        if( (i == j+1) || (j == i+1))
            A(i,j) = 1;
        else if( i == j)
            A(i,j) = 4;
           end
        end
    end
end                                          %这一块是系数矩阵

[Q,R]=qr(A);
DF = R\(Q\b);                                %用 QR 分解法求解

if( p == 1)
    df = f(1);
else
    df = DF(p-1);                            %这里是求出第 p 个节点处的导数值
end
```

【例 5-8】 辛普森数值微分法应用实例之一。分别采用 5，10，100 个离散点的辛普森数值微分法求函数 $f=e^x$ 在 $x=2.5$ 的导数值。

解：在 MATLAB 命令窗口中输入：

```
>> df = CISimpson('exp(x)',2.5)                    %采用默认的 5 个离散点
```

输出计算结果为：

```
df =
   13.8179
>> df = CISimpson('exp(x)',2.5,10)                 %采用 10 个离散点计算
```

输出计算结果为：

```
df =
   13.4528
>> df = CISimpson('exp(x)',2.5,100)                %采用 100 个离散点计算
```

输出计算结果为：

```
df =
   13.4637
```

而函数 $f=e^x$ 在 $x=2.5$ 处的导数值为 $e^{2.5}=12.1825$。

辛普森数值微分公式的缺点在于：如果端点的导数没有给定，则它的精度会比较低，当然如果端点的导数未知，可以采用其他的精度更高的方法来近似端点导数值，上面列出的代码中是采用中点微分公式来近似的，精度显然比较低。

【例 5-9】 辛普森数值微分法应用实例之二。用辛普森数值微分法求下表所列数据点在 $x=1$ 处的导数值。

x	0	0.5	1	1.5	2	2.5	3
y	0	0.125	1	3.375	8	15.625	27

解：在 MATLAB 命令窗口中输入：

```
>> x=0:0.5:3;
>> y=[0 0.125 1 3.375 8 15.625 27];
>> df=DISimpson(x,y,7,3)
```

输出计算结果为：

```
df =
    3.0308
```

表格中的数据点是根据函数 $f=x^3$ 来离散的，$f=x^3$ 在 $x=1$ 处的导数为 3.00，辛普森数值微分法的结果还是比较好的。

5.2.5 理查森外推算法

理查森外推算法的迭代公式如下：

$$G_1(h)=\frac{f(x+h/2)-f(x-h/2)}{h}$$

$$G_{n+1}(h)=\frac{G_n(\frac{h}{2})-(\frac{1}{2})^{2n}G_n(h)}{1-(\frac{1}{2})^{2n}}$$

当迭代步数充分大时，$G(h)$ 收敛到 $f'(x)$。此算法是一种金字塔式的算法，底层是 $G_1(h)$，$G_1(\frac{h}{2})$，…，$G_1(\frac{h}{2^n})$，第二层是 $G_2(h)$，$G_2(\frac{h}{2})$，…，$G_2(\frac{h}{2^{n-1}})$，顶层是 $G_{n-1}(h)$。

在 MATLAB 中编程实现的理查森外推算法的函数为：Richason。

功能：理查森外推算法求已知函数在某点的导数值。

调用格式：d*f* = Richason (func, *x*0, *n*, *h*)。

其中，func 为函数名；

*x*0 为求导点；

n 为将已知函数离散的数据点数；

h 为离散步长；

d*f* 为导数值。

理查森外推算法的 MATLAB 程序代码如下：

```
function df = Richason(func, x0, n, h)
%理查森外推算法求已知函数 func 在 x0 点处的导数值
%函数名: func
%求导点: x0
%将已知函数离散的数据点数: n
%离散步长: h
%导数值: df

if nargin == 3
    h = 1;
else if (nargin == 3 && h == 0.0)
        disp('h不能为0! ');
        return;
    end
end

for(i=1:n)
    y1 = subs(sym(func), findsym(sym(func)),x0+h/(2^i));
    y2 = subs(sym(func), findsym(sym(func)),x0-h/(2^i));
    G(i) = 2^(i-1)*(y1-y2)/h;                %求得金字塔的底层值
end

G1 = G;
for(i=1:n-1)
    for(j=(i+1):n)
        G1(j)=(G(j)-(0.5)^(2*i)*G(j-1))/(1-(0.5)^(2*i)); %求得金字塔的每层值
    end
    G = G1;
```

```
end

df = G(n);      %顶层值就是所需的导数值
```

【例 5-10】 理查森外推算法求取导数应用实例。采用理查森外推算法求函数 $y=2^x$ 在 $x=1$ 处的导数值。

解：在 MATLAB 命令窗口中输入：

```
>> df =Richason('2^x',1, 8)
```

输出计算结果为：

```
df =
    1.3863
```

而函数 $y=2^x$ 在 $x=1$ 处的导数的准确值为 2*log(2)=1.3863，经过试验可以发现，对于各种函数，理查森外推算法都能达到很高的精度。

5.3 小结

数值微分和数值积分在工程中都有很多的应用，毕竟几乎没有哪个工程问题不会涉及微分与积分，而一般情况下能够求出具体的微分和积分的表达式的情况很少，因此近来数值微分和数值积分发展了很多精度高的算法，而且高阶导数的求法和奇异积分的处理也有了很大的进步。

第 6 章

积分计算

数值积分可用于计算解析定义的函数的积分，也可以计算以列表形式给出的函数的积分，其基本原理是采用多项式近似原函数，然后用该多项式的积分近似原函数的积分。

数值积分是一种十分基础而且重要的运算。MATLAB 既提供了求不定积分的函数，也提供了求定积分的函数，方便进行数值积分求解。

6.1 MATLAB 中的不定积分函数

函数 int 的功能十分强大，它不但能计算普通的初等函数的不定积分，还能计算各类非初等函数的不定积分，例如贝塞尔函数的不定积分。它的调用格式如下。

（1）int（F）

该函数用于对符号表达式 F 求不定积分，自变量为 F 中的符号自变量，如果 F 为常数，自变量为 x；

（2）int（F,s）

该函数计算符号表达式 F 对自变量 s 的不定积分；

【例 6-1】 MATLAB 中求不定积分应用实例。采用 int 函数，计算下列不定积分：$f=\int x\sin x\mathrm{d}x$，$f=\int y\sin x\mathrm{d}x$，$f=\int 4\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> f=int('sin(x)*x')
  f =
     sin(x)-x*cos(x)
>> f=int('sin(x)*y','x')
  f =
```

```
      -y*cos(x)
>> f=int('4')
  f =
      4*x
```

由计算结果可知，$\int x\sin x\mathrm{d}x=\sin x-x\cos x$，$\int y\sin x\mathrm{d}x=-y\cos x$，$\int 4\mathrm{d}x=4x$。

int 函数是最基本的求积分的函数，也是最通用的求积分的函数。

6.2 MATLAB 中的定积分函数

6.2.1 定积分计算函数

MATLAB 中求定积分的函数有 int、trapz、quad 和 quadl，下面分别予以介绍。

（1）int 是通用的方法，适用性比较广，精度一般，其调用格式为：

- int(*F*, *a*, *b*)：计算符号表达式 *F* 从 *a* 到 *b* 的定积分；
- int(*F*, *s*, *a*, *b*)：计算符号表达式 *F* 对自变量 *s* 从 *a* 到 *b* 的定积分。

（2）trapz 通过计算若干梯形面积的和来近似某函数的积分，其调用格式为：

- trapz(*Y*)：用等距节点法近似计算 ***Y*** 的积分。若 ***Y*** 是一向量，则 trapz(**Y**)为 ***Y*** 的积分；若 ***Y*** 是一矩阵，则 trapz(**Y**)为 ***Y*** 的每一列的积分；
- trapz(*X*，*Y*)：用梯形法计算 *Y* 在 *X* 点上的积分。

（3）quad 和 quadl 是基于数学上的正方形概念来计算函数的面积的，这些积分函数的操作方式一样。为获得更准确的结果，两个函数在所需的区间都要计算被积函数。其调用格式为：

- quad(*F*, *a*, *b*)：从 *a* 到 *b* 计算函数 *F* 的数值积分，误差为 10^{-6}。若给 *F* 输入向量 ***x***，应返回向量 ***y***，即 *F* 是一单值函数；
- quad(*F*, *a*, *b*, tol)：用指定的绝对误差 tol 代替默认误差；
- quadl(*F*, *a*, *b*)：用高精度进行数值积分计算，效率可能比 quad 更高。

此外，与简单的梯形比较，这两个函数进行更高阶的近似，而且 quad8 比 quad 更精确。

【例 6-2】 MATLAB 中求定积分应用实例。计算定积分 $f=\int_0^1(x^3+\sin x)\mathrm{d}x$

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=int('x^3+sin(x)', 1,3)
q =
    20-cos(3)+cos(1)
```

由计算结果可知，$\int_0^1(x^3+\sin x)\mathrm{d}x=20-\cos(3)+\cos(1)$。

6.2.2 二重积分计算函数

MATLAB 提供了计算重积分的函数，如下所述。

（1）dblquad 函数用于矩形区域上的二重积分的数值计算，其调用格式如下。

- dblquad(F, $x1$, $x2$, $y1$, $y2$)：调用函数 quad 在区域[$x1$，$x2$，$y1$，$y2$]上计算二元函数 $z=F(x, y)$的二重积分；
- dblquad(fun, $x1$, $x2$, $y1$, $y2$, tol)：用指定的精度 tol 代替默认精度 10^{-6}，再进行计算；
- dblquad(F, $x1$, $x2$, $y1$, $y2$, tol, method)：用指定的算法 method 代替默认算法 quad。method 的取值为 quadl，或是用户指定的、与命令 quad 与 quadl 有相同调用次序的函数；

（2）quad2dggen 用于任意区域上二元函数的数值积分，其调用格式如下。

- q = quad2dggen(fun，xlower，xupper，ymin，ymax)：在由[xlower，xupper，ymin，ymax]指定的区域上计算二元函数 z=f(x，y)的二重积分。
- q = dblquad(fun，xlower，xupper，ymin，ymax，tol)：用指定的精度 tol 代替默认精度 10^{-6}，再进行计算。
- q = dblquad(fun，xmin，xmax，ymin，ymax，tol，method)：用指定的算法 method 代替默认算法。method 的取值为默认算法，或是用户指定的、与默认命令有相同调用次序的函数句柄。
- q = dblquad(fun，xlower，xupper，ymin，ymax，tol，method,$p1$，$p2$，…)：将可选参数 $p1$，$p2$，..等传递给函数 fun(x，y，p1，p2，…)。若 tol=[]，method=[]，则使用默认精度和算法。

【例 6-3】 MATLAB 中求取重积分应用实例。计算二重积分 $\iint\limits_{\substack{-1\leqslant x\leqslant 1,\\ 0\leqslant y\leqslant 2}} \sin x\sqrt{y}\,\mathrm{d}x\mathrm{d}y$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=dblquad('sin(x)*sqrt(y)',-1,1,0,2)
q =
    -4.4306e-018
```

所以 $\int_{-1}^{1}\int_{0}^{2}(\sin x\sqrt{y})\mathrm{d}x\mathrm{d}y=-4.4306\mathrm{e}-018$。理论上根据二重积分的性质，上面的积分值应该为 0。

6.2.3 三重积分计算

MATLAB 中还有如下的求三元重积分的函数。

- triplequad(F, $x1$, $x2$, $y1$, $y2$, $z1$, $z2$) ：调用函数 triplequad 在区域[$x1$，$x2$，$y1$，$y2$,$z1$,$z2$]上计算三元函数 F=F(x，y，z)的三重积分；
- triplequad(F, $x1$, $x2$, $y1$, $y2$, $z1$, $z2$, tol)：用指定的精度 tol 代替默认精度 10^{-6}，再进行计算；
- triplequad(F, $x1$, $x2$, $y1$, $y2$, $z1$, $z2$, tol, method)：用指定的算法 method 代替默认算法 triplequad。；

【例 6-4】 MATLAB 中求取三重积分应用实例。计算三重积分 $\int_0^{\pi}\int_0^1\int_{-1}^1(y\sin x+z\cos x)$ dxdydz 。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> F = @(x,y,z)y*sin(x)+z*cos(x);
>> q = triplequad(F,0,pi,0,1,-1,1)
q =
    2.0000
```

所以 $\int_0^{\pi}\int_0^1\int_{-1}^1(y\sin x+z\cos x)\mathrm{d}x\mathrm{d}y\mathrm{d}z=2.000$ 。

6.3 梯形法数值积分

梯形法数值积分采用的梯形公式是最简单的数值积分公式，其计算表达式为：

$$\int_a^b f(x)\mathrm{d}x\approx\frac{b-a}{2}[f(a)+f(b)]$$

由于用梯形公式来求积分十分粗糙，误差也比较大，后来改进后提出了复合梯形公式：

$$\int_a^b f(x)\mathrm{d}x\approx\frac{h}{2}[f(a)+2\sum_{k=1}^{n-1}f(x_k)+f(b)]$$

其中 $$h=\frac{b-a}{n}$$ 。

在 MATLAB 中编程实现的复合梯形公式的函数为：CombineTraprl。

功能：复合梯形公式求函数的数值积分。

调用格式：[I, step]= CombineTraprl(f, a, b, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值；

step 为积分划分的区间个数

复合梯形公式的 MATLAB 代码如下：

```
function [I,step] = CombineTraprl(f,a,b,eps)
%复合梯形公式求函数 f 在区间[a,b]上的定积分
%函数名： f
%积分下限： a
%积分下限： b
%积分精度： eps
%积分值： I
%积分划分的子区间个数： step

if(nargin==3)
```

```
    eps=1.0e-4;                                %默认精度为 0.0001
end

n=1;
h=(b-a)/2;
I1=0;
I2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h;
while abs(I2-I1)>eps
    n=n+1;
    h=(b-a)/n;
    I1=I2;
    I2=0;
    for i=0:n-1                %第 n 次的复合梯形公式积分
        x=a+h*i;               %i=0 和 n-1 时，分别代表积分区间的左右端点
        x1=x+h;
        I2=I2+(h/2)*(subs(sym(f),findsym(sym(f)),x)+...
              subs(sym(f),findsym(sym(f)),x1));
    end
end

I=I2;
step=n;
```

【例 6-5】 复合梯形法求取数值积分实例。利用复合梯形法计算定积分 $\int_2^4 \frac{1}{x^2-1}\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=CombineTraprl('1/(x^2-1)',2,4)
q =
    0.2945
s =
    15
>> [q,s]=CombineTraprl('1/(x^2-1)',2,4,1.0e-6)
q =
    0.2939
s =
    66
```

所以从复合梯形公式可得出 $\int_2^4 \frac{1}{x^2-1}\mathrm{d}x \approx 0.2939$。

从第二步结果可以看出，为了达到 1.0e–6 的积分精度，复合梯形公式采用了 66 个积分子区间。

6.4 辛普森法数值积分

辛普森公式的数值积分公式为：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}[f(a)+4f(\frac{a+b}{2})+f(b)]$$

它的几何意义为用通过三点 $(a,f(a))$，$(\frac{a+b}{2},f(\frac{a+b}{2}))$，$(b,f(b))$ 的抛物线围成的曲

边形面积来代替给定函数的积分。精度高一些的还有辛普森3/8公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{8}[f(a)+3f(\frac{2a+b}{3})+3f(\frac{a+2b}{3})+f(b)]$$

同梯形公式一样，也有复合辛普森公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+\frac{1}{2}})+f(x_{k+1})]$$

$$f(x_0)=f(a), f(x_n)=f(b)$$

其中 $x_{k+\frac{1}{2}}=\frac{x_k+x_{k+1}}{2}$，$h=\frac{b-a}{n}$。

在 MATLAB 中编程实现的辛普森系列公式的函数为：IntSimpson。

功能：辛普森系列公式求函数的数值积分。

调用格式：[*I*, step] = IntSimpson(f, a, b, type, eps)。

其中，f 为函数名；

a 为积分下限；

b 为积分上限；

type 为辛普森公式的类型；

eps 为积分精度；

I 为积分值；

step 为积分划分的区间个数。

辛普森系列公式的 MATLAB 代码如下：

```
function [I,step] = IntSimpson(f,a,b,type,eps)
%辛普森系列公式求函数 f 在区间[a,b]上的定积分
%函数名：f
%积分下限：a
%积分上限：b
%辛普森公式的类型：type（可取 1，2，3）
%积分精度：eps
%积分值： I
%积分划分的子区间个数：step

if(type==3 && nargin==4)
    disp('缺少参数！');
end

I=0;

switch type
    case 1,                    %辛普森公式
        I=((b-a)/6)*(subs(sym(f),findsym(sym(f)),a)+...
            4*subs(sym(f),findsym(sym(f)),(a+b)/2)+...
            subs(sym(f),findsym(sym(f)),b));
        step=1;

    case 2,                    %辛普森 3/8 公式
        I=((b-a)/8)*(subs(sym(f),findsym(sym(f)),a)+...
```

```
            3*subs(sym(f),findsym(sym(f)),(2*a+b)/3)+ ...
          3*subs(sym(f),findsym(sym(f)),(a+2*b)/3)+subs(sym(f),findsym
            (sym(f)),b));
        step=1;

    case 3,                    %复合辛普森公式
        n=2;
        h=(b-a)/2;
        I1=0;
        I2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),
                b))/h;
        while abs(I2-I1)>eps
            n=n+1;
            h=(b-a)/n;
            I1=I2;
            I2=0;
            for i=0:n-1
                x=a+h*i;
                x1=x+h;
                I2=I2+(h/6)*(subs(sym(f),findsym(sym(f)),x)+...
                    4*subs(sym(f),findsym(sym(f)),(x+x1)/2)+...
                    subs(sym(f),findsym(sym(f)),x1));
            end
        end
        I=I2;
        step=n;
end
```

【例 6-6】 辛普森法求取数值积分实例。利用辛普森法计算积分 $\int_0^{10}\sin x\mathrm{d}x$ 。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=IntSimpson('sin(x)',0,10,1)
q =
   -7.2995
s =
     1
>> [q,s]=IntSimpson('sin(x)',0,10,2)
q =
    0.0084
s =
     1
>> [q,s]=IntSimpson('sin(x)',0,10,3)
q =
    1.8393
s =
    13
```

上面的例子是分别用辛普森公式、辛普森 3/8 公式和复合辛普森公式算出来的结果，当然用复合辛普森公式算出来的才是正确的结果，即 $\int_0^{10}\sin x\mathrm{d}x\approx 1.8393$ 。

函数 sin(x)在区间[0,10]的积分的准确值为–cos(10)+1，大约为 1.8391，由此可见如果积分区间不是被积函数的单调区间的话，辛普森公式和辛普森 3/8 公式的误差将是很大的，而复合辛普森公式显然不存在这种问题。

6.5 牛顿-科茨法数值积分

牛顿-科茨积分公式是通过在 n 个等距节点上用 $n-1$ 阶多项式对被积函数进行插值来逼近被积函数，并对逼近函数求积分来获得的。其积分公式的表达式如下：

$$\begin{cases} \int_a^b f(x)\mathrm{d}x \approx (b-a)\sum_{k=0}^{n} C_k f(x_k) \\ f(x_0) = f(a) \\ \\ f(x_n) = f(b) \end{cases}$$

梯形公式对应于上式中的 $n=1$ 的情况，辛普森公式对应于上式中的 $n=2$ 的情况。一些比较常用的积分公式列出如下。

（1）科茨公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{90}[7f(a)+32f(\frac{3a+b}{4})+12f(\frac{a+b}{2})+32f(\frac{a+3b}{4})+7f(b)]$$

（2）牛顿-科茨六点公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{288}[19f(a)+75f(\frac{4a+b}{5})+50f(\frac{3a+2b}{5})+50f(\frac{2a+3b}{5})+75f(\frac{a+4b}{5})+19f(b)]$$

（3）牛顿-科茨七点公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{840}[41f(a)+216f(\frac{5a+b}{6})+27f(\frac{2a+b}{3})+272f(\frac{a+b}{2})$$
$$+27f(\frac{a+2b}{3})+216f(\frac{a+5b}{6})+41f(b)]$$

当 $n \leqslant 6$ 时，n 越大，积分精度越高；但是当 $n>6$ 时，积分的稳定性得不到保证，因此一般情况下 n 最大取 6 就够了。

在 MATLAB 中编程实现的牛顿-科茨系列公式的函数为：NewtonCotes。

功能：牛顿-科茨系列公式求函数的数值积分。

调用格式：*I* = NewtonCotes(f, a, b, type)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

type 为牛顿-科茨公式的类型；

I 为积分值。

牛顿-科茨系列公式的 MATLAB 代码如下：

```
function I = NewtonCotes(f,a,b,type)
%牛顿一科茨系列公式求函数 f 在区间[a,b]上的定积分
%函数名: f
%积分下限: a
%积分上限: b
```

```
%辛普森公式的类型：type（可取 1，2，3）
%积分值： I

if(type==3 && nargin==4)
    eps=1.0e-4;                          %默认精度为 0.0001
end

I=0;

switch type
    case 1,                    %科茨公式
        I=((b-a)/90)*(7*subs(sym(f),findsym(sym(f)),a)+...
            32*subs(sym(f),findsym(sym(f)),(3*a+b)/4)+...
            12*subs(sym(f),findsym(sym(f)),(a+b)/2)+...
            32*subs(sym(f),findsym(sym(f)),(a+3*b)/4)+...
             7*subs(sym(f),findsym(sym(f)),b));

    case 2,                    %牛顿—科茨六点公式
        I=((b-a)/288)*(19*subs(sym(f),findsym(sym(f)),a)+...
            75*subs(sym(f),findsym(sym(f)),(4*a+b)/5)+...
            50*subs(sym(f),findsym(sym(f)),(3*a+2*b)/5)+...
            50*subs(sym(f),findsym(sym(f)),(2*a+3*b)/5)+...
            75*subs(sym(f),findsym(sym(f)),(a+4*b)/5)+...
            19*subs(sym(f),findsym(sym(f)),b));

    case 3,                    %牛顿—科茨七点公式
        I=((b-a)/840)*(41*subs(sym(f),findsym(sym(f)),a)+...
            216*subs(sym(f),findsym(sym(f)),(5*a+b)/6)+...
            27*subs(sym(f),findsym(sym(f)),(2*a+b)/3)+...
            272*subs(sym(f),findsym(sym(f)),(a+b)/2)+...
            27*subs(sym(f),findsym(sym(f)),(a+2*b)/3)+...
            216*subs(sym(f),findsym(sym(f)),(a+5*b)/6)+...
            41*subs(sym(f),findsym(sym(f)),b));
end
```

【例 6-7】 牛顿-科茨系列公式求取数值积分实例。利用牛顿-科茨系列公式计算积分 $\int_0^{10}\sin x\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=NewtonCotes('sin(x)',0,10,1)  %科茨公式计算
q =
    3.7613
>> q=NewtonCotes('sin(x)',0,10,2)    %牛顿—科茨六点公式计算
q =
    2.7865
>> q=NewtonCotes('sin(x)',0,10,3)    %牛顿—科茨七点公式
q =
    1.5296
```

上面的例子是分别用科茨公式、牛顿-科茨六点公式和牛顿-科茨公式七点公式算出来的结果，当然第三个结果才是准确的，即 $\int_0^{10}\sin x\mathrm{d}x\approx 1.5296$。这个例子说明低阶的牛顿-科茨公式也存在同辛普森公式一样的缺点。

6.6 高斯系列公式数值积分

6.6.1 高斯公式

高斯积分公式的思想是用 n 个不等距的节点 x_1，x_2,…，x_n 对被积函数进行插值，然后对插值后的函数进行积分，其积分公式为：

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{k=0}^{n} A_k f(x_k)$$

如果积分区间不是[−1,1]，则需转换到此区间：

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}t + \frac{a+b}{2})dt$$

其中系数 A_k，节点 x_k 与 n 的关系如表 6-1 所示。

表 6-1 高斯公式节点和系数表

n	x_k	A_k
0	0.0000000	2.0000000
1	±0.5773503	1.0000000
2	±0.7745967	0.55555556
	0.0000000	0.88888889
3	±0.8611363	0.3478548
	±0.3398810	0.6521452
4	±0.9061793	0.2369269
	±0.5384693	0.4786287
	0.0000000	0.5688889

在 MATLAB 中编程实现的高斯公式的函数为：IntGauss。

功能：高斯公式求函数的数值积分。

调用格式：I = IntGauss(f, a, b, n, AK, XK)。

其中，f 为函数名；

a 为积分下限；

b 为积分上限；

n 为积分所用项数；

AK 为自定义高斯公式中的系数；

XK 为自定义高斯公式中的节点坐标；

I 为积分值。

高斯公式的 MATLAB 代码如下：

```
function I = IntGauss(f,a,b,n,AK,XK)
%高斯公式求函数 f 在区间[a,b]上的定积分
%函数名: f
```

```
%积分下限：a
%积分上限：b
%积分所用项数：n
%自定义高斯公式中的系数：AK
%自定义高斯公式中的节点坐标：XK
%积分值： I

%AK 为自定义系数
%XK 为自定义节点
if(n<5 && nargin == 4)
    AK = 0;
    XK = 0;
Else                                          %如果 n>4，则节点和系数由调用者给出
    XK1=((b-a)/2)*XK+((a+b)/2);
    I=((b-a)/2)*sum(AK.*subs(sym(f),findsym(f),XK1));
end

ta = (b-a)/2;
tb = (a+b)/2;


switch n
    case 0, %n=0 时，采用表 6-1 中对应的系数 xk 和 Ak 进行计算
       I=2*ta*subs(sym(f),findsym(sym(f)),tb);

    case 1, %n=1 时，采用表 6-1 中对应的系数 xk 和 Ak 进行计算
       I=ta*(subs(sym(f),findsym(sym(f)),ta*0.5773503+tb)+...
          subs(sym(f),findsym(sym(f)),-ta*0.5773503+tb));

    case 2, %n=2 时，采用表 6-1 中对应的系数 xk 和 Ak 进行计算
       I=ta*(0.55555556*subs(sym(f),findsym(sym(f)),ta*0.7745967+tb)+...
          0.55555556*subs(sym(f),findsym(sym(f)),-ta*0.7745967+tb)+...
          0.88888889*subs(sym(f),findsym(sym(f)),tb));

    case 3, %n=3 时，采用表 6-1 中对应的系数 xk 和 Ak 进行计算
       I=ta*(0.3478548*subs(sym(f),findsym(sym(f)),ta*0.8611363+tb)+...
          0.3478548*subs(sym(f),findsym(sym(f)),-ta*0.8611363+tb)+...
          0.6521452*subs(sym(f),findsym(sym(f)),ta*0.3398810+tb)...
          +0.6521452*subs(sym(f),findsym(sym(f)),-ta*0.3398810+tb));
   case 4,  %n=4 时，采用表 6-1 中对应的系数 xk 和 Ak 进行计算
       I=ta*(0.2369269*subs(sym(f),findsym(sym(f)),ta*0.9061793+tb)+...
          0.2369269*subs(sym(f),findsym(sym(f)),-ta*0.9061793+tb)+...
          0.4786287*subs(sym(f),findsym(sym(f)),ta*0.5384693+tb)...
          +0.4786287*subs(sym(f),findsym(sym(f)),-ta*0.5384693+tb)+...
          0.5688889*subs(sym(f),findsym(sym(f)),tb));
end
```

【例 6-8】 高斯公式数值积分应用实例之一。利用高斯公式数值积求解积分 $\int_0^1 x^9 \mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGauss('x^9',0,1,3)
```

```
q =
    0.0999
```

所以由高斯公式可得到 $\int_0^1 x^9 \mathrm{d}x \approx 0.0999$。

可以证明高斯公式对多项式积分是十分精确的。

【例 6-9】 高斯公式数值积分应用实例之二。利用高斯公式数值积求解积分 $\int_0^{10} \sin x \mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGauss('sin(x)',0,10,4)
q =
    1.8167
```

所以由高斯公式可得到 $\int_0^{10} \sin x \mathrm{d}x \approx 1.8167$。

从【例子 6-9】可以看出，高斯公式对于被积函数在积分区间上不单调的情况下不存在同辛普森公式一样的缺点。

6.6.2 高斯-拉道公式

高斯-拉道求积公式如下：

$$I(f) = \int_{-1}^{1} f(x)\mathrm{d}x \approx \frac{2}{n^2} f(-1) + \sum_{k=2}^{n} A_k f(x_k)$$

前几个高斯-拉道节点和系数的取值与 n 的关系如表 6-2 所示。

表 6-2 高斯-拉道公式节点和系数表

n	x_k	A_k
2	−1 0.333333	0.5 1.5
3	−1 −0.289898 0.689898	0.222222 0.752806 1.024972
4	−1 −0.575319 0.181066 0.822824	0.125000 0.657689 0.776387 0.440925
5	−1 −0.720480 −0.167181 0.446314 0.885792	0.080000 0.446207 0.623653 0.562712 0.287427

在 MATLAB 中编程实现的高斯-拉道公式的函数为：IntGaussLada。

功能：高斯-拉道公式求函数的数值积分。

调用格式：I = IntGaussLada(f, a, b, n, AK, XK)。

其中，f 为函数名；

a 为积分下限；

b 为积分上限；

n 为积分所用项数；

AK 为自定义高斯-拉道公式中的系数；

XK 为自定义高斯-拉道公式中的节点坐标；

I 为积分值；

高斯-拉道公式的 MATLAB 代码如下：

```
function I = IntGaussLada(f,a,b,n,AK,XK)
%高斯—拉道公式求函数 f 在区间[a,b]上的定积分
%函数名： f
%积分下限： a
%积分上限： b
%积分所用项数： n
%自定义高斯—拉道公式中的系数： AK
%自定义高斯—拉道公式中的节点坐标： XK
%积分值： I

if(n<6 && nargin == 4)
    AK = 0;
    XK = 0;
else                                    %如果 n>5，则节点和系数由调用者给出
    XK1=((b-a)/2)*XK+((a+b)/2);
    I=((b-a)/2)*((2/n/n)*subs(sym(f),findsym(sym(f)),a)+...
              sum(AK.*subs(sym(f),findsym(sym(f)),XK1)));
end

ta = (b-a)/2;
tb = (a+b)/2;


switch n                %以下是对不同的 n 采取不同的节点和系数
    case 2,             %n=2 时，采用表 6-2 中对应的系数进行计算
        I=ta*0.5*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+...
            1.5*ta*subs(sym(f),findsym(sym(f)),ta*(1/3)+tb);

    case 3,             %n=3 时，采用表 6-2 中对应的系数进行计算
        I=ta*((2/9)*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+...
            0.752806*subs(sym(f),findsym(sym(f)),ta*(-0.289898)+tb)+...
            1.024972*subs(sym(f),findsym(sym(f)),ta*0.689898+tb));

    case 4,             %n=4 时，采用表 6-2 中对应的系数进行计算
        I=ta*(0.125*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+...
            0.657689*subs(sym(f),findsym(sym(f)),-ta*0.575319+tb)+...
            0.776387*subs(sym(f),findsym(sym(f)),ta*0.181066+tb)+...
            0.440925*subs(sym(f),findsym(sym(f)),ta*0.822824+tb));

    case 5,             %n=5 时，采用表 6-2 中对应的系数进行计算
```

```
        I=ta*(0.08*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+...
            0.446207*subs(sym(f),findsym(sym(f)),-ta*0.720480+tb)+...
            0.623653*subs(sym(f),findsym(sym(f)),-ta*0.167181+tb)+...
            0.562712*subs(sym(f),findsym(sym(f)),ta*0.446314+tb)+...
            0.287427*subs(sym(f),findsym(sym(f)),ta*0.885792+tb));
end
```

【例 6-10】 高斯-拉道公式数值积分应用实例。利用高斯-拉道公式数值积求解积分 $\int_0^{10} e^{-x}\sin x dx$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,3)
q =
   -0.0421
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,4)
q =
    0.3310
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,5)
q =
    0.5035
```

所以由高斯-拉道公式可得到 $\int_0^{10} e^{-x}\sin x dx \approx 0.5035$。

此积分的准确值为 0.5000、因此随着 n 的增大，高斯系列公式的精度是越来越高的。

6.6.3 高斯-洛巴托公式

高斯-洛巴托求积公式如下：

$$I(f)=\int_{-1}^{1} f(x)dx \approx \frac{2}{n(n-1)}[f(-1)+f(1)]+\sum_{k=2}^{n-1} A_k f(x_k)$$

高斯-洛巴托公式的前几个节点和系数取值见表 6-3。

表 6-3 高斯-洛巴托公式节点和系数表

n	x_k	A_k
3	0 ±1	1.333333 0.333333
4	±0.447214 ±1	0.833333 0.166666
5	0 ±0.654654 ±1	0.711111 0.544444 0.100000
6	±0.285232 ±0.765055 ±1	0.554858 0.378475 0.066667

在 MATLAB 中编程实现的高斯-洛巴托公式的函数为：IntGaussLobato。

功能：高斯-洛巴托公式求函数的数值积分。

调用格式：*I*=IntGaussLobato(*f*, *a*, *b*, *n*, *AK*, *XK*)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

n 为积分所用项数；

AK 为自定义高斯-洛巴托公式中的系数；

XK 为自定义高斯-洛巴托公式中的节点坐标；

I 为积分值。

高斯-洛巴托公式的 MATLAB 代码如下：

```
function I = IntGaussLobato(f,a,b,n,AK,XK)
%高斯-洛巴托公式求函数 f 在区间[a,b]上的定积分
%函数名：f
%积分下限：a
%积分上限：b
%积分所用项数：n
%自定义高斯—洛巴托公式中的系数：AK
%自定义高斯—洛巴托公式中的节点坐标：XK
%积分值： I

if(n<7 && nargin == 4)
    AK = 0;
    XK = 0;
else                                    %如果 n>6，则节点和系数由调用者给出
    XK1=((b-a)/2)*XK+((a+b)/2);
    I=((b-a)/2)*((2/n/(n-1))*(subs(sym(f),findsym(sym(f)),a)+...
           subs(sym(f),findsym(sym(f)),b))+...
            sum(AK.*subs(sym(f),findsym(sym(f)),XK1)));
end

ta = (b-a)/2;
tb = (a+b)/2;
switch n                                %以下是对不同的 n 采取不同的节点和系数
    case 3,   %n=3 时，采用表 6-3 中对应的系数进行计算
       I=ta*((1/3)*(subs(sym(f),findsym(sym(f)),a)+...
          subs(sym(f),findsym(sym(f)),b))+...
          1.333333*subs(sym(f),findsym(sym(f)),tb));

    case 4,    %n=4 时，采用表 6-3 中对应的系数进行计算
       I=ta*((1/6)*(subs(sym(f),findsym(sym(f)),a)+...
          subs(sym(f),findsym(sym(f)),b))+0.833333*...
          (subs(sym(f),findsym(sym(f)),ta*0.447214+tb)+...
          subs(sym(f),findsym(sym(f)),-ta*0.447214+tb)));

    case 5,    %n=5，采用表 6-3 中对应的系数进行计算
        I=ta*((1/10)*(subs(sym(f),findsym(sym(f)),a)+...
           subs(sym(f),findsym(sym(f)),b))+0.544444*...
          (subs(sym(f),findsym(sym(f)),ta*0.654654+tb)+...
          subs(sym(f),findsym(sym(f)),-ta*0.654654+tb))+...
          0.711111*subs(sym(f),findsym(sym(f)),tb));
```

```
    case 6,    %n=6，采用表 6-3 中对应的系数进行计算
        I=ta*((1/15)*(subs(sym(f),findsym(sym(f)),a)+...
            subs(sym(f),findsym(sym(f)),b))+0.554858*...
           (subs(sym(f),findsym(sym(f)),ta*0.285232+tb)+...
           subs(sym(f),findsym(sym(f)),-ta*0.285232+tb))+...
           0.378475*(subs(sym(f),findsym(sym(f)),ta*0.765055+tb)+...
           subs(sym(f),findsym(sym(f)),-ta*0.765055+tb)));
end
```

【例 6-11】 高斯-洛巴托公式数值积分应用实例。利用高斯—洛巴托公式数值积求解积分 $\int_0^{10} e^{-x}\sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,4)
q =
    0.0993
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,5)
q =
    0.4560
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,6)
q =
    0.5075
```

所以由高斯-洛巴托公式可得到 $\int_0^{10} e^{-x}\sin x\mathrm{d}x \approx 0.5075$。

上面求出的值可以和积分的准确值 0.5000 对照比较。如果积分节点个数为 n 的话，高斯公式的积分精度为 $2n-1$，高斯-拉道公式的积分精度为 $2n-2$，高斯-洛巴托公式的积分精度为 $2n-3$。但是如果知道被积函数在区间的端点上取值为 0 的话，宜采用高斯-拉道公式或高斯-洛巴托公式。

6.7 区间逐次分半法数值积分

区间逐次分半法本质上是一种复合积分法，它通过把积分区间逐次分半，以达到想要的积分精度。下面讲述常见的三种区间逐次分半法积分，分别是区间逐次分半梯形公式数值积分、区间逐次分半辛普森数值积分和区间逐次分半布尔数值积分。

6.7.1 区间逐次分半梯形公式数值积分

区间逐次分半梯形公式为：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}\sum_{k=1}^{2^n}[f(x_{k-1})+f(x_k)]$$

$$f(x_0)=f(a), f(x_{2^n})=f(b)$$

$$h=\frac{b-a}{2^n}$$

在 MATLAB 中编程实现的区间逐次分半梯形公式的函数为：DDTraprl。

功能：区间逐次分半梯形公式求函数的数值积分。

调用格式：[*I*, *step*]=DDTraprl(*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值；

step 为积分划分的区间次数。

区间逐次分半梯形公式的 MATLAB 代码如下：

```
function [I,step] = DDTraprl(f,a,b,eps)
%区间逐次分半梯形公式求函数 f 在区间[a,b]上的定积分
%函数名： f
%积分下限： a
%积分上限： b
%积分精度： eps
%积分值： I
%积分划分的子区间次数： step

if(nargin==3)
    eps=1.0e-4;
end;
n=1;                                %n 为划分子区间的次数，也就是公式中的 n
h=(b-a);
I2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h/2;
Tol=1;
while tol>eps
    n=n+1;
    h=h/2;                          %区间逐次分半
    I1=I2;
    I2=0;
    for i=0:(2^n-1)
        x=a+h*i;
        x1=x+h;
        I2=I2+(h/2)*(subs(sym(f),findsym(sym(f)),x)+...
            subs(sym(f),findsym(sym(f)),x1));        %梯形公式
    end
    tol=abs(I2-I1);
end

I=I2;
step=n;
```

【例 6-12】 区间逐次分半梯形公式数值积分应用实例之一。利用区间逐次分半梯形公式数值积求解积分 $\int_0^{10} e^{-x}\sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDTraprl('exp(-x)*sin(x)',0,10,4)
q =
    0.5000
s =
    10
```

所以由区间逐次分半梯形公式可得到 $\int_0^{10} e^{-x}\sin x\mathrm{d}x \approx 0.5000$。

【例 6-13】 区间逐次分半梯形公式数值积分应用实例之二。利用区间逐次分半梯形公式数值积求解积分 $\int_0^{10}\log(x^2+1)\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDTraprl('log(x^2+1)',0,10)
q =
   29.0935
s =
     9
```

可见，由区间逐次分半梯形公式可得到 $\int_0^{10}\log(x^2+1)\mathrm{d}x \approx 29.0935$。

6.7.2 区间逐次分半辛普森公式数值积分

区间逐次分半辛普森公式是：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{3}\sum_{k=1}^{2^{n-1}}[f(x_{2k-2})+4f(x_{2k-1})+f(x_{2k})]$$

$$f(x_0)=f(a), f(x_{2^n})=f(b)$$

$$h=\frac{b-a}{2^n}$$

在 MATLAB 中编程实现的区间逐次分半辛普森公式的函数为：DDSimpson。

功能：区间逐次分半辛普森公式求函数的数值积分。

调用格式：[*I*, *step*]= DDSimpson(*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值；

step 为积分划分的区间次数。

区间逐次分半辛普森公式的 MATLAB 代码如下：

```
function [I,step] = DDSimpson(f,a,b,eps)
%区间逐次分半辛普森公式求函数 f 在区间[a,b]上的定积分
%函数名: f
%积分下限: a
%积分上限: b
```

```
%积分精度：eps
%积分值： I
%积分划分的子区间次数：step

if(nargin==3)
    eps=1.0e-4;
end;
n=1;                                          %n 为划分子区间的次数，也就是公式中的 n
h=b-a;
I1=0;
I2=(subs(sym(f),findsym(sym(f)),a)+...
    subs(sym(f),findsym(sym(f)),b))/h/2;
tol=1;
while tol>eps
    n=n+1;
    h=h/2;                                    %区间逐次分半
    I1=I2;
    I2=0;
    for i=0:(2^(n-1)-1)
        x=a+h*2*i;
        x1=x+h;
        x2=x1+h;
        I2=I2+(h/3)*(subs(sym(f),findsym(sym(f)),x)+...
            4*subs(sym(f),findsym(sym(f)),x1)+...
            subs(sym(f),findsym(sym(f)),x2));          %辛普森公式
    end
    tol=abs(I2-I1);
end
I=I2;
step=n;
```

【例 6-14】 区间逐次分半辛普森公式数值积分应用实例。利用区间逐次分半辛普森公式数值积求解积分 $\int_0^{10} e^{-x}\sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDSimpson('exp(-x)*sin(x)',0,10)
q =
    0.5000
s =
     8
```

所以由区间逐次分半辛普森公式可得到 $\int_0^{10} e^{-x}\sin x\mathrm{d}x \approx 0.5000$。

对于同样的积分精度，区间逐次分半辛普森公式比区间逐次分半梯形公式划分的子区间次数要少。

6.7.3 区间逐次分半布尔公式数值积分

区间逐次分半布尔公式：

$$\begin{cases} \int_a^b f(x)\mathrm{d}x \approx \dfrac{2h}{45}\sum_{k=1}^{4^{n-1}}[7f(x_{4k-4})+32f(x_{4k-3})+12f(x_{4k-2})+32f(x_{4k-1})+7f(x_{4k})] \\ f(x_0)=f(a), f(x_{4^n})=f(b) \\ h=\dfrac{b-a}{4^n} \end{cases}$$

在 MATLAB 中编程实现的区间逐次分半布尔公式的函数为：DDBuer。

功能：区间逐次分半布尔公式求函数的数值积分。

调用格式：[*I*, *step*]=DDBuer(*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值；

step 为积分划分的区间次数。

区间逐次分半布尔公式的 MATLAB 代码如下：

```
function [I,step] = DDBuer(f,a,b,eps)
%区间逐次分半布尔公式求函数 f 在区间[a,b]上的定积分
%函数名： f
%积分下限： a
%积分上限： b
%积分精度： eps
%积分值： I
%积分划分的子区间次数： step

if(nargin==3)
    eps=1.0e-4;
end;
n=1;                          %n 为划分子区间的次数，也就是公式中的 n
h=b-a;
I1=0;
I2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h/2;
tol=1;
while tol>eps
    n=n+1;
    h=h/4;                    %注意是除以 4!
    I1=I2;
    I2=0;
    for i=0:(4^(n-1)-1)
        x=a+h*4*i;
        x1=x+h;
        x2=x1+h;
        x3=x2+h;
        x4=x3+h;
        I2=I2+(2*h/45)*(7*subs(sym(f),findsym(sym(f)),x)+...
            32*subs(sym(f),findsym(sym(f)),x1)+...
            12*subs(sym(f),findsym(sym(f)),x2)+...
            32*subs(sym(f),findsym(sym(f)),x3)+...
```

```
            7*subs(sym(f),findsym(sym(f)),x4));                    %布尔公式
    end
    tol=abs(I2-I1);
end
I=I2;
step=n;
```

【例 6-15】 区间逐次分半布尔公式数值积分应用实例。利用区间逐次分半布尔公式数值积求解积分 $\int_0^{10} e^{-x}\sin x dx$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDBuer('exp(-x)*sin(x)',0,10)
q =
    0.5000
s =
     5
```

所以由区间逐次分半布尔公式可得到 $\int_0^{10} e^{-x}\sin x dx \approx 0.5000$。

对于同样的积分精度，区间逐次分半布尔公式的速度是最快的。

6.8 龙贝格积分法

龙贝格积分法是用里查森外推算法来加快复合梯形求积公式的收敛速度，它的算法如下，其中 $T_m^{(i)}$ 是一系列逼近原定积分的龙贝格积分值。

（1）计算

$$T_1^{(0)} = \frac{b-a}{2}[f(a)+f(b)]$$

（2）对 $k=1,2,3,\cdots$，计算下列各步：

$$T_1^{(k)} = \frac{1}{2}[T_1^{(k-1)} + \frac{b-a}{2^{k-1}}\sum_{j=1}^{2k-1} f(a+\frac{(2j-1)(b-a)}{2^k})]$$

对 $m=1,2,\cdots,k$ 和 $i=k,k-1,k-2,\cdots$，计算：

$$T_{m+1}^{i-1} = \frac{4^m T_m^i - T_m^{i-1}}{4^m - 1}$$

（3）精度控制。

上面的计算过程如表 6-4 所示。

表 6-4 龙贝格积分计算表格

$T_1^{(0)}$			
$T_1^{(1)}$	$T_2^{(0)}$		
$T_1^{(2)}$	$T_2^{(1)}$	$T_3^{(0)}$	
$T_1^{(3)}$	$T_2^{(2)}$	$T_3^{(1)}$	
$T_1^{(4)}$	$T_2^{(3)}$	$T_3^{(2)}$	
⋮	⋮	⋮	…

随着计算的步骤的增加，$T_m^{(i)}$ 越来越逼近积分 $\int_a^b f(x)\mathrm{d}x$。下面的是用 $T_m^{(m)}$ 来逼近 $\int_a^b f(x)\mathrm{d}x$ 的 MATLAB 代码。

在 MATLAB 中编程实现的龙贝格积分法的函数为：Roberg。

功能：龙贝格积分法求函数的数值积分。

调用格式：[*I*, *step*]= Roberg(*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值；

step 为积分划分的子区间次数。

龙贝格积分的 MATLAB 代码如下：

```
function [I,step]=Roberg(f,a,b,eps)
%龙贝格积分法求函数 f 在区间[a,b]上的定积分
%函数名：f
%积分下限：a
%积分上限：b
%积分精度：eps
%积分值： I
%积分划分的子区间次数：step
    if(nargin==3)
        eps=1.0e-4;
    end;
    M=1;
    tol=10;
    k=0;
    T=zeros(1,1);
    h=b-a;
    T(1,1)=(h/2)*(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f))
,b)); %初始值

    while tol>eps
        k=k+1;
        h=h/2;
        Q=0;
        for i=1:M
            x=a+h*(2*i-1);
            Q=Q+subs(sym(f),findsym(sym(f)),x);
        end
        T(k+1,1)=T(k,1)/2+h*Q;
        M=2*M;
        for j=1:k
            T(k+1,j+1)=T(k+1,j)+(T(k+1,j)-T(k,j))/(4^j-1);
        end
        tol=abs(T(k+1,j+1)-T(k,j));
    end
```

```
I=T(k+1,k+1);
step=k;
```

【例 6-16】 龙贝格积分法数值积分应用实例之一。利用龙贝格积分法求解积分 $\int_{-1}^{1} x^2 \mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=Roberg('x^2',-1,1)
q =
    0.6667
s =
     2
```

所以由龙贝格积分法可得到 $\int_{-1}^{1} x^2 \mathrm{d}x \approx 0.6667$。

【例 6-17】 龙贝格积分法数值积分应用实例之二。利用龙贝格积分法求解积分 $\int_0^1 \frac{\sin x}{\sin x + \cos x} \mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=Roberg('sin(x)/(sin(x)+cos(x))',0,1)
q =
    0.3383
s =
     3
```

所以由龙贝格积分法可得到 $\int_0^1 \frac{\sin x}{\sin x + \cos x} \mathrm{d}x \approx 0.3383$。

6.9 自适应法求积分

自适应积分法是一种比较经济而且快速的求积分的方法。它能自动地在被积函数变化剧烈的地方增多节点，而在被积函数变化平缓的地方减少节点。因此它是一种不均匀区间的积分方法。按照子区间上的积分方式它可以分为自适应辛普森积分法和自适应梯形积分法。通常采用前者作为子区间的积分方式。

自适应积分法的基本步骤如下：

（1）将积分区间 $[a,b]$ 分成两个相等的 1 级子区间 $[a,a+\frac{h}{2}]$ 和 $[a+\frac{h}{2},a+h]$，且 $h=b-a$；

（2）在上述两个 1 级子区间上用辛普森积分得到积分 $I^{(1)}_{a,a+\frac{h}{2}}$ 和 $I^{(1)}_{a+\frac{h}{2},a+h}$；

（3）将子区间 $[a,a+\frac{h}{2}]$ 分成两个相等的 2 级子区间 $[a,a+\frac{h}{2^2}]$ 和 $[a+\frac{h}{2^2},a+\frac{h}{2}]$；

（4）采用辛普森积分计算得到：

$$I^{(2)}_{a,a+\frac{h}{2}} = I^{(1)}_{a,a+\frac{h}{2^2}} + I^{(1)}_{a+\frac{h}{2^2},a+\frac{h}{2}}$$

（5）比较 $I^{(2)}_{a,a+\frac{h}{2}}$ 和 $I^{(1)}_{a,a+\frac{h}{2}}$，如果 $\left|I^{(1)}_{a,a+\frac{h}{2}}-I^{(2)}_{a,a+\frac{h}{2}}\right|<10\times\frac{\varepsilon}{2}$，其中 ε 为整体积分所需精度，则认为子区间 $[a,a+\frac{h}{2}]$ 上的积分 $I^{(1)}_{a,a+\frac{h}{2}}$ 已达到所需精度，不需要再细分；否则就需要再细分，对每个 2 级子区间做同样的判断。

1 级子区间 $[a+\frac{h}{2},a+h]$ 的操作过程完全与上面相同。

在 MATLAB 中编程实现的自适应辛普森积分法的函数为：SmartSimpson。

功能：自适应辛普森积分法求函数的数值积分。

调用格式：*I*=SmartSimpson(*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为积分下限；

b 为积分上限；

eps 为积分精度；

I 为积分值。

自适应辛普森积分的 MATLAB 代码如下：

```
function I=SmartSimpson(f,a,b,eps)
%自适应辛普森积分法求函数 f 在区间[a,b]上的定积分
%函数名:  f
%积分下限: a
%积分上限: b
%积分精度: eps
%积分值:  I

if(nargin==3)
    eps=1.0e-4;
end;
e=5*eps;
I=SubSmartSimpson(f,a,b,e);

function q=SubSmartSimpson(f,a,b,eps)
QA=IntSimpson(f,a,b,1,eps);
QLeft=IntSimpson(f,a,(a+b)/2,1,eps);
QRight=IntSimpson(f,(a+b)/2,b,1,eps);

if(abs(QLeft+QRight-QA)<=eps)
    q=QA;
else
    q=SubSmartSimpson(f,a,(a+b)/2,eps)+SubSmartSimpson(f,(a+b)/2,b,eps);
%递归公式
end
```

【例 6-18】 自适应辛普森积分公式数值积分求解举例。

计算积分 $\int_0^1 x\sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=SmartSimpson('x*sin(x)',0,1)
q =
    0.3011
```

所以由自适应辛普森积分公式可得到 $\int_0^1 x\sin x\mathrm{d}x \approx 0.3011$。

【例 6-19】 自适应辛普森积分公式数值积分应用实例。利用自适应辛普森积分公式数值积分法求解积分 $\int_1^2 \frac{1}{\sin x+\sqrt{x}}\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=SmartSimpson('1/(sqrt(x)+sin(x))',1,2)
q =
    0.4622
```

所以由自适应辛普森积分公式可得到 $\int_1^2 \frac{1}{\sin x+\sqrt{x}}\mathrm{d}x \approx 0.4622$。

自适应辛普森积分应用十分广泛，不管被积函数多复杂，它都能快速地得到高精度的结果。MATLAB 中的积分函数 quad 就是采用的自适应辛普森积分方法。

6.10 样条函数求积分

MATLAB 中的样条工具箱中提供了求样条函数的积分的函数 fnint。函数 fnint 的用法如下：

q=fnint(Y)

它表示求取样条函数 *Y* 的积分。

注意

在用函数 fnint 求积分之前，必须用函数 csape 对被积函数进行样条插值拟合。

【例 6-20】 样条函数求取积分应用实例。利用样条函数求解积分 $\int_0^3 \sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> x=0:0.1:3;
>> y=sin(x);
>> Y=csape(x,y,'second', [0, 0])    %对被积函数进行样条插值拟合
>> q=fnval(fnint(Y),3)
q =
    1.9900
```

所以由样条积分可得到 $\int_0^3 \sin x\mathrm{d}x \approx 1.9900$。

6.11 简单的奇异积分

对于一般的奇异积分，没有通用的数值方法来求出结果，只能针对具体的情况采取不同的方法。对奇异积分的处理方法有：积分变量替换法、奇异性的解析处理、分解法、康托洛维奇法和菲隆法等。下面介绍奇异积分中比较简单的情况的处理，即积分区间为无穷区间，在积分区间上被积函数无奇异点。

6.11.1 高斯-拉盖尔公式

高斯—拉盖尔公式有两种形式：

$$\int_0^{\infty} e^{-x} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k)$$

$$\int_0^{\infty} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k e^{x_k} f(x_k)$$

系数和节点的值如表 6-5 所示。

表 6-5 高斯-拉盖尔公式节点和系数表

n	x_k	A_k
2	-0.585786	0.853553
	3.414214	0.146447
3	0.415575	0.711093
	2.294280	0.278518
	6.289945	0.0103893
4	0.322548	0.603154
	1.745761	0.357419
	4.536620	0.0388879
	9.395071	0.000539295
5	0.263560	0.521756
	1.413403	0.398667
	3.596426	0.0759424
	7.085810	0.00361176
	12.640801	0.0000233700

在 MATLAB 中编程实现的高斯-拉盖尔公式的函数为：IntGaussLager

功能：高斯-拉盖尔公式求函数 0 到正无穷的近似积分。

调用格式：*I* =IntGaussLager(*f*, *n*, *AK*, *XK*)。

其中，*f* 为函数名；

n 为积分所用项数；

AK 为自定义高斯-拉盖尔公式中的系数；

XK 为自定义高斯-拉盖尔公式中的节点坐标；

I 为积分值。

高斯-拉盖尔公式的 MATLAB 代码如下：

```
function I = IntGaussLager(f,n,AK,XK)
%高斯-拉盖尔公式求函数 f 在 0 到正无穷的近似积分
%函数名： f
%积分所用项数：n
%自定义高斯-拉盖尔公式中的系数：AK
%自定义高斯-拉盖尔公式中的节点坐标：XK
%积分值： I

if(n<6 && nargin == 2)
    AK = 0;
    XK = 0;
else
    I=sum(AK.*subs(sym(f),findsym(f),XK));
end

switch n
    case 2,                 %n=2 时，采用表 6-5 中对应的系数进行计算
        I=0.853553*subs(sym(f),findsym(sym(f)),-0.585786)+...
            0.146447*subs(sym(f),findsym(sym(f)),3.414214);

    case 3,                 %n=3 时，采用表 6-5 中对应的系数进行计算
        I=0.711093*subs(sym(f),findsym(sym(f)),0.415575)+...
            0.278518*subs(sym(f),findsym(sym(f)),2.294280)+...
            0.0103893*subs(sym(f),findsym(sym(f)),6.289945);

    case 4,                 %n=4 时，采用表 6-5 中对应的系数进行计算
        I=0.603154*subs(sym(f),findsym(sym(f)),0.322548)+...
            0.357419*subs(sym(f),findsym(sym(f)),1.745761)+...
            0.0388879*subs(sym(f),findsym(sym(f)),4.536620)+...
            0.000539295*subs(sym(f),findsym(sym(f)),9.395071);

    case 5,                 %n=5 时，采用表 6-5 中对应的系数进行计算
        I=0.521756*subs(sym(f),findsym(sym(f)),0.263560)+...
            0.398667*subs(sym(f),findsym(sym(f)),1.413403)+...
            0.0759424*subs(sym(f),findsym(sym(f)),3.596426)+...
            0.00361176*subs(sym(f),findsym(sym(f)),7.085810)+...
            0.0000233700*subs(sym(f),findsym(sym(f)),12.640801);
end
```

【例 6-21】 高斯-拉盖尔公式数值积分应用实例。采用高斯-拉盖尔公式计算积分 $\int_0^{\infty} e^{-x}\sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLager('sin(x)',4)
q =
    0.5049
>> q=IntGaussLager('sin(x)',5)
q =
    0.4989
```

所以由高斯-拉盖尔公式可得出 $\int_0^{\infty} e^{-x}\sin x\mathrm{d}x \approx 0.4989$。

$\int_0^{\infty} e^{-x}\sin x\mathrm{d}x$ 的准确结果为 0.5000,可见当 n 等于 5 时，高斯-拉盖尔公式的精度已经很高了。

6.11.2 高斯-埃尔米特公式

高斯-埃尔米特公式也有两种形式：

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k)$$

$$\int_{-\infty}^{\infty} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k e^{x_k^2} f(x_k)$$

系数和节点的取值如表 6-6 所示。

表 6-6 高斯-埃尔米特公式节点和系数表

n	x_k	A_k
2	±0.707107	0.886227
3	0.000000 ±1.224745	1.181636 0.295409
4	±0.524648 ±1.650680	0.544444 0.100000
5	0.000000 ±0.958572 ±2.020183	0.945309 0.393619 0.199532

在 MATLAB 中编程实现的高斯-埃尔米特公式的函数为：IntGaussHermite。

功能：高斯-埃尔米特公式求函数负无穷到正无穷的近似积分。

调用格式：*I* =IntGaussHermite(*f*, *n*, *AK*, *XK*)。

其中，*f* 为函数名；

n 为积分所用项数；

AK 为自定义高斯-埃尔米特公式中的系数；

XK 为自定义高斯-埃尔米特公式中的节点坐标；

I 为积分值。

高斯-埃尔米特公式的 MATLAB 代码如下：

```
function I = IntGaussHermite(f,n,AK,XK)
%高斯-埃尔米特公式求函数 f 在负无穷到正无穷值间的近似积分
%函数名： f
%积分所用项数：n
%自定义高斯-埃尔米特公式中的系数：AK
%自定义高斯-埃尔米特公式中的节点坐标：XK
%积分值： I
```

```
if(n<6 && nargin == 2)
    AK = 0;
    XK = 0;
else
    I=sum(AK.*subs(sym(f),findsym(sym(f)),XK));
end

switch n
    case 2,          %n=2 时，采用表 6-6 中对应的系数进行计算
        I=0.886227*(subs(sym(f),findsym(sym(f)),-0.707107)+...
            subs(sym(f),findsym(f),0.707107));

    case 3,          %n=3 时，采用表 6-6 中对应的系数进行计算
        I=1.181636*subs(sym(f),findsym(sym(f)),0)+...
            0.295409*(subs(sym(f),findsym(sym(f)),1.224745)+...
            subs(sym(f),findsym(sym(f)),-1.224745));

    case 4,          %n=4 时，采用表 6-6 中对应的系数进行计算
        I=0.544444*(subs(sym(f),findsym(sym(f)),0.524648)+...
            subs(sym(f),findsym(sym(f)),-0.524648))+...
            0.100000*(subs(sym(f),findsym(sym(f)),1.650680)+...
            subs(sym(f),findsym(sym(f)),-1.650680));

    case 5,          %n=5 时，采用表 6-6 中对应的系数进行计算
        I=0.945309*subs(sym(f),findsym(sym(f)),0)+...
            0.393619*(subs(sym(f),findsym(sym(f)),0.958572)+...
            subs(sym(f),findsym(sym(f)),-0.958572))+...
            0.199532*(subs(sym(f),findsym(sym(f)),2.020183)+...
            subs(sym(f),findsym(sym(f)),-2.020183));
    end
```

【例 6-22】 高斯-埃尔米特公式数值积分应用实例。采用高斯-埃尔米特公式计算积分 $\int_{-\infty}^{\infty} e^{-x^2}\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussHermite('1',4)
q =
    1.2889
>> q=IntGaussHermite('1',5)
q =
    2.1316
```

可见，由高斯-埃尔米特公式可得出 $\int_{-\infty}^{\infty} e^{-x^2}\mathrm{d}x \approx 2.1316$。而积分 $\int_{-\infty}^{\infty} e^{-x^2}\mathrm{d}x$ 的准确结果约为 1.7725、可见当 n 等于 5 时，高斯-埃尔米特公式的精度还比较差，要想提高精度，还得取更大的 n。

6.12 重积分的数值计算

对积分区域为矩形的重积分的数值计算与定积分相似，相应的有梯形公式和辛普森公

式等。以下的重积分的数值计算都基于矩形的积分区域$a \leqslant x \leqslant A, b \leqslant y \leqslant B$。

6.12.1 梯形公式

对二重积分$\int_b^B \int_a^A f(x,y)\mathrm{d}x\mathrm{d}y$有如下的梯形公式：

$$\int_b^B \int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{4}[f(a,b)+f(A,b)+f(a,B)+f(A,B)]$$

梯形公式的精度显然不高，要想提高精度，可采用下面的复合梯形公式：

$$\int_b^B \int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{4mn}\sum_{i=0}^{n}\sum_{j=0}^{m}[C_{ij}f(x_i,y_j)]$$

其中$x_i = a + \frac{i(A-a)}{n}, y_j = b + \frac{j(B-b)}{m}$

$$[C] = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 1 & 2 & 2 & \cdots & 2 & 2 & 1 \end{bmatrix}_{(n+1)(m+1)}$$

在 MATLAB 中编程实现的二重积分复合梯形公式的函数为：DblTraprl。

功能：用复合梯形公式求函数的二重数值积分。

调用格式：q = DblTraprl(f, a, A, b, B, m, n)。

其中，f为函数名；

a为x积分下限；

A为x积分上限；

b为y积分下限；

B为y积分上限；

m为y方向上的划分区间数；

n为x方向上的划分区间数；

q为积分值。

复合梯形公式的 MATLAB 代码如下：

```
function q=DblTraprl(f,a,A,b,B,m,n)
%用复合梯形公式求函数的二重数值积分
%函数名： f
%x 积分下限：a
%x 积分上限：A
%y 积分下限：b
%y 积分上限：B
```

```
%y 方向上的划分区间数：m
%x 方向上的划分区间数：n
%积分值： q

if(m==1 && n==1)                %梯形公式
    q=((B-b)*(A-a)/4)*(subs(sym(f),findsym(sym(f)),{a,b})+...
        subs(sym(f),findsym(sym(f)),{a,B})+...
        subs(sym(f),findsym(sym(f)),{A,b})+...
        subs(sym(f),findsym(sym(f)),{A,B}));

else                            %复合梯形公式
    C=4*ones(n+1,m+1);
    C(1,:)=2;
    C(:,1)=2;
    C(n+1,:)=2;
    C(:,m+1)=2;
    C(1,1)=1;
    C(1,m+1)=1;
    C(n+1,1)=1;
    C(n+1,m+1)=1;               %C 矩阵
end

F=zeros(n+1,m+1);
q=0;
for i=0:n
   for j=0:m
      x=a+i*(A-a)/n;
      y=b+j*(B-b)/m;
      F(i+1,j+1)=subs(sym(f),findsym(sym(f)),{x,y});
      q=q+F(i+1,j+1)*C(i+1,j+1);
   end
end

q=((B-b)*(A-a)/4/m/n)*q;
```

【例 6-23】 复合梯形公式计算重积分应用实例。采用复合梯形公式计算重积分：$\int_0^1\int_0^1 \sin(xy)\mathrm{d}x\mathrm{d}y$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=DblTraprl('sin(x*y)',0,1,0,1,15,10)
q =
    0.2397
```

所以 $\int_0^1\int_0^1 \sin(xy)\mathrm{d}x\mathrm{d}y \approx 0.2397$。

6.12.2 辛普森公式

对二重积分 $\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y$ 有如下的辛普森公式：

$$\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{9}[f(a,b)+f(a,B)+f(A,b)$$
$$+f(A,B)+4f(\frac{A-a}{2},b)+4f(a,\frac{B-b}{2})+4f(\frac{A-a}{2},B)$$
$$+4f(A,\frac{B-b}{2})+16f(\frac{A-a}{2},\frac{B-b}{2})]$$

同样有精度更高的复合辛普森公式：

$$\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{36mn}\sum_{i=0}^{2n}\sum_{j=0}^{2m}[C_{ij}f(x_i,y_j)]$$

其中 $x_i=a+\frac{i(A-a)}{2n}$，$y_j=b+\frac{j(B-b)}{2m}$

$$[\boldsymbol{C}]=\begin{bmatrix}1&4&2&4&2&\cdots&4&2&4&1\\4&16&8&16&8&\cdots&16&8&16&4\\2&8&4&8&4&\cdots&8&4&8&2\\\vdots&\vdots&\vdots&\vdots&\vdots&&\vdots&\vdots&\vdots&\vdots\\2&8&4&8&4&\cdots&8&4&8&2\\4&16&8&16&8&\cdots&16&8&16&8\\1&4&2&4&2&\cdots&4&2&4&1\end{bmatrix}_{(2n+1)(2m+1)}$$

在 MATLAB 中编程实现的二重积分复合辛普森公式的函数为：DblSimpson。

功能：用复合辛普森公式求函数的二重数值积分。

调用格式：q= DblSimpson(f, a, A, b, B, m, n)。

其中，f 为函数名；

a 为 x 积分下限；

A 为 x 积分上限；

b 为 y 积分下限；

B 为 y 积分上限；

m 为 y 方向上的划分区间数；

n 为 x 方向上的划分区间数；

q 为积分值。

复合辛普森公式的 MATLAB 代码如下：

```
function q=DblSimpson(f,a,A,b,B,m,n)
%用复合梯形公式求函数的二重数值积分
%函数名： f
%x 积分下限：a
%x 积分上限：A
%y 积分下限：b
%y 积分上限：B
%y 方向上的划分区间数：m
%x 方向上的划分区间数：n
%积分值： q
```

```
if(m==1 && n==1)                    %辛普森公式
    q=((B-b)*(A-a)/9)*(subs(sym(f),findsym(sym(f)),{a,b})+...
        subs(sym(f),findsym(sym(f)),{a,B})+...
        subs(sym(f),findsym(sym(f)),{A,b})+...
        subs(sym(f),findsym(sym(f)),{A,B})+...
        4*subs(sym(f),findsym(sym(f)),{(A-a)/2,b})+...
        4*subs(sym(f),findsym(sym(f)),{(A-a)/2,B})+...
        4*subs(sym(f),findsym(sym(f)),{a,(B-b)/2})+...
        4*subs(sym(f),findsym(sym(f)),{A,(B-b)/2})+...
        16*subs(sym(f),findsym(sym(f)),{(A-a)/2,(B-b)/2}));

else                                %复合辛普森公式
    q=0;
    for i=0:n-1
        for j=0:m-1
            x=a+2*i*(A-a)/2/n;
            y=b+2*j*(B-b)/2/m;
            x1=a+(2*i+1)*(A-a)/2/n;
            y1=b+(2*j+1)*(B-b)/2/m;
            x2=a+2*(i+1)*(A-a)/2/n;
            y2=b+2*(j+1)*(B-b)/2/m;

           q=q+subs(sym(f),findsym(sym(f)),{x,y})+...
               subs(sym(f),findsym(sym(f)),{x,y2})+...
               subs(sym(f),findsym(sym(f)),{x2,y})+...
               subs(sym(f),findsym(sym(f)),{x2,y2})+...
               4*subs(sym(f),findsym(sym(f)),{x,y1})+...
               4*subs(sym(f),findsym(sym(f)),{x2,y1})+...
               4*subs(sym(f),findsym(sym(f)),{x1,y})+...
               4*subs(sym(f),findsym(sym(f)),{x1,y2})+...
               16*subs(sym(f),findsym(sym(f)),{x1,y1});
        end
    end
end

q=((B-b)*(A-a)/36/m/n)*q;
```

【例 6-24】 复合辛普森公式计算重积分应用实例。采用复合辛普森公式计算重积分 $\int_0^1\int_0^1 e^x \sin(xy)\mathrm{d}x\mathrm{d}y$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=DblSimpson('exp(x)*sin(x*y)',0,1,0,1,15,15)
q =
    0.4771
```

所以 $\int_0^1\int_0^1 e^x \sin(xy)\mathrm{d}x\mathrm{d}y \approx 0.4771$。

6.13 小结

数值积分相对数值微分要容易些，因此求数值积分的方法比数值微分要多，而且有很多精度十分高的数值积分方法正不断在发展中。奇异积分是数值积分比较棘手的问题，必须根据具体的问题采用不同的方法。

第 7 章

非线性方程求解

非线性方程是常见的一类方程，非线性方程（组）的理论远不如线性方程（组）成熟和有效，特别是非线性方程组解的存在唯一性还没有完全解决，判断其解的存在性和解的个数几乎没有可行的方法。利用 MATLAB 提供的函数，可以求解一些简单的非线性方程；通过编程，可以解决一些较为复杂的非线性方程。

通过本章，读者能够熟练掌握 MATLAB 中的非线性方程求解相关的函数，而且能通过编程实现多种求解非线性方程的数值算法。

7.1 MATLAB 中的非线性方程求根函数

MATLAB 中求解非线性方程的函数主要是以下两个：fzero 和 fsolve。其中 fzero 可用来求解非线性方程，fsolve 可用来求解非线性方程和非线性方程组。

7.1.1 fzero 函数

fzero 函数用于求解单变量非线性方程。它的基本用法及功能如下：

1．x = fzero (f, $x0$)　求函数 f 在 x0 附近的根；

2．x = fzero(f, $x0$, options)　使用优化选项，options 可选的值很多，具体可参考 MATLAB 帮助；

3．[x,fval] = fzero(f, $x0$, options)　返回值中的 fval= $f(x)$；

4．[x,fval,exitflag] = fzero(f, $x0$, options)　其中 exitflag 标志着求解状态。

函数 fzero 的具体用法见下面的例子。

【例 7-1】 非线性方程求解函数 fzero 的应用实例。采用 fzero 函数求方程 $x^2+x-1=0$ 在 $x=0.5$ 附近的根。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> x=fzero('x^2+x-1',0.5)
```

输出计算结果为：

```
x =
   0.6180
```

从结果可知，方程在 $x=0.5$ 附近的根为 $x=0.6180$。

函数 fzero 还可以求函数在某个区间上的一个根，例如，求取方程 $x^2+x-1=0$ 在[0,2]区间上的根，继续在 MATLAB 命令窗口中输入下列命令：

```
>> x=fzero('sin(x)-0.5',[0 2])
```

输出计算结果为：

```
x =
    0.5236
```

从结果可知，方程在[0,2]区间上的根为 $x=0.5236$。

需要注意的是，当用 fzero 求函数在某个区间上的一个根时，此函数在区间两端点处的值符号要相反才行。

函数 fzero 还有一些特殊的用法，例如，求取函数 $f(x,y)=x^y-3$ 当 $y=2$ 时在 $x=2$ 附近的根，在 MATLAB 命令窗口中输入下列命令：

```
>> u=inline('x^y-3', 'x','y'); %建立非线性方程 f(x,y)=x^y-3
>> c=fzero(u,2,[ ],2)
```

输出计算结果为：

```
c =
    1.7321
```

从结果可知，所求的根为 $x=1.7321$。

优化选项 options 的使用方法如下，使用 optimset 函数，其中参数 TolX 选项用来设置求解精度，在 MATLAB 命令窗口中输入下列命令：

```
>> opt=optimset('TolX',1.0e-8);
>> x=fzero('sin(x)-0.5',0.5,opt)
x =
    0.5236
>> opt=optimset('TolX',1.0e-2);
>> x=fzero('sin(x)-0.5',0.5,opt)
x =
    0.5283
```

从运算结果可以看出，不同精度对根有影响。

7.1.2 fsolve 函数

fsolve 函数用于求解多变量非线性方程组。它的基本用法及功能如下：

1. x = fsolve(fun, $x0$)　用 fun 定义向量函数，其定义方式可以参考例子；
2. x = fsolve(fun, $x0$, options)　options 为优化选项；
3. [x, val] = fsolve(…)　val=F(x)，即函数值向量；
4. [x, val, exitflag] = fsolve(…)　exitflag 标志着求解状态；
5. [x, val, exitflag, output] = fsolve(…)　output 包含着优化后的结果信息；
6. [x, val, exitflag, output, jacobian] = fsolve(…)　jacobian 为解 x 处的 Jacobian 阵。

其余参数与前面参数相似。

另外，符号数学工具箱中也有一个可用于求解方程组的函数 solve，它可求各种类型方程组的解析解。当找不到解析解时，solve 会自动寻求一个近似解，并且精度很高。

需要注意的是，函数 fsolve 不是 MATLAB 符号工具箱中的函数，它位于优化工具箱内，而 solve 是一个符号函数。

【例 7-2】 非线性方程组求解函数 fsolve 应用实例之一。采用 fsolve 函数求方程组 $\begin{cases} x^2 - y - 2 = 0 \\ y^2 - 2x - 4 = 0 \end{cases}$ 在 $(x, y) = (2, 2)$ 附近的根。

解：首先新建一个.m 文件，命名为 myfun.m，键入下列内容：

```
function F=myfun(x)
F=[x(1)^2-x(2)-2;x(2)^2-2*x(1)-4];
```

在 MATLAB 命令窗口中输入下列命令：

```
>> x0=[2 2];
>> opt=optimset('Display','iter');
>> [r,val]=fsolve(@myfun,x0,opt)
```

输出计算结果为：

```
                                    Norm of      First-order  Trust-region
 Iteration  Func-count     f(x)          step      optimality        radius
     0          3              16                                      16             1
     1          6        0.699004             1          4.62             1
     2          9     0.000118156      0.123965        0.0356           2.5
     3         12     4.3305e-011     0.00304521     1.96e-005           2.5
     4         15    6.63221e-024     1.8626e-006    6.78e-012           2.5
Optimization terminated: first-order optimality is less than options.TolFun.

r =
    2.2143    2.9032
val =
  1.0e-011 *
    0.2075
    0.1525
```

由计算结果可知，方程组的一个根为 $(x, y) = (2.2143, 2.9032)$。

【例 7-3】 非线性方程组求解函数 fsolve 应用实例之二。采用 fsolve 函数求矩阵方程 $\boldsymbol{x}^2=\begin{bmatrix}-11 & -8\\ 24 & -3\end{bmatrix}$ 的一个根，初始解向量为 $\boldsymbol{x}0$=[1，1；1，1]。

解：首先新建一个.m 文件，命名为 myfun.m，键入下列内容：

```
function F=myfun(x)
F=[x(1)^2-x(2)-2;x(2)^2-2*x(1)-4];
```

在 MATLAB 命令窗口中输入下列命令：

```
>> x0=[1,1;1,1];
>> r=fsolve(@matrixfun,x0)
```

输出计算结果为：

```
Optimization terminated: first-order optimality is less than options.TolFun.
r =
    1.0000   -2.0000
    6.0000    3.0000
```

可以验证矩阵 $\begin{bmatrix}1 & -2\\ 6 & 3\end{bmatrix}$ 的平方刚好等于 $\begin{bmatrix}-11 & -8\\ 24 & -3\end{bmatrix}$。

7.2 非线性方程求根的其他数值方法

下面讲述的几种数值方法，在不加说明的情况下，都是在某个区间内求得方程的一个根。其理论依据为：如果 $f(a)f(b)<0$，则在区间 $[a,b]$ 上方程 $f(x)=0$ 至少存在一个实根。

大体上可以把求解非线性方程的方法分为夹逼法（二分法和黄金分割法）和迭代法（其余）两类，其中迭代法的内容十分丰富，包括一些加速和优化的技巧。下面进行讲述。

7.2.1 二分法

二分法的具体求解步骤如下。

（1）计算函数 $f(x)$ 在区间 $[a,b]$ 中点的函数值 $f(\frac{a+b}{2})$，并作下面的判断：

如果 $f(a)f(\frac{a+b}{2})<0$，转到（2）；

如果 $f(a)f(\frac{a+b}{2})>0$，令 $a=\frac{a+b}{2}$，转到（1）；

如果 $f(a)f(\frac{a+b}{2})=0$，则 $x=\frac{a+b}{2}$ 为一个根。

（2）如果 $\left|a-\frac{a+b}{2}\right|<\varepsilon$（ε为预先给定的精度），则 $x=\frac{b+3a}{4}$ 为一个根，否则令 $b=\frac{a+b}{2}$，转到（1）。

在 MATLAB 中编程实现的二分法的函数为：HalfInterval。

功能：用二分法求函数在某个区间上的一个零点。

调用格式：root=HalfInterval (*f*,*a*,*b*,eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

二分法的 MATLAB 程序代码如下：

```
function root=HalfInterval(f,a,b,eps)
%二分法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);                    %两端点的函数值
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    root=FindRoots(f,a,b,eps);                        %调用求解子程序
end

function r=FindRoots(f,a,b,eps)
f_1=subs(sym(f),findsym(sym(f)),a);
f_2=subs(sym(f),findsym(sym(f)),b);
mf=subs(sym(f),findsym(sym(f)),(a+b)/2);              %中点函数值
if(f_1*mf>0)
    t=(a+b)/2;
    r=FindRoots(f,t,b,eps);                           %右递归
else
    if(f_1*mf==0)
        r=(a+b)/2;
    else
        if(abs(b-a)<=eps)
            r=(b+3*a)/4;                              %输出根
        else
```

```
            s=(a+b)/2;
            r=FindRoots(f,a,s,eps);                    %左递归
        end
    end
end
```

【例 7-4】 二分法求解非线性方程应用实例。采用二分法求方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=HalfInterval('x^3-3*x+1',0,1)
```

输出计算结果为：

```
r =
    0.3473
```

可见，方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根为 $x=0.3473$。

7.2.2 黄金分割法

二分法是把求解区间的长度逐次减半，而黄金分割法是把求解区间逐次缩短为前次的0.618倍。它的求解步骤如下：

（1）设 $t_1=a+(1-0.618)*(b-a)$，$t_2=a+0.618*(b-a)$，且 $f_1=f(t_1),f_2=f(t_2)$；

（2）如果 $|t_1-t_2|<\varepsilon$ (给定的最小区间长度)，则输出方程的根为 $\frac{t_1+t_2}{2}$；否则转到（3）；

（3）如果 $f_1*f_2<0$，则令 $a=t_1$，$b=t_2$，转（1），否则如果 $f_1*f(a)>0$，令 $a=t_2$，反之令 $b=t_1$，转到（1）。

在 MATLAB 中编程实现的黄金分割法的函数为：hj。

功能：用黄金分割法求函数在某个区间上的一个零点。

调用格式：root= hj (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

黄金分割法的 MATLAB 程序代码如下：

```
function root=hj(f,a,b,eps)
%黄金分割法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root
```

```
if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    t1=a+(b-a)*0.382;
    t2=a+(b-a)*0.618;
    f_1=subs(sym(f),findsym(sym(f)),t1);
    f_2=subs(sym(f),findsym(sym(f)),t2);
    tol=abs(t1-t2);
    while(tol>eps)                              %精度控制
        if(f_1*f_2<0)
            a=t1;                               %区间两端点都调整
            b=t2;
        else
            fa=subs(sym(f),findsym(sym(f)),a);
            if(f_1*fa>0)
                a=t2;                           %同号左端点调整
            else
                b=t1;                           %异号右端点调整
            end
        end
        t1=a+(b-a)*0.382;
        t2=a+(b-a)*0.618;
        f_1=subs(sym(f),findsym(sym(f)),t1);
        f_2=subs(sym(f),findsym(sym(f)),t2);
        tol=abs(t2-t1);
    end
    root=(t1+t2)/2;                             %输出根
end
```

【例 7-5】 黄金分割法求解非线性方程应用实例。采用黄金分割法求方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=hj('x^3-3*x+1',0,1)
```

输出计算结果为：

```
r =
    0.3473
```

可见，方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根为 $x=0.3473$，与二分法的结果相同。

7.2.3 不动点迭代法

求方程 $f(x)=0$ 的根，可以先把方程改写成如下的形式：

$$x = f(x) + x$$

于是得到不动点迭代法的其中一种迭代公式：

$$x_n = f(x_{n-1}) + x_{n-1}$$

此种不动点迭代法很有可能不收敛，因为它的本质是求函数 $y=f(x)+x$ 与直线 $y=x$ 的交点，而它们不一定存在交点。即使收敛，其速度也十分慢，因此有了艾肯特加速迭代与史蒂芬森加速迭代。

在 MATLAB 中编程实现的不动点迭代法的函数为：StablePoint。

功能：用不动点迭代法求函数的一个零点。

调用格式：[root,n] = StablePoint (*f*, *x*0, eps)。

其中，*f* 为函数名；

*x*0 为初始迭代向量；

eps 为根的精度；

root 为求出的函数零点；

n 为迭代步数。

不动点迭代法的 MATLAB 程序代码如下：

```
function [root,n]=StablePoint(f,x0,eps)
%用不动点迭代法求函数的一个零点
%初始迭代向量：x0
%根的精度：eps
%求出的函数零点：root
%迭代步数：n

if(nargin==2)
    eps=1.0e-4;
end

tol=1;
root=x0;
n=0;
while(tol>eps)
    n=n+1;
    r1=root;
    root=subs(sym(f),findsym(sym(f)),r1)+r1;          %迭代的核心公式
    tol=abs(root-r1);
end
```

【例 7-6】 不动点迭代法求解非线性方程应用实例。采用不动点迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> [r,n]=StablePoint('1/sqrt(x)+x-2',0.5)
r =
    0.3820
n =
     4
```

从计算结果可以看出，经过 4 步迭代，得出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根为 $x=0.3820$。

1. 艾肯特加速

艾肯特加速是在计算出 x_n, x_{n+1}, x_{n+2} 后，对 x_{n+1} 作以下修正：

$$\hat{x}_{n+1}=x_n-\frac{(x_{n+1}-x_n)^2}{x_{n+2}-2x_{n+1}+x_n}$$

然后用 $\hat{x}_{n+1}$ 来逼近方程的根。

艾肯特加速是先用不动点迭代法算出一系列 $\{x_n\}$，再对此系列作修正得到系列 $\{\hat{x}_n\}$，用后者来逼近方程的根。

在 MATLAB 中编程实现的艾肯特加速迭代法的函数为：AtkenStablePoint。

功能：用艾肯特加速迭代法求函数的一个零点。

调用格式：[root, *n*] = AtkenStablePoint (*f*, *x*0, eps)。

其中，*f* 为函数名；

*x*0 为初始迭代向量；

eps 为根的精度；

root 为求出的函数零点；

n 为迭代步数。

艾肯特加速的 MATLAB 程序代码如下：

```
function [root,n]=AtkenStablePoint(f,x0,eps)
%用艾肯特加速迭代法求函数 f 的一个零点
%初始迭代向量：x0
%根的精度：eps
%求出的函数零点：root
%迭代步数：n

if(nargin==2)
    eps=1.0e-4;
end

tol=1;
root=x0;
x(1:2)=0;
n=0;
m=0;
a2=x0;
while(tol>eps)
    n=n+1;
    a1=a2;
    r1=root;
    root=subs(sym(f),findsym(sym(f)),r1)+r1;          %求出根
```

```
    x(n)=root;                                          %把所有的根都保存下来
    if(n>2)
        m=m+1;
        a2=x(m)-(x(m+1)-x(m))^2/(x(m+2)-2*x(m+1)+x(m)); %对保存的根进行艾肯特
%修正
        tol=abs(a2-a1);
    end
end
root=a2;
```

【例 7-7】 艾肯特加速迭代法求解非线性方程应用实例。采用艾肯特加速迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> [r,n]= AtkenStablePoint('1/sqrt(x)+x-2',0.5)
r =
    0.3820
n =
     4
```

从计算结果可以看出，经过 4 步迭代，得出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根为 $x=0.3820$。这个方程比较简单，所以通过简单的几步迭代就可得出根。从下面的结果可以看出艾肯特加速迭代法的优点。

```
>> [r,n]=AtkenStablePoint('1/sqrt(x)+x-2',0.999)
r =
    1.0000
n =
     4
>> [r,n]=StablePoint('1/sqrt(x)+x-2',0.999)
r =
    0.3820
n =
    21
```

上面的例子说明采用初始值 0.999 时，经过 4 步迭代艾肯特加速迭代法，可以得到方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的另一个根 $x=1$，而普通的不动点迭代法却得不到。

2. 史蒂芬森加速

史蒂芬森加速与艾肯特加速不同的地方在于前者是在迭代的同时就进行修正，它的迭代公式为：

$$y_n = f(x_n)+x_n$$
$$z_n = f(y_n)+y_n$$
$$x_{n+1} = x_n - \frac{(y_n-x_n)^2}{z_n-2y_n+x_n}$$

在 MATLAB 中编程实现的史蒂芬森加速迭代法的函数为：StevenStablePoint。

功能：用史蒂芬森加速迭代法求函数的一个零点。

调用格式：[root, *n*] = StevenStablePoint (*f*, *x*0, eps)。

其中，*f* 为函数名；

*x*0 为初始迭代向量；

eps 为根的精度；

root 为求出的函数零点；

n 为迭代步数。

史蒂芬森加速的 MATLAB 程序代码如下：

```
function [root,n]=StevenStablePoint(f,x0,eps)
%用史蒂芬森加速迭代法求函数 f 的一个零点
%初始迭代向量：x0
%根的精度：eps
%求出的函数零点：root
%迭代步数：n

if(nargin==2)
    eps=1.0e-4;
end

tol=1;
root=x0;
n=0;
while(tol>eps)
    n=n+1;
    r1=root;
    y=subs(sym(f),findsym(sym(f)),r1)+r1;
    z=subs(sym(f),findsym(sym(f)),y)+y;
    root=r1-(y-r1)^2/(z-2*y+r1);                    %对每次算出的根立即进行修正
    tol=abs(root-r1);
end
```

【例 7-8】 史蒂芬森加速迭代法求解非线性方程应用实例。采用史蒂芬森加速迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> [r,n]=StevenStablePoint('1/sqrt(x)+x-2',0.999)
r =
    1.0000
n =
     2
>> [r,n]=StevenStablePoint('1/sqrt(x)+x-2',0.5)
r =
    0.3820
n =
     4
```

从上例可以看出，史蒂芬森加速迭代法不仅能求出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根

$x=0.3820$，而且它比艾肯特加速迭代法更快地求出另一个根 $x=1$。

7.2.4 弦截法

弦截法的算法过程如下：

（1）过两点 $(a,f(a))$，$(b,f(b))$ 作一直线，它与 x 轴有一个交点，记为 x_1；

（2）如果 $f(a)f(x_1)<0$，过两点 $(a,f(a))$，$(x_1,f(x_1))$ 作一直线，它与 x 轴的交点记为 x_2，否则过两点 $(b,f(b))$，$(x_1,f(x_1))$ 作一直线，它与 x 轴的交点记为 x_2；

（3）如此下去，直到 $|x_n-x_{n-1}|<\varepsilon$，就可以认为 x_n 为 $f(x)=0$ 在区间 $[a,b]$ 上的一个根。

（4）x_k 的递推公式为：

$$\begin{cases} x_k=a-\dfrac{x_{k-1}-a}{f(x_{k-1})-f(a)}f(a) & , \quad f(a)f(x_{k-1})<0 \\ x_k=b-\dfrac{x_{k-1}-b}{f(x_{k-1})-f(b)}f(b) & , \quad f(a)f(x_{k-1})>0 \end{cases}$$

且 $x_1=a-\dfrac{b-a}{f(b)-f(a)}f(a)$。

在 MATLAB 中编程实现的弦截法的函数为：Secant。

功能：用弦截法求函数在某个区间上的一个零点。

调用格式：root= Secant (*f*,*a*,*b*,eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

弦截法的 MATLAB 程序代码如下：

```
function root=Secant(f,a,b,eps)
%弦截法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
```

```
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    root=a-(b-a)*fa/(fb-fa);                          %迭代初始值
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        s=fx*fa;
        if(s==0)
            root=r1;
        else
            if(s>0)
                root=b-(r1-b)*fb/(fx-fb);             %用递推公式 2
            else
                root=a-(r1-a)*fa/(fx-fa);             %用递推公式 1
            end
        end
        tol=abs(root-r1);
    end
end
```

【例 7-9】 弦截法求解非线性方程应用实例。采用弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=Secant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为：

```
r =
    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。

7.2.5 史蒂芬森弦截法

史蒂芬森弦截法是弦截法的一种变形，它的递推公式为：

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f(x_{k-1}+f(x_{k-1}))-f(x_{k-1})} f(x_{k-1})$$

且有

$$x_1 = a - \frac{f(a)}{f(a+f(a))-f(a)} f(a)。$$

史蒂芬森弦截法比弦截法的精度要高，收敛速度要快。

在 MATLAB 中编程实现的史蒂芬森弦截法的函数为：StevenSecant。

功能：用史蒂芬森弦截法求函数在某个区间上的一个零点。

调用格式：root=Secant (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

史蒂芬森弦截法的 MATLAB 程序代码如下：

```
function root=StevenSecant(f,a,b,eps)
%史蒂芬森弦截法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    faa=subs(sym(f),findsym(sym(f)),fa+a);
    root=a-fa*fa/(faa-fa);                          %迭代初始值
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        v=fx+r1;
        fxx=subs(sym(f),findsym(sym(f)),v);
        root=r1-fx*fx/(fxx-fx);                     %递推的核心公式
        tol=abs(root-r1);
    end
end
```

【例 7-10】 史蒂芬森弦截法求解非线性方程应用实例。采用史蒂芬森弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=StevenSecant('sqrt(x)+log(x)-2',1,4)
Warning: Log of zero.
> In sym.subs at 127
  In StevenSecant at 22
Warning: Log of zero.
> In sym.subs at 127
  In StevenSecant at 28
r =
     1
```

结果出现错误，改正方法之一是把求解区间缩短：

```
>> r=StevenSecant('sqrt(x)+log(x)-2',1.5,2)
r =
    1.8773
```

这样就得出了正确的根 1.8773。从上面的例子可以看出求解区间的选择也是很重要的，一般来说，求解区间越短越好。

7.2.6 抛物线法

弦截法其实是用不断缩短的直线来近似函数 $f(x)$ 的，而抛物线法则采用三个点来近似函数 $f(x)$，并且用抛物线与横轴的交点来逼近函数 $f(x)$ 的根。它的算法过程如下：

（1）选定初始值 x_0, x_1, x_2，并计算 $f(x_0), f(x_1), f(x_2)$ 和以下差分：

$$f[x_2, x_1] = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$

一般取 $x_0 = a$，$x_1 = b$，$a < x_2 < b$。注意不要使三点共线。

（2）用牛顿插值法对三点 $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$ 进行插值得到一条抛物线，它有两个根：

$$x_3 = x_2 + \frac{-B \pm \sqrt{B^2 - 4AC}}{2C},$$

其中

$$A = f(x_2),\ C = f[x_2, x_1, x_0],$$

$$B = f[x_2, x_1] + f[x_2, x_1, x_0](x_2 - x_1)。$$

两个根中只取靠近 x_2 的那个根，即 ± 号取与 B 同号，

即

$$x_3 = x_2 - \frac{2A}{B + \mathrm{sgn}(B)\sqrt{B^2 - 4AC}}$$

（3）用 x_1,x_2,x_3 代替 x_0,x_1,x_2，重复以上步骤，并有以下递推公式：

$$x_{n+1}=x_n-\frac{2A_n}{B_n+\mathrm{sgn}(B_n)\sqrt{B_n^2-4A_nC_n}},$$

其中

$$A_n=f(x_n),C_n=f[x_n,x_{n-1},x_{n-2}],$$
$$B_n=f[x_n,x_{n-1}]+f[x_n,x_{n-1},x_{n-2}](x_n-x_{n-1})。$$

（4）进行精度控制。

在 MATLAB 中编程实现的抛物线法的函数为：Parabola。

功能：用抛物线法求函数在某个区间上的一个零点。

调用格式：root = Parabola (*f*, *a*, *b*, *x*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

x 为初始迭代点；

eps 为根的精度；

root 为求出的函数零点。

抛物线法的 MATLAB 程序代码如下：

```
function root=Parabola(f,a,b,x,eps)
%抛物线法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%初始迭代点：x
%根的精度：eps
%求出的函数零点：root

if(nargin==4)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
```

```
    fx=subs(sym(f),findsym(sym(f)),x);
    d1=(fb-fa)/(b-a);
    d2=(fx-fb)/(x-b);
    d3=(d2-d1)/(x-a);
    B=d2+d3*(x-b);
    root=x-2*fx/(B+sign(B)*sqrt(B^2-4*fx*d3));
    t=zeros(3);
    t(1)=a;
    t(2)=b;
    t(3)=x;
    while(tol>eps)
        t(1)=t(2);                                          %保存三个点
        t(2)=t(3);
        t(3)=root;
        f1=subs(sym(f),findsym(sym(f)),t(1));               %计算三点的函数值
        f2=subs(sym(f),findsym(sym(f)),t(2));
        f3=subs(sym(f),findsym(sym(f)),t(3));
        d1=(f2-f1)/(t(2)-t(1));                             %计算三个差分
        d2=(f3-f2)/(t(3)-t(2));
        d3=(d2-d1)/(t(3)-t(1));
        B=d2+d3*(t(3)-t(2));                                %计算算法中的 B
        root=t(3)-2*f3/(B+sign(B)*sqrt(B^2-4*f3*d3));
        tol=abs(root-t(3));
    end
end
```

【例 7-11】 抛物线法求解非线性方程应用实例之一。采用抛物线法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=Parabola('sqrt(x)+log(x)-2',1,4,2)
r =
    1.8773
```

【例 7-12】 抛物线法求解非线性方程应用实例之二。分别从两个区间[0.5,1.5]和[0.1,1]上采用抛物线法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=Parabola('1/sqrt(x)+x-2',0.5,1.5,0.8)
r =
     1
>> r=Parabola('1/sqrt(x)+x-2',0.1,1,0.5)
r =
    0.3820
```

由此可以看出抛物线法也能求出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的两个根。

7.2.7 牛顿法

弦截法本质上是一种割线法，它从两端向中间逐渐逼近方程的根；牛顿法本质上是一

种切线法，它从一端向一个方向逼近方程的根，其递推公式为：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

初始值可以取 $f'(a)$ 和 $f'(b)$ 的较大者，这样可以加快收敛速度。

和牛顿法有关的还有简化牛顿法和牛顿下山法。

在 MATLAB 中编程实现的牛顿法的函数为：NewtonRoot。

功能：用牛顿法求函数在某个区间上的一个零点。

调用格式：root= NewtonRoot (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

牛顿法的 MATLAB 程序代码如下：

```
function root=NewtonRoot(f,a,b,eps)
%牛顿法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root
if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                                    %求导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)                                          %初始值取两端点导数较大者
        root=a-fa/dfa;
```

```
    else
        root=b-fb/dfb;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);      %求该点的导数值
        root=r1-fx/dfx;                               %迭代的核心公式
        tol=abs(root-r1);
    end
end
```

【例 7-13】 牛顿法求解非线性方程应用实例。采用牛顿法求方程 $\sqrt{x}-x^3+2=0$ 的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=NewtonRoot('sqrt(x)-x^3+2',0.5,2)
```

输出计算结果为：

```
r =
    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

需要注意的是，初始值的选择不要使得其导数为 0。

1. 简化牛顿法

将牛顿法的递推公式改为：

$$x_{n+1}=x_n-\lambda f(x_n)$$

为保证收敛，系数 λ 只须满足 $0<\lambda<\dfrac{2}{f'(x_n)}$。如果 λ 取常数 $\dfrac{1}{f'(x_0)}$，则称为简化牛顿法。

在 MATLAB 中编程实现的简化牛顿法的函数为：SimpleNewton。

功能：用简化牛顿法求函数在某个区间上的一个零点。

调用格式：root= SimpleNewton (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点。

简化牛顿法的 MATLAB 代码如下：

```
function root=SimpleNewton(f,a,b,eps)
%简化牛顿法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
```

```
%根的精度：eps
%求出的函数零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                                    %求导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);

    if(dfa>dfb)                                          %初始值取两端点导数较大者
        df0=1/dfa;
        root=a-df0*fa;
    else
        df0=1/dfb;
        root=b-df0*fb;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        root=r1-df0*fx;                                  %迭代的核心公式，其中df0为常数
        tol=abs(root-r1);
    end
end
```

【例7-14】 简化牛顿法求解非线性方程应用实例。采用简化牛顿法求方程 $\sqrt{x}-x^3+2=0$ 的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=SimpleNewtonRoot('sqrt(x)-x^3+2',1.2,2)
```

输出计算结果为：

```
r =
    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

2. 牛顿下山法

牛顿下山法收敛速度很快，很容易就可得出方程的根。牛顿下山法的递推公式为：

$$x_{n+1}=x_n-\alpha\frac{f(x_n)}{f'(x_n)}(0<\alpha\leqslant 1)$$

α 称为下山因子，它的确定方法如下：

（1）先取 $\alpha=1$，求出 x_{n+1}；

（2）判断条件 $|f(x_{n+1})|<|f(x_n)|$ 是否满足；不满足的话令 $\alpha=0.5\alpha$；

（4）不断验证直到 $|f(x_{n+1})|<|f(x_n)|$。

在 MATLAB 中编程实现的牛顿下山法的函数为：NewtonDown。

功能：用牛顿下山法求函数在某个区间上的一个零点。

调用格式：root= NewtonDown (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数零点；

牛顿下山法的 MATLAB 程序代码如下：

```
function root=NewtonDown(f,a,b,eps)
%牛顿下山法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));
```

```
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)                                          %迭代初始值的选取
        root=a;
    else
        root=b;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);
        toldf=1;
        alpha=2;
        while toldf>0                                    %此循环为下山因子的选取过程
            alpha=alpha/2;
            root=r1-alpha*fx/dfx;                        %迭代的核心公式
            fv=subs(sym(f),findsym(sym(f)),root);
            toldf=abs(fv)-abs(fx);
        end
        tol=abs(root-r1);
    end
end
```

【例 7-15】 牛顿法下山求解非线性方程应用实例。采用牛顿下山法求方程 $\sqrt{x}-x^3+2=0$ 的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=NewtonDown('sqrt(x)-x^3+2',1,2)
```

输出计算结果为：

```
r =
    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

7.2.8 两步迭代法

两步迭代法的一般公式为：

$$y_n = g(x_n)$$
$$x_{n+1} = h(y_n)$$

它比较常用的有以下两种形式：

(1) $$\begin{cases} y_n = x_n - \dfrac{f(x_n)}{f'(x_n)} \\ x_{n+1} = y_n - \dfrac{f(y_n)}{f'(x_n)} \end{cases}$$

$$(2)\begin{cases} y_n = x_n - \dfrac{f(x_n)}{f'(x_n)} \\ x_{n+1} = y_n - \dfrac{f(y_n)(y_n - x_n)}{2f(y_n) - f(x_n)} \end{cases}$$

在 MATLAB 中编程实现的两步迭代法的函数为：TwoStep。

功能：用两步迭代法求函数在某个区间上的一个零点。

调用格式：root= TwoStep (*f*, *a*, *b*, type,eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

type 为两步迭代法的形式；

eps 为根的精度；

root 为求出的函数零点。

两步迭代法的 MATLAB 代码如下：

```
function root=TwoStep(f,a,b,type,eps)
%两步迭代法求函数 f 在区间[a,b]上的一个零点
%函数名：f
%区间左端点：a
%区间右端点：b
%两步迭代法的形式：type（可取 1，2）
%根的精度：eps
%求出的函数零点：root

if(nargin==4)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)
        root=a;
```

```
    else
        root=b;
    end
    while(tol>eps)
        if(type==1)                                          %第一种形式的两步迭代法
            r1=root;
            fx1=subs(sym(f),findsym(sym(f)),r1);
            dfx=subs(sym(fun),findsym(sym(fun)),r1);
            r2=r1-fx1/dfx;                                   %第一步迭代
            fx2=subs(sym(f),findsym(sym(f)),r2);
            root=r2-fx2/dfx;                                 %第二步迭代
            tol=abs(root-r1);
        else                                                 %第二种形式的两步迭代法
            r1=root;
            fx1=subs(sym(f),findsym(sym(f)),r1);
            dfx=subs(sym(fun),findsym(sym(fun)),r1);
            r2=r1-fx1/dfx;                                   %第一步迭代
            fx2=subs(sym(f),findsym(sym(f)),r2);
            root=r2-fx2*(r2-r1)/(2*fx2-fx1);                 %第二步迭代
            tol=abs(root-r1);
        end
    end
end
```

【例 7-16】 两步迭代法求解非线性方程应用实例。利用两步迭代法的两种形式求方程 $\lg x-\sin(x)+1=0$ 在区间[0.1，3]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=TwoStep('(log(x)-sin(x)+1)',0.1,3,1)
r =
    0.7013
>> r=TwoStep('(log(x)-sin(x)+1)',0.1,3,2)
r =
    0.7013
```

从上例可以看出，两种形式的两步迭代法得出的结果是一样的。

7.2.9 重根的迭代方法

如果方程存在重根，那么前面的方法都可能不收敛或者收敛速度很慢，因此针对二重根的情况有以下的迭代公式：

$$x_{n+1}=x_n-\frac{f'(x_n)f(x_n)}{[f'(x_n)]^2-f(x_n)f''(x_n)}$$

但它要求是二阶导数，如果方程比较复杂，计算量则比较大。

在 MATLAB 中编程实现的重根的迭代法的函数为：MultiRoot。

功能：用迭代法求函数在某个区间上的一个多重零点。

调用格式：root = MultiRoot (*f*, *a*, *b*, eps)。

其中，*f* 为函数名；

a 为区间左端点；

b 为区间右端点；

eps 为根的精度；

root 为求出的函数多重零点。

重根的迭代法的 MATLAB 代码如下：

```
function root=MultiRoot(f,a,b,eps)
%求函数 f 在区间[a,b]上的一个多重零点
%函数名：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求出的函数多重零点：root

if(nargin==3)
    eps=1.0e-4;
end

f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end

if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                                    %一阶导数
    ddf=diff(sym(fun));                                  %二阶导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)
        root=a;
    else
        root=b;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);         %一阶导数值
        ddfx=subs(sym(ddf),findsym(sym(ddf)),r1);        %二阶导数值
        root=r1-fx*dfx/(dfx*dfx-fx*ddfx);                %迭代公式
        tol=abs(root-r1);
    end
end
```

【例 7-17】 求解非线性方程综合应用实例。用重根迭代法、两步迭代法和弦截法求方程 $x^2(\sin x - x + 2) = 0$ 在区间[-2，3]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=MultiRoot('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r =
 0
>> r=TwoStep('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r =
 -1.0147e-005
>> r=Secant('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r =
    2.5542
```

方程 $x^2(\sin x - x + 2) = 0$ 有一个二重根 0，用其他方法得出的精度都不好，而用特殊的重根迭代法可以得到精确的结果。

7.3 非线性方程组的数值解法

非线性方程组的数值解法大多来源于非线性方程的解法，只不过数的计算换成了矩阵的计算。

7.3.1 不动点迭代法

非线性方程组的不动点迭代法与非线性方程的不动点迭代法基本一样。

在 MATLAB 中编程实现的非线性方程组的不动点迭代法的函数为：mulStablePoint。

功能：用不动点迭代法求非线性方程组的一个解。

调用格式：[*r*, *n*] = mulStablePoint (*x*0, eps)。

其中，*x*0 为初始迭代向量；

eps 为迭代精度；

r 为求出的解向量；

n 为迭代步数。

不动点迭代法的 MATLAB 代码如下：

```
function [r,n]=mulStablePoint(x0,eps)
%不动点迭代法求非线性方程组的一个解
%初始迭代向量：x0
%迭代精度：  eps
%解向量：    r
%迭代步数：  n

if nargin==1
    eps=1.0e-4;
end

r=myf(x0);
n=1;
tol=1;
```

```
while tol>eps
    x0=r;
    r=myf(x0);                          %迭代公式
    tol=norm(r-x0);                     %注意矩阵的误差求法，norm 为矩阵的欧几里德范数
    n=n+1;
    if(n>100000)                        %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

【例 7-18】 不动点迭代法求解非线性方程组应用实例。采用不动点迭代法求下面方程组的一个根。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0 \end{cases}$$

初始迭代值取（0，0）。

解：首先建立 myf.m 函数文件，输入以下内容：

```
function f=myf(x)
f(1)=0.5*sin(x(1))+0.1*cos(x(2)*x(1))-x(1);
f(2)=0.5*cos(x(1))-0.1*sin(x(2))-x(2);
```

在 MATLAB 命令窗口中输入：

```
>> [r,n]=mulStablePoint([0,0])
r =
    0.1979    0.4470
n =
    11
```

由计算结果可知，初始迭代值取(0,0)时，用 11 步迭代，得到原方程组的一组解(0.1979, 0.4470)。

7.3.2 牛顿法

设非线性方程组为 $\boldsymbol{F}(x_1, x_2, \cdots x_n) = 0$，其中 $\boldsymbol{x} = [x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n]^T$，

$$\boldsymbol{F} = \begin{bmatrix} f_1(x_1, x_2, \cdots x_n) \\ f_2(x_1, x_2, \cdots x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots x_n) \end{bmatrix}$$

牛顿法的迭代公式为：

$$\boldsymbol{x}^{n+1} = \boldsymbol{x}^n - \left(\boldsymbol{F}'(\boldsymbol{x}^n)\right)^{-1} \boldsymbol{F}(\boldsymbol{x}^n)$$

求解步骤为：

（1）给出初始值 $\boldsymbol{x}^0$；

（2）对 $n = 1, 2, 3\cdots$ 计算 $\boldsymbol{F}(\boldsymbol{x}^n)$ 和 $\boldsymbol{F}'(\boldsymbol{x}^n)$；

（3）求出 $\boldsymbol{x}^{n+1}$，并进行精度控制。

更一般的牛顿法迭代公式为：

$$\boldsymbol{x}^{n+1}=\boldsymbol{x}^{n}-\left(\boldsymbol{A}(\boldsymbol{x}^{n})\right)^{-1}\boldsymbol{F}(\boldsymbol{x}^{n})$$

当 $\boldsymbol{A}(\boldsymbol{x}^{n})=\boldsymbol{F}'(\boldsymbol{x}^{0})$ 时，就得到简化牛顿法。

在 MATLAB 中编程实现的非线性方程组的牛顿迭代法的函数为：mulNewton。

功能：用牛顿迭代法求非线性方程组的一个解。

调用格式：[*r*, *n*] = mulNewton (*x*0, eps)。

其中，*x*0 为初始迭代向量；

eps 为迭代精度；

r 为求出的解向量；

n 为迭代步数。

牛顿法的 MATLAB 代码如下：

```
function [r,n]=mulNewton(x0,eps)
%牛顿迭代法求非线性方程组的一个解
%初始迭代向量：x0
%迭代精度：  eps
%解向量：    r
%迭代步数：  n

if nargin==1
   eps=1.0e-4;
end

r=x0-myf(x0)/dmyf(x0);
n=1;
tol=1;
while tol>eps
    x0=r;
    r=x0-myf(x0)/dmyf(x0);                          %核心迭代公式
    tol=norm(r-x0);
    n=n+1;
    if(n>100000)                                    %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

在 MATLAB 中编程实现的非线性方程组的简化牛顿迭代法的函数为：mulSimNewton。

功能：用简化牛顿迭代法求非线性方程组的一个解。

调用格式：[*r*, *n*] = mulSimNewton (*x*0, eps)。

其中，*x*0 为初始迭代向量；

eps 为迭代精度；

r 为求出的解向量；

n 为迭代步数。

简化牛顿法的 MATLAB 代码如下：

```
function [r,n]= mulSimNewton (x0,eps)
%简化牛顿迭代法求非线性方程组的一个解
%初始迭代向量：x0
%迭代精度： eps
%解向量：   r
%迭代步数： n

if nargin==1
    eps=1.0e-4;
end

r=x0-myf(x0)/dmyf(x0);                                    %核心迭代公式
c= dmyf(x0);
n=1;
tol=1;
while tol>eps
    x0=r;
    r=x0-myf(x0)/c;
    tol=norm(r-x0);
    n=n+1;
    if(n>100000)                                          %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

【例 7-19】 牛顿法求解非线性方程组应用实例。采用牛顿法和简化牛顿法求下面方程组的一个根。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0 \end{cases}$$

初始迭代值取（0，0）。

解：首先建立 myf.m 函数文件，输入以下内容：

```
function f=myf(x)
f(1)=0.5*sin(x(1))+0.1*cos(x(2)*x(1))-x(1);
f(2)=0.5*cos(x(1))-0.1*sin(x(2))-x(2);
f=[f(1) f(2)];
```

再建立 dmyf.m 导数的雅可比矩阵，输入以下内容：

```
function df=dmyf(x)
df=[0.5*cos(x(1))-0.1*x(2)*sin(x(2)*x(1))-1-0.1*x(1)*sin(x(2)*x(1));
    -0.5*sin(x(1))-0.1*cos(x(2))-1];
```

然后在 MATLAB 命令窗口中输入：

```
>> [r,n]=mulNewton([0,0])
r =
    0.1979    0.4470
n =
     5
>> [r,n]=mulSimNewton([0,0])
```

```
r =
    0.1979    0.4470
n =
     4
```

由计算结果可知，初始迭代值取（0, 0）时，用 6 步迭代，牛顿法和简化牛顿法就得到方程组的一组解（0.1979,0.4470）。

从上面的例子可以看出，牛顿法的收敛速度很快，收敛步数和初始值的选取有很大的关系，初始值选得好，用很少的步骤就可以得到根，选得不好可能会不收敛。

7.3.3 牛顿下山法

牛顿下山法的迭代公式为：

$$\boldsymbol{x}^{n+1}=\boldsymbol{x}^{n}-\omega\left(\boldsymbol{F}'(\boldsymbol{x}^{n})\right)^{-1}\boldsymbol{F}(\boldsymbol{x}^{n})$$

ω 的取值范围为 $0<\omega\leqslant 1$，为了保证收敛，还要求 ω 的取值使得：

$$\left\|\boldsymbol{F}(\boldsymbol{x}^{n+1})\right\|<\left\|\boldsymbol{F}(\boldsymbol{x}^{n})\right\|$$

可以用逐次减半法来确定 ω。为了减少计算量，还可以用差商来代替偏导数。

在 MATLAB 中编程实现的非线性方程组的牛顿下山法的函数为：mulDNewton。

功能：用牛顿下山法求非线性方程组的一个解。

调用格式：[*r*, *n*]= mulDNewton (*x*0,eps)。

其中，*x*0 为初始迭代向量；

eps 为迭代精度；

r 为求出的解向量；

n 为迭代步数。

牛顿下山法的 MATLAB 代码如下：

```
function [r,n]=mulDNewton(x0,eps)
%牛顿下山法求非线性方程组的一个解
%初始迭代向量：x0
%迭代精度： eps
%解向量：   r
%迭代步数： n

if nargin==1
    eps=1.0e-4;
end

r=x0-myf(x0)/dmyf(x0);
n=1;
tol=1;
while tol>eps
    x0=r;
    ttol=1;
    w=1;
    F1=norm(myf(x0));
```

```
    while ttol>=0                                    %下面的循环是选取下山因子 w 的过程
        r=x0-w*myf(x0)/dmyf(x0);                     %核心的迭代公式
        ttol=norm(myf(r))-F1;
        w=w/2;
    end
    tol=norm(r-x0);
    n=n+1;
    if(n>100000)                                     %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

【例 7-20】 牛顿下山法求解非线性方程组应用实例。采用牛顿下山法求下面方程组的一个根。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0 \end{cases}$$

其初始迭代值取（0,0）。

解：首先建立 myf.m 函数文件，输入以下内容：

```
function f=myf(x)
f(1)=0.5*sin(x(1))+0.1*cos(x(2)*x(1))-x(1);
f(2)=0.5*cos(x(1))-0.1*sin(x(2))-x(2);
f=[f(1) f(2)];
```

再建立 dmyf.m 导数的雅可比矩阵，输入以下内容：

```
function df=dmyf(x)
df=[0.5*cos(x(1))-0.1*x(2)*sin(x(2)*x(1))-1  -0.1*x(1)*sin(x(2)*x(1));
    -0.5*sin(x(1))   -0.1*cos(x(2))-1];
```

然后，在 MATLAB 命令窗口中输入：

```
>> [r,n]=mulDNewton([0,0])
```

输出计算结果为：

```
r =
    0.1979    0.4470
n =
     5
```

由计算结果可知，初始迭代值取（0, 0）时，用 5 步迭代得到了方程组的一组解（0.1979,0.4470），牛顿下山法也是一种快速且有效的求解非线性方程组的方法。

7.3.4 拟牛顿法

拟牛顿法是为了减少计算导数而带来的计算量的一种迭代法，它的迭代过程如下：

$$\boldsymbol{x}^{n+1} = \boldsymbol{x}^n - \boldsymbol{A}_n^{-1}\boldsymbol{F}(\boldsymbol{x}^n);$$

$$\boldsymbol{y}^n = \boldsymbol{x}^{n+1} - \boldsymbol{x}^n;$$

$$z^n = \boldsymbol{F}(\boldsymbol{x}^{n+1}) - \boldsymbol{F}(\boldsymbol{x}^n);$$

$$\boldsymbol{A}_{n+1} = \boldsymbol{A}_n + \frac{(z^n - \boldsymbol{A}_n \boldsymbol{y}^n)(\boldsymbol{y}^n)^T}{\|\boldsymbol{y}^n\|}$$

拟牛顿法的思想是用比较简单的矩阵 $\boldsymbol{A}_n$ 来近似 $\boldsymbol{F}'(\boldsymbol{x}^n)$， $\boldsymbol{A}_0$ 可选为单位阵。

在 MATLAB 中编程实现的非线性方程组的拟牛顿法的函数为：mulVNewton。

功能：用拟牛顿法求非线性方程组的一个解。

调用格式：[*r*, *n*] = mulVNewton (x0, *A*, eps)。

其中，*x*0 为初始迭代向量；

eps 为迭代精度；

r 为求出的解向量；

n 为迭代步数。

拟牛顿法的 MATLAB 代码如下：

```
function [r,n]=mulVNewton(x0,eps)
%拟牛顿法求非线性方程组的一个解
%初始迭代向量：x0
%迭代精度：  eps
%解向量：    r
%迭代步数：  n

if nargin==1
   A=eye(length(x0));                          %A 取为单位阵
   else
     if nargin==2
          eps=1.0e-4;
      end
   end

   r=x0-myf(x0)/A;
   n=1;
   tol=1;
   while tol>eps
      x0=r;
      r=x0-myf(x0)/A;
      y=r-x0;
      z=myf(r)-myf(x0);
      A1=A+(z-y*A)'*y/norm(y);                 %调整 A
      A=A1;
      n=n+1;
      if(n>100000)                             %迭代步数控制
         disp('迭代步数太多，可能不收敛！');
         return;
      end
      tol=norm(r-x0);
   end
```

【例 7-21】 拟牛顿法求解非线性方程组应用实例。采用拟牛顿法求下面方程组的一个根。

$$\begin{cases}0.5\sin x_1+0.1\cos(x_1x_2)-x_1=0\\0.5\cos x_1-0.1\sin x_2-x_2=0\end{cases}$$

其初始迭代值取（0.5,0.5）。

解：和上面的例子一样，先建立 myf.m 函数文件和 dmyf.m 导数的雅克比矩阵，在 MATLAB 命令窗口中输入：

```
>> [r,n]=mulVNewton([0.5,0.5])
```

输出计算结果为：

```
r =
    0.1980    0.4470
n =
    27
```

由计算结果可知，初始迭代值取（0.5，0.5）时，用 27 步迭代得到了方程组的一组解（0.1980，0.4470）。

拟牛顿法中重要的是要选好 $\boldsymbol{A}_0$，否则，迭代的效果会很差。

7.4 小结

本章讲述了利用MATLAB求非线性方程在某个区间上的根的方法，包括采用MATLAB中提供的函数进行求解，以及编程实现的一些经典的数值方法。非线性方程形式多样，前面讲述的不可能囊括所有的解法，求解时关键是要认真分析方程的特征，采取行之有效的算法。

第 8 章

常微分方程求解

常微分方程作为微分方程的基本类型之一，在自然界与工程界有很广泛的应用。很多问题的数学表述都可以归结为常微分方程的定解问题。很多偏微分方程问题，也可以化为常微分方程问题来近似求解。MATLAB 中提供了求解函数和求解器，可以实现常微分方程的解析求解和数值求解，而且还可以通过编程实现一些常见的数值解法。

通过本章，读者能够熟练使用 MATLAB 的求解函数和求解器，以及通过编程进行常微分方程的求解。

8.1 MATLAB 中的常微分方程求解函数

一般微分方程式描述系统内部变量的变化率如何受系统内部变量和外部激励，如输入的影响。当常微分方程式能够解析求解时，可用 MATLAB 的符号工具箱中的功能找到精确解；在常微分方程难以获得解析解的情况下，使用 MATLAB 的常微分方程求解器 solver，可以方便地在数值上求解。

8.1.1 常微分方程符号解函数 dsolve

一阶常微分方程式(first-order ordinary differential equation, ODE)可写为

$$y' = g(x, y)$$

其中 x 为独立变量，而 y 是 x 的函数。它的解是

$$y = f(x, y)$$

该解可以满足 $y'=f'=g(x,y)$，在初始条件 $y(x_0) = y_0$ 下，可以得到唯一解。

MATLAB 常微分方程符号解的语法是：

```
dsolve('equation','condition')
```

其中，equation 代表常微分方程式即 $y' = g(x,y)$，且须以 Dy 代表一阶微分项 y'，D2y 代表二阶微分项 y''，condition 则为初始条件。

函数 dsolve 用来解符号常微分方程、方程组，如果没有初始条件，则求出通解，如果有初始条件，则求出特解。

dsolve 的调用格式如下：

（1）dsolve('equation')

给出微分方程的解析解，表示为 t 的函数；

（2）dsolve('equation', 'condition')

给出微分方程初值问题的解，表示为 t 的函数；

（3）dsolve('equation', 'v')

给出微分方程的解析解，表示为 v 的函数；

（4）dsolve('equation', 'condition', 'v')

给出微分方程初值问题的解，表示为 v 的函数。

【例 8-1】 常微分方程符号解求解实例之一。计算微分方程 $\frac{dy}{dx}+3xy=xe^{-x^2}$ 的通解。

```
>>  dsolve('Dy+3*x*y=x*exp(-x^2)')
```

输出为

```
ans =
(1/3*exp(-x*(x-3*t))+C1)*exp(-3*x*t)
```

由于系统默认的自变量是 t，显然系统把 x 当作常数，把 y 当作 t 的函数求解. 输入命令：

```
>>  dsolve('Dy+3*x*y=x*exp(-x^2)','x')
```

输出为

```
ans =
exp(-x^2)+exp(-3/2*x^2)*C1
```

可知通解为 $y=e^{-x^2}+e^{-\frac{3}{2}x^2}*C_1$，其中 C_1 为常数。

【例 8-2】 常微分方程符号解求解实例之二。计算微分方程 $xy'+2y-e^x=0$ 在初始条件 $y|_{x=1}=2e$ 下的特解。

```
>>   dsolve('x*Dy+2*y-exp(x)=0','y(1)=2*exp(1)','x')
```

输出为

```
ans =
 (exp(x)*x-exp(x)+2*exp(1))/x^2
```

可知特解为 $y=\frac{xe^x-e^x+2e}{x^2}$。

【例 8-3】 常微分方程符号解求解实例之三。求 $y''+2y'+e^x=0$ 的通解。

```
>>  dsolve('D2y+2*Dy+exp(x)=0','x')
```

输出为

```
ans =
 -1/3*exp(x)-1/2*exp(-2*x)*C1+C2
```

可知通解为 $y=-\frac{1}{3}e^x-\frac{1}{2}e^{-2x}*C_1+C_2$，其中 C_1，C_2 为常数。

8.1.2 求解器 solver

在求常微分方程数值解方面，MATLAB 具有丰富的函数，将其统称为 solver，其一般格式为：

```
[T,Y]=solver(odefun,tspan,y0)
```

该函数表示在区间 tspan=[t_0, t_f]上，用初始条件 y0 求解显式常微分方程 $y'=f(t,y)$。

odefun 为显式常微分方程 $y'=f(t,y)$ 中的 $f(t,y)$，tspan 为求解区间，要获得问题在其他指定点 $t_0,t_1,t_2\cdots$ 上的解，则令 $tspan=t_0,t_1,t_2,\cdots t_f$（要求 t_i 单调），y0 为初始条件。

solver 为命令 ode45、ode23、ode113、ode15s、ode23s、ode23t、ode23tb 之一，其中 ode45、ode23、ode113 属于非刚性 ODE 类型，这些命令的特点如表 8-1 所示；ode15s、ode23s、ode23t、ode23tb 属于刚性 ODE 类型，这些命令的特点如表 8-2 所示。

表 8-1 非刚性 ODE 求解命令

求解器 solver	功　能	说　明
ode45	一步算法：4，5 阶 Runge-Kutta 方程；累计截断误差达 $(\Delta x)^3$	大部分场合的首选算法
ode23	一步算法：2，3 阶 Runge-Kutta 方程；累计截断误差达 $(\Delta x)^3$	使用于精度较低的情形
ode113	多步法：Adams 算法；高低精度均可到 $10^{-3}\sim10^{-6}$	计算时间比 ode45 短

表 8-2 刚性 ODE 求解命令

求解器 solver	功　能	说　明
ode23t	采用梯形算法	适度刚性情形
ode15s	多步法：Gear's 反向数值微分；精度中等	若 ode45 失效时，可尝试使用
ode23s	一步法：2 阶 Rosebrock 算法；低精度	当精度较低时，计算时间比 ode15s 短
ode23tb	梯形算法；低精度	当精度较低时，计算时间比 ode15s 短

函数 ode45 与 ode23 是常使用的求解方法，函数 ode45 的使用与 ode23 完全一样。两个函数的差别在于必须与所用的内部算法相关。两个函数都运用了基本的龙格-库塔(Runge-Kutta)数值积分法的变形。

ode23 运用一个组合的 2/3 阶龙格-库塔-芬尔格(Runge-Kutta-Fehlerg)算法，而 ode45 运用组合的 4/5 阶龙格-库塔-芬尔格算法。

一般地，ode45 可取较多的时间步。所以，要保持与 ode23 相同误差时，在 t_0 和 t_f 之间

可取较少的时间步。然而，在同一时间内，ode23 每时间步至少调用 3 次，而 ode45 每时间步至少调用 6 次。

正如使用高阶多项式内插常常得不到最好的结果一样，ode45 也不总是比 ode23 好。如果 ode45 产生的结果，对作图间隔太大，则必须在更细的时间区间内，对数据进行内插，比如用函数 interp1。这个附加时间点会使 ode23 更有效。作为一条普遍规则，在所计算的导数中，如有重复的不连续点，为保持精度致使高阶算法减少时间步长，这时低阶算法更有效。

采用求解器 solver 求解 ODE 的基本过程如下：

（1）根据问题所属学科中的规律、定律、公式，用微分方程与初始条件进行描述。

$$F(y, y', \ldots y^{(n-1)}, t) = 0$$

$$y(0) = y_0, y'(0) = y_1, \ldots y^{(n-1)}(0) = y_{n-1}$$

写为向量的形式为 $\boldsymbol{y} = [y; y(1); y(2), \ldots, y(m-1)]$，$n$ 与 m 可以不等。

（2）运用数学中的变量替换：y_n=y(n-1)，y_{n-1}=y(n-2)，…，y_2=y_1=y，把高阶（大于 2 阶）的方程（组）写成一阶微分方程组：

$$\boldsymbol{y}' = \begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} f_1(t, y) \\ f_2(t, y) \\ \vdots \\ f_n(t, y) \end{bmatrix}$$

相应的初始条件为：

$$\boldsymbol{y}_0 = \begin{bmatrix} y_1(0) \\ y_2(0) \\ \vdots \\ y_n(0) \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

（3）根据（1）与（2）的结果，编写能计算导数的 M-函数文件 odefile。

（4）将文件 odefile 与初始条件传递给求解器 solver 中的一个，运行后就可得到 ODE 的、在指定时间区间上的解列向量 $\boldsymbol{y}$（其中包含 y 及不同阶的导数）。

【例 8-4】 求解器 solver 应用实例。采用 ode45 求解描述某非刚性体的运动方程的微分方程：$\begin{cases} y_1' = y_2 y_3 \\ y_2' = -y_1 y_3 \\ y_3' = 0.51 y_1 y_2 \end{cases}$，其初始条件为 $\begin{cases} y_1(0) = 0 \\ y_2(0) = 1 \\ y_3(0) = 1 \end{cases}$，并画出解的图形。

解：编写函数文件 rigid.m：

```
function dy = rigid(t,y)
dy = zeros(3,1);    % 行向量
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

再在命令窗口中执行：

```
options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5]);
[T,Y] = ode45(@rigid,[0 12],[0 1 1],options);
plot(T,Y(:,1),'-',T,Y(:,2),'-.',T,Y(:,3),'.')
```

输出的图形结果为图 8-1 所示。

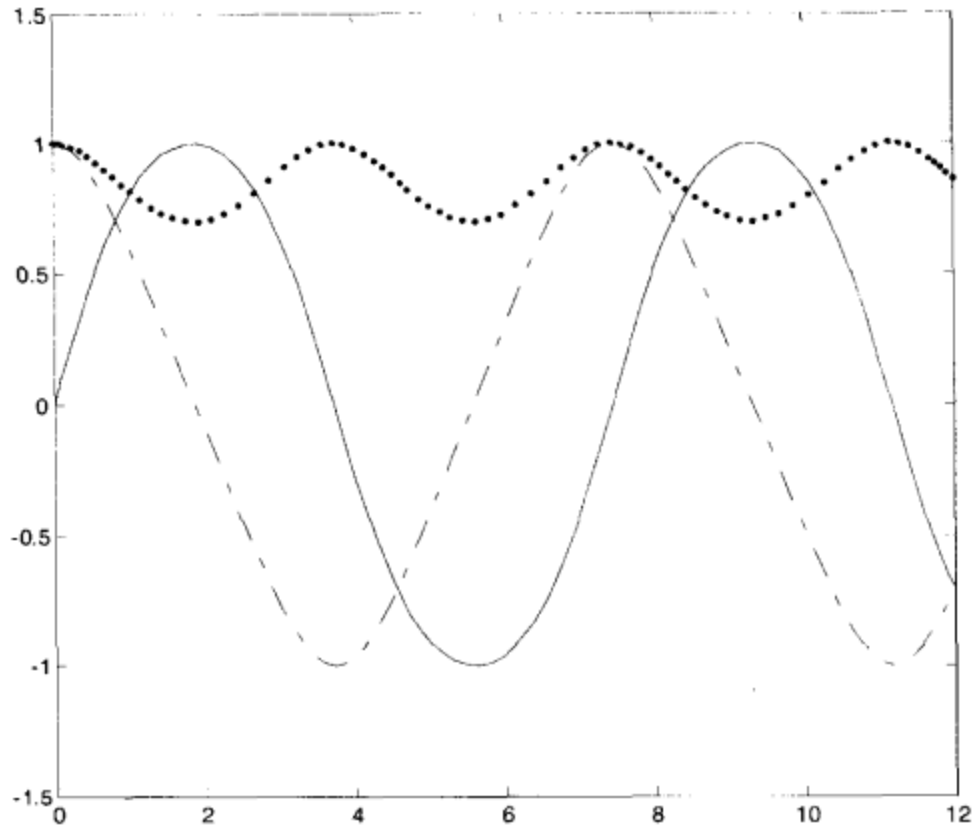

图 8-1 非刚性体运动的微分方程图

8.2 欧拉法（Euler）

8.2.1 简单欧拉法

欧拉法是简单有效的常微分方程数值解法，欧拉法有多种形式的算法，其中简单欧拉法是一种单步递推算法。

1. 基本原理

对于一个简单的一阶微分方程：

$$y' = f(x, y)$$

其中初始值为：

$$y(x_0) = y_0$$

欧拉方法等同于将函数微分转换为数值差分，如下式，

$$y'(x) = \frac{y(x+h) - y(x)}{h}$$

故原方程变形为：

$$y_{n+1} = y_n + hf(x_n, y_n)$$

2. 算法的程序实现

在 MATLAB 中编程实现的欧拉法函数为：MyEuler。

功能：以欧拉法求解常微分方程。

调用格式：[outx,outy] = MyEuler(fun, *x*0, *xt*, *y*0, PointNum)。

其中，fun 为函数名；

*x*0 为自变量区间初值；

xt 为自变量区间终值；

*y*0 为函数在 *x*0 处的值；

PointNum 为自变量在[*x*0, *xt*]上所取的点数；

outx 为输出自变量区间上所取点的 *x* 值；

outy 为对应点上的函数值。

欧拉法的 MATLAB 程序代码如下：

```
function [outx,outy]=MyEuler(fun,x0,xt,y0,PointNum)
%用前向差分的欧拉方法解微分方程 y'=fun
%函数 f（x，y）：fun
%自变量的初值和终值：x0,xt
%y0 表示函数在 x0 处的值，输入初值为列向量形式
%自变量在[x0,xt]上取的点数：PointNum
%outx:所取的点的 x 值
%outy:对应点上的函数值
if nargin<5 | PointNum<=0          %如果函数仅输入 4 个参数值，则 PointNum 默认值为 100
    PointNum=100;
end
if nargin<4                        %y0 默认值为 0
    y0=0;
end
h=(xt-x0)/PointNum;                %计算步长 h
x=x0+[0:PointNum]'*h;              %自变量数组
y(1，：) = y0（：）';               %输入存为列向量
for k = 1:PointNum
    f=feval(fun,x(k),y(k,:));      %计算 f(x,y)在每个迭代点的值
    f=f(:)';
    y(k + 1, :) =y(k, :) +h*f;     %对于所取的点 x 迭代计算 y 值
end
outy=y;
outx=x;
plot(x,y)                          %画出方程解的函数图
```

3. 实例分析

【例 8-5】 欧拉法求解常微分方程应用实例。用欧拉法求常微分方程：

$$\begin{cases} y' = \sin x + y \\ y(x_0) = 1, x_0 = 0 \end{cases}$$

当 $h = 0.2$ 和 $h = 0.4$ 时的数值解，与精确解进行对比，并画出解的图形。

解：首先建立函数文件 myfun01.m：

```
function f=myfun01(x,y)
f=sin(x)+y;
```

通过在相同的积分区间上设定不同的取点数，从而改变步长，可得到不同的欧拉解。以下程序即可实现此功能，并给出了几乎等于精确解的微分方程符号解，如文件 ex0801.m 所示。

```
%ex0801.m
function ex0601()
[x1,y1]=MyEuler('myfun01',0,2*pi,1,16);                %欧拉法所得的解
h1=2*pi/15 %计算取步长
[x11,y11]=MyEuler('myfun01',0,2*pi,1,32);              %欧拉法所得的解
h2=pi/15 %计算步长
y=dsolve('Dy=y+sin(t)','y(0)=1');                      %该常微分方程的符号解
for k=1:33
    t(k)=x11(k);
    y2(k)=subs(y,t(k));                                %求其对应点的离散解
end
plot(x1,y1,'+b',x11,y11,'og',x11,y2,'*r')
legend('h=0.4 的欧拉法解','h=0.2 的欧拉解','符号解')
```

运行程序后，输出如图 8-2 所示的图形。

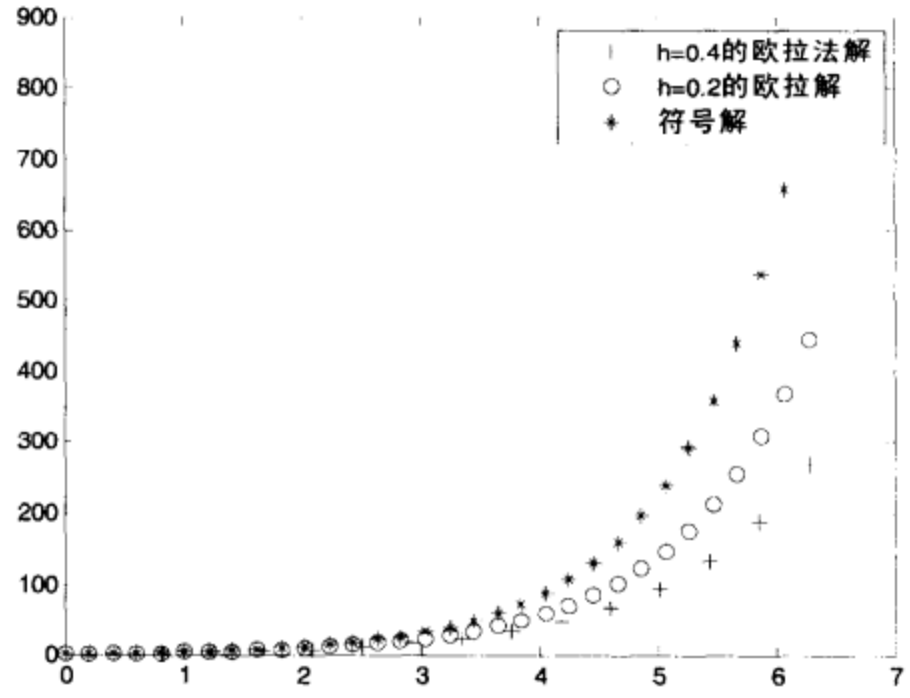

图 8-2 欧拉法所得方程解与符号解（精确解）对比

从图 8-2 中可以看出，用欧拉法得到的解和用符号法得到得解之间存在一定的误差，且取的步长越小，欧拉解越接近精确解。

由于向前差商比较简单，此处仅举向前差商的例子。另外还有两种差商，分别为向后差商和中心差商。有兴趣的读者可以参考相关资料，编写相应的 MATLAB 程序，其基本思想是一致的。

8.2.2 改进的欧拉法

改进的欧拉法实际上是通过预估-校正过程对梯形公式进行修正，从而简化迭代过程的

方法，又称为 Henu 算法。

1. 基本原理

改进的欧拉法的具体求解过程如下：

对方程 $y' = f(x, y)$ 两边由 x_n 到 x_{n+1} 积分，并利用梯形公式，可得：

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]$$

其中初始值为：

$$y(x_0) = y_0$$

由上可知，所得到的迭代式为隐式格式。计算中为了保证一定的精确度，避免迭代过程不菲的计算量，一般先用显式公式算出初始值，再用隐式公式进行一次修正。此过程称为预估-校正过程。

具体迭代公式如下：

$$\overline{y}_{n+1} = y_n + hf(x_n, y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \overline{y}_{n+1})]$$

2. 算法的程序实现

在 MATLAB 中编程实现的改进的欧拉法函数为：MyEulerPro。

功能：用改进的欧拉法求解常微分方程。

调用格式：[Xout,Yout]=MyEulerPro(fun, *x*0, *x*t, *y*0, PointNumber)。

其中，fun 为函数名；

*x*0 为自变量区间初值；

*x*t 为自变量区间终值；

*y*0 为函数在 *x*0 处的值；

PointNumber 为自变量在[*x*0, *x*t]上所取的点数；

Xout 为输出自变量区间上所取点的 *x* 值；

Yout 为对应点上的函数值。

改进的欧拉法的 MATLAB 程序代码如下：

```
function [Xout,Yout]=MyEulerPro(fun,x0,xt,y0,PointNumber)
% 用改进的欧拉法解微分方程 y'=fun
%函数 f（x，y）：fun
%自变量的初值和终值：x0,xt
%y0 表示函数在 x0 处的值，输入初值为列向量形式
%自变量在[x0,xt]上取的点数：PointNumber
%Xout:所取的点的 x 值
%Yout:对应点上的函数值
if nargin<5 | PointNumer<=0        %如果函数仅输入 4 个参数值，则 PointNumer 默认值为 100
    PointNumer=100;
end
if nargin<4  %y0 默认值为 0
```

```
    y0=0;
end
h=(xt-x0)/PointNumber;                          %计算所取的两离散点之间的距离
x=x0+[0:PointNumber]'*h;                        %表示出离散的自变量 x
y(1,:)=y0(:)';
for i=1:PointNumber                             %迭代计算过程
    f1=h*feval(fun,x(i),y(i,:));
    f1=f1(:)';
    f2=h*feval(fun,x(i+1),y(i,:)+f1);
    f2=f2(:)';
    y(i+1,:)=y(i,:)+1/2*(f1+f2);
end
Xout=x;
Yout=y;
```

3. 实例分析

【例 8-6】 改进的欧拉法求解常微分方程应用实例。利用改进的欧拉法求常微分方程：

$$\begin{cases} y' = \sin x + y \\ y(x_0) = 1, x_0 = 0 \end{cases}$$

即对【例 8-4】用改进的欧拉法求解，比较两种解法的结果，并画出解的图形。

解： 编写 MATLAB 程序 ex0802.m：

```
function ex0802()
%ex0802 比较改进的欧拉法，简单欧拉方法以及微分方程符号解
[x3,y3]=MyEulerPro('myfun01',0,2*pi,1,128);
[x,y1]=MyEuler('myfun01',0,2*pi,1,128);      %欧拉法所得的解
y=dsolve('Dy=y+sin(t)','y(0)=1');            %该常微分方程的符号解
for k=1:129
    t(k)=x(k);
    y2(k)=subs(y,t(k));                      %求其对应点的离散解
end
plot(x,y1,'-b',x3,y3,'og',x,y2,'*r')
legend('简单欧拉法解','改进欧拉解','符号解')
```

运行程序后，输出如图 8-3 所示的图形。

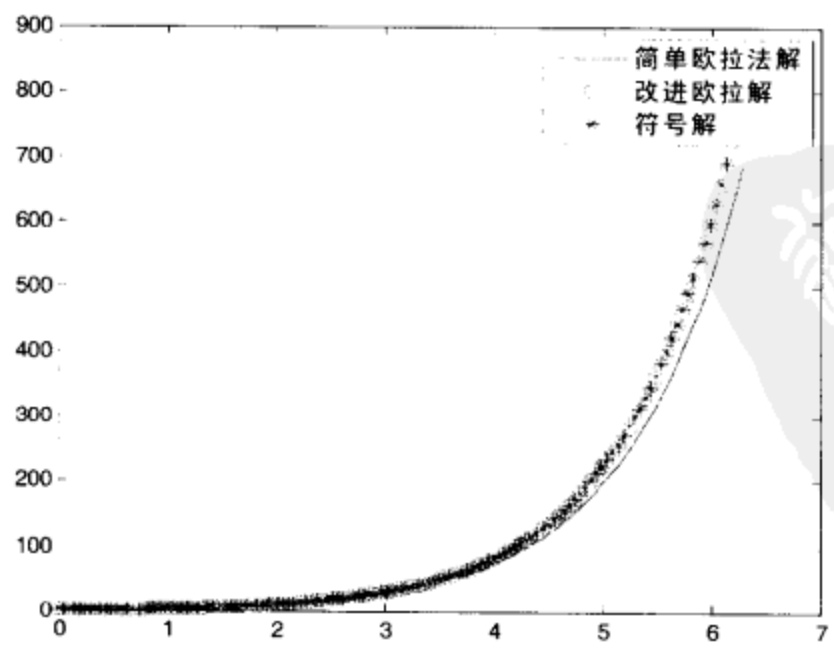

图 8-3　改进的欧拉法解、欧拉解、精确解的对比图

如图 8-3 所示，改进欧拉解与精确解几乎完全吻合，而简单欧拉解与精确解还有一定的误差。因此可得，改进的欧拉法较之简单欧拉法更为精确。

8.3 龙格-库塔法（Runge-Kutta）

尽管改进的欧拉法相对于简单欧拉法较为精确。但是对于很多实际的问题，运用这两种方法仍然得不到足够精确的解。龙格-库塔法是较之前两种方法计算精度更高的方法。

1. 基本原理

在龙格-库塔法中，四阶龙格-库塔法的局部截断误差约为 $o(h^5)$，被广泛应用于解微分方程的初值问题。其算法公式为：

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3)$$

其中：

$$\begin{cases} k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ k_3 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\ k_4 = f(x_n + h, y_n + hk_3) \end{cases}$$

2. 四阶龙格-库塔算法编程实现

在 MATLAB 中编程实现的四阶龙格-库塔算法函数为：MyRunge_Kutta。

功能：用四阶龙格-库塔算法求解常微分方程。

调用格式：[*x*, *y*]=MyRunge_Kutta(fun, *x*0, *xt*, *y*0, PointNum, varargin)。

其中，fun 为函数名；

*x*0 为自变量区间初值；

xt 为自变量区间终值；

y0 为函数在 *x*0 处的值；

PointNum 为自变量在[*x*0, *xt*]上所取的点数；

Varargin 为可选项参数；

x 为输出自变量区间上所取点的 *x* 值；

y 为对应点上的函数值。

四阶龙格-库塔法的 MATLAB 程序代码如下：

```
function [x,y] = MyRunge_Kutta(fun,x0,xt,y0,PointNum,varargin)
%Runge-Kutta 方法解微分方程形为 y'(t)=f(x,y(x))
%此程序可解高阶的微分方程。只要将其形式写为上述微分方程的向量形式
%函数 f（x，y）：fun
```

```
%自变量的初值和终值：x0,xt
%y0 表示函数在 x0 处的值，输入初值为列向量形式
%自变量在[x0,xt]上取的点数：PointNum
%varargin 为可选输入项可传适当参数给函数 f(x,y)
%x:所取的点的 x 值
%y:对应点上的函数值

if nargin < 4 | PointNum <= 0
    PointNum= 100;
end
if nargin < 3
    y0 = 0;
end
y(1,:) = y0(:)';                                          %初值存为行向量形式
h = (xt-x0)/(PointNum-1);                                 %计算步长
x = x0+[0:PointNum]'*h;                                   %得 x 向量值
for k = 1:PointNum  %迭代计算
f1 = h*feval(fun,x(k),y(k,:),varargin{:});
f1 = f1(:)';                                              %得公式中 k1
f2 = h*feval(fun,x(k) + h/2,y(k,:) + f1/2,varargin{:});
f2 = f2(:)';                                              %得公式中 k2
f3 = h*feval(fun,x(k) + h/2,y(k,:) + f2/2,varargin{:});
f3 = f3(:)';                                              %得公式中 k3
f4 = h*feval(fun,x(k) + h,y(k,:) + f3,varargin{:});
f4 = f4(:)';                                              %得公式中 k4
y(k + 1,:) = y(k,:) + (f1 + 2*(f2 + f3) + f4)/6;          %得 y(n+1)
end
```

3. 实例分析

【例 8-7】 龙格-库塔法求解常微分方程应用实例。采用龙格-库塔法求解微分方程，

$$\begin{cases} y' = -y + 1 \\ y(x_0) = 0, x_0 = 0 \end{cases}$$

比较与简单欧拉法、改进的欧拉法解的求解结果以及各种方法的运行时间，并画出解的图形。

解：容易得到该微分方程的真值解为：

$$y = 1 - \exp(-x)$$

MATLAB 程序如下：

```
ex0803.m
%用三种不同方法解微分方程
clear, clf                                                %清内存中的变量
x0=0;
xt=2;
Num=100;
h=(xt-x0)/(Num-1);
x =x0+[0:Num]*h;
a = 1;
yt = 1-exp(-a*x);                                         %真值解
fun = inline('-y + 1','x','y');                           %用 inline 构造函数 f(x,y)
y0 = 0;                                                   %设定函数初值
```

```
PointNum = 4;                                    %设定取点数
[x1,ye] = MyEuler(fun,x0,xt,y0,PointNum);
[x2,yh] = MyEulerPro(fun,x0,xt,y0,PointNum);
[x3,yr] = MyRunge_Kutta(fun,x0,xt,y0,PointNum);
plot(x,yt,'k', x1,ye,'b:', x2,yh,'g:', x3,yr,'r:')
legend('真值','简单欧拉法解','改进的欧拉法解','Runge_Kutta 法解')
hold on
plot(x1,ye,'bo',x2,yh,'b+', x3,yr,'r*')
PointNum= 1000;                                  %估计各算法运行 PointNum 步迭代的时间
tic  %计时开始
[x1,ye] = MyEuler(fun,x0,xt,y0,PointNum);
time_Euler = toc                                 %得到欧拉法的运行时间
tic
[x1,yh] = MyEulerPro(fun,x0,xt,y0,PointNum);
time_EulerPro = toc                              %得到改进的欧拉法运行时间
tic
[x1,yr] = MyRunge_Kutta(fun,x0,xt,y0,PointNum);
time_RK4 = toc                                   %得到 Runge_Kutta 法运行时间
```

运行后得到：

```
>> ex0803
time_Euler =
    0.2544
time_EulerPro =
    0.4887
time_RK4 =
    1.0195
```

运行的结果如图 8-4 所示。

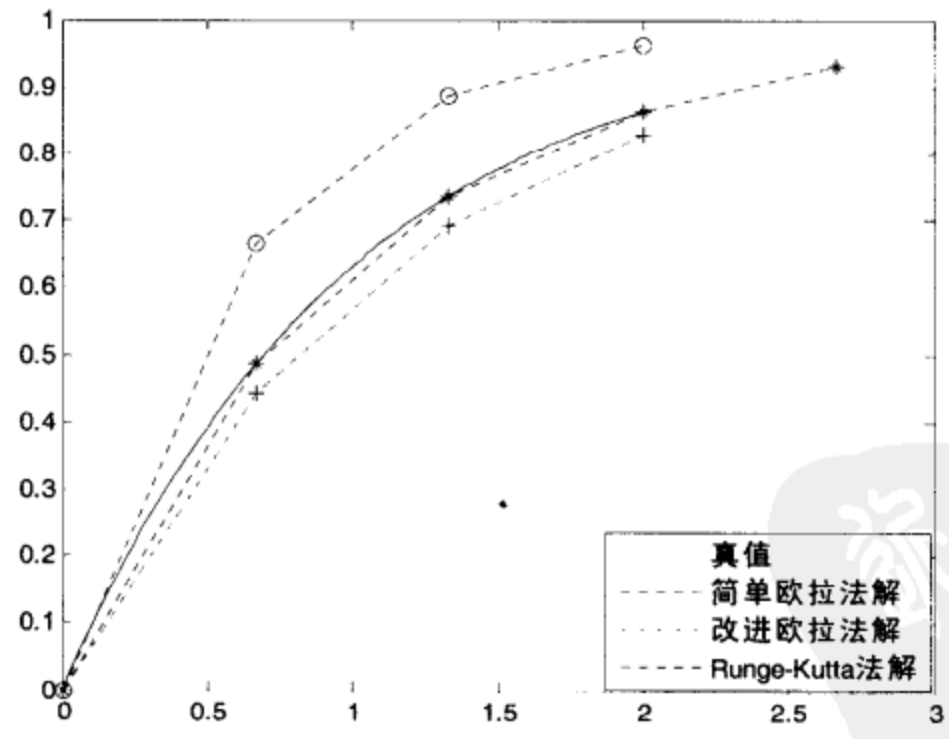

图 8-4　三种不同方法所得结果

由运算结果可知，龙格-库塔法可得到与真值几乎相同的微分方程数值解。但其相应的程序运行时间较长，几乎是简单欧拉法的五倍（欧拉法运行时间是 0.2544s，而龙格-库塔法运行时间是 1.0195s）。

4. 求解器 solver 中的龙格-库塔法求解

前面讲到，求解器 solver 中的 ode23 采用二阶、三阶龙格-库塔法；ode45 采用四阶、五阶龙格-库塔法。

【例 8-8】 求解器 solver 中的龙格-库塔法求解应用实例之一。分别采用二、三、四、五阶龙格-库塔方法求解以下方程，

$$\begin{cases} y' = 3y - 4e^{-x}, \text{且} 0 \leqslant x \leqslant 1 \\ y(x_0) = 1, x_0 = 0 \end{cases}$$

解法一：用二阶、三阶龙格-库塔函数求解。

编写程序文件 ex0804a.m：

```
%ex0804a.m 用 ode23 得到微分方程解并计算出该算法运行时间
fun =inline('-3*y^2+2*x.^2+3*x','x','y');        %用 inline 构造函数 f(x,y)
[x,y]=ode23(fun,[0,1],1);                        %可得到 x,y 输出向量值
ode23(fun,[0,1],1),hold on                       %可得到输出的函数图
```

结果如图 8-5a 所示。

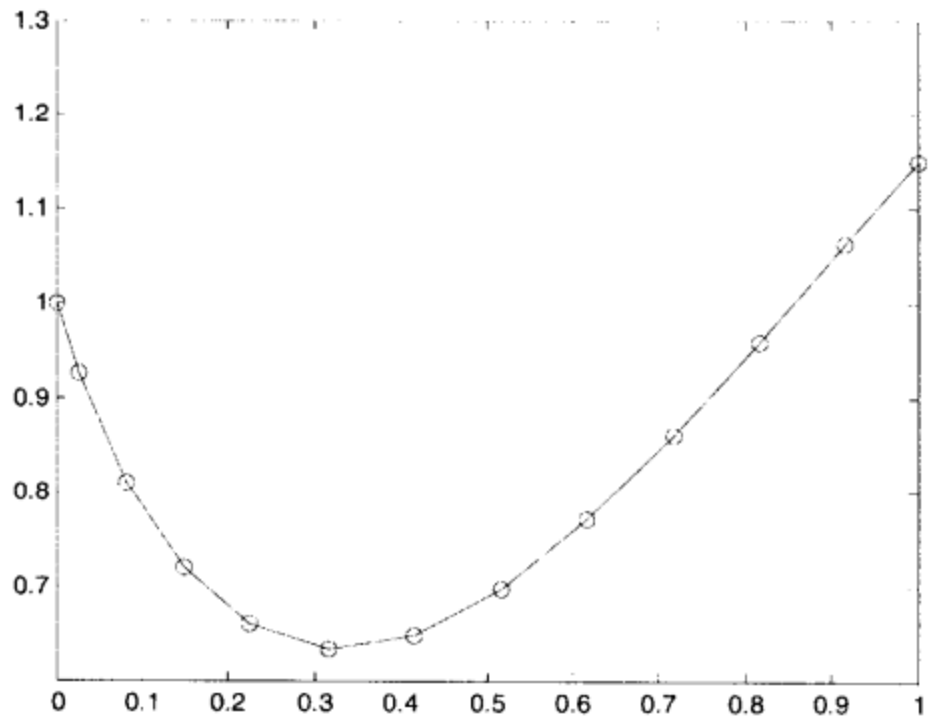

图 8-5a 二、三阶龙格-库塔法解微分方程

解法二：用四阶、五阶龙格-库塔函数求解。

编写程序文件 ex0804b.m：

```
%ex0804b.m 用 ode45 得到微分方程解,并计算出该算法运行时间
fun =inline('-3*y^2+2*x.^2+3*x','x','y');        %用 inline 构造函数 f(x,y)
ode45(fun,[0,1],1),hold on                       %可得到输出的函数图
tic
[x,y]=ode45(fun,[0,1],1);
t1=toc
tic
[x,y]=ode23(fun,[0,1],1);
t2=toc
```

命令窗口显示 t1、t2 值。

```
>> ex0604b
t1 =
  0.0256
t2 =
0.0166
```

如图 8-5b 所示显示函数的解结果。

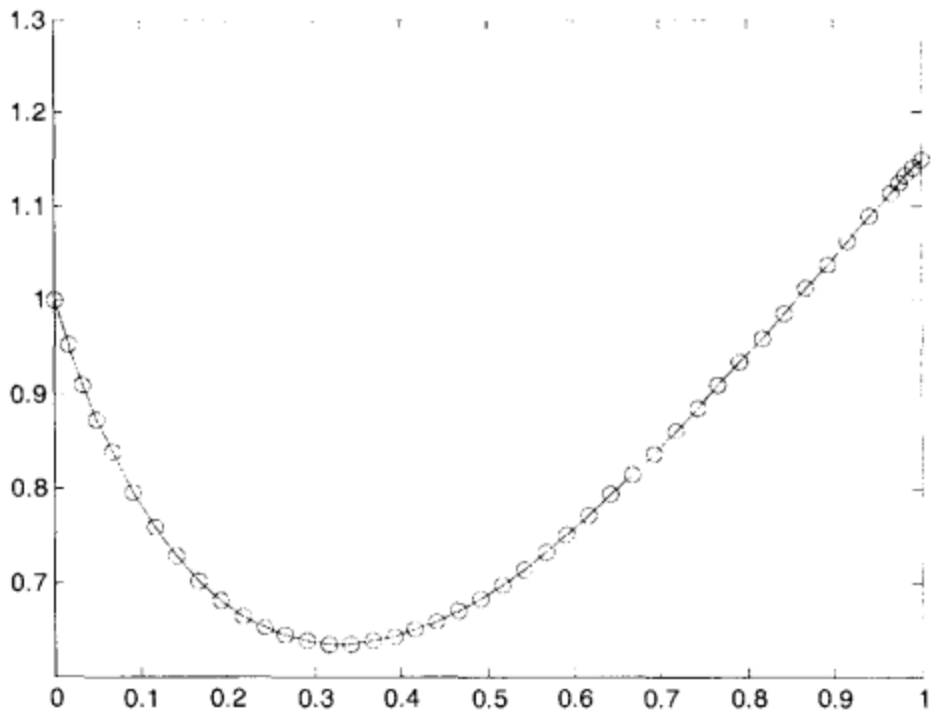

图 8-5b　四阶、五阶龙格-库塔算法解微分方程

由图 8-5a 和图 8-5b 所示可知用两个函数求解得到的结果相同。而 ode23 比 ode45 的运行速度要快一些。

【例 8-9】 求解器 solver 中的龙格-库塔法求解应用实例之二。采用 ode45 求解如下二阶方程：

$$\begin{cases} y'(1) = y(1)[1 - 0.1y(2)] \\ y'(2) = y(2)[-0.5 + 0.02y(1)] \end{cases}$$

初始条件为：$x = 0$ 时，$y(1) = 25$，$y(2) = 2$，并画出解的图形。

解：用 ode45 求解。首先建立函数文件 myfun02.m：

```
myfun02.m
function dx=myfun02(x,y)
dx=zeros(2,1);
dx(1)=y(1)*(1-0.1*y(2));
dx(2)=y(2)*(-0.5+0.02*y(1));
```

编写 ex0805.m 对微分方程求解。

```
% ex0805.m
[x,y]=ode45('myfun02',[0 15],[25 2]);
plot(x,y(:,1),'-',x,y(:,2),'*')          %画出 y（1），y(2)的函数图
```

运行程序，输出结果如图 8-6 所示。

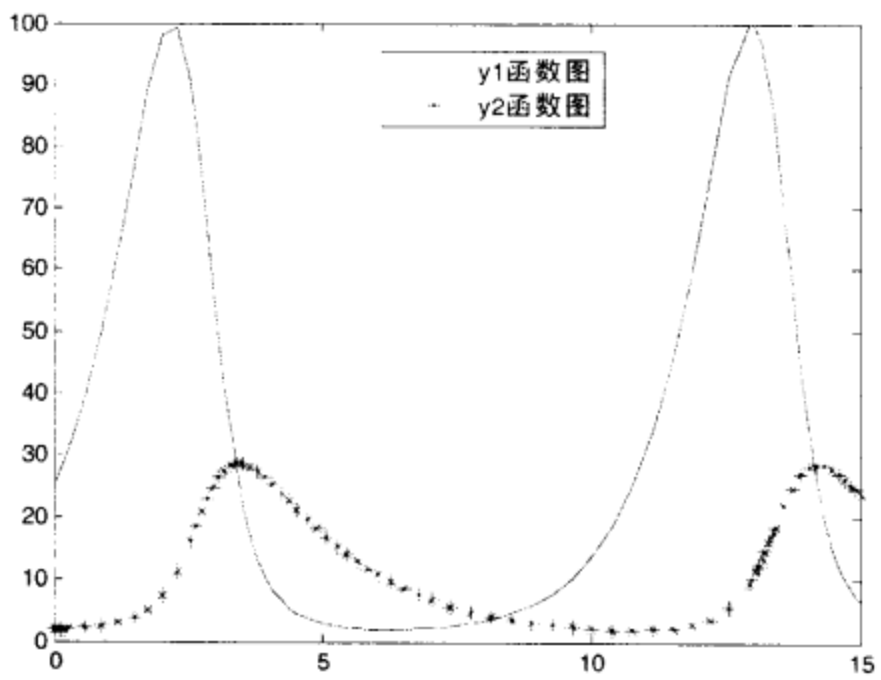

图 8-6 例 8-9 方程的解

8.4 预估-校正法(predictor-corrector)

所谓预估校正算法是指算法中主要包括的两个过程，预估过程和校正过程。前面讲到的改进的欧拉法是一种简单的预估校正算法。此外还有两种比较常用的形式，它们分别是 Adams-Bashforth-Moulton（ABM）方法和 Hamming 方法。

8.4.1 Adams-Bashforth-Moulton 方法

Adams-Bashforth-Moulton 法是最常用的传统的预测-校正法。

1. 基本原理

Adams-Bashforth-Moulton（ABM）方法与上面几种方法的主要不同是：它属于多步算法，y_{n+1}的值是由(x_n, y_n)，(x_{n-1}, y_{n-1})，(x_{n-2}, y_{n-2})，(x_{n-3}, y_{n-3})这几个点通过预估校正过程求得的，要分以下两步进行。

第一步：用四阶的拉格朗日多项式来近似得到$f(x, y)$的积分，并用前四个点的f值表示。从而可得y_{n+1}的预估表达式p_{n+1}：

$$p_{n+1} = y_n + \frac{h}{24}(-9f_{n-3} + 37f_{n-2} - 59f_{n-1} + 55f_n)$$

第二步：对p_{n+1}的预估值进行校正，得到y_{n+1}的校正值c_{n+1}。

$$c_{n+1} = y_n + \frac{h}{24}(f_{n-2} - 5f_{n-1} + 19f_n + 9f_{n+1})$$

通常使用的是改进的 ABM 算法。即在上述迭代式的基础上，再加入更正公式，因此可得如下的迭代运算公式：

$$p_{n+1} = y_n + \frac{h}{24}(-9f_{n-3} + 37f_{n-2} - 59f_{n-1} + 55f_n)$$

$$m_{n+1} = p_{n+1} + \frac{251}{270}(c_n - p_n)$$

$$c_{n+1} = y_n + \frac{h}{24}(f_{n-2} - 5f_{n-1} + 19f_n + 9f(x_{n+1}, m_{n+1}))$$

$$y_{n+1} = c_{n+1} - \frac{19}{270}(c_{n+1} - p_{n+1})$$

由于算法采用多步近似，因而其具有较小的截断误差 $o(h^5)$，其计算精度优于前面介绍的众多方法。

2. 算法的程序实现

MATLAB 中的 ode113 函数就是该算法的程序实现，由于迭代过程较复杂，此处不给出具体程序。有兴趣的读者可以参考 ode113 的实现程序。其具体命令格式与上一节介绍的 ode23 和 ode45 相同。

8.4.2 Hamming 方法

Hamming 方法是另一种形式的多步预估-校正法，其截断误差也是 $o(h^5)$，运算效率与 ABM 相当。下面对其进行具体介绍。

1. 基本原理

Hamming 预估-校正公式为：

$$p_{n+1} = y_{n-3} + \frac{4h}{3}(2f_{n-2} - f_{n-1} + 2f_n)$$

$$m_{n+1} = p_{n+1} + \frac{112}{121}(c_n - p_n)$$

$$c_{n+1} = \frac{1}{8}[9y_n - y_{n-2} + 3h(-f_{n-1} + 2f_n + f(x_{n+1}, m_{n+1}))]$$

$$y_{n+1} = c_{n+1} - \frac{9}{121}(c_{n+1} - p_{n+1})$$

2. 算法程序实现

在 MATLAB 中编程实现的 Hamming 算法函数为：MyHamming。

功能：以 Hamming 算法求解常微分方程。

调用格式：[*x*, *y*]=MyHamming(*f*, *x*0, *xt*, *y*0, *N*, *KC*)。

其中，*f* 为函数名；

*x*0 为自变量区间初值；

xt 为自变量区间终值；

*y*0 为函数在 *x*0 处的值；

N 为自变量在[*x*0, *xt*]上所取的点数；

KC 为确定是否使用改正公式；

x 为输出自变量区间上所取点的 *x* 值；

y 为对应点上的函数值。

Hamming 算法的 MATLAB 程序代码如下：

```
function [x,y]=MyHamming(f,x0,xt,y0,N,KC)
% 用 hamming 方法解向量微分方程 y' (t) = f(t,y(t))
%函数 f (x, y) : f
%自变量的初值和终值: x0,xt
%函数在 x0 处的值: y0
%自变量在[x0,xt]上取的点数: N
%选择是否使用改正公式: KC
%x:所取的点的 x 值
%y:对应点上的函数值

if nargin < 6,
    KC = 1;                                          %默认使用更正公式
end
if nargin < 5 | N <= 0
    N = 100;                                         %最大迭代数默认为 100
end
if nargin < 4
    y0 = 0;                                          %初值默认为 0
end
y0 = y0(:)';                                         %使 y0 为行向量
h = (xt-x0)/(N-1);                                   %步长
xt0 = x0+2*h;
[x,y] = MyRunge_Kutta(f,x0,xt0,y0,3);                %用 Runge-Kutta 得初始三点值
x = [x(1:3)' x(4):h:xt]'; %重新调整 x 向量，使其前三点与 Runge-Kutta 法计算的三点
                          %相同
for k = 2:4
    F(k-1,:) = feval(f,x(k),y(k,:)); %计算 f2 f3 f4
end
p = y(4,:);                                          % 预估量初值
c = y(4,:);                                          %校正量初值
h34 = h/3*4;                                         %预估公式中系数
KC11 = KC*112/121;                                   %更正公式中系数
KC91 = KC*9/121;                                     %最后计算 y 值的公式中更正项的
                                                      系数
h312 = 3*h*[-1 2 1];                                 %校正公式中各 f 项系数
for k = 4:N-1
   p1 = y(k-3,:) + h34*(2*(F(1,:) + F(3,:)) - F(2,:)); %p(n+1)计算公式
   m1 = p1 + KC11*(c - p);                           %m(n+1)计算公式
   c1 = (-y(k - 2,:) + 9*y(k,:) +...
   h312*[F(2:3,:); feval(f,x(k + 1),m1)])/8;         %c(n+1)计算公式
   y(k+1,:) = c1 - KC91*(c1 - p1);                   %y(n+1)计算公式
   p = p1;                                           %更新预测值
   c = c1;                                           %更新校正值
   F = [F(2:3,:);feval(f,x(k + 1),y(k + 1,:))];
end
```

【例 8-10】 预估-校正法求解常微分方程应用实例。采用 ABM 法与 Hamming 法解方程：

$$\begin{cases} y' = -y+1 \\ y(x_0)=0, x_0=0 \end{cases}$$

并画出解的图形。

解：在 MATLAB 中编写 ex0806.m 来实现求解。

```
%ex0806.m 用 ode113(),MyHamming(),解微分方程 y'=-y+1
clear, clf
x0 = 0;                                          %积分区域开始值
xt = 10;                                         %积分区域终止值
y0 = 0;                                          %y 初值
N = 50;                                          %离散点数
fun66 = inline('-y+1','x','y');                  %需要求解的微分方程
f66 = inline('1-exp(-t)','t');                   %方程的精确解析解
tic
[x113,y113] = ode113(fun66,[x0,xt],y0);
t113 = toc
tic
[x1,yH] = MyHamming(fun66,x0,xt,y0,N);
tH = toc
yt1 = f66(x1);                                   %在精确解上取离散点
subplot(1,2,1)                                   %将窗口分为一行两列，在第一个图中作
                                                 %出各种方法的解的曲线
plot(x1,yt1,'*r',x1,yH,'og')
legend('精确解','Hamming 解')
title('Hamming 法所得的解')
yt113=f66(x113);
subplot(122 )
plot(x113,yt113,'*r', x113, y113,'og')
legend('精确解','ABM 解')
title('ode113 即 ABM 算法所得的解')
```

运行程序，输出结果如图 8-7 所示。

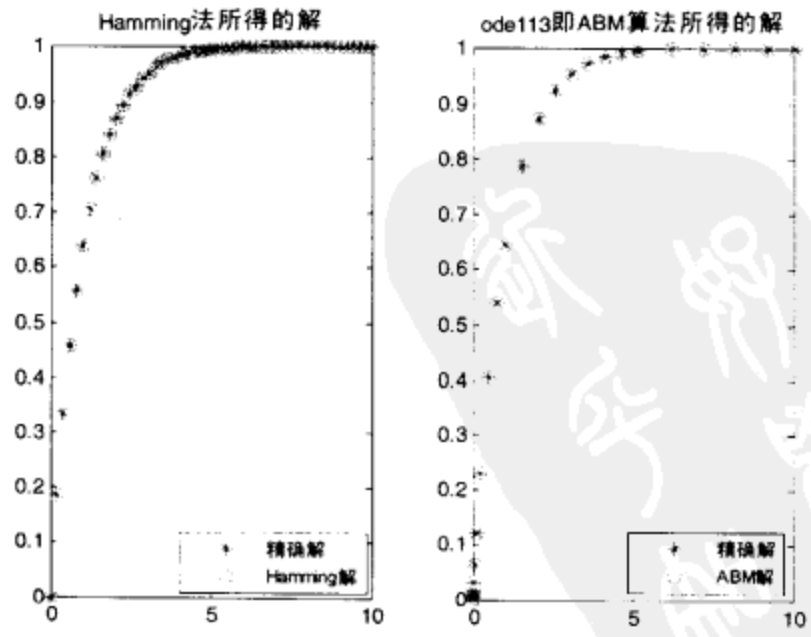

图 8-7 用 ABM 法与 Hamming 法解微分方程

8.5 常微分方程求解综合实例

下面通过几个实例来综合讲述常微分方程的求解。

【例 8-11】 多阶常微分方程求解实例之一。采用自编函数 MyEulerPro()以及 MyRunge_Kutta()求解多阶微分方程：$x^3\frac{d^3y}{dx^3}-2\frac{d^2y}{dx^2}-3\frac{dy}{dx}=3e^{2x}$，其中 $y(1)=1$，$y'(1)=10$，$y''(1)=30$，$x\in[1,10]$，并画出解的图形。

解：对于此类多阶微分方程，先将其变换为三个一阶微分方程组，再利用上面几节介绍的方法求解。对于该微分方程，可得下面的式子：

$$\begin{cases}\dfrac{dy_1}{dx}=y_2\\ \dfrac{dy_2}{dx}=y_3\\ \dfrac{dy_3}{dx}=\dfrac{2}{x^3}y_3+\dfrac{3}{x^3}y_2+\dfrac{3e^{2x}}{x^3}\end{cases}$$

写成矩阵方程形式为：

$$\frac{dY}{dx}=\frac{d}{dx}\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix}=\begin{bmatrix}0&1&0\\0&0&1\\0&\dfrac{3}{x^3}&\dfrac{2}{x^3}\end{bmatrix}Y+\begin{bmatrix}0\\0\\\dfrac{3e^{2x}}{x^3}\end{bmatrix}$$

方程变成上述形式后，其求解过程与一阶微分方程相同，不过值得注意的是，y 的初值输入应为向量的形式。具体见下面的程序。

首先编写函数 myfun03.m：

```
function dy=myfun03(x,y)
dy=zeros(3,1)                                  %初始化变量 dy
dy(1)=y(2);                                    %dy(1)表示 y 的一阶导数,其等于 y 的第二列值
dy(2)=y(3);                                    %dy(2)表示 y 的二阶导数
dy(3)=2*y(3)/x^3+3*y(2)/x^3+3*exp(2*x)/x^3  %dy(3)表示 y 的三阶导数
```

用自编函数 MyEulerPro 以及 MyRunge_Kutta 求解，程序如下：

```
%ex0807.m 用自编函数 MyEulerPro（）,MyRunge_Kutta（）求多阶微分方程
x0=1;                                          %设定函数变量起始值
xt=10;                                         %设定函数变量终止值
y0=[1 10 30];                                  %y 初值
N=100;
[x,yE]=MyEulerPro('myfun03',x0,xt,y0,N);
[x,yR]=MyRunge_Kutta('myfun03',x0,xt,y0,N);
```

```
plot(x,yE(:,1),'r*',x,yR(:,1),'go')
legend('改进的欧拉法解','龙格-库塔法解')
```

运行程序，输出结果如图 8-8 所示。

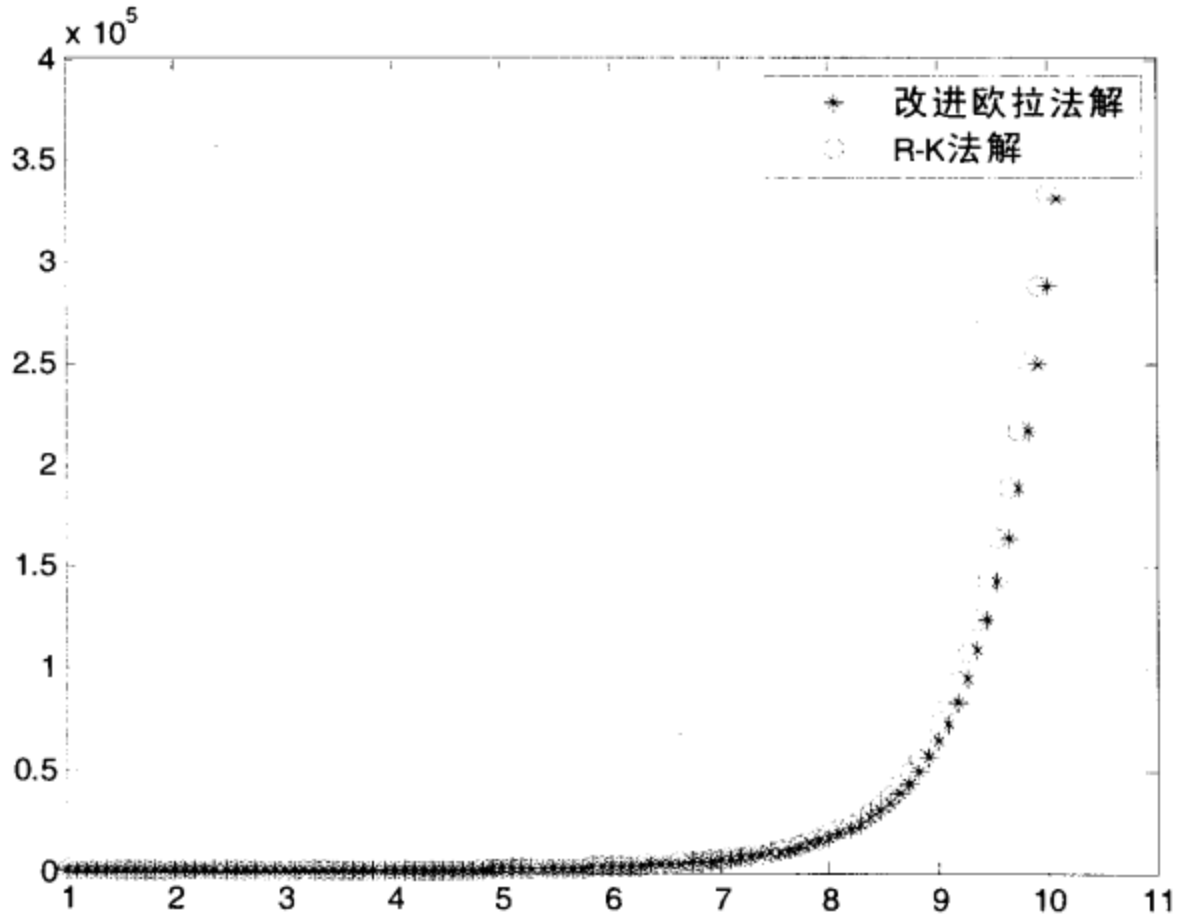

图 8-8　用两种方法求多阶微分方程的解

【例 8-12】　多阶常微分方程求解实例之二。用 MATLAB 功能函数 ode23、ode45、ode113 求解多阶微分方程 $x^3\frac{d^3y}{dx^3}-2\frac{d^2y}{dx^2}-3\frac{dy}{dx}=3e^{2x}$，其中 $y(1)=1$，$y'(1)=10$，$y''(1)=30$，$x\in[1,10]$，并画出解的图形，即求解【例 8-11】中的微分方程。

解：在 MATLAB 中编写以下程序实现求解。

```
%ex0808 用 ode23 ode45 ode113 解多阶微分方程
clear,clc
[x23,y23]=ode23('myfun03',[1,10],[1 10 30]);
[x45,y45]=ode45('myfun03',[1,10],[1 10 30]);
[x113,y113]=ode113('myfun03',[1,10],[1 10 30]);
figure(1)                                          %第一幅图
plot(x23,y23(:,1),'*r',x45,y45(:,1),'ob',x113,y113(:,1),'+g')
legend('ode23 解','ode45 解','ode113 解')
title('ODE 函数求解结果')
figure(2)
plot(x45,y45)                                      %以 ode45 为例作出函数以及其各阶导数图
legend('y','y 一阶导数','y 二阶导数')
title('y,y 一阶导数,y 二阶导数函数图')
```

运行程序，输出结果如图 8-9a、图 8-9b 所示。

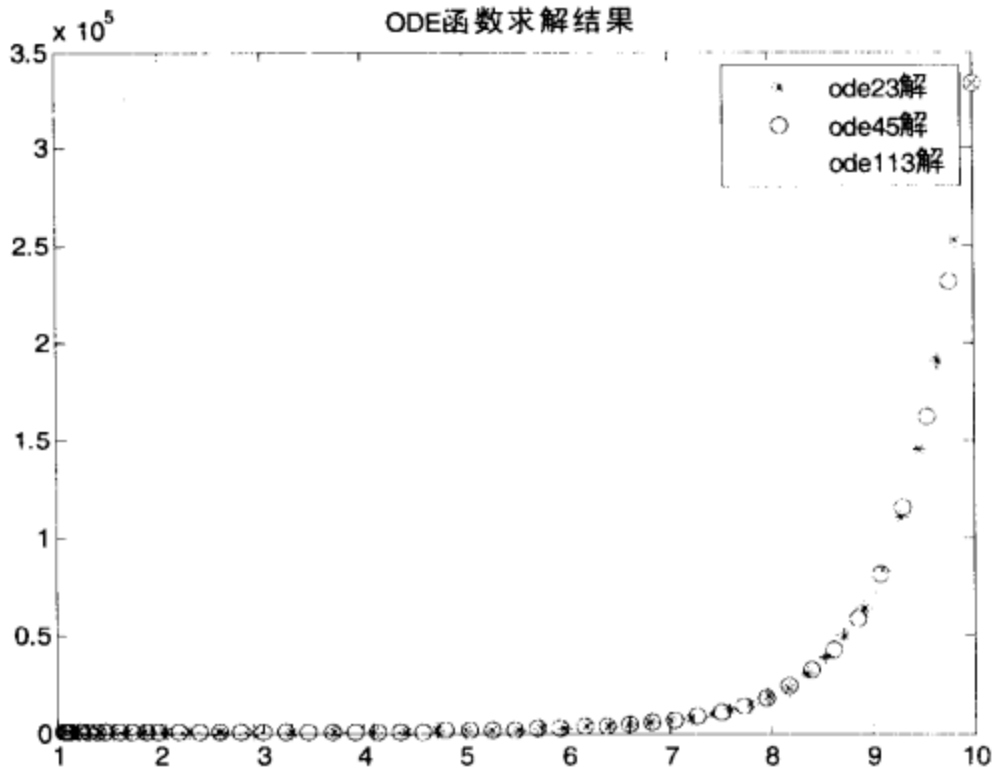

图 8-9a MATLAB 中 ODE 功能函数求解结果

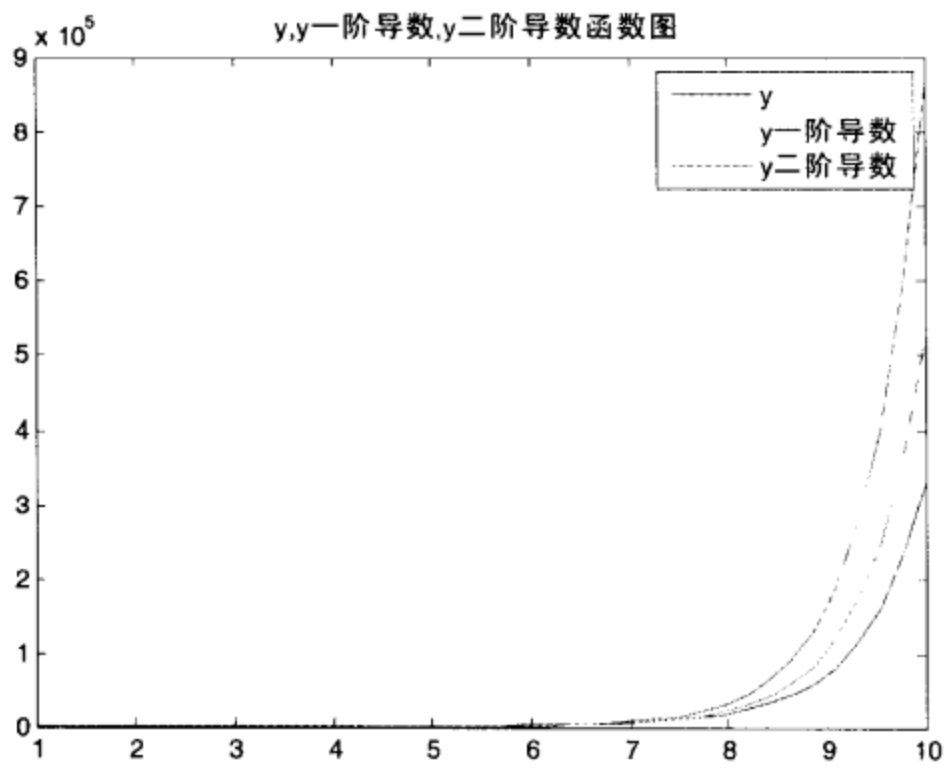

图 8-9b y 以及其一阶、二阶导数图

8.6 差分方程求解

差分方程可由常微分方程离散化而得到，对于差分方程的求解在实际应用中有重要的作用。下面仅讲述求解差分方程的几种常见方法。

8.6.1 用 filter 函数解差分方程

设 N 阶差分方程为：

$$y(n)=\sum_{r=0}^{M}b_r x(n-r)-\sum_{k=1}^{N}a_k y(n-k)$$

在 MATLAB 中，可以用函数 filter()求解差分方程。函数 filter 有多种使用形式，常用的调用格式为：

$$y=\text{filter}(\boldsymbol{B},\boldsymbol{A},x)$$

其中，$\boldsymbol{A}=[a_0,a_1,\cdots,a_N]$，$\boldsymbol{B}=[b_0,b_1,\cdots,b_M]$，$x$ 为输入信号序列。

【例 8-13】 差分方程求解实例之一。求如下差分方程所表示的离散系统的单位脉冲响应：

$$y(n)-0.7y(n-1)-0.6y(n-2)+y(n-3)=x(n)$$

并绘制单位脉冲响应图。

解：系统的单位脉冲响应即为 $x(n)=\delta(n)$ 时的差分方程解 $y(n)$，利用 filter 函数编程求解，在 MATLAB 中编写程序 ex0809 来实现。

```
%ex0809.m 求解离散系统单位脉冲响应
clear,clc                                    %清内存
A=[1,-0.7,-0.6,1];                           %A 系数向量
B=[1];                                       %B 系数向量
n=[-20:100];                                 %作图的脉冲响应长度
x=[n==0];                                    %输入脉冲 等价于
                                              x=[zeros(1,20),1,zeros(1,100)];
y=filter(B,A,x);                             %求得脉冲响应
stem(n,y);                                   %作图
title('冲激响应');
grid
```

运行程序，输出结果如图 8-10 所示。

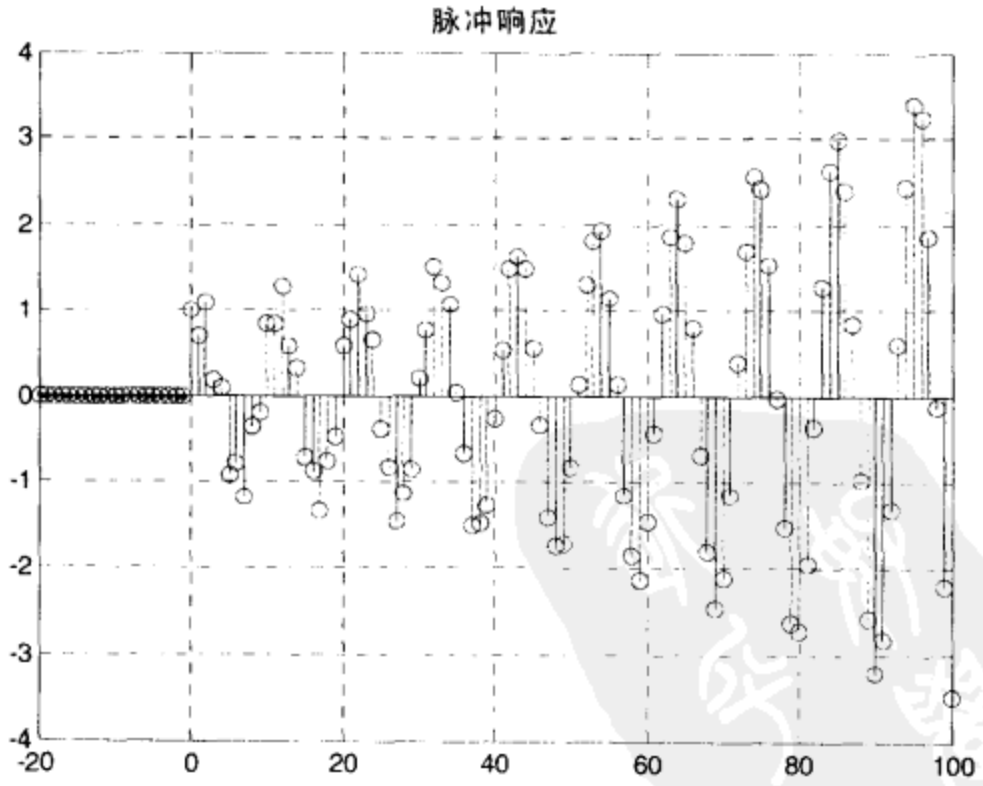

图 8-10 单位脉冲响应图

【例 8-14】 差分方程求解实例之二。求下列差分方程：

$$y(n)-0.7y(n-1)-0.6y(n-2)+y(n-3)=x(n)+0.5x(n-1)$$

其中 $x=[0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0]$，对应于 n 的值为从-9 到 9，并绘制输入响应和输出响应图。

解：在 MATLAB 中编写程序文件 ex0810.m 来实现求解。

```
%ex0810.m 求差分方程的解
A=[1,-0.7,-0.6,1];
B=[1,0.5];
n=[-9:9];
x=[zeros(1,5),ones(1,8),zeros(1,6)];
y=filter(B,A,x)
subplot(211)
stem(n,x);
title('输入响应')
subplot(212)
stem(n,y)
title('输出响应')
```

MATLAB 命令窗口的输出为：

```
y =
  Columns 1 through 10
   0         0         0         0         0    1.0000    2.2000    3.6400
4.3680    4.5416
  Columns 11 through 19
   3.6599    2.4189    0.8476   -1.1153   -2.6910   -3.4005   -2.8797
-1.3650    0.7171
```

并得到如图 8-11 所示的结果。

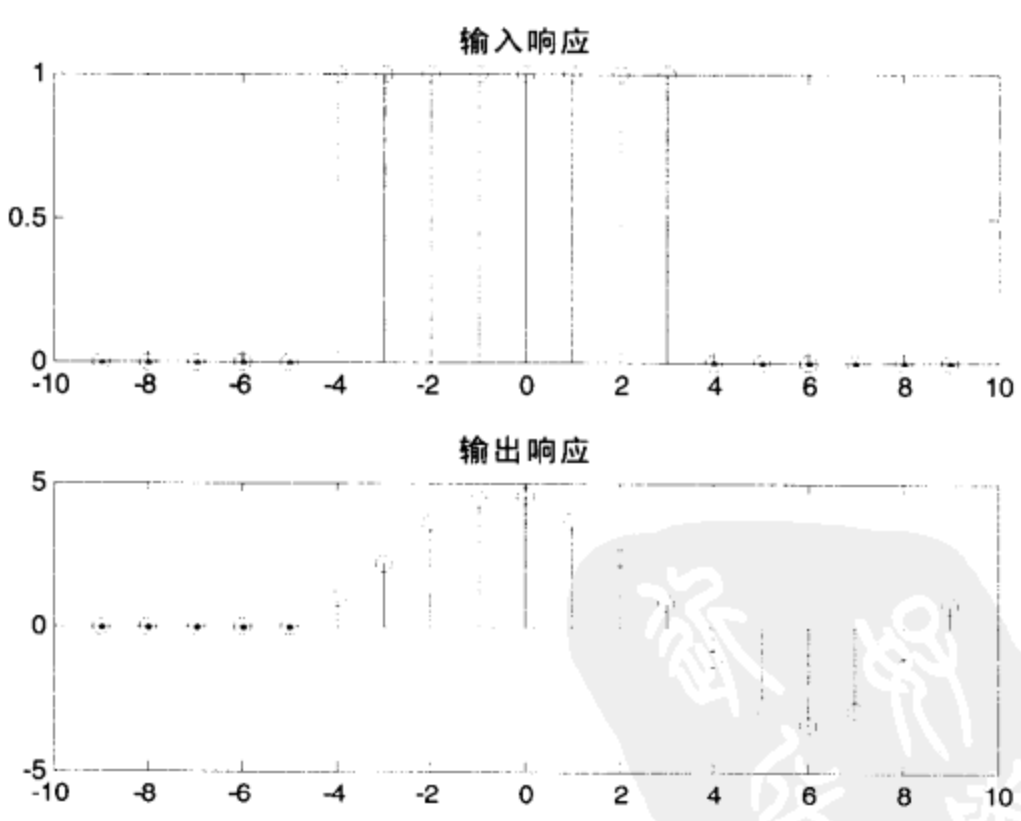

图 8-11 输入、输出对比图

8.6.2 递推法解差分方程

上述两个例题均是利用 filter 函数求解的，未附加条件。对于带有附加条件的差分方程，

通常采用递推法求解。

设 N 阶常系数差分方程如下：

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{r=0}^{M} b_r x(n-r)$$

附加条件为：　　$y[-k]$，　$k = 1,2,\cdots,N$

将差分方程变形为后推方程，便可迭代计算出 $n \geqslant 0$ 的 $y[n]$：

$$y(n) = \frac{1}{a_0}\{\sum_{r=0}^{M} b_r x(n-r) - \sum_{k=1}^{N} a_k y(n-k)\}$$

若要求 $-k$ 之前时刻的 $y(n)$ 值，需要将差分方程变形为前推方程：

$$y(n-N) = \frac{1}{a_N}\{\sum_{r=0}^{M} b_r x(n-r) - \sum_{k=0}^{N-1} a_k y(n-k)\}$$

【例 8-15】 递推算法求解差分方程实例。用递推算法求解差分方程：

$$y(n) - y(n-1) = 3x(n) + 2x(n-1) - x(n-2)$$

已知输入 $x[n] = \delta[n], y(-1) = 0$。画出 $y(n)$ 在坐标点[-10,20]上的图。

解：在 MATLAB 中编写程序 ex0811.m 来实现。

```
%ex0811.m 递推法解差分方程
clear,clc
%求 y(0)到 y(20)
x0=[0,0,1];                          %设定后推方程的迭代初值
y=zeros(1,31);                       %y 初值设为零
y(10)=0; %y(-1)=0
y0=0;                                %后推初值
for i=1:21
    y0=y0+[-1,2,3]*x0';              %用后推方程计算 n>=0 的 y 值
    y(i+10)=y0;
    x0=[x0(2:3),0];
end
%求 y(-2)到 y(-10)
x0=[0,0,0];                          %前推方程迭代初始化
y0=0;
for i=1:9
    y0=y0+[1,-2,-3]*x0';             %用前推方程迭代计算 n<-1 的 y 值
    y(10-i)=y0;
    x0=[0,x0(1:2)];
end
y
n=[-10:20];
stem(n,y)
```

命令窗口输出为：

```
>> ex0811
y =
  Columns 1 through 17
0     0     0     0     0     0     0     0     0     0     3     5     4     4
4     4     4
  Columns 18 through 31
4     4     4     4     4     4     4     4     4     4     4     4     4     4
```

图 8-12 绘出了坐标[-10,20]上的 y 值。此程序仅计算-10 到 20 之间的 y 值，依此步骤重复计算可得 y 的解为：$y(n)=\begin{cases}0, & n<0\\3, & n=0\\5, & n=1\\4, & n\geq 2\end{cases}$。

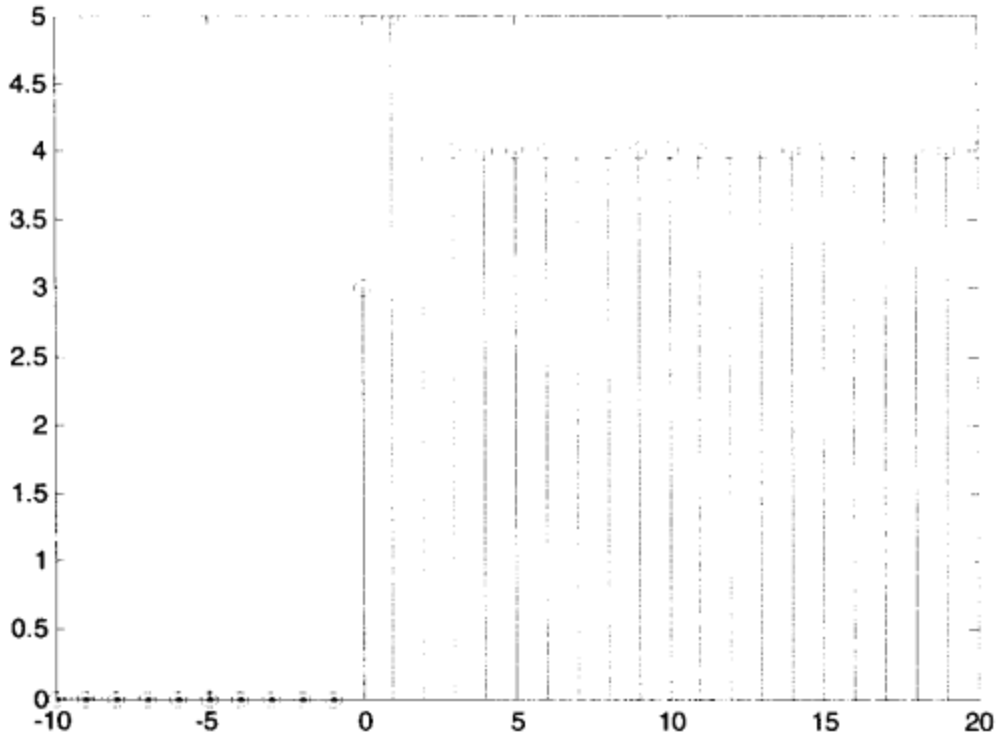

图 8-12　递推法所得 y 值

8.6.3　利用 z 反变换求解

利用 z 反变换求解差分方程多用于手工求解系统脉冲响应的过程中。

MATLAB 提供了符号运算工具箱，可方便地进行 Z 变换和 Z 反变换，进行 Z 反变换的函数是 iztrans，iztrans 有多种调用格式，其常用的调用格式为：

$$f=\text{iztrans}(F)$$

表示求 F 的 z 变换原函数 f(n)。

此处仅通过一例简单介绍一下该方法。

【例 8-16】　利用 z 变换求解差分方程实例。用单边 z 反变换求解差分方程的脉冲响应（求 $n\geqslant 0$ 的值）：

$$y(n)-y(n-1)=x(n)$$

其中　　$y(-1)=0$。

解：可知单位脉冲响应 $h(n)$ 为 $x(n)=\delta(n)$ 时的输出 $y(n)$。对差分方程两边进行单边变换可得：$Y(z)-z^{-1}Y(z)-y(-1)=X(z)$，当 $x(n)=\delta(n)$ 时，$X(z)=1$，$Y(z)=H(z)$。再由 $y(-1)=0$，故可得：

$$H(z)=\frac{1}{1-z^{-1}}$$

用 MATLAB 中的 iztrans 求其反变换，可得 $h(n)$ 的关于变量 n 的表达式。

具体见程序文件 ex0812.m，如下所示。

```
%ex0812.m 利用 z 变换求脉冲响应
syms z
hz=1/(1-z^(-1));
hn=iztrans(hz)
```

命令窗口输出结果为：

```
>>ex0812
hn
  =1
```

8.7 小结

本章讲述了 MATLAB 的求解函数和求解器及其常见用法，还讲述了采用 MATLAB 解决常微分方程初值问题的几种方法，包括欧拉法（Euler's method）、预测－修正法（Predictor-Corrector Method）和龙格-库塔法（Runge-Kutta method），最后讲述了常系数差分方程的三种基本解法。

本章所举的例子很多并不是单纯用一种方法求解的，很多例子的程序中都涉及了几种算法的比较，包括计算精度及计算时间的比较。读者在选择算法解微分方程时也要考虑到这两个方面。

差分方程求解法的实际应用中，filter 函数多被用来求系统脉冲响应，或者作为求滤波器输出。由于现实生活中的系统为因果系统，其满足 $y(-k)=0$、故用递归法不需附加条件便可求脉冲响应，其应用也很普遍。z 反变换求解多用在手工计算中。

第 9 章

偏微分方程求解

科学与工程中的大量现象与规律都可以用偏微分方程来描述，偏微分方程的定解问题成了表述科学与工程中的现象和规律的重要数学工具，应用极其广泛。

MATLAB 提供了简单易用、功能强大的工具箱来求解偏微分方程数值，同时，通过编程可以实现一些偏微分方程的求解。

通过本章的学习，读者不仅能掌握 MATLAB 的偏微分方程工具箱（PDE Toolbox）的使用，而且还能掌握利用 MATLAB 对椭圆型偏微分方程、抛物线型偏微分方程和双曲线型偏微分方程这三大类偏微分方程进行数值求解的方法。

9.1 偏微分方程概述

所谓偏微分方程是指含两个以上自变量的微分方程。本章主要讨论含自变量 x 和 y 的二阶偏微分方程。它可表示成下式的形式：

$$A(x,y)\frac{\partial^2 u}{\partial^2 x}+B(x,y)\frac{\partial^2 u}{\partial x\partial y}+C(x,y)\frac{\partial^2 u}{\partial^2 y}=f(x,y,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial y})$$

其中，自变量取值范围是：

$$x_0 \leqslant x \leqslant x_f, y_0 \leqslant y \leqslant y_f$$

边界条件为：

$$u(x,y_0)=b_{y_0}(x), u(x,y_f)=b_{y_f}(x),$$
$$u(x_0,y)=b_{x_0}(y), u(x_f,y)=b_{y_f}(y),$$

根据偏微分方程中的 B、A、C 系数之间的关系，可以把偏微分方程分为 3 类：

（1）椭圆型偏微分方程，它满足：

$$B^2-4AC<0$$

（2）抛物线型偏微分方程，它满足：

$$B^2-4AC=0$$

（3）双曲线型偏微分方程，它满足

$$B^2-4AC>0$$

这三种方程分别对应稳定状态、扩散状态和振荡状态。由于偏微分方程的解析解很难获得，本章主要讲述它们采用 MATLAB 进行的数值求解法。

9.2 椭圆偏微分方程

以下仅以一类特殊的椭圆偏微分方程 Helmhotz 方程为例，介绍椭圆偏微分方程的求解。

9.2.1 常规 Helmholtz 方程的数值解

Helmholtz 方程是椭圆偏微分方程中的一种，它满足：

$$\nabla^2 u(x,y)+g(x,y)u(x,y)=\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}+g(x,y)u(x,y)=f(x,y)$$

其自变量取值范围 D 为：

$$x_0 \leqslant x \leqslant x_f, y_0 \leqslant y \leqslant y_f$$

边界条件为：

$$u(x,y_0)=b_{y_0}(x),\quad u(x,y_f)=b_{y_f}(x),$$
$$u(x_0,y)=b_{x_0}(y),\quad u(x_f,y)=b_{y_f}(y),$$

如果 Helmholtz 方程中 $g(x,y)=0$，则它被称为 Possion 方程。

如果同时满足 $g(x,y)=0$ 和 $f(x,y)=0$，则它被称为 Laplace 方程。

1. 求解原理描述

为了采用不同的方法求解，对区域 D 进行分割：将 $x_0 \leqslant x \leqslant x_f$ 范围内的 x 轴等分为 M_x 段，每段长为：$\Delta x=(x_f-x_0)/M_x$；同理将 y 轴分割成 M_y 段，每段长为 $\Delta y=(y_f-y_0)/M_y$。

在 Helmholtz 方程中，利用三点中心差分估计二阶导数：

$$\frac{\partial^2 u(x,y)}{\partial x^2}\Big|_{x_j,y_i}\approx\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{\Delta x^2}$$

其中：

$$x_j=x_0+j\Delta x, y_i=y_0+i\Delta y$$

$$\frac{\partial^2 u(x,y)}{\partial y^2}\Big|_{x_j,y_i}\approx\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\Delta y^2}$$

其中：

$$u_{i,j}=u(x_j,y_i)$$

因此，对于自变量区间上的每一个内点(x_j, y_i)，可以得到如下等式：

$$\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{\Delta x^2}+\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\Delta y^2}+g_{i,j}u_{i,j}=f_{i,j}$$

其中：

$$u_{i,j}=u(x_j,y_i), f_{i,j}=f(x_j,y_i), g_{i,j}=g(x_j,y_i)$$

在区域D上共有$(M_y-1)(M_x-1)$个内点，因此可以建立$(M_y-1)(M_x-1)$个方程。

但当M_x, M_y较大时，联立求解方程组显然不现实，迭代方法可以解决这个问题。

迭代法首先把边界条件变形为如下形式：

$$u_{i,j}=r_y(u_{i,j+1}+u_{i,j-1})+r_x(u_{i+1,j}+u_{i-1,j})+r_{xy}(g_{i,j}u_{i,j}-f_{i,j})$$

$$u_{i,0}=b_{x0}(y_i),\ u_{i,M_x}=b_{xf}(y_i),\ u_{0,j}=b_{y0}(x_i),\ u_{M_y,j}=b_{yf}(x_j)$$

其中：

$$\frac{\Delta y^2}{2(\Delta x^2+\Delta y^2)}=r_y,\quad \frac{\Delta x^2}{2(\Delta x^2+\Delta y^2)}=r_x,\quad \frac{\Delta x^2\Delta y^2}{2(\Delta x^2+\Delta y^2)}=r_{xy}$$

迭代算法需要迭代初值的先验知识，采用边界值的平均作初值具有较好的效果。

2. 求解程序 MATLAB 的实现

在 MATLAB 中编程实现的 Helmholtz 方程求解程序为：Helmholtz。

功能：迭代法求解 Helmholtz 方程。

调用格式：[*u*, *x*, *y*] = Helmholtz(*f*, *g*, *bx*0, *bxf*, *by*0, *byf*, *D*, *Mx*, *My*, MinErr, MaxIter)。

其中，*f*，*g* 为函数名；

*bx*0 为在边界 *x*=*x*0 上的函数值；

bxf 为在边界 *x*=*xf* 上的函数值；

*by*0 为在边界 *y*=*y*0 上的函数值；

byf 为在边界 *y*=*yf* 上的函数值；

D 为边界点值，长度为四的列向量，其元素分别为 ***x*0**，***xf***，***y*0**，***yf***；

Mx 为沿 *x* 轴的区间数；

My 为沿 *y* 轴的区间数；

MinErr 为误差控制因子；

MaxIter 为最大迭代次数；

[*u*, *x*, *y*]:方程 *u*（*x*，*y*）在（*x*，*y*）点的函数值。

迭代法求解 Helmholtz 方程的 MATLAB 程序代码如下：

```
function [u,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter)
%解方程： u_xx + u_yy + g(x,y)u = f(x,y)
%f，g：函数名；
%bx0：在边界 x=x0 上的函数值；
%bxf：在边界 x=xf 上的函数值；
%by0：在边界 y=y0 上的函数值；
%byf：在边界 y=yf 上的函数值；
%D：边界点值，长度为四的列向量，其元素分别为 x0，xf，y0，yf；
```

```
%Mx：沿 x 轴的区间数；
%My：沿 y 轴的区间数；
%MinErr：误差控制因子；
%MaxIter：最大迭代次数；
%[u,x,y]:方程 u（x，y）在（x，y）点的函数值。

x0 = D(1); xf = D(2); y0 = D(3); yf = D(4);
dx = (xf - x0)/Mx; x = x0 + [0:Mx]*dx;      %构造内点数组
dy = (yf - y0)/My; y = y0 + [0:My]'*dy;
Mx1 = Mx + 1; My1 = My + 1;
%边界条件
for m = 1:My1
    u(m,[1 Mx1])=[bx0(y(m)) bxf(y(m))];      %左右边界
end
for n = 1:Mx1
    u([1 My1],n) = [by0(x(n)); byf(x(n))];  %上下边界
end
%边界平均值作迭代初值
sum_of_bv = sum(sum([u(2:My,[1 Mx1]) u([1 My1],2:Mx)']));
u(2:My,2:Mx) = sum_of_bv/(2*(Mx + My - 2));
for i = 1:My
    for j = 1:Mx
        F(i,j) = f(x(j),y(i)); G(i,j) = g(x(j),y(i));
    end
end
dx2 = dx*dx; dy2 = dy*dy; dxy2 = 2*(dx2 + dy2);
rx = dx2/dxy2; ry = dy2/dxy2; rxy = rx*dy2;
for itr = 1:MaxIter
    for j = 2:Mx
        for i = 2:My
            u(i,j) = ry*(u(i,j + 1)+u(i,j - 1)) + rx*(u(i + 1,j)+u(i - 1,j))+
                     rxy*(G(i,j)*u(i,j)- F(i,j));          %迭代公式
        end
    end
    if itr > 1 & max(max(abs(u - u0))) < MinErr  %循环结束条件
        break;
    end
    u0 = u;
end
```

3. 实例分析

下面给出求解 Helmholtz 方程以及其特例 possion 方程和 Laplace 方程的实例。

【例 9-1】 迭代法求解 Helmholtz 方程应用实例。求以下 Helmholtz 方程的数值解：

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}+\sqrt{x}u(x,y)=x^2+y^2$$

自变量取值：$0\leqslant x\leqslant 4$，$0\leqslant y\leqslant 4$

边界：$u(0,y)=y^2$，$u(4,y)=16\cos(y)$，$u(x,0)=x^2$，$u(x,4)=16\cos(x)$

解：可知 $f(x,y)=x^2+y^2$，$g(x,y)=\sqrt{x}$。

在 MATLAB 中编写函数 ex0901 来实现求解，代码如下。

```
function ex0901()
%调用函数 Helmholtz 求满足一定初始条件和边界条件的 Helmholtz 方程的数值解

f = inline('x^2+y^2','x','y');
g = inline('sqrt(x)','x','y');
x0 = 0; xf = 4; y0 = 0; yf = 4;          %自变量取值范围
Mx = 30;My = 30;                         %等分段数
bx0 = inline('y^2','y');                 %边界条件
bxf = inline('4^2*cos(y)','y');
by0 = inline('x^2','x');
byf = inline('4^2*cos(x)','x');
D = [x0 xf y0 yf]; MaxIter = 400; MinErr = 1e-4;
[U,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter);
clf, mesh(x,y,U);
xlabel('x')
ylabel('y')
zlabel('U')
xlabel('x')
ylabel('y')
zlabel('U')
```

运行结果如图 9-1 所示。

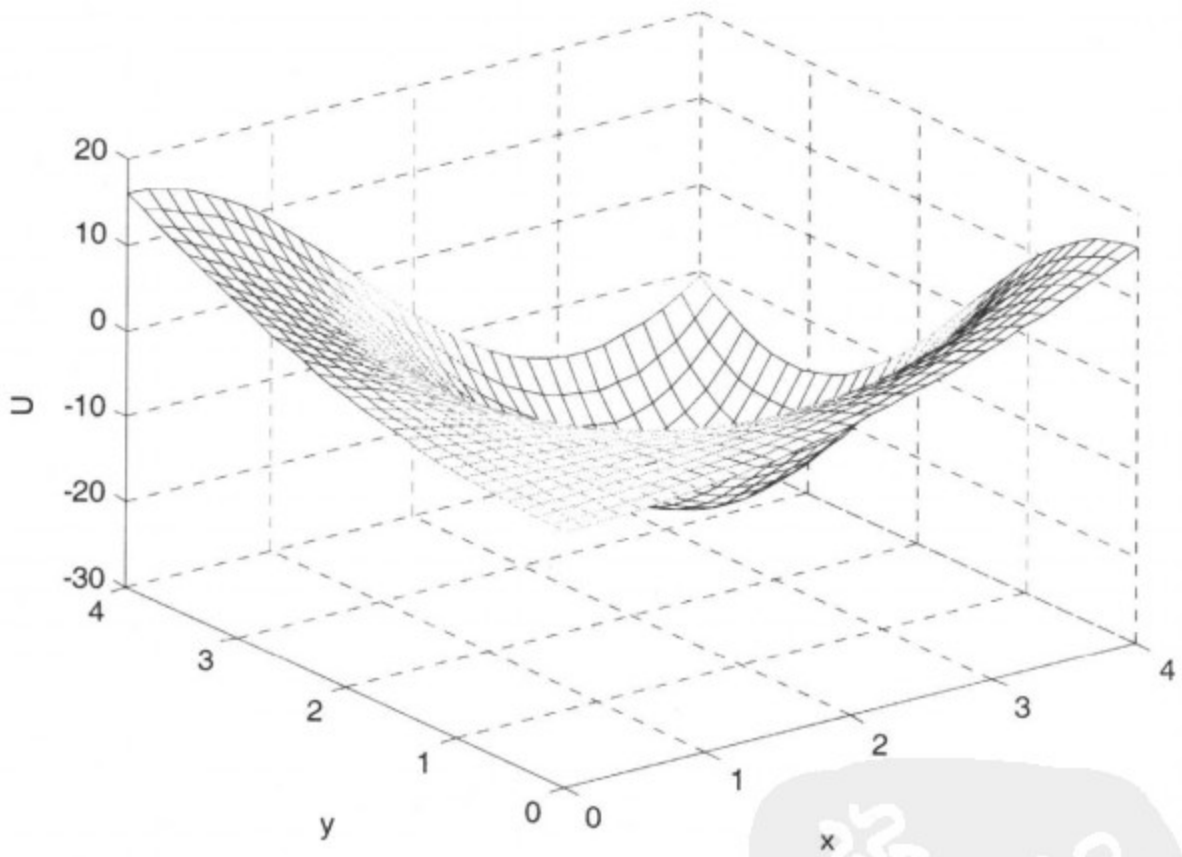

图 9-1 【例 9-1】的 Helmholtz 方程的数值解

若 $g(x,y)=0$，演变为 Possion 方程。此时，只需将程序 ex0901.m 中的代码“g=inline('sqrt(x)','x','y');”改为“g=inline('0','x','y');”即可。可得其数值解的图形如图 9-2 所示。

若二者均为 0，则演变为 Laplace 方程，即将“g=inline('sqrt(x)','x','y');”改为“g=inline('0','x','y');”，将“f=inline('x^2+y^2','x','y');”改为‘f=inline('0','x','y');，可得其数值解为如图 9-3 所示。

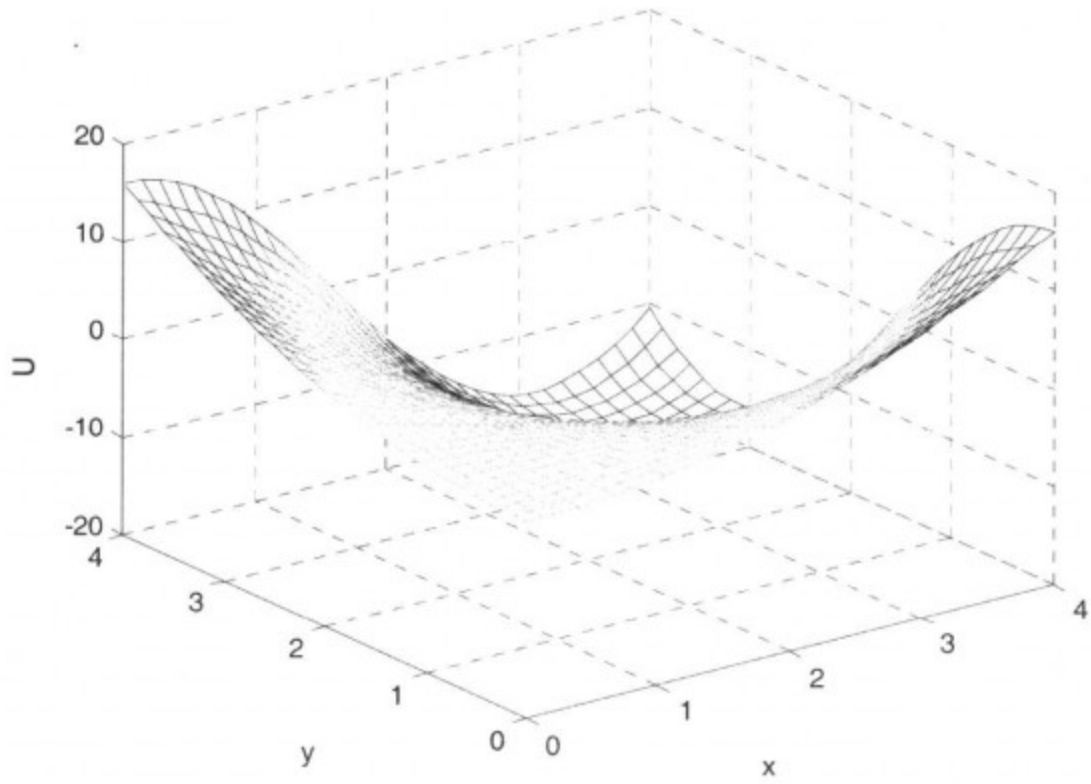

图 9-2 【例 9-1】的 Possion 方程的数值解

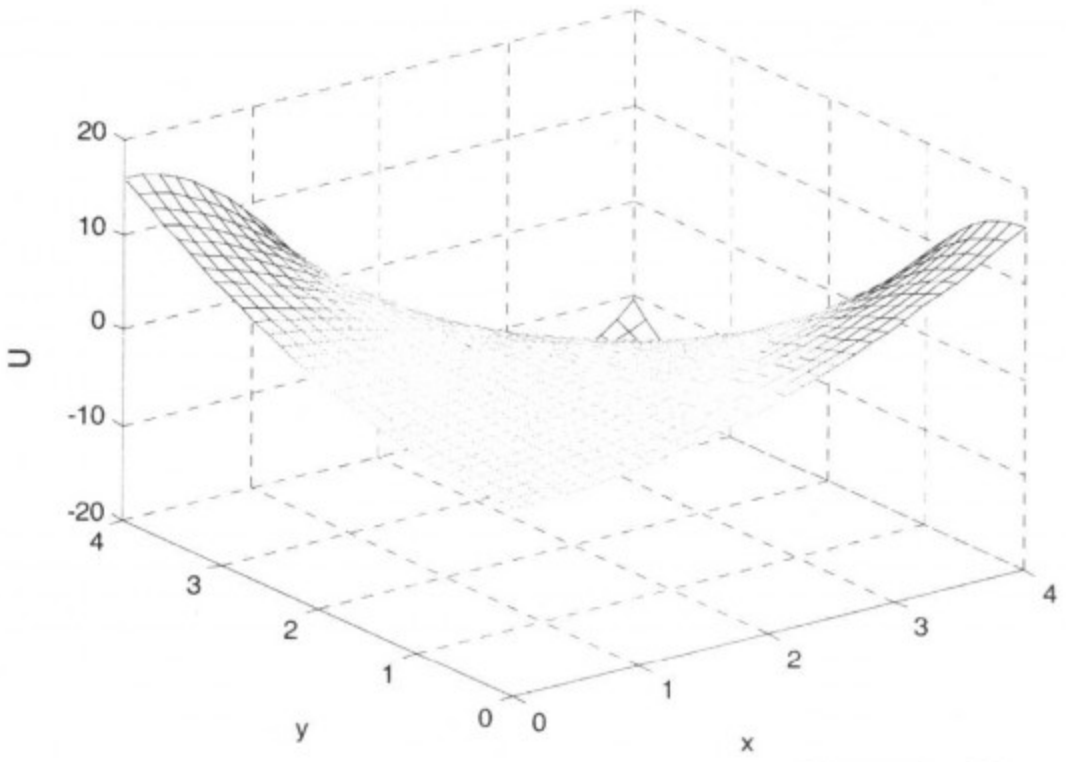

图 9-3 【例 9-1】的 Laplace 方程的数值解

9.2.2 满足牛顿边值条件的 Helmholtz 方程

接下来将讨论具有牛顿边值条件的 Helmholtz 方程，所谓牛顿边值条件是指：

$$\frac{\partial u(x,y)}{\partial x}|_{x=x_0} = b'_{x_0}(y)。$$

1. 求解原理描述

对于满足牛顿边值条件的 Helmholtz 方程，将左边界微分方程用差分方程估计，可以

得到：

$$\frac{u_{i,1}-u_{i,-1}}{2\Delta x}\approx b'_{x_0}(y_i),\ u_{i,-1}\approx u_{i,1}-2b'_{x_0}(y_i)\Delta x,\quad i=1,2\cdots M_y-1。$$

结合边界点，可以得到：

$$\begin{aligned}u_{i,0}&=r_y(u_{i,1}+u_{i,-1})+r_x(u_{i+1,0}+u_{i-1,0})+r_{xy}(g_{i,0}u_{i,0}-f_{i,0})\\&=r_y(u_{i,1}+u_{i,1}-2b'_{x_0}(y_i)\Delta x)+r_x(u_{i+1,0}+u_{i-1,0})+r_{xy}(g_{i,0}u_{i,0}-f_{i,0})\\&=2r_yu_{i,1}+r_x(u_{i+1,0}+u_{i-1,0})+r_{xy}(g_{i,0}u_{i,0}-f_{i,0}-2b'_{x_0}(y_i)/\Delta x)。\end{aligned}$$

类似地，如果在 $y=y_0$ 处也满足牛顿条件，可以得到：

$$u_{0,j}=r_y(u_{0,j+1}+u_{0,j-1})+2r_xu_{1,j}+r_{xy}(g_{0,j}u_{0,j}-f_{0,j}-2b'_{y_0}(x_i)/\Delta y),\quad i=1,2\cdots M_y-1$$

特别地，在边界点 (x_0,y_0) 处，可以得到：

$$u_{0,0}=2(r_yu_{0,1}+r_xu_{1,0})+r_{x,y}(g_{0,0}u_{0,0}-f_{0,0}-2(b'_{x_0}(y_0)/\Delta x+2(b'_{y_0}(x_0)/\Delta y))。$$

2. 求解程序 MATLAB 实现

在 MATLAB 中编程实现满足牛顿边值条件的 Helmholtz 方程求解程序为：Helmholtz。

功能：求解满足牛顿边值条件的 Helmholtz 方程。

调用格式：[*u*, *x*, *y*] = Helmholtz_Newton(*f*, *g*, *dbx*, *bx*0, *bxf*, *by*0, *byf*, ***D***, *Mx*, *My*, MinErr, MaxIter)。

其中，*f*，*g* 为函数名；

dbx 为牛顿边界函数；

*bx*0 为在边界 *x*=*x*0 上的函数值；

bxf 为在边界 *x*=*xf* 上的函数值；

*by*0 为在边界 *y*=*y*0 上的函数值；

byf 为在边界 *y*=*yf* 上的函数值；

D 为边界点值，长度为 4 的列向量，其元素分别为 *x*0，*xf*，*y*0，*yf*；

Mx 为沿 *x* 轴的区间数；

My 为沿 *y* 轴的区间数；

MinErr 为误差控制因子；

MaxIter 为最大迭代次数；

[*u*, *x*, *y*]为方程 *u*（*x*，*y*）在（*x*，*y*）点的函数值。

迭代法求解满足牛顿边值条件的 Helmholtz 方程的 MATLAB 程序代码如下：

```
function [u,x,y] = Helmholtz_Newton(f,g,dbx,bx0,bxf,by0,byf,D,Mx,My,MinErr,
                                    MaxIter)
%解方程： u_xx + u_yy + g(x,y)u = f(x,y)
%dbx：牛顿边界函数
%其他条件同 Helmholtz 函数

x0 = D(1); xf = D(2); y0 = D(3); yf = D(4);
dx = (xf - x0)/Mx; x = x0 + [0:Mx]*dx;          %构造内点数组
dy = (yf - y0)/My; y = y0 + [0:My]'*dy;
Mx1 = Mx + 1; My1 = My + 1;
for i = 1:My
```

```
    for j = 1:Mx
        F(i,j) = f(x(j),y(i)); G(i,j) = g(x(j),y(i));
    end
end
for i=1:My
    DBX(1,i)=dbx(x(1),y(i));
end
dx2 = dx*dx; dy2 = dy*dy; dxy2 = 2*(dx2 + dy2);
rx = dx2/dxy2; ry = dy2/dxy2; rxy = rx*dy2;
%边界条件
for m = 1:My1
    u(m,[1 Mx1])=[bx0(y(m)) bxf(y(m))];          %左右边界
end
for n = 1:Mx1
    u([1 My1],n) = [by0(x(n)); byf(x(n))];       %上下边界
end
%边界平均值作迭代初值
sum_of_bv = sum(sum([u(2:My,[1 Mx1]) u([1 My1],2:Mx)']));
u(2:My,2:Mx) = sum_of_bv/(2*(Mx + My - 2))
for itr = 1:MaxIter
    for m = 2:(My1-1)                            %x 轴牛顿边值条件迭代
        u(m,1)=2*ry*u(m,2)+rx*(u(m+1,1)+u(m-1,1))+rxy*(G(m,1)*u(m,1)-
                                                    F(m,1)- 2*dbx(x(1),y(m))/dx);
    end
    for m=2:(Mx1-1)                              %y 轴牛顿边值条件迭代
        u(1,m)=ry*(u(1,m+1)+u(1,m-1))+2*rx*u(2,m)+rxy*(G(1,m)*u(1,m)- F(1,m)
               -2*dbx(x(m),y(1))/dy);
    end
    for j = 2:Mx
        for i = 2:My
            u(i,j) = ry*(u(i,j + 1)+u(i,j - 1)) + rx*(u(i + 1,j)+u(i - 1,j))+
                     rxy*(G(i,j)*u(i,j)- F(i,j));        %迭代公式
        end
    end
    if itr > 1 & max(max(abs(u - u0))) < MinErr %循环结束条件
        break;
    end
    u0 = u;
end
bx0 = inline('y^2','y');                         %边界条件
bxf = inline('4^2*cos(y)','y');
by0 = inline('x^2','x');
byf = inline('4^2*cos(x)','x');
D = [x0 xf y0 yf]; MaxIter = 400; MinErr = 1e-4;
[U,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter);
clf, mesh(x,y,U)
```

3. 实例分析

【例 9-2】 迭代法求解满足牛顿边值条件的 Helmholtz 方程应用实例。求【例 9-1】中满足如下牛顿边值条件的数值解：

$$\frac{\partial u}{\partial x}\Big|_{x=x_0} = y^2, \frac{\partial u}{\partial y}\Big|_{y=x_0} = x^2$$

解：在 MATLAB 中编写函数 ex0902 来实现求解：

```
function ex0902()
%调用函数 Helmholtz_Newton 解满足一定边值条件的方程： u_xx + u_yy + g(x,y)u =
%f(x,y)

f = inline('x^2+y^2','x','y');          %构造函数 f(x,y)=(x^2+y^2)
g = inline('sqrt(x)','x','y');          %构造函数 f(x,y)=√x
x0 = 0; xf = 4; y0 = 0; yf = 4;         %自变量取值范围
Mx = 30;My = 30;                        %等分段数
dbx=inline('x^2+y^2','x','y');          %牛顿边值
bx0 = inline('y^2','y');                %边界条件
bxf = inline('4^2*cos(y)','y');
by0 = inline('x^2','x');
byf = inline('4^2*cos(x)','x');
D = [x0 xf y0 yf]; MaxIter = 100; MinErr = 1e-4;
[U,x,y] = Helmholtz_Newton(f,g,dbx,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter);
clf, mesh(x,y,U)
xlabel('x')
ylabel('y')
zlabel('U')
```

运行结果如图 9-4 所示。

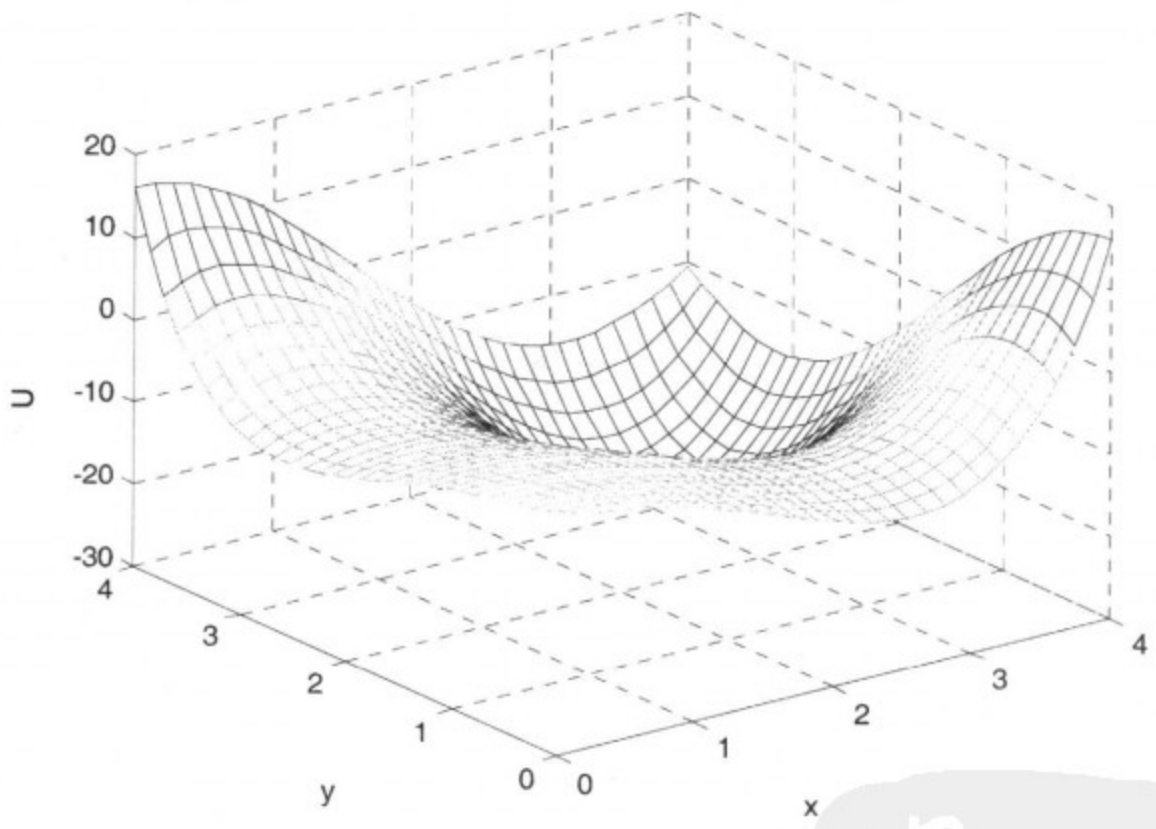

图 9-4　牛顿边值的 Helmholtz 方程数值解

通过对比图 9-4 与图 9-1 可以发现，两者的区别也主要体现在边界上，这是由于后者具有牛顿边值条件造成的。

9.3　抛物线偏微分方程

一维热传导方程就是一个典型的抛物线偏微分方程，它用下式的形式表示：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}$$

其中：

$$0\leqslant x\leqslant x_f, 0\leqslant t\leqslant T$$

此外，还须给出边界条件和初值：

$$u(0,t)=b_0(t),\quad u(x_f,t)=b_{xf}(t),\quad u(x,0)=u_0(x)$$

本节讨论的一维抛物线型方程的数值解法有显式前向欧拉法、隐式后向欧拉法、Grank-Nicholson 方法。在本节最后将介绍二维抛物线的求解。

9.3.1 显式前向欧拉法

下面讨论利用显式前向欧拉法解抛物线型偏微分方程。

1. 原理描述

对于抛物线偏微分，仍然可采用有限项的差分方法，因此对时间轴和 x 轴进行划分：将 $[0,x_f]$ 等分成 M 段，每段长度为 $\Delta x=x_f/M$；将时间轴等分为 N 段，每段长度为 $\Delta t=T/N$。

将抛物线偏微分方程

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}$$

的左边用中心差分估计，右边用前向差分估计，就可以得到下面的差分方程：

$$A\frac{u_{i+1}^k-2u_i^k+u_{i-1}^k}{\Delta x^2}=\frac{u_i^{k+1}-u_i^k}{\Delta t}$$

采用显式前向欧拉算法对上式进行迭代求解，即得到形如下式的变形：

$$u_i^{k+1}=r(u_{i+1}^k+u_{i-1}^k)+(1-2r)u_i^k$$

其中：

$$r=A\frac{\Delta t}{\Delta x^2},\quad k=1,2\cdots M-1$$

2. 求解程序 MATLAB 实现

在 MATLAB 中编程实现的显式前向欧拉法求解抛物线方程的程序为：EF_Euler。

功能：显式前向欧拉法求解一维抛物线方程。

调用格式：[*u*, *x*, *t*] = EF_Euler(*A*, *xf*, *T*, *it*0, *bx*0, *bxf*, *M*, *N*)。

其中，*A* 为抛物线偏微分方程系数；

xf 为 *x* 取值下限；

T 为 *t* 取值下限；

*it*0 为在边界 *t*=0 上的函数值；

*bx*0 为在边界 *x*=0 上的函数值；

bxf 为在边界 *x*=*xf* 上的函数值；

M 为沿 *x* 轴的等分段数；

N 为沿 *t* 轴的等分段数；

[*u*,*x*,*t*]:方程 *u*（*x*，*t*）在（*x*，*t*）点的函数值。

显示前向欧拉法求解抛物线方程的 MATLAB 程序代码如下：

```
function [u,x,t] = EF_Euler(A,xf,T,it0,bx0,bxf,M,N)
%解方程 A u_xx = u_t ,  0 <= x <= xf, 0 <= t <= T
%初值: u(x,0) = it0(x)
% 边界条件: u(0,t) = bx0(t), u(xf,t) = bxf(t)
% M :  x 轴的等分段数
% N :  t 轴的等分段数

dx = xf/M; x = [0:M]'*dx;
dt = T/N; t = [0:N]*dt;
for i = 1:M + 1
   u(i,1) = it0(x(i));
end
for n = 1:N + 1
   u([1 M + 1],n) = [bx0(t(n)); bxf(t(n))];
end
r = A*dt/dx/dx, r1 = 1 - 2*r;
if(r>0.5)
   disp('r>0.5,unstability');
end
for k = 1:N
   for i = 2:M
      u(i,k+1) = r*(u(i + 1,k) + u(i-1,k)) + r1*u(i,k); %(11.2.3)
   end
end
```

3. 实例分析

【例 9-3】 显式前向欧拉法求解一维抛物线方程应用实例。求满足下列条件的热传导方程的数值解：

$$A=0.5,0\leqslant x\leqslant 2,0\leqslant t\leqslant 0.1$$

解：热传导方程为典型的一维抛物线偏微分方程，如下所示：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}$$

代入未知参量，并考虑所给变量取值范围。

在 MATLAB 中编写函数 ex0903 来实现求解：

```
function ex0903()
%调用函数 EF_Euler 解热传导方程

A = 0.5;                                          %方程系数
it0 = inline('sin(pi*x)','x');                    %初始条件
bx0 = inline('0'); bxf = inline('0');             %边界条件
xf = 2; M = 50; T = 0.1; N = 100;
[u1,x,t] = EF_Euler(A,xf,T,it0,bx0,bxf,M,N);
figure(1),clf,mesh(t,x,u1)
xlabel('x')
```

```
ylabel('y')
zlabel('U')
title('r>0.5')
M=50;
[u1,x,t] = EF_Euler(A,xf,T,it0,bx0,bxf,M,N);
figure(2),clf,,mesh(t,x,u1)
    xlabel('x')
    ylabel('y')
    zlabel('U')
    title('r<0.5')
```

结果如图 9-5，图 9-6 所示。

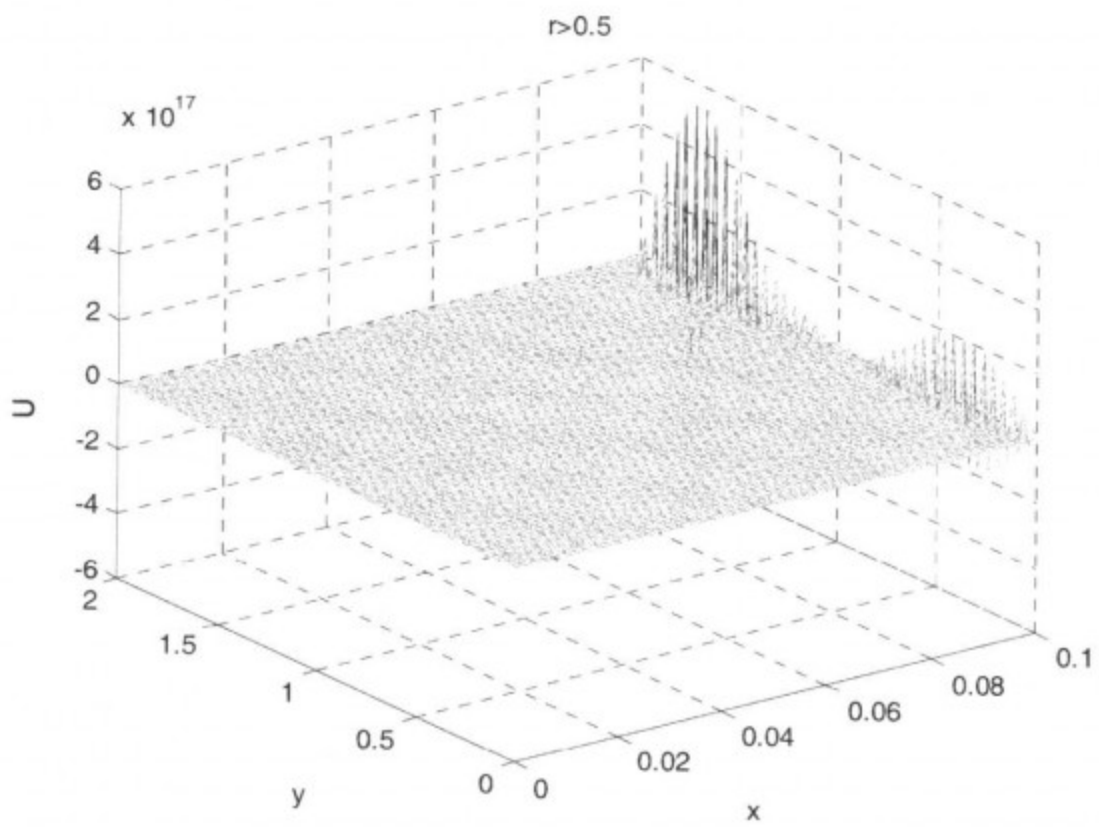

图 9-5　利用显式前向欧拉法得到的抛物线方程数值解：不稳定区域

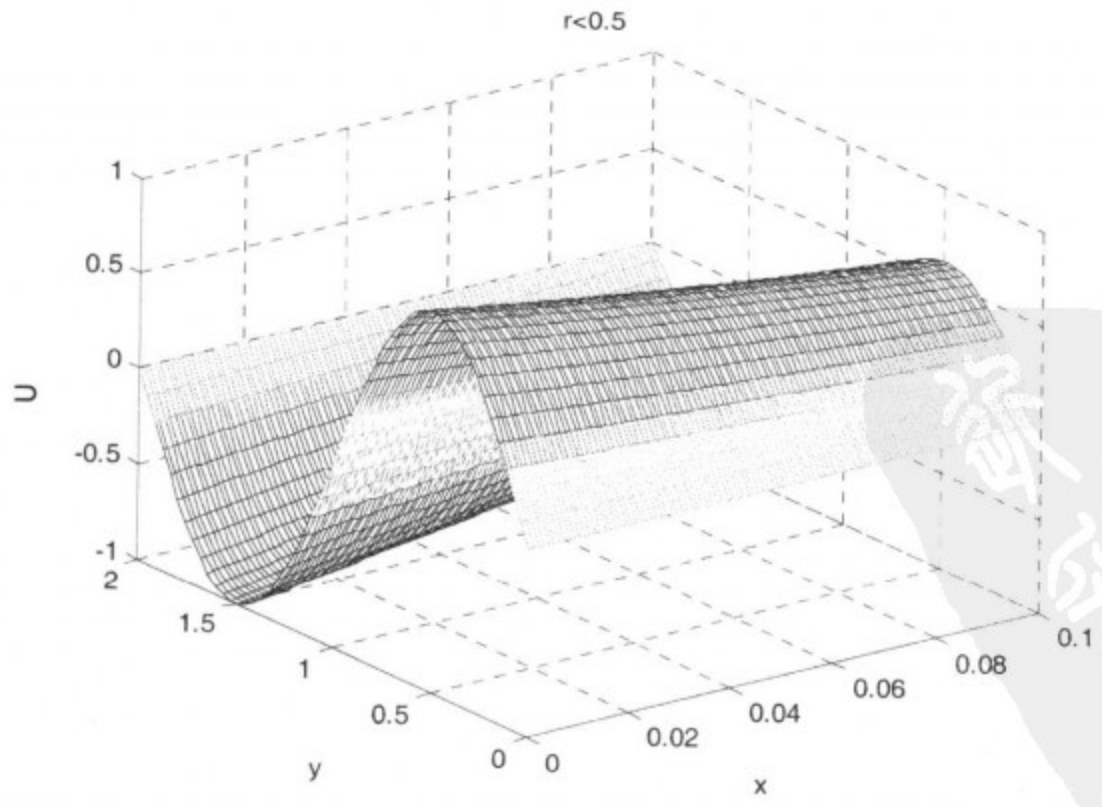

图 9-6　利用显式前向欧拉法得到的抛物线方程数值解：稳定区域

9.3.2　隐式后向欧拉法

下面讨论利用隐式后向欧拉法解抛物线型偏微分方程。

1．求解原理描述

采用后向差分估计式

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}$$

的右边，左边仍然由中心差分估计，可以得到：

$$A\frac{u_{i+1}^k-2u_i^k+u_{i-1}^k}{\Delta x^2}=\frac{u_i^k-u_i^{k-1}}{\Delta t}$$

写成迭代式：

$$-ru_{i-1}^k+(1+2r)u_i^k-ru_{i+1}^k=u_i^{k-1}$$

其中：

$$r=A\frac{\Delta t}{\Delta x^2},\quad i=1,2,...,M-1$$

若 u_0^k，u_M^k 的值满足 Dirichlet 型边界条件，则 $-ru_{i-1}^k+(1+2r)u_i^k-ru_{i+1}^k=u_i^{k-1}$ 式可写成矩阵形式：

$$\begin{bmatrix}1+2r & -r & 0 & & 0 & 0\\ -r & 1+2r & -r & & 0 & 0\\ 0 & -r & 1+2r & \cdots & 0 & 0\\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & 1+2r & -r\\ 0 & 0 & 0 & & 0 & 1+2r\end{bmatrix}\begin{bmatrix}u_1^k\\ u_2^k\\ u_3^k\\ \cdots\\ u_{M-2}^k\\ u_{M-1}^k\end{bmatrix}=\begin{bmatrix}u_1^k+ru_0^k\\ u_2^{k-1}\\ u_3^{k-1}\\ \cdots\\ u_{M-2}^{k-1}\\ u_{M-1}^{k-1}+ru_M^k\end{bmatrix}。$$

下面讨论牛顿边值情形，即满足：

$$\frac{\partial u}{\partial x}|_{x=0}=b_0'(t)$$

用差分估计微分，可以得到：

$$\frac{u_1^k-u_{-1}^k}{2\Delta x}=b_0'(k)$$

等式 $-ru_{i-1}^k+(1+2r)u_i^k-ru_{i+1}^k=u_i^{k-1}$ 中取 k=0，得到：

$$-ru_{-1}^k+(1+2r)u_0^k-ru_1^k=u_0^{k-1}$$

综上可得：

$$(1+2r)u_0^k-2ru_1^k=u_0^{k-1}-2rb_0'(k)\Delta x。$$

此时，上述矩阵式可以改写为：

$$\begin{bmatrix} 1+2r & -r & 0 & & 0 & 0 \\ -r & 1+2r & -r & & 0 & 0 \\ 0 & -r & 1+2r & \cdots & 0 & 0 \\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & 1+2r & -r \\ 0 & 0 & 0 & & 0 & 1+2r \end{bmatrix} \begin{bmatrix} u_0^k \\ u_1^k \\ u_2^k \\ \cdots \\ u_{M-2}^k \\ u_{M-1}^k \end{bmatrix} = \begin{bmatrix} u_0^{k-1} - 2rb_0'(k)\Delta x \\ u_1^{k-1} \\ u_2^{k-1} \\ \cdots \\ u_{M-2}^{k-1} \\ u_{M-1}^{k-1} + ru_M^k \end{bmatrix}。$$

上述两个矩阵式方程都是三对角矩阵，且对角占优保证了算法的收敛性，因此用后向隐式欧拉法解这类问题具有较高效率。

2. 求解程序的 MATLAB 实现

在 MATLAB 中编程实现的隐式后向欧拉法求解抛物线方程的程序为：IB_Euler。

功能：隐式后向欧拉法求解一维抛物线方程。

调用格式：[*u*, *x*, *t*] = IB_Euler(*A*, *xf*, *T*, *it*0, *bx*0, *bxf*, *M*, *N*)。

其中，*A* 为抛物线偏微分方程系数；

xf 为 *x* 取值下限；

*it*0 为在边界 $t=0$ 上的函数值；

*bx*0 为在边界 $x=0$ 上的函数值；

bxf 为在边界 $x=xf$ 上的函数值；

M 为沿 *x* 轴的等分段数；

N 为沿 *t* 轴的等分段数；

[*u*, *x*, *t*]为方程 *u*（*x*，*t*）在（*x*，*t*）点的函数值。

隐式后向欧拉法求解抛物线方程的 MATLAB 程序代码如下：

```
function [u,x,t] = IB_Euler(A,xf,T,it0,bx0,bxf,M,N)
%解方程 A1 u_xx = u_t ,  0 <= x <= xf, 0 <= t <= T
%初值: u(x,0) = it0(x)
% 边界条件: u(0,t) = bx0(t), u(xf,t) = bxf(t)
% M :  x 轴的等分段数
% N :  t 轴的等分段数

dx = xf/M; x = [0:M]'*dx;
dt = T/N; t = [0:N]*dt;
for i = 1:M + 1, u(i,1) = it0(x(i)); end
for n = 1:N + 1, u([1 M + 1],n) = [bx0(t(n)); bxf(t(n))]; end
r = A*dt/dx/dx; r2 = 1 + 2*r;
for i = 1:M - 1
    P(i,i) = r2;                          %构造三对角矩阵
    if i > 1
        P(i - 1,i) = -r; P(i,i - 1) = -r;
    end
end
for k = 2:N + 1
    b = [r*u(1,k); zeros(M - 3,1); r*u(M + 1,k)] + u(2:M,k - 1); %Eq.(11.2.9)
    u(2:M,k) = linsolve(P,b);
end
```

3. 实例分析

【例 9-4】 隐式后向欧拉法求解一维抛物线方程应用实例。利用隐式后向欧拉法求满足下列条件的热传导方程的数值解：

$$A=0.5,\quad 0\leqslant x\leqslant 2,\quad 0\leqslant t\leqslant 0.1$$

即利用隐式后向欧拉法求解【例 9-3】。

解：热传导方程为典型的一维抛物线偏微分方程，如下所示：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}$$

代入未知参量，并考虑所给变量取值范围。

在 MATLAB 中编写函数 ex0904 来实现求解：

```
function ex0904()
%调用 IB_Euler 解热传导方程
A= 0.5;              %方程系数
it0 = inline('sin(pi*x)','x');          %初始条件
bx0 = inline('0'); bxf = inline('0'); %边界条件
xf = 2; M = 80; T = 0.1; N = 100;
[u1,x,t] = IB_Euler(A,xf,T,it0,bx0,bxf,M,N);
mesh(t,x,u1)
xlabel('x')
ylabel('y')
zlabel('U')
```

运行结果如图 9-7 所示。

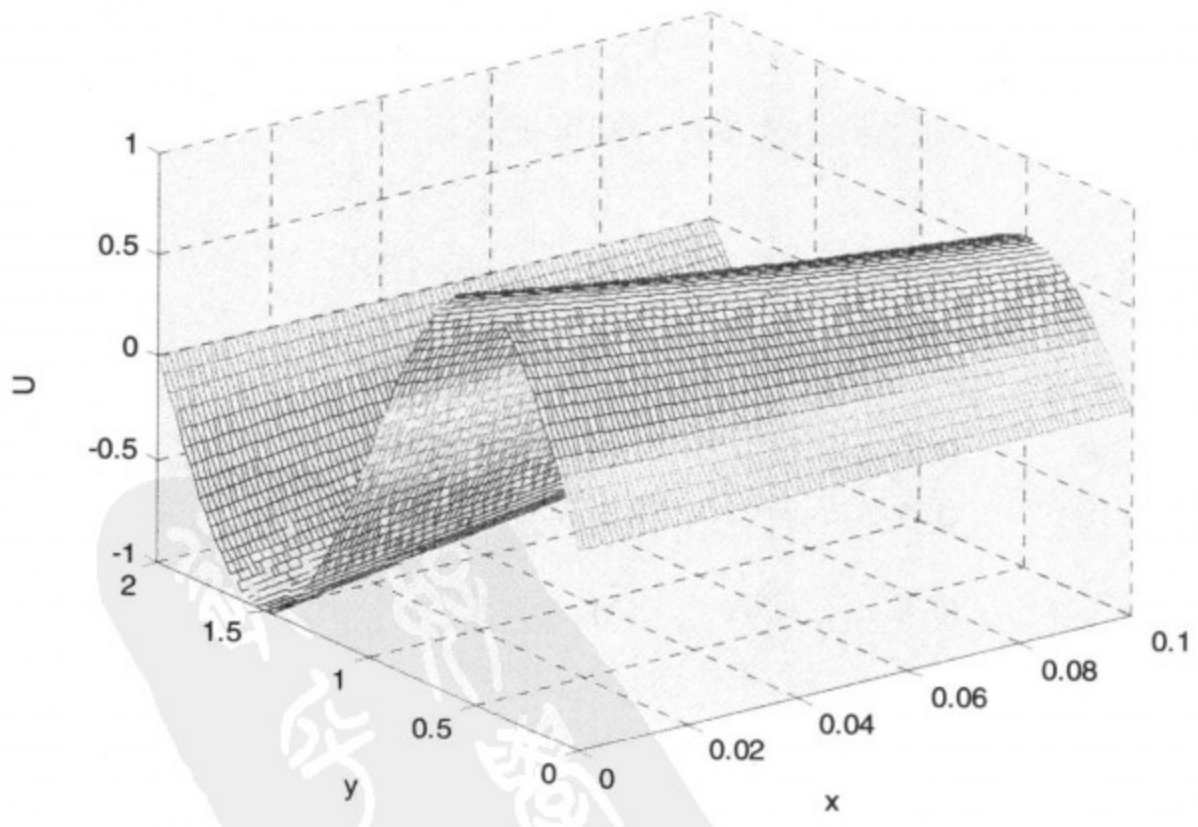

图 9-7　利用隐式后向欧拉法得到的抛物线方程数值解

从图 9-7 可以看出，无不稳定的 r 区间，因此与【例 9-3】相比，采用隐式后向欧拉法具有更好的鲁棒性。

9.3.3 Grank-Nicholson 方法

下面讨论利用 Grank-Nicholson 方法解抛物线型偏微分方程。

1. 原理描述

本节讨论对隐式后向欧拉法的一种改进。

对于等式：

$$A\frac{u_{i+1}^{k}-2u_{i}^{k}+u_{i-1}^{k}}{\Delta x^{2}}=\frac{u_{i}^{k}-u_{i}^{k-1}}{\Delta t}$$

其左边的差分估计在时间点 k 进行；如果把右边的差分估计看做是时间步长为 $\Delta t/2$ 的中心差分，则估计在点 k 和 k-1 的中点进行。等式左右的估计点不在同一个时刻。为了消除这个不一致性，需要将等式左右的估计放在同一个时间点进行。

在时间点 k 和 k+1 对等式两边进行差分估计，可以得到：

$$\frac{A}{2}\left(\frac{u_{i+1}^{k+1}-2u_{i}^{k+1}+u_{i-1}^{k+1}}{\Delta x^{2}}+\frac{u_{i+1}^{k}-2u_{i}^{k}+u_{i-1}^{k}}{\Delta x^{2}}\right)=\frac{u_{i}^{k+1}-u_{i}^{k}}{\Delta t}$$

对上式进行变形就得到 Grank-Nicholson 迭代方法：

$$-ru_{i+1}^{k+1}+2(1+r)u_{i}^{k+1}-ru_{i-1}^{k+1}=ru_{i+1}^{k}+2(1-r)u_{i}^{k}+ru_{i-1}^{k}$$

其中，

$$r=A\frac{\Delta t}{\Delta x^{2}}。$$

如果方程在 x_0/x_M 处具有牛顿边值条件，则可把迭代式写成三对角矩阵形式：

$$\begin{bmatrix}2+2r & -r & 0 & \cdots & 0 & 0\\ -r & 2+2r & -r & & 0 & 0\\ 0 & -r & 2+2r & & 0 & 0\\ & & & \cdots & & \\ 0 & 0 & 0 & & 2+2r & -r\\ 0 & 0 & 0 & & 0 & 2+2r\end{bmatrix}\begin{bmatrix}u_1^k\\ u_2^k\\ u_3^k\\ \cdots\\ u_{M-1}^k\\ u_M^k\end{bmatrix}=$$

$$\begin{bmatrix}2-2r & r & 0 & \cdots & 0 & 0\\ r & 2-2r & r & & 0 & 0\\ 0 & r & 2-2r & & 0 & 0\\ & & & \cdots & & \\ 0 & 0 & 0 & & 2-2r & r\\ 0 & 0 & 0 & & 0 & 2-2r\end{bmatrix}\begin{bmatrix}u_1^k\\ u_2^k\\ u_3^k\\ \cdots\\ u_{M-1}^k\\ u_M^k\end{bmatrix}+\begin{bmatrix}r(u_0^{k+1}-u_0^k)\\ 0\\ 0\\ \cdots\\ 0\\ 2r(b_M'(k+1)+b_M'(k))\end{bmatrix}$$

该算法具有高效率和衡稳定性，即对于任意 r，算法均收敛。

2. 求解程序的 MATLAB 实现

在 MATLAB 中编程实现的 Grank-Nicholson 法求解抛物线方程的程序为：Grank_Nicholson。

功能：Grank-Nicholson 法求解一维抛物线方程。

调用格式：[*u*, *x*, *t*] = Grank_Nicholson(*A*, *xf*, *T*, *it*0, *bx*0, *bxf*, *M*, *N*)。

其中，*A* 为抛物线偏微分方程系数；

xf 为 *x* 取值下限；

T 为 *t* 取值下限；

*it*0 为在边界 *t*=0 上的函数值；

*bx*0 为在边界 *x*=0 上的函数值；

bxf 为在边界 *x*=*xf* 上的函数值；

M 为沿 *x* 轴的等分段数；

N 为沿 *t* 轴的等分段数；

[*u*,*x*,*t*]:方程 *u*（*x*，*t*）在（*x*，*t*）点的函数值。

Grank-Nicholson 法求解抛物线方程的 MATLAB 程序代码如下：

```
function [u,x,t] = Grank_Nicholson(A,xf,T,it0,bx0,bxf,M,N)
%解方程 A u_xx = u_t ,  0 <= x <= xf, 0 <= t <= T
%初值: u(x,0) = it0(x)
% 边界条件: u(0,t) = bx0(t), u(xf,t) = bxf(t)
% M :  x 轴的等分段数
% N :  t 轴的等分段数
dx = xf/M; x = [0:M]'*dx;
dt = T/N; t = [0:N]*dt;
for i = 1:M + 1, u(i,1) = it0(x(i)); end
for n = 1:N + 1, u([1 M + 1],n) = [bx0(t(n)); bxf(t(n))]; end
r = A*dt/dx/dx
r1 = 2*(1 - r); r2 = 2*(1 + r);
for i = 1:M - 1
    P(i,i) = r1; %(11.2.17)
    Q(i,i)=r2;
    if i > 1
        P(i - 1,i) = -r; P(i,i - 1) = -r;
        Q(i - 1,i) = r; Q(i,i - 1) = r;
    end
end
for k = 2:N + 1
    b=Q*u(2:M,k-1)+[r*(u(1,k)+u(1,k-1));zeros(M-2,1)];
    u(2:M,k) = linsolve(P,b); %(11.2.17)
end
bx0 = inline('y^2','y');                    %边界条件
bxf = inline('4^2*cos(y)','y');
by0 = inline('x^2','x');
byf = inline('4^2*cos(x)','x');
D = [x0 xf y0 yf]; MaxIter = 400; MinErr = 1e-4;
[U,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter);
clf, mesh(x,y,U)%axis([0 4 0 4 -100 100]);
```

3. 实例分析

【例 9-5】 Grank-Nicholson 法求解一维抛物线方程应用实例。利用 Grank-Nicholson 法求满足下列条件的热传导方程的数值解：

$$A=0.5,\quad 0\leqslant x\leqslant 2,\quad 0\leqslant t\leqslant 0.1$$

即利用 Grank-Nicholson 法求解【例 9-3】。

解：热传导方程为典型的一维抛物线偏微分方程，如下所示：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial u(x,t)}{\partial t}。$$

代入未知参量，并考虑所给变量取值范围。在 MATLAB 中编写函数 ex0905 来实现求解：

```
function ex0905()
A= 0.5;                                          %方程系数
it0 = inline('sin(pi*x)','x');                   %初始条件
bx0 = inline('0'); bxf = inline('0');            %边界条件
xf = 2; M = 25; T = 0.1; N = 100;
[u,x,t] = Grank_Nicholson(A,xf,T,it0,bx0,bxf,M,N);
mesh(t,x,u)
xlabel('x')
ylabel('y')
zlabel('U')
```

结果如图 9-8 所示。

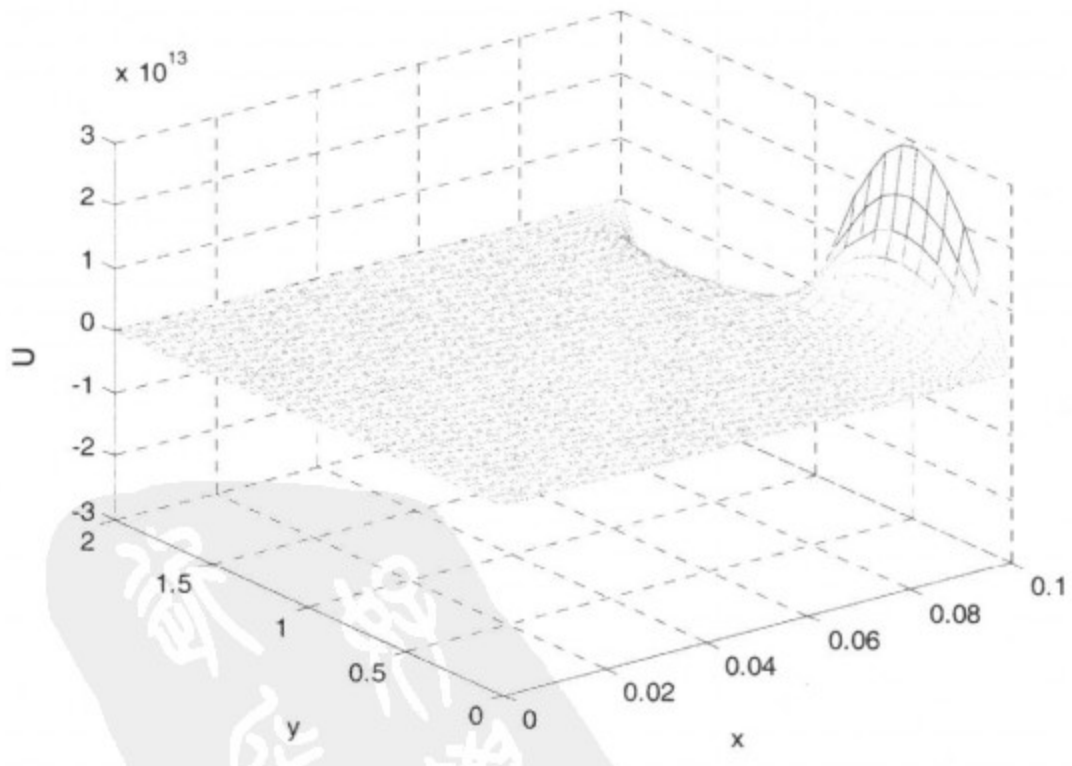

图 9-8　利用 Grank-Nicholson 得到的抛物线方程数值解

结论：在前 3 节中，用三种方法求解一维热传导方程。其中，显式前向欧拉法迭代式的算法思路清晰，容易编程实现，但存在不稳定区域。必须限制 $r<0.5$。隐式后向欧拉法和 Grank-Nicholson 方法利用三对角矩阵进行求解，无不稳定区域，但要求矩阵方程的解，耗

时更多，且程序可读性不如显示前向欧拉算法。

9.3.4 二维抛物线方程

形如下式的二维热传导方程为抛物线型偏微分方程，

$$A(\frac{\partial^2 u(x,y,t)}{\partial x^2}+\frac{\partial^2 u(x,y,t)}{\partial y^2})=\frac{\partial u(x,y,t)}{\partial t}$$

其中，

$$x_0 \leqslant x \leqslant x_f, y_0 \leqslant y \leqslant y_f, 0 \leqslant t \leqslant T$$

边界条件是，

$$u(x_0,y,t)=b_{x0}(y,t),\ u(x_f,y,t)=b_{xf}(y,t)$$
$$u(x,y_0,t)=b_{y0}(x,t),\ u(x,y_f,t)=b_{yf}(x,t),$$

初值为：

$$u(x,y,0)=i_0(x,y)。$$

1. 原理描述

与前面类似，用差分进行估计，可以得到：

$$A(\frac{u_{i,j+1}^{k+1}-2u_{i,j}^{k+1}+u_{i,j-1}^{k+1}}{\Delta x^2}+\frac{u_{i+1,j}^{k}-2u_{i,j}^{k}+u_{i-1,j}^{k}}{\Delta y^2})=\frac{u_{i,j}^{k+2}-u_{i,j}^{k+1}}{\Delta t}$$

变形为迭代式如下：

$$-r_y(u_{i-1,j}^{k+1}+u_{i+1,j}^{k+1})+(1+2r_y)u_{i,j}^{k+1}=r_x(u_{i,j-1}^{k}+u_{i,j+1}^{k})+(1-2r_x)u_{i,j}^{k}$$
$$-r_x(u_{i,j-1}^{k+2}+u_{i,j+1}^{k+2})+(1+2r_x)u_{i,j}^{k+2}=r_y(u_{i-1,j}^{k+1}+u_{i+1,j}^{k+1})+(1-2r_y)u_{i,j}^{k+1}$$

其中：

$$r_x=\frac{A\Delta t}{\Delta x^2},\ r_y=\frac{A\Delta t}{\Delta y^2}$$

$$\Delta x=\frac{x_f-x_0}{M_x},\ \Delta y=\frac{y_f-y_0}{M_y},\Delta t=\frac{T}{N}。$$

2. 求解程序的 MATLAB 实现

在 MATLAB 中编程实现的求解二维抛物线方程的程序为：TDE。

功能：求解二维抛物线方程。

调用格式：[*u*, *x*, *y*, *t*] = TDE(*A*, *D*, *T*, *ixy*0, *bxyt*, *bxf*, *Mx*, *My*, *N*)。

其中，*A* 为抛物线偏微分方程系数；

D 为包含函数分别在 *x*，*y* 轴上的边界值；

T 为 *t* 取值下限；

*ixy*0 为在边界 *t*=0 上的函数值；

bxyt 为边界取值函数；

Mx 为沿 *x* 轴的等分段数；

My 为沿 *y* 轴的等分段数；

N 为沿 *t* 轴的等分段数；

[*u*,*x*,*y*,*t*]为方程 *u*（*x*,*y*,*t*）在（*x*,*y*,*t*）点的函数值。

求解二维抛物线方程的 MATLAB 程序代码如下：

```
function [u,x,y,t] = TDE(A,D,T,ixy0,bxyt,Mx,My,N)
%解方程 u_t = c(u_xx + u_yy) for D(1) <= x <= D(2), D(3) <= y <= D(4), 0 <= t<= T
%初值: u(x,y,0) = ixy0(x,y)
% 边界条件: u(x,y,t) = bxyt(x,y,t) for (x,y)cB
% Mx/My :  x 轴和 y 轴的等分段数
% N :  t 轴的等分段数
dx = (D(2) - D(1))/Mx; x = D(1)+[0:Mx]*dx;
dy = (D(4) - D(3))/My; y = D(3)+[0:My]'*dy;
dt = T/N; t = [0:N]*dt;
%初始化 u
for j = 1:Mx + 1
    for i = 1:My + 1
        u(i,j) = ixy0(x(j),y(i));
    end
end
rx = A*dt/(dx*dx)
rx1 = 1 + 2*rx; rx2 = 1 - 2*rx;
ry = A*dt/(dy*dy)
ry1 = 1 + 2*ry; ry2 = 1 - 2*ry;
for j = 1:Mx - 1
    P(j,j) = ry1;
    if j > 1
        P(j - 1,j) = -ry; P(j,j-1) = -ry;
    end
end
for i = 1:My - 1 %(11.2.21b)
    Q(i,i) = rx1;
    if i > 1
        Q(i - 1,i) = -rx; Q(i,i - 1) = -rx;
    end
end
for k = 1:N
    u_1 = u; t = k*dt;
        for i = 1:My + 1                          %边界条件
            u(i,1) = feval(bxyt,x(1),y(i),t);
            u(i,Mx+1) = feval(bxyt,x(Mx+1),y(i),t);
        end
        for j = 1:Mx + 1
            u(1,j) = feval(bxyt,x(j),y(1),t);
            u(My+1,j) = feval(bxyt,x(j),y(My + 1),t);
        end
        if mod(k,2) == 0
            for i = 2:My
                jj = 2:Mx;
                bx = [ry*u(i,1) zeros(1,Mx - 3) ry*u(i,My + 1)] +rx*(u_1(i-1,jj)+ u_1
                     (i + 1,jj)) + rx2*u_1(i,jj);
                u(i,jj) = linsolve(P,bx')';
            end
```

```
        else
            for j = 2:Mx
                ii = 2:My;
                by = [rx*u(1,j); zeros(My-3,1); rx*u(Mx + 1,j)] + ry*(u_1(ii,j-1) +
                     u_1(ii,j + 1)) + ry2*u_1(ii,j);
                u(ii,j) = linsolve(Q,by);
            end
        end
    end
```

3. 实例分析

【例 9-6】 二维抛物线方程求解应用实例。求满足下列条件的二维热传导方程的解：

$$A=10^{-3},\quad x_0\leqslant x\leqslant 5,\quad y_0\leqslant y\leqslant 5,\quad 0\leqslant t\leqslant 1000$$

边界条件为：

$$b(x,y,t)=x^2\sin y-y^2\sin x$$

初值为：

$$u(x,y,0)=0\text{。}$$

解：在 MATLAB 中编写函数 ex0906 来实现求解：

```
function ex0906()
A = 1e-3;
it0 = inline('0','x','y');          %(E9.3.2a)
bxyt = inline('x^2*sin(y)-y^2*sin(x)','x','y','t'); %(E9.3.2b)
D = [0 5 0 5]; T = 1000; Mx = 50; My = 50; N = 50;
[u,x,y,t] = TDE(A,D,T,it0,bxyt,Mx,My,N);
mesh(x,y,u)
xlabel('x')
ylabel('y')
zlabel('U')
```

运行结果见图 9-9。

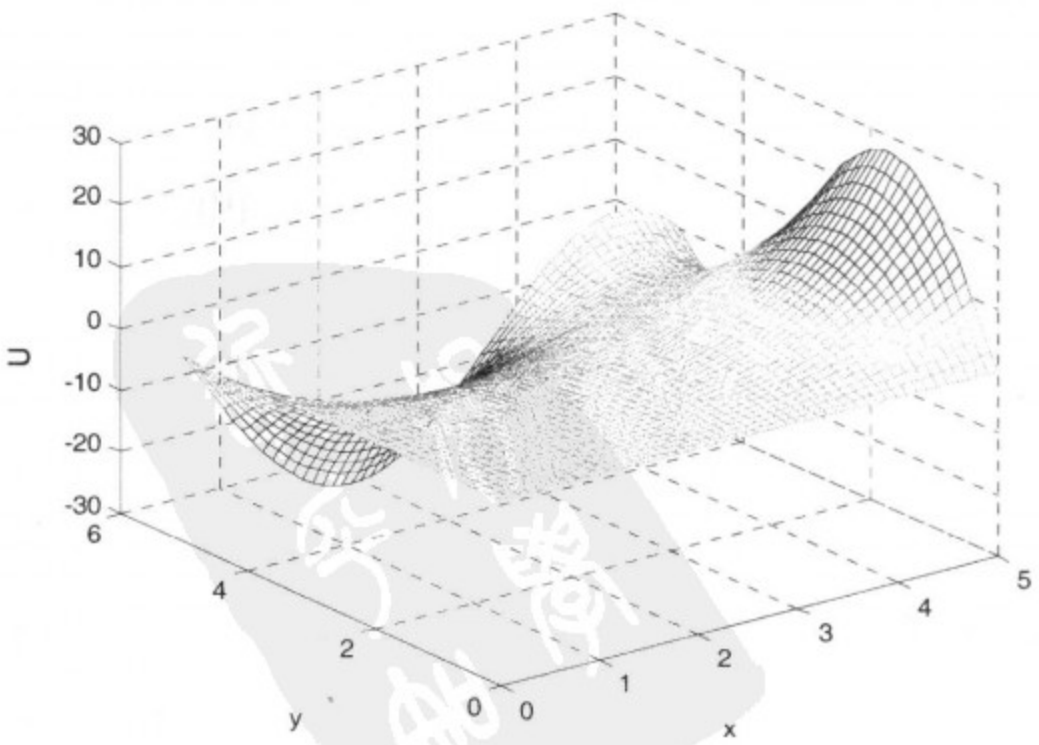

图 9-9　二维抛物线方程的数值解

9.4 双曲线偏微分方程

形如下式的一维波动方程就是一个典型的双曲线方程：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial^2 u(x,t)}{\partial t^2}$$

其中：

$$0\leqslant x\leqslant x_f,\quad 0\leqslant t\leqslant T$$

边界条件为：

$$u(0,t)=b_0(t),\ u(x_f,t)=b_{xf}(t)$$

初值为：

$$u(x,0)=i_0(x),\ \frac{\partial u}{\partial t}(x,0)=i_0'(x)$$

9.4.1 显式中心差分法

1. 原理描述

与抛物线方程的解法类似，在上述双曲线方程的两端均用三点中心差分估计，得到：

$$A\frac{u_{i+1}^k-2u_i^k+u_{i-1}^k}{\Delta x^2}=\frac{u_{i+1}^k-2u_i^k+u_{i-1}^k}{\Delta t^2}$$

改写为迭代式：

$$u_i^{k+1}=r(u_{i+1}^k+u_{i-1}^k)+2(1-r)u_i^k-u_i^{k-1}$$

其中：

$$r=A\frac{\Delta t^2}{\Delta x^2}$$

迭代初值为：

$$u_i^1=\frac{1}{2}r(u_{i+1}^0+u_{i-1}^0)+(1-r)u_i^0-i_0'(x_i)\Delta t$$

另，稳定性要求 $r\leqslant 1$，而算法的精度随着 r 的增大而提高，因此，选择 $r=1$。

2. 求解程序 MATLAB 实现

在 MATLAB 中编程实现的显示中心差分法求解双曲线偏微分方程的程序为：ECD_Wave。

功能：显式中心差分法求解一维抛物线方程。

调用格式：[*u*, *x*, *t*] = ECD_Wave(*A*, *xf*, *T*, *it*0, *i1t*0, *bx*0, *bxf*, *M*, *N*)。

其中，*A* 为双曲线偏微分方程系数；

xf 为 *x* 取值下限；

T 为 *t* 取值下限；

*it*0 为在边界 *t*=0 上的函数值；

i1t0 为在边界 t=0 处的函数导数值；

bx0 为在边界 x=0 上的函数值；

bxf 为在边界 x=*xf* 上的函数值；

M 为沿 x 轴的等分段数；

N 为沿 t 轴的等分段数；

[*u*,*x*,*t*]为方程 u（x，t）在（x，t）点的函数值。

下面给出程序实现。

```
function [u,x,t] = ECD_Wave(A,xf,T,it0,i1t0,bx0,bxf,M,N)
%解方程 a u_xx = u_tt for 0<=x<=xf, 0<=t<=T
% 初始条件: u(x,0) = it0(x), u_t(x,0) = i1t0(x)
% 边界条件: u(0,t)= bx0(t), u(xf,t) = bxf(t)
% M : 沿 x 轴的等分段数
% N : 沿 y 轴的等分段数
dx = xf/M; x = [0:M]'*dx;
dt = T/N; t = [0:N]*dt;
for i = 1:M + 1
    u(i,1) = it0(x(i));
end
for k = 1:N + 1
    u([1 M + 1],k) = [bx0(t(k)); bxf(t(k))];
end
r = A*(dt/dx)^ 2; r1 = r/2; r2 = 2*(1 - r);
u(2:M,2) = r1*u(1:M - 1,1) + (1 - r)*u(2:M,1) + r1*u(3:M + 1,1) +
           dt*i1t0(x(2:M)); %(11.3.4)
for k = 3:N + 1
    u(2:M,k) = r*u(1:M - 1,k - 1) + r2*u(2:M,k-1) + r*u(3:M + 1,k - 1)-
               u(2:M,k - 2); %(11.3.3)
end
```

3. 实例分析

【例 9-7】 显式中心差分法求解一维波动方程应用实例。求波动方程的解如下：

$$A\frac{\partial^2 u(x,t)}{\partial x^2}=\frac{\partial^2 u(x,t)}{\partial t^2}$$

其中，

$$0\leqslant x\leqslant 1,\quad 0\leqslant t\leqslant 2$$

$$u(x,0)=\sin x-x^2,\ \frac{\partial u}{\partial t}(x,0)=0$$

$$u(0,t)=0,\ u(1,t)=0$$

解：在 MATLAB 中编写函数 ex0907 来实现求解，具体代码如下。

```
function ex0907()
A= 1;
it0 = inline('x-x^2','x'); i1t0 = inline('0');
bx0t = inline('0'); bxft = inline('0');
xf =1; M = 10; T = 2; N = 50;
[u,x,t] = ECD_Wave(A,xf,T,it0,i1t0,bx0t,bxft,M,N);
```

```
figure(1), clf
mesh(t,x,u)                                    %作出三维图
xlabel('x')
ylabel('y')
zlabel('U')
figure(2), clf
%作出二维直观图
for n = 1:8
    subplot(2,4,n);
    plot(x,u(:,4*n)),axis([0 xf -0.3 0.3])
end
```

运行结果如图 9-10 和图 9-11 所示。

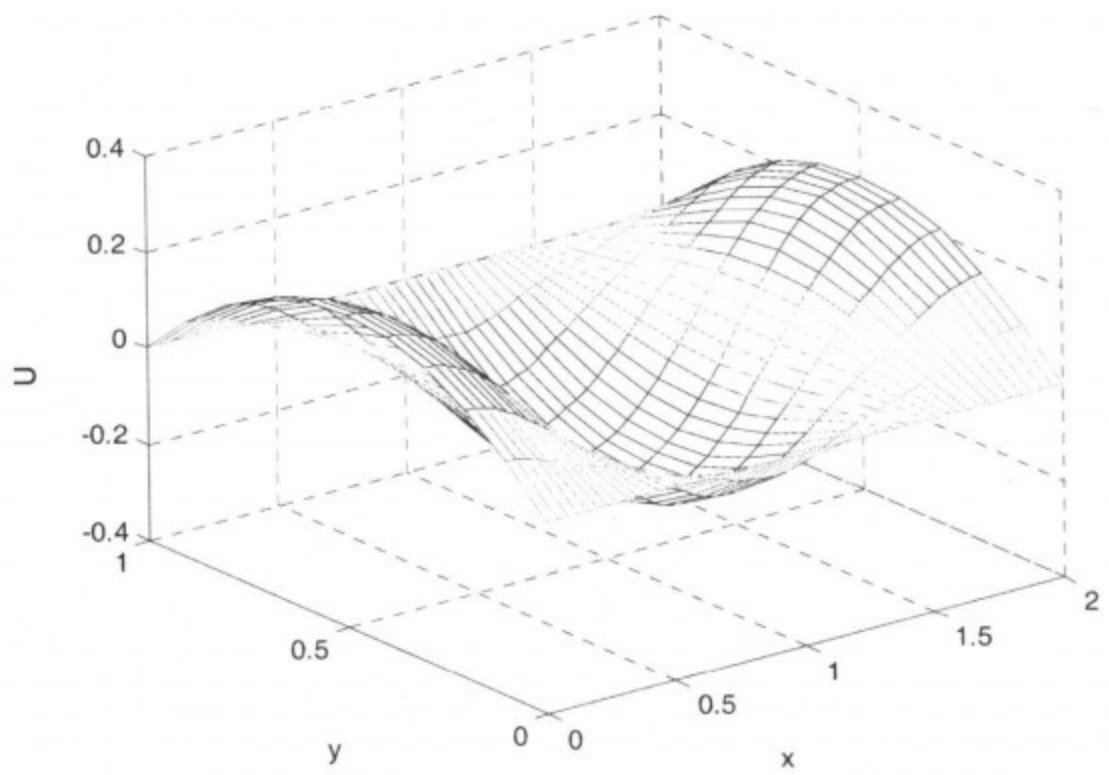

图 9-10　一维波动方程的数值解

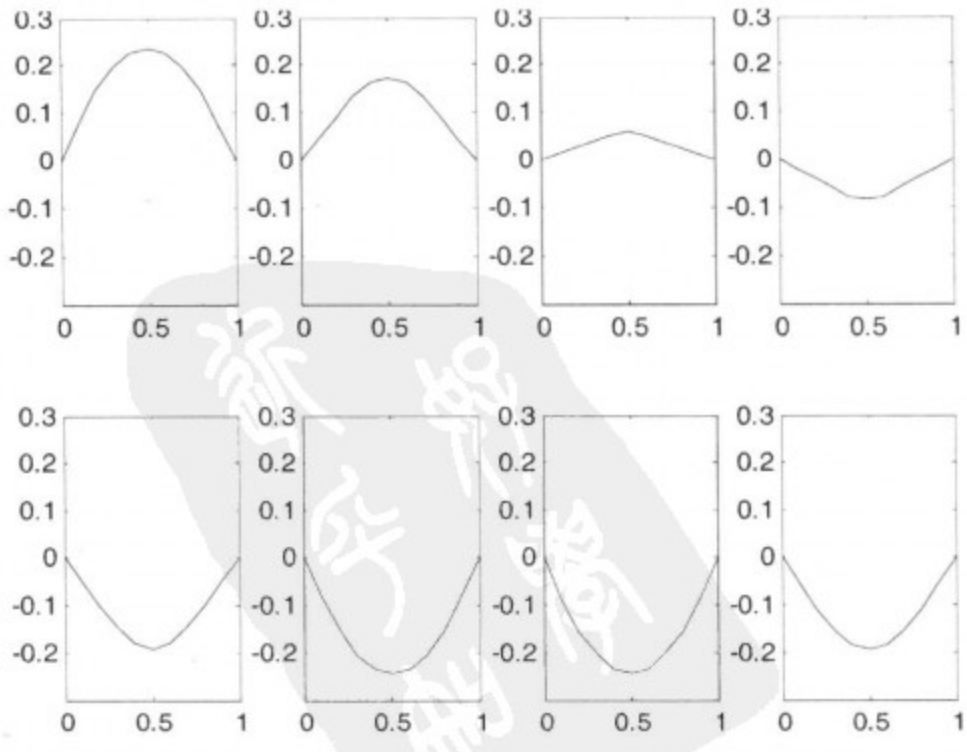

图 9-11　一维波动方程的二维直观图

图 9-11 所示是每隔 4 个单位所作的波动方程的二维直观图。当这个间隔取得更小时，读者可以观察波动曲线近似连续的变化。

9.4.2 二维双曲线型方程

本节讨论一个二维波动方程，形如：

$$A(\frac{\partial^2 u(x,y,t)}{\partial x^2}+\frac{\partial^2 u(x,y,t)}{\partial y^2})=\frac{\partial u(x,y,t)}{\partial t^2}$$

其中：

$$x_0 \leqslant x \leqslant x_f, \quad y_0 \leqslant y \leqslant y_f, \quad 0 \leqslant t \leqslant T$$

边界条件为：

$$u(x_0,y,t)=b_{x0}(y,t),\ u(x_f,y,t)=b_{xf}(y,t)$$
$$u(x,y_0,t)=b_{y0}(x,t),\ u(x,y_f,t)=b_{yf}(x,t),$$

初值为：

$$u(x,y,0)=i_0(x,y),\ \frac{\partial u}{\partial t}(x,y,0)=i_0'(x,y)。$$

1. 原理描述

与前面类似，用差分进行估计，可以得到：

$$A(\frac{u_{i,j+1}^k-2u_{i,j}^k+u_{i,j-1}^k}{\Delta x^2}+\frac{u_{i+1,j}^k-2u_{i,j}^k+u_{i-1,j}^k}{\Delta y^2})=\frac{u_{i,j}^{k+1}-2u_{i,j}^k+u_{i,j}^{k-1}}{\Delta t^2}$$

变形为迭代式：

$$u_{i,j}^{k+1}=r_x(u_{i,j+1}^k+u_{i,j-1}^k)+2(1-r_x-r_y)u_{i,j}^k+r_y(u_{i+1,j}^k+u_{i-1,j}^k)-u_{i,j}^{k-1}$$

其中：

$$r_x=\frac{A\Delta t^2}{\Delta x^2},\ r_y=\frac{A\Delta t^2}{\Delta y^2}$$

$$\Delta x=\frac{x_f-x_0}{M_x},\ \Delta y=\frac{y_f-y_0}{M_y},\ \Delta t=\frac{T}{N}$$

迭代初值为：

$$u_{i,j}^1=\frac{1}{2}\{r_x(u_{i,j+1}^0+u_{i,j-1}^0)+r_y(u_{i+1,j}^0+u_{i-1,j}^0)\}+2(1-r_x-r_y)u_{i,j}^0+i_0'(x_j,x_i)\Delta t$$

2. 求解程序的 MATLAB 实现

下面给出程序实现。

在 MATLAB 中编程实现的求解二维双曲线偏微分方程的程序为：Wave2。

功能：显式中心差分法求解二维双曲线方程。

调用格式：[*u*, *x*, *t*] = Wave2(*A*, *D*, *T*, *it*0, *i1t*0, *bxyt*, *Mx*, *My*, *N*)。

其中，*A* 为双曲线偏微分方程系数；

D 为 *x*，*y* 取值二维区域；

T 为 *t* 取值下限；

*it*0 为在边界 *t*=0 上的函数值；

*i1t*0 为在边界 *t*=0 处的函数导数值；

bxyt 为在边界取值函数；

Mx 为沿 *x* 轴的等分段数；

My 为沿 *y* 轴的等分段数；

N 为沿 *t* 轴的等分段数；

[*u*, *x*, *t*]为方程 *u*（*x*，*t*）在（*x*，*t*）点的函数值。

具体程序代码如下所示。

```
function [u,x,y,t] = Wave2(A,D,T,it0,i1t0,bxyt,Mx,My,N)
%解方程 a(u_xx + u_yy) = u_tt for D(1) <= x <= D(2), D(3) <= y <= D(4), 0 <= t <= T
% 初始条件: u(x,y,0) = it0(x,y), u_t(x,y,0) = i1t0(x,y)
% 边界条件: u(x,y,t) = bxyt(x,y,t) for (x,y) on Boundary
% Mx/My  x 轴和 y 轴的等分段数
% N = 时间轴的等分段数
dx = (D(2)-D(1))/Mx; x = D(1)+[0:Mx]*dx;
dy = (D(4)- D(3))/My; y = D(3)+[0:My]'*dy;
dt = T/N; t = [0:N]*dt;
%初始化
u = zeros(My+1,Mx + 1); ut = zeros(My + 1,Mx + 1);
for j = 2:Mx
    for i = 2:My
        u(i,j) = it0(x(j),y(i)); ut(i,j) = i1t0(x(j),y(i));
    end
end
adt2 = A*dt*dt; rx = adt2/(dx*dx); ry = adt2/(dy*dy);
rxy1 = 1- rx - ry; rxy2 = rxy1*2;
u_1 = u;
for k = 0:N
    t = k*dt;
    for i = 1:My + 1                    %边界条件
        u(i,[1 Mx + 1]) = [bxyt(x(1),y(i),t) bxyt(x(Mx + 1),y(i),t)];
    end
    for j = 1:Mx + 1
        u([1 My + 1],j) = [bxyt(x(j),y(1),t); bxyt(x(j),y(My + 1),t)];
    end
    if k == 0
        for i = 2:My
            for j = 2:Mx %(11.3.8)
                u(i,j) = 0.5*(rx*(u_1(i,j - 1) + u_1(i,j + 1))...
                    + ry*(u_1(i - 1,j)+u_1(i + 1,j))) + rxy1*u(i,j) + dt*ut(i,j);
            end
        end
    else
        for i = 2:My
            for j = 2:Mx                %11.3.7
                u(i,j) = rx*(u_1(i,j - 1)+ u_1(i,j + 1))+ ry*(u_1(i - 1,j) +
                        u_1(i + 1,j)) + rxy2*u(i,j) -u_2(i,j);
```

```
            end
        end
    end
    u_2 = u_1; u_1 = u;
end
```

3. 实例分析

【例 9-8】 显式中心差分法求解二维波动方程应用实例。解下列二维波动方程：

$$0.5\left(\frac{\partial^2 u(x,y,t)}{\partial x^2}+\frac{\partial^2 u(x,y,t)}{\partial y^2}\right)=\frac{\partial u(x,y,t)}{\partial t^2}$$

其中，

$$0\leqslant x\leqslant 2,\quad 0\leqslant y\leqslant 2,\quad 0\leqslant t\leqslant 2$$

$$u(0,y,t)=0, u(2,y,t)=0$$

$$u(x,0,t)=0, u(x,y_f,t)=0$$

$$u(x,y,0)=\cos(\pi x)\sin(\pi y), \frac{\partial u}{\partial t}(x,y,0)=0$$

解： 在 MATLAB 中编写函数 ex0908 来实现求解，具体代码如下所示。

```
function ex0908()
it0 = inline('cos(pi*x)*sin(pi*y)','x','y');
i1t0 = inline('0','x','y'); bxyt = inline('0','x','y','t');
A= 0.5; D = [0 2 0 2]; T = 2; Mx = 40; My = 40; N = 40;
[u,x,y,t] = Wave2(A,D,T,it0,i1t0,bxyt,Mx,My,N);
mesh(x,y,u)
xlabel('x')
ylabel('y')
zlabel('U')
```

运行结果如图 9-12 所示。

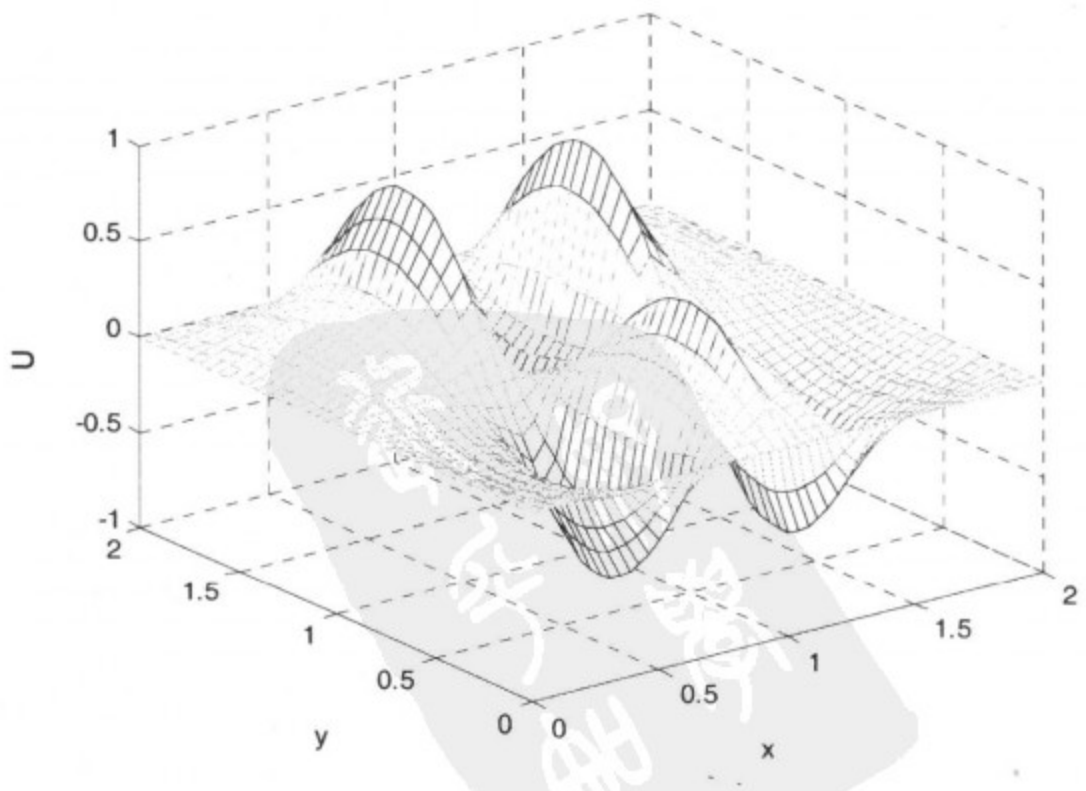

图 9-12　二维波动方程的数值解

9.5 有限元法

有限元法是求解偏微分方程数值解的另一种方法。它既能处理形如[X,Y,Z]的规则边界问题，也能处理不规则边界问题。其应用范围较广。

1. 原理描述

对于下式所示的偏微分方程：

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}+g(x,y)u(x,y)=f(x,y)$$

其中，区域 D 上的边界可以描述为：

$$u(x,y)=b(x,y)$$

利用有限元求解的步骤如下所示。

（1）将二维区域 D 划分为 N_s 个区域，$\{S_1,S_2\cdots S_{N_s}\}$，不要求每个区域均相等；

（2）在边界上确定 N_n 个节点，由外而内地进行编号 $\{N_1,N_2\cdots N_b\}$，$\{N_{b+1},N_{b+2}\cdots N_n\}$；

（3）定义基函数和插值函数：

$$\phi_n(x,y)=\{\phi_{n,s},s=1,2,\cdots,N_s\}\forall(x,y)\in D$$

$$\phi_{n,s}(x,y)=p_{n,s}(1)+p_{n,s}(2)x+p_{n,s}(3)y$$

这样的定义可以保证 ϕ_n 仅在 n 节点取值为 1，在其他节点取值均为 0。

偏微分方程的解可以由 ϕ_n 线性表示：

$$u(x,y)=c^T\varphi(x,y)=\sum_{n=1}^{N_n}c_n\phi_n=\sum_{n=1}^{N_b}c_n\phi_n+\sum_{n=N_b+1}^{N_n}c_n\phi_n=\boldsymbol{c_1^T}\varphi_1+\boldsymbol{c_2^T}\varphi_2$$

其中：

$$\varphi_1=[\phi_1,\phi_2\cdots\phi_{N_b}]^T,\mathbf{c}_1=[c_1,c_2\cdots c_{N_b}]^T$$

$$\varphi_2=[\phi_{N_b+1},\phi_{N_b+2}\cdots\phi_{N_n}]^T,\mathbf{c}_2=[c_{N_b+1},c_{N_b+2}\cdots c_{N_n}]^T$$

对每一个子区域，该等式可以写为：

$$\phi_s(x,y)=\sum_{n=1}^{N_n}c_n\phi_{n,s}(x,y)=\sum_{n=1}^{N_s}c_n(p_{n.s}(1)+p_{n.s}(2)x+p_{n.s}(3)y)$$

（4）根据边界条件确定 $\boldsymbol{c_1}$ 中的节点系数；

（5）根据等式 $A_2\boldsymbol{c_2}=\boldsymbol{d}$ 确定内节点的系数；

其中：

$$A_1=\sum_{s=1}^{N_s}\left\{\left[\frac{\partial}{\partial x}\varphi_{2,s}\right]\left[\frac{\partial}{\partial x}\varphi_{1,s}\right]^T+\left[\frac{\partial}{\partial y}\varphi_{2,s}\right]\left[\frac{\partial}{\partial y}\varphi_{1,s}\right]^T-g(x_s,y_s)\varphi_{2,s}{\varphi_{1,s}}^T\right\}\Delta S_s$$

$$A_2=\sum_{s=1}^{N_s}\left\{\left[\frac{\partial}{\partial x}\varphi_{2,s}\right]\left[\frac{\partial}{\partial x}\varphi_{2,s}\right]^T+\left[\frac{\partial}{\partial y}\varphi_{2,s}\right]\left[\frac{\partial}{\partial y}\varphi_{2,s}\right]^T-g(x_s,y_s)\varphi_{2,s}{\varphi_{2,s}}^T\right\}\Delta S_s$$

$$\boldsymbol{d} = -A_1\boldsymbol{c}_1 - \sum_{s=1}^{N_s} f(x_s, y_s)\varphi_{2,s}\Delta S$$

满足以下条件时：

$$A_1\boldsymbol{c}_1 + A_2\boldsymbol{c}_2 + \sum_{s=1}^{N_s} f(x_s, y_s)\varphi_{2,s}\Delta S_s = 0$$

最小化表达式为：

$$I = \iint_R \left\{ \left(\frac{\partial}{\partial x}u(x,y)\right)^2 + \left(\frac{\partial}{\partial y}u(x,y)\right)^2 - g(x,y)u^2(x,y) + 2f(x,y)u(x,y) \right\} \boldsymbol{dxdy}$$

$$= \iint_R \left\{ \boldsymbol{c}^T \frac{\partial}{\partial x}\varphi \frac{\partial}{\partial x}\varphi^T\boldsymbol{c} + \boldsymbol{c}^T \frac{\partial}{\partial y}\varphi \frac{\partial}{\partial y}\varphi^T\boldsymbol{c} - g(x,y)\varphi_2\varphi^T\boldsymbol{c} + f(x,y)\varphi_2 \right\} \mathrm{dxdy}$$

可以求得偏微分方程的解。

2. 求解程序 MATLAB 实现

在进行偏微分方程的求解之前，首先直观地给出基函数的图形。

分下面步骤进行。

（1）首先通过函数 fem_basis_ftn 产生基函数，使得 ϕ_n 仅在 n 处取 1，其余处均取 0。

在 MATLAB 中编程实现的产生基函数的程序为：fem_basis_ftn。

功能：产生基函数。

调用格式：p = fem_basis_ftn(N,S)。

其中，$\boldsymbol{N}$ 为节点坐标矩阵；

$\boldsymbol{S}$ 为子区域的节点列表矩阵，第 i 行为第 i 个子区域内的所有节点编号；

$\boldsymbol{P}$ 为基函数 ftn phi_i 系数。

具体程序代码如下所示。

```
function p = fem_basis_ftn(N,S)
%p(i,s,1:3): 基函数 ftn phi_i 系数
% 共 s 个子区域
%N(n,1:2) : 第 n 个节点的 x 和 y 坐标
%S(s,1:3) : 第 s 个子区域的节点#s
N_n = size(N,1); %总共节点数
N_s = size(S,1); % 总子区域个数

for n = 1:N_n
    for s = 1:N_s
        for i = 1:3
            A(i,1:3) = [1 N(S(s,i),1:2)];
            b(i) = (S(s,i) == n); %第 n 个基函数仅在节点 n 为 1
        end
        pnt=A\b';
        for i=1:3, p(n,s,i) = pnt(i); end
    end
end
bx0 = inline('y^2','y');                    %边界条件
bxf = inline('4^2*cos(y)','y');
```

```
by0 = inline('x^2','x');
byf = inline('4^2*cos(x)','x');
D = [x0 xf y0 yf]; MaxIter = 400; MinErr = 1e-4;
[U,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,D,Mx,My,MinErr,MaxIter);
clf, mesh(x,y,U)%axis([0 4 0 4 -100 100]);
```

(2) 通过函数 fem_coef 为每一个节点和子区域构造基函数 $\phi_{n,s}(x,y)$，求得系数向量 $\boldsymbol{c}$；

在 MATLAB 中编程实现产生的每一个节点和子区域构造的基函数为：fem_coef。

功能：每一个节点和子区域构造基函数。

调用格式：[U,*c*] = fem_coef(*f*, *g*, *p*, *c*, *N*, *S*, *N_i*)。

其中，*f* 为拉普拉斯方程等式右边的函数 *f*(*x*,*y*)；

g 为拉普拉斯方程等式左边方程 *g*(*x*,*y*)；

c 为边界点的取值；

N 为总节点数；

S 为子区域的节点列表矩阵，第 *i* 行为第 *i* 个子区域内的所有节点编号；

N_i 为区域内点数。

```
function [U,c] = fem_coef(f,g,p,c,N,S,N_i)
%p(i,s,1:3): 基函数 ftn phi_i 系数
%c = [ .1 1 . 0 0 .] 边界节点取 1，内节点取 0
%N(n,1:2) : 第 n 个节点的 x 和 y 坐标
%S(s,1:3) : 第 s 个子区域的节点#s
%N_i : 内节点个数
%U(s,1:3) : 每个区域的 p1 + p2(s)x + p3(s)y 的坐标
N_n = size(N,1); % 总共节点数
N_s = size(S,1); % 总共子区域数
d=zeros(N_i,1);
N_b = N_n-N_i;
for i = N_b+1:N_n
    for n = 1:N_n
        for s = 1:N_s
            xy = (N(S(s,1),:) + N(S(s,2),:) + N(S(s,3),:))/3; %重心
            p_vctr = [p([i n],s,1) p([i n],s,2) p([i n],s,3)];
            tmpg(s) = sum(p(i,s,2:3).*p(n,s,2:3))-g(xy(1),xy(2))*p_vctr(1,:)*
                      [1 xy]'*p_vctr(2,:)*[1 xy]';
            dS(s) = det([N(S(s,1),:) 1; N(S(s,2),:) 1;N(S(s,3),:) 1])/2;
            %子区域
            if n == 1, tmpf(s) = -f(xy(1),xy(2))*p_vctr(1,:)*[1 xy]'; end
        end
        A12(i - N_b,n) = tmpg*abs(dS)';
    end
    d(i-N_b) = tmpf*abs(dS)';
end
d = d - A12(1:N_i,1:N_b)*c(1:N_b)';
c(N_b + 1:N_n) = A12(1:N_i,N_b+1:N_n)\d;
for s = 1:N_s
    for j = 1:3, U(s,j) = c*p(:,s,j); end
end
```

（3）得出基函数图形。

在 MATLAB 中编程实现绘制基函数图形的函数：Show_Basis。

功能：绘制基函数图形。

调用格式：Show_Basis()。

```
function Show_Basis()
%N = [-1 1;1 1;1 -1;-1 -1;0.2 0.5];      %节点集合
N = [1 0;0 1;1 2;2 1;1.2 1.5];           %节点集合
N_n = size(N,1);                         %总节点数
S = [1 2 5;2 3 5;3 4 5;1 4 5];           %区域集合
N_s = size(S,1);                         %总区域数
figure(1), clf
for s = 1:N_s
    nodes = [S(s,:) S(s,1)];
    for i = 1:3
        plot([N(nodes(i),1) N(nodes(i + 1),1)],[N(nodes(i),2) N(nodes(i+1),2)]),
    hold on
    end
end
text(0.8,0.6,'S1');text(0.8,1.6,'S2');text(1.4,1.5,'S3'),text(1.4,0.6,'S4');
%基函数
p = fem_basis_ftn(N,S);
%x0 = -1; xf = 1; y0 = -1; yf = 1;       %二维区域
x0 = 0; xf = 2; y0 = 0; yf = 2;          %二维区域
figure(2), clf
Mx = 50; My = 50;
c = [0 1 2 3 0]; %节点的值
dx = (xf - x0)/Mx; dy = (yf - y0)/My;
xi = x0 + [0:Mx]*dx; yi = y0 + [0:My]*dy;
i_ns = [1 2 3 4 5];                      %节点标号
%迭代计算五个节点的基函数
for itr = 1:5
    i_n = i_ns(itr);
    if itr == 1
        for i = 1:length(xi)
            for j = 1:length(yi)
                Z(j,i) = 0;
                for s = 1:N_s
                    if inpolygon(xi(i),yi(j), N(S(s,:),1),N(S(s,:),2)) > 0
                        Z(j,i) = p(i_n,s,1) + p(i_n,s,2)*xi(i) + p(i_n,s,3)*yi(j);
                        break;
                    end
                end
            end
        end
        subplot(321), mesh(xi,yi,Z);title(itr) %节点1的基函数
    else
        c1 = zeros(size(c)); c1(i_n) = 1;
        subplot(320 + itr)
        trimesh(S,N(:,1),N(:,2),c1)      %节点 2~5 的基函数
        title(itr);
    end
end
```

```
c = [0 1 2 3 0];
subplot(326)
trimesh(S,N(:,1),N(:,2),c);title('基函数的线型组合')
```

运行结果如图 9-13 和图 9-14 所示。

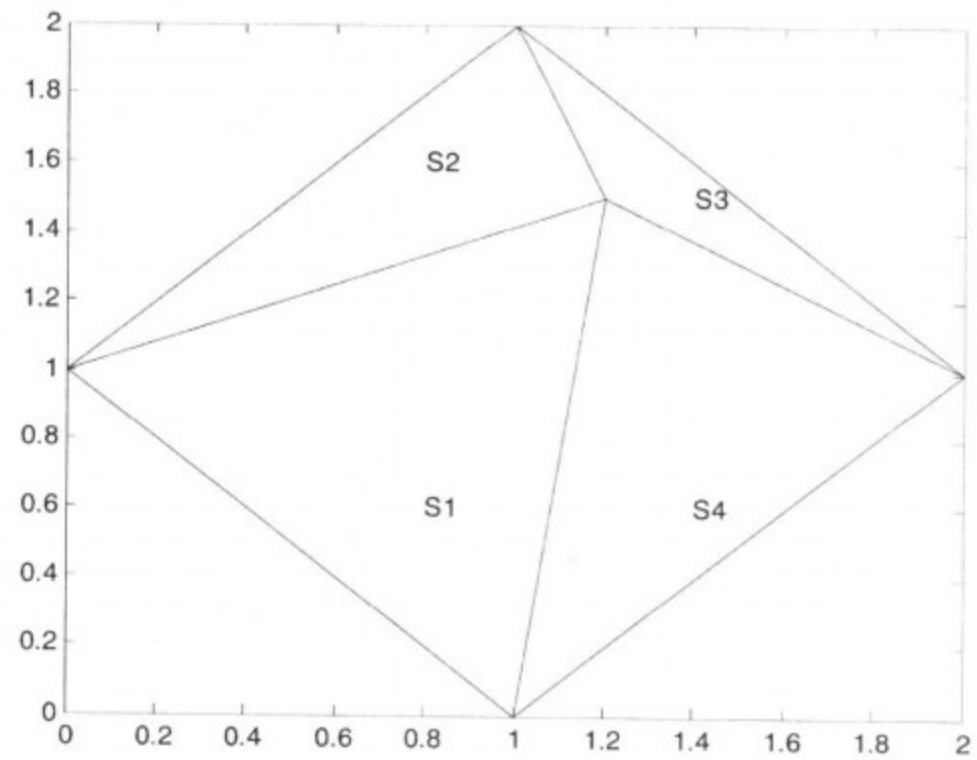

图 9-13　节点和子区域

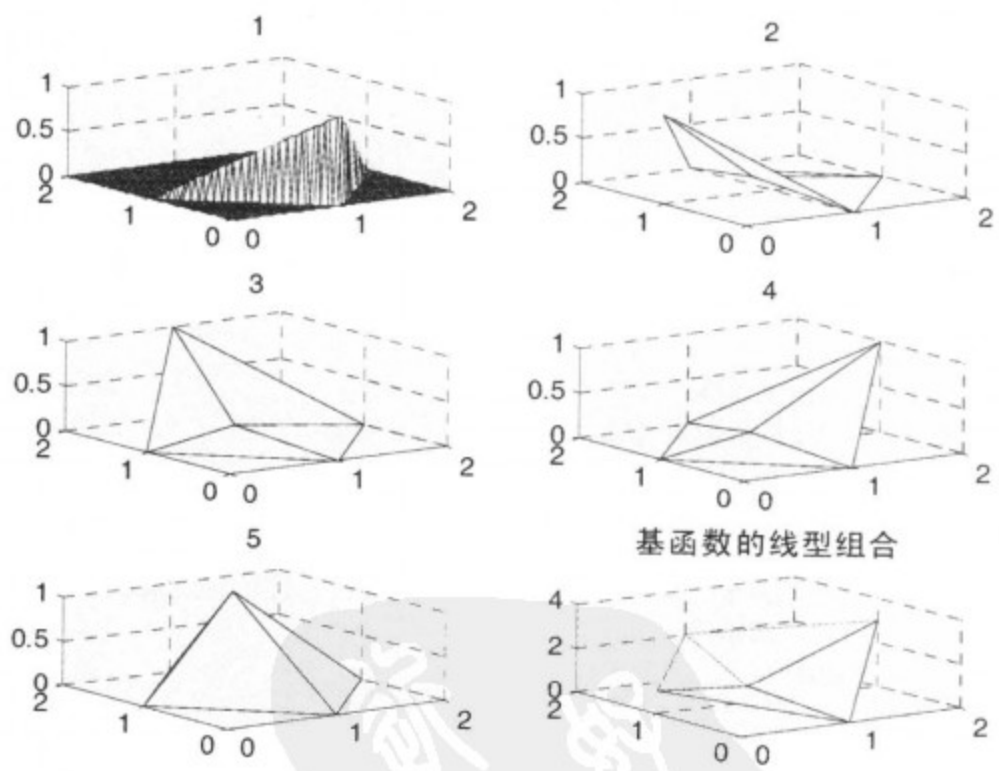

图 9-14　基函数及其线性组合

图 9-14 中图（1）~（5）分别为节点 1~5 的基函数，最后一个图为基函数的线性组合。

3. 实例分析

【例 9-9】 有限元法求解偏微分方程应用实例。采用有限元法求解如下拉普拉斯方程：

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}=f(x,y)$$

其中：

$$-1\leqslant x\leqslant 1,\ -1\leqslant y\leqslant 1$$

$$f(x,y)=\begin{cases}1,(x,y)=(0.5,0.5)\\1,(x,y)=(-0.5,0.5)\\0,(x,y)\in elsewhere\end{cases}$$

边界条件满足：$u(x,y)=0$。

解：首先进行区域分割，可以取 12 个边界点和 19 个内点。将区域 D 分割为 36 个子区域。注意到只有在内点 (0.5,0.5),(–0.5,–0.5) 附近，函数 $f(x,y)$ 的值不为零，其余地方均为零。

因此可以推测在这两个点附近，$u(x,y)$ 的值会存在剧烈的变化：出现波峰或者波谷。故在这两个点附近取较密集的内点，可以增加解的精确度。具体程序代码如下所示。

```
function ex0909()
%产生节点
N = [-1 0;-1 -1;-1/2 -1;0 -1;1/2 -1; 1 -1;1 0;1 1;1/2 1; 0 1;
    -1/2 1;-1 1; -1/2 -1/4; -5/8 -7/16;-3/4 -5/8;-1/2 -5/8;
    -1/4 -5/8;-3/8 -7/16; 0 0; 1/2 1/4;5/8 7/16;3/4 5/8;
    1/2 5/8;1/4 5/8;3/8 7/16;-9/16 -17/32;-7/16 -17/32;
    -1/2 -7/16;9/16 17/32;7/16 17/32;1/2 7/16];
N_b = 12;                                    %边缘节点数
%分割区域
S = [1 11 12;1 11 19;10 11 19;4 5 19;5 7 19; 5 6 7;1 2 15; 2 3 15;
    3 15 17;3 4 17;4 17 19;13 17 19;1 13 19;1 13 15;7 8 22;8 9 22;
    9 22 24;9 10 24; 10 19 24; 19 20 24;7 19 20; 7 20 22;13 14 18;
    14 15 16;16 17 18;20 21 25;21 22 23;23 24 25;14 26 28;
    16 26 27;18 27 28; 21 29 31;23 29 30;25 30 31;
    26 27 28; 29 30 31];                     %子区域
f11 = '-(norm([x y]+[0.5 0.5])<0.01)-(norm([x y]-[0.5 0.5]) < 0.01)';
 f=inline(f11,'x','y');
g=inline('0','x','y');
N_n = size(N,1);                             %总节点数
N_i = N_n - N_b;                             %内部节点数
c = zeros(1,N_n);                            %边界节点和内部节点的取值
p = fem_basis_ftn(N,S);
[U,c] = fem_coef(f,g,p,c,N,S,N_i);
%输出图形
figure(1), clf, trimesh(S,N(:,1),N(:,2),c)
N_s = size(S,1);                             %子区域总数
x0 = -1; xf = 1; y0 = -1; yf = 1;
Mx = 16; dx = (xf - x0)/Mx; xi = x0+[0:Mx]*dx;
My = 16; dy = (yf - y0)/My; yi = y0+[0:My]*dy;
for i = 1:length(xi)
    for j = 1:length(yi)
        for s = 1:N_s                        %该点属于该区域
            if inpolygon(xi(i),yi(j), N(S(s,:),1),N(S(s,:),2)) > 0
                Z(i,j) = U(s,:)*[1 xi(i) yi(j)]';
```

```
                break;
            end
        end
    end
end
figure(2), clf, mesh(xi,yi,Z)
bx0 = inline('0'); bxf = inline('0');
by0 = inline('0'); byf = inline('0');
[U,x,y] = Helmholtz(f,g,bx0,bxf,by0,byf,[x0 xf y0 yf],Mx,My,1e-4,100);
figure(3), clf, mesh(x,y,U)
```

运行结果如图 9-15、图 9-16 和图 9-17 所示。

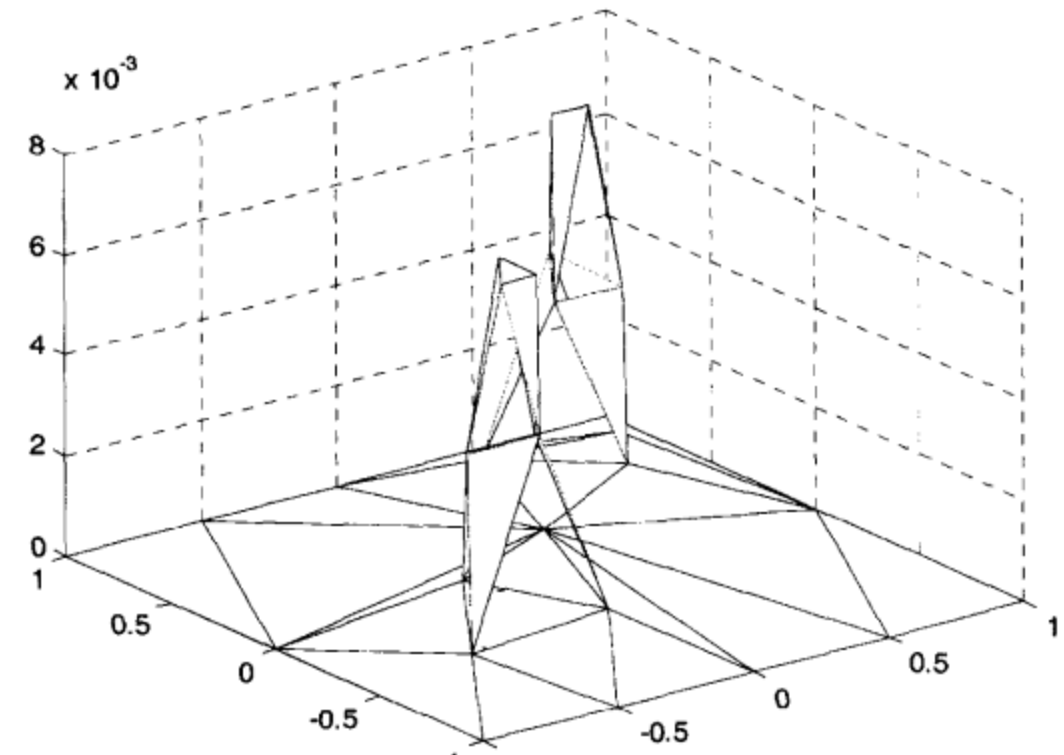

图 9-15　拆分区域 D 为 36 个子区域

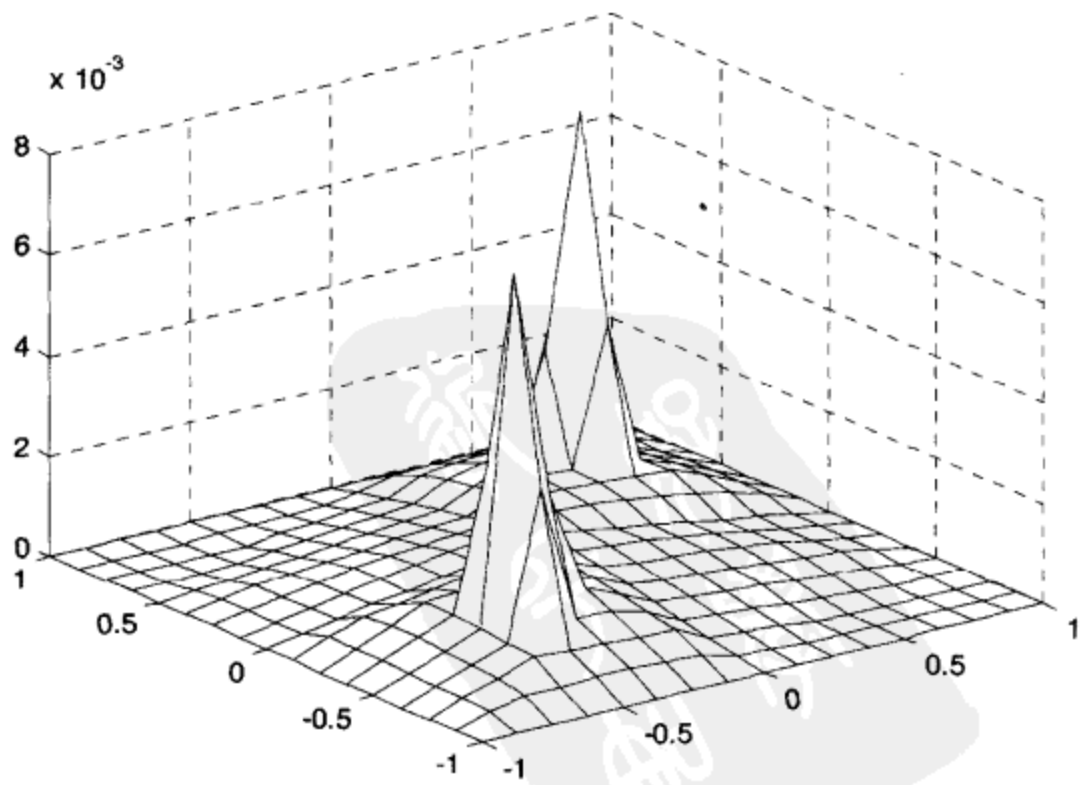

图 9-16　由有限元法得出的拉普拉斯方程数值解

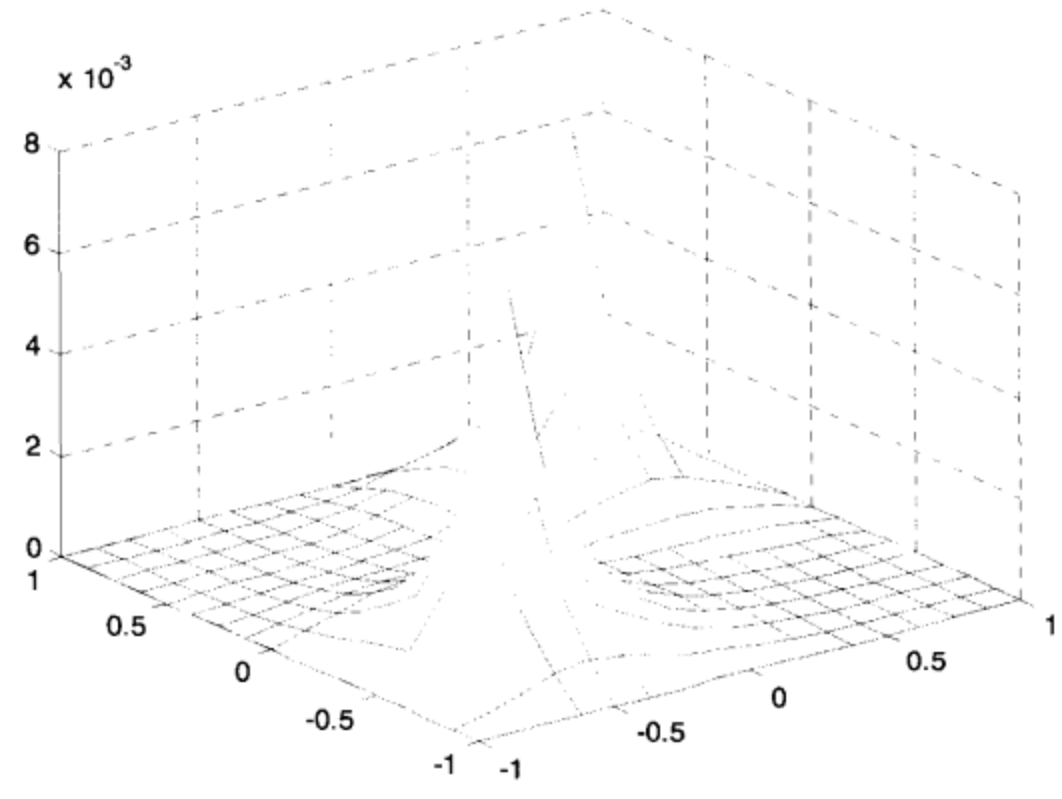

图 9-17 由【例 9-1】的解法求得的拉普拉斯方程数值解

9.6 偏微分方程求解工具（PDETOOL）的使用

本节主要介绍使用 MATLAB 的 GUI 工具求解偏微分方程。

9.6.1 可由 PDETOOL 求解的基本偏微分方程

可采用 PDETOOL 求解的基本偏微分方程有椭圆偏微分方程、抛物线偏微分方程、双曲线偏微分方程和特征值型偏微分方程。

1. 椭圆偏微分方程

椭圆偏微分方程形如：

$$-\nabla\cdot(c\nabla u)+au=f$$

边界条件为：$hu=r$（dirichlit 条件）或者 $n\cdot c\nabla u+qu=g$（广义黎曼条件）。

如果 u 是在矩形区域内的一个标量算子，则上述方程变为：

$$-c(\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2})+au(x,y)=f(x,y)。$$

2. 抛物线偏微分方程

抛物线偏微分方程形如：

$$-\nabla\cdot(c\nabla u)+au+\mathrm{d}\frac{\partial u}{\partial t}=f$$

在区域Ω上，取值范围为：$0 \leqslant t \leqslant T$。边界条件与上述椭圆偏微分方程同，初始条件为$u(t_0)$。

3. 双曲线偏微分方程

双曲线偏微分方程形如：

$$-\nabla \cdot (c\nabla u) + au + \mathrm{d}\frac{\partial^2 u}{\partial t^2} = f$$

在区域Ω上，取值范围为：$0 \leqslant t \leqslant T$。边界条件与上述椭圆偏微分方程同，初始条件为$u(t_0)/u'(t_0)$。

4. 特征值型偏微分方程

特征值型偏微分方程形如：

$$-\nabla \cdot (c\nabla u) + au = \lambda du \text{。}$$

边界条件同上述椭圆偏微分方程。

PDE 工具箱还能处理如下类型的偏微分方程：

$$-\nabla \cdot (c_{11}\nabla u_1) - \nabla \cdot (c_{12}\nabla u_2) + a_{11}u_1 + a_{12}u_2 = f_1$$
$$-\nabla \cdot (c_{21}\nabla u_1) - \nabla \cdot (c_{22}\nabla u_2) + a_{21}u_1 + a_{22}u_2 = f_2$$

在区域Ω上，具有 Dirichlet 边值条件：

$$\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$$

或者牛顿边值条件：

$$n \cdot (c_{11}\nabla u_1) + n \cdot (c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1$$
$$n \cdot (c_{21}\nabla u_1) + n \cdot (c_{22}\nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2$$

以及混合边界条件：

$$\boldsymbol{c} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix},\quad \boldsymbol{a} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},\quad \boldsymbol{f} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix},\quad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$\boldsymbol{h} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix},\quad \boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix},\quad \boldsymbol{q} = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix},\quad \boldsymbol{g} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

9.6.2 PDETOOL 的使用说明

PDE 工具箱采用有限元法解偏微分方程。具体使用步骤为：

（1）在 MATLAB 命令窗中输入 pdetool 命令，按回车键可以启动 PDE 工具箱。

此工具箱提供方便的用户交互命令，用户可以调整坐标的比例、网格线等各种显示参数。在 Options 菜单下选择 Grid 命令，打开网格。

'Aplication'选项可用于选择所要解的偏微分方程类型。

（2）在 Draw 模式下，使用 CSG 对话框创建区域Ω。

在下拉菜单中选择二维图形的类型，再拖动到指定位置，可用鼠标控制图形的大小。

（3）在 Boundary 模式下，在各个边界段上给出边界条件。

（4）在 PDE 模式下，确定方程类型、系数，也可以在不同子区域上设置不同的系数。

（5）在 Mesh 模式下，生成网格，自动控制网格参数。

（6）在 Solve 模式下，解偏微分方程并输出结果。在解抛物线和双曲线时，需要设定初始条件和时间区间。

（7）在 Plot 模式下，可以选择作图选项，使用 Color，Height 和 Vector 等作图。对于 抛物线型和双曲线型方程，还可以生成动画解。

下面通过一个实际例子说明工具箱的使用。

【例 9-10】 偏微分方程工具箱应用实例。利用偏微分方程工具箱，求解【例 9-1】所示的椭圆微分方程的拉普拉斯形式：

$$\nabla^2 u(x,y)=\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}=0$$

其自变量取值范围 D 为：

$$0\leqslant x\leqslant 4,\quad 0\leqslant y\leqslant 4$$

边界条件为：

$$u(x,0)=x^2,\ u(x,4)=16\cos(x),$$

$$u(0,y)=y^2,\ u(4,y)=16\cos(y)。$$

解：第一步，输入命令 pdetool 打开工具箱，调整 x 坐标范围为[0,5]，y 坐标范围为[0,5]。通过 options 选项的 Axes Limits 设定，如图 9-18 所示。

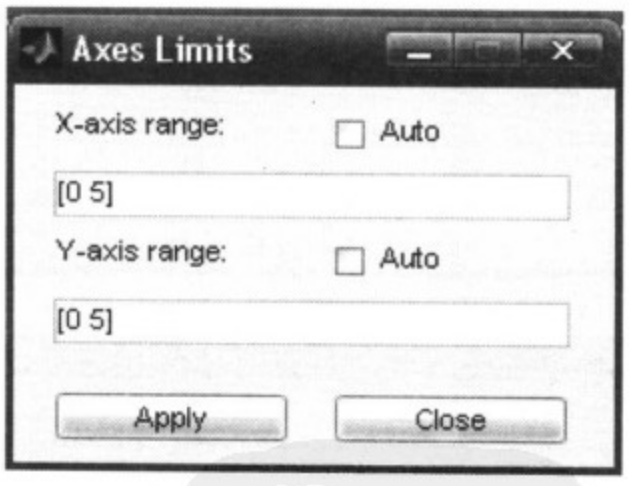

图 9-18　设定坐标范围

第二步，设定矩形区域。单击工具栏中的按钮“ ▭ ”，拖动鼠标画出一矩形，并双击该矩形，设定矩形的大小，如图 9-19 所示。

第三步：设置边界条件。单击工具栏中的按钮“ ?Ω| ”，并双击矩形区域的相应的边线在弹出的对话框中设定边界条件。如图 9-20 所示，为其中两条边的边界条件设定框，其余两条边的设定与此类似。

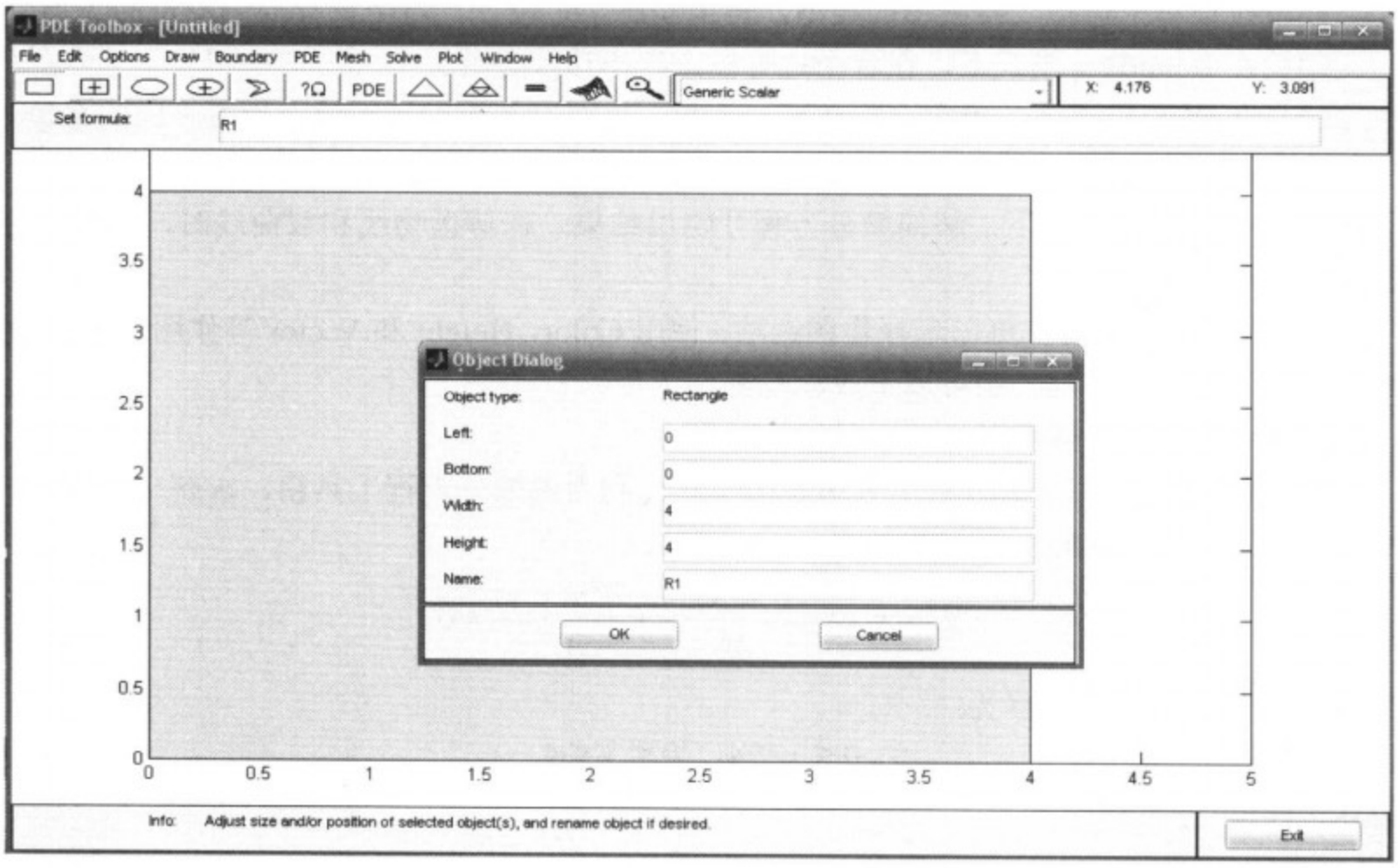

图 9-19　设定矩形区域边界

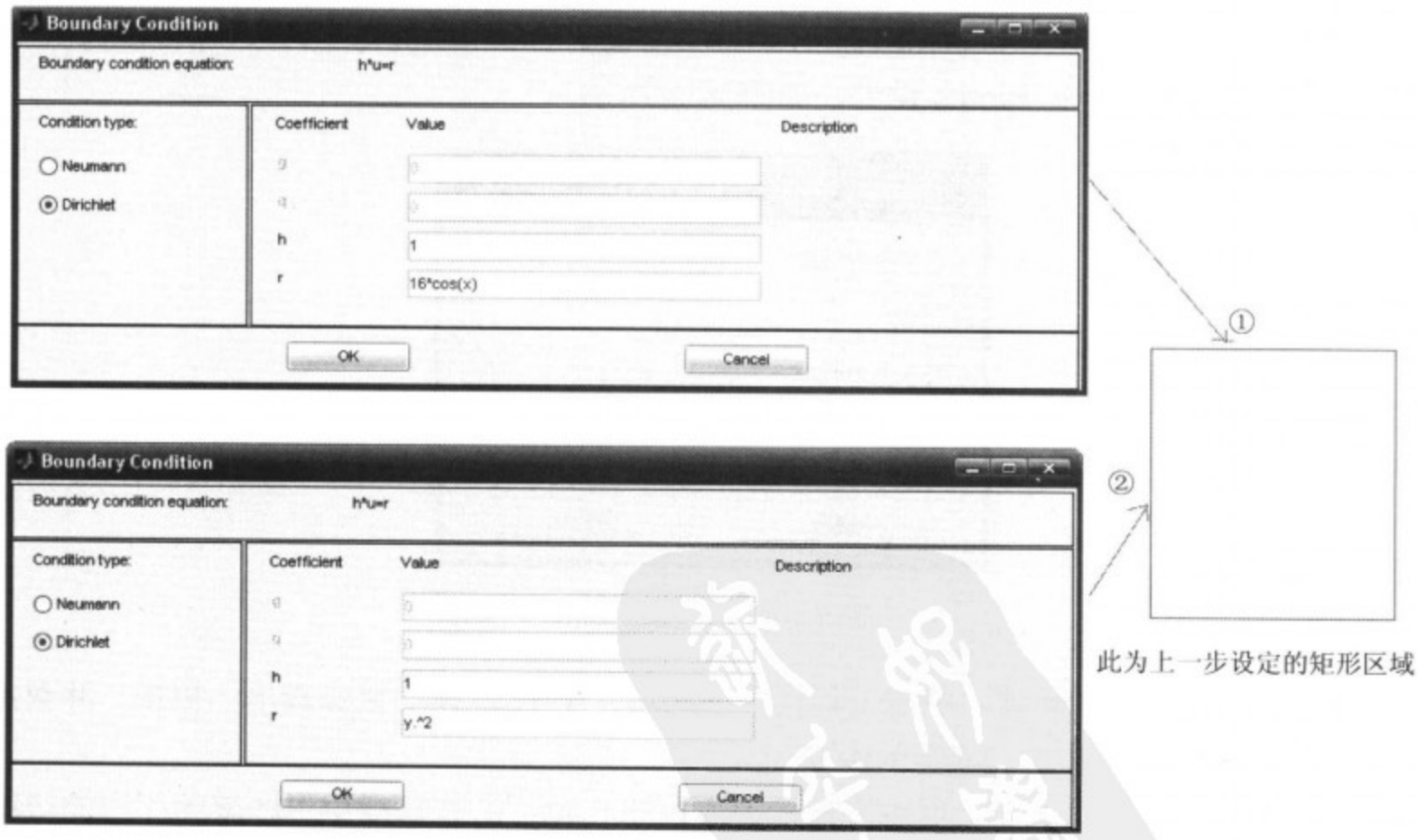

图 9-20　设定边界条件

注：图中的小矩形区域为第二步中设定的矩形区域，箭头指向的两边界分别为：

① $n(t, 4)=16\cos t$　　② $n(0, y)=y^2$

第四步，设定方程。单击工具栏中的按钮“ PDE ”在 PDE 模式下选择方程类型，如图 9-21 所示，并在其中设定参数。

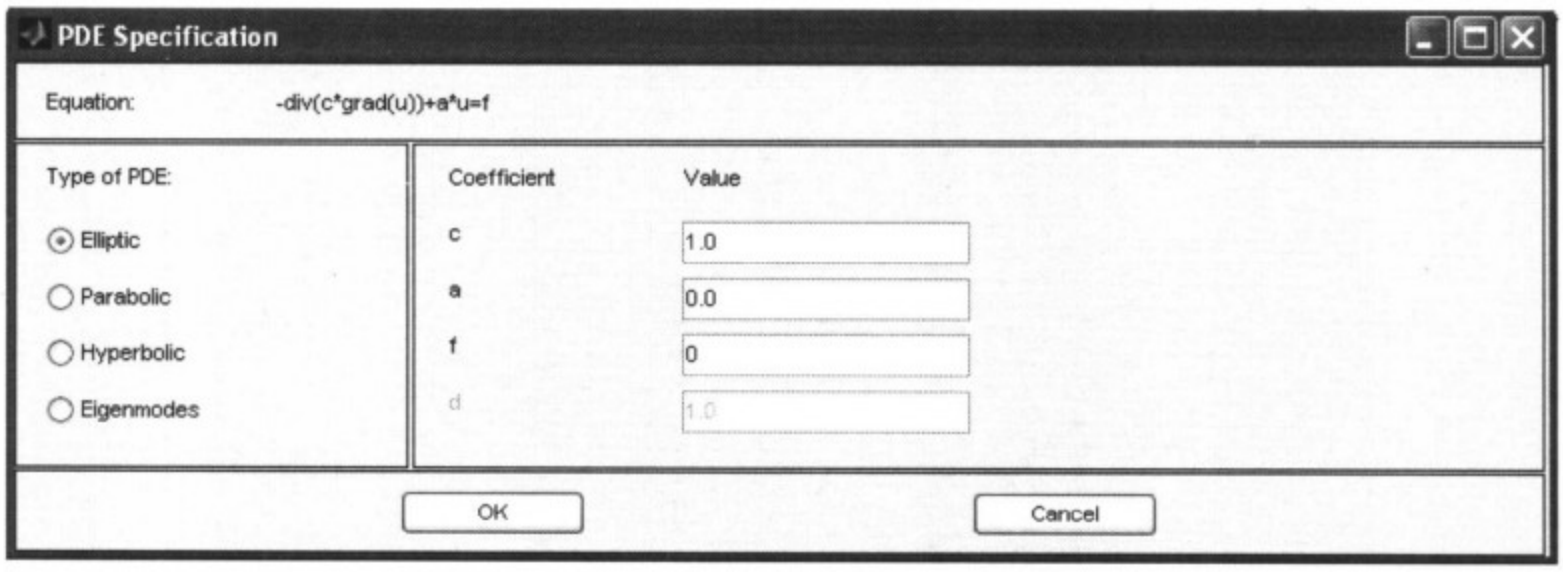

图 9-21　PDE 类型选定和参数设定

第五步，单击工具栏中的按钮“ △ ”拆分区域为若干子区域，如图 9-22 所示。

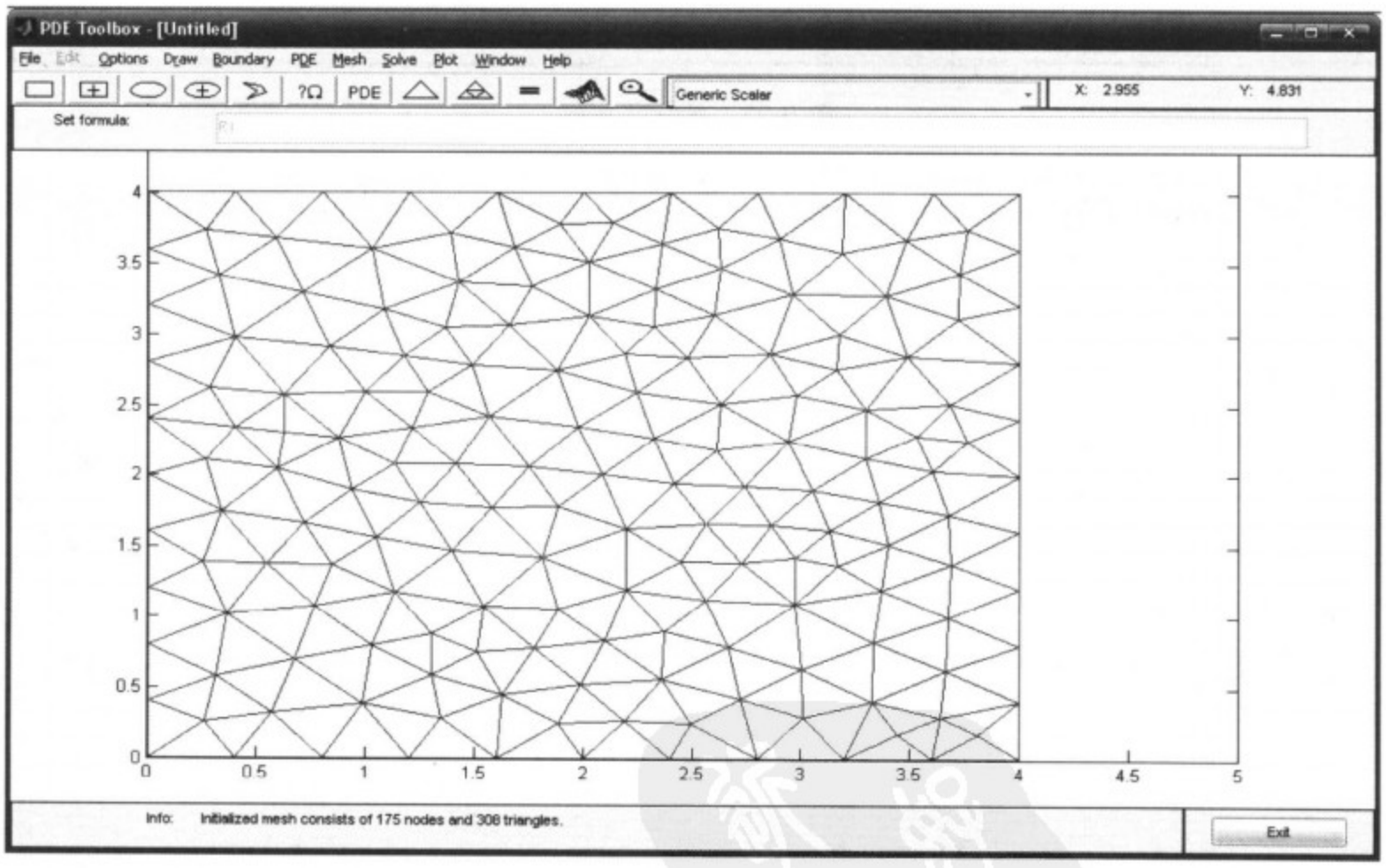

图 9-22　子区域的产生

第六步，单击工具栏中的按钮“ ”，将子区域细化，从而保证结果更精确，如图 9-23 所示。

第七步，单击工具栏中的按钮“ ”，设置所画曲线的特性，如图 9-24 所示，并作出其解的三维图。

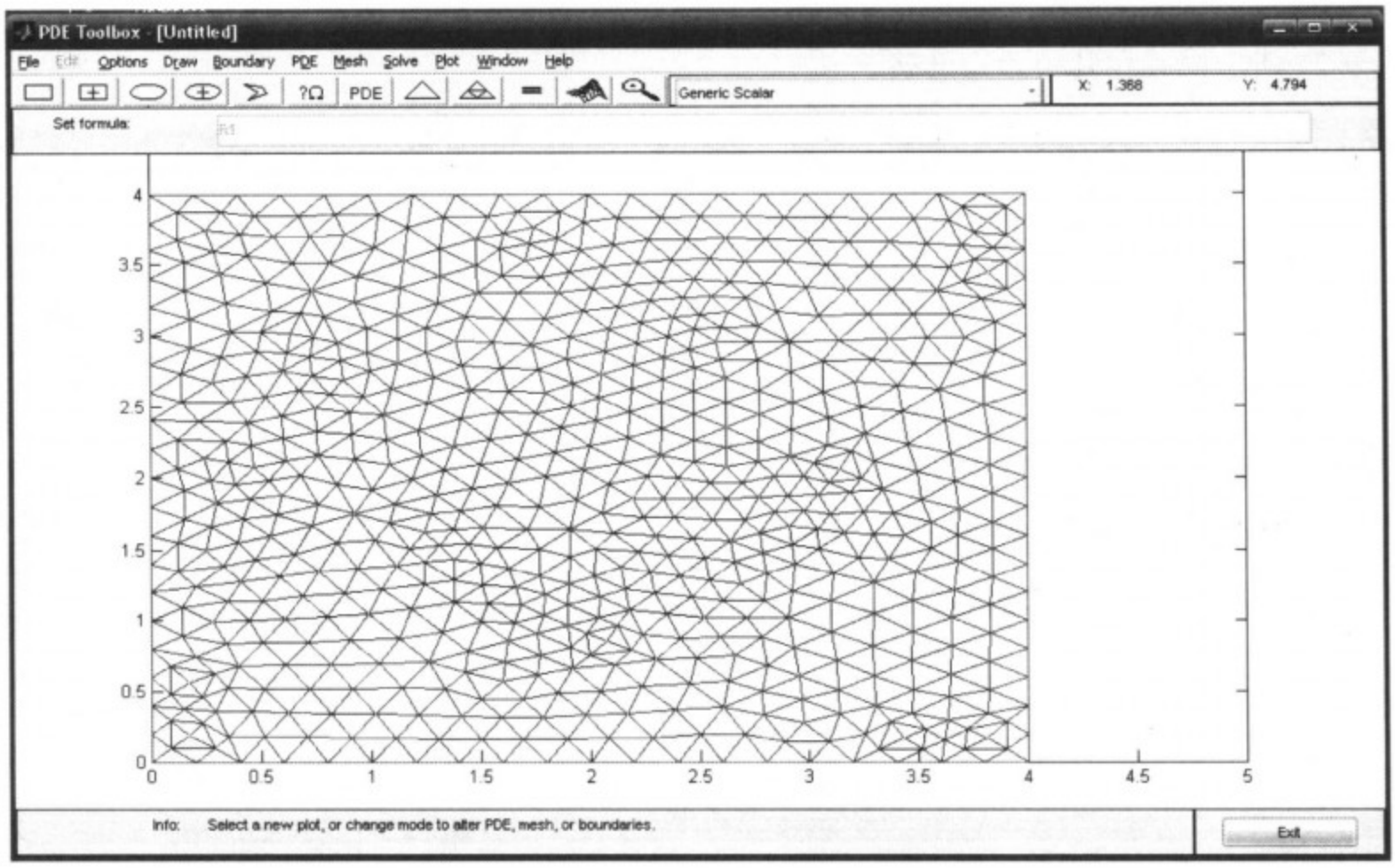

图 9-23　细化子区域

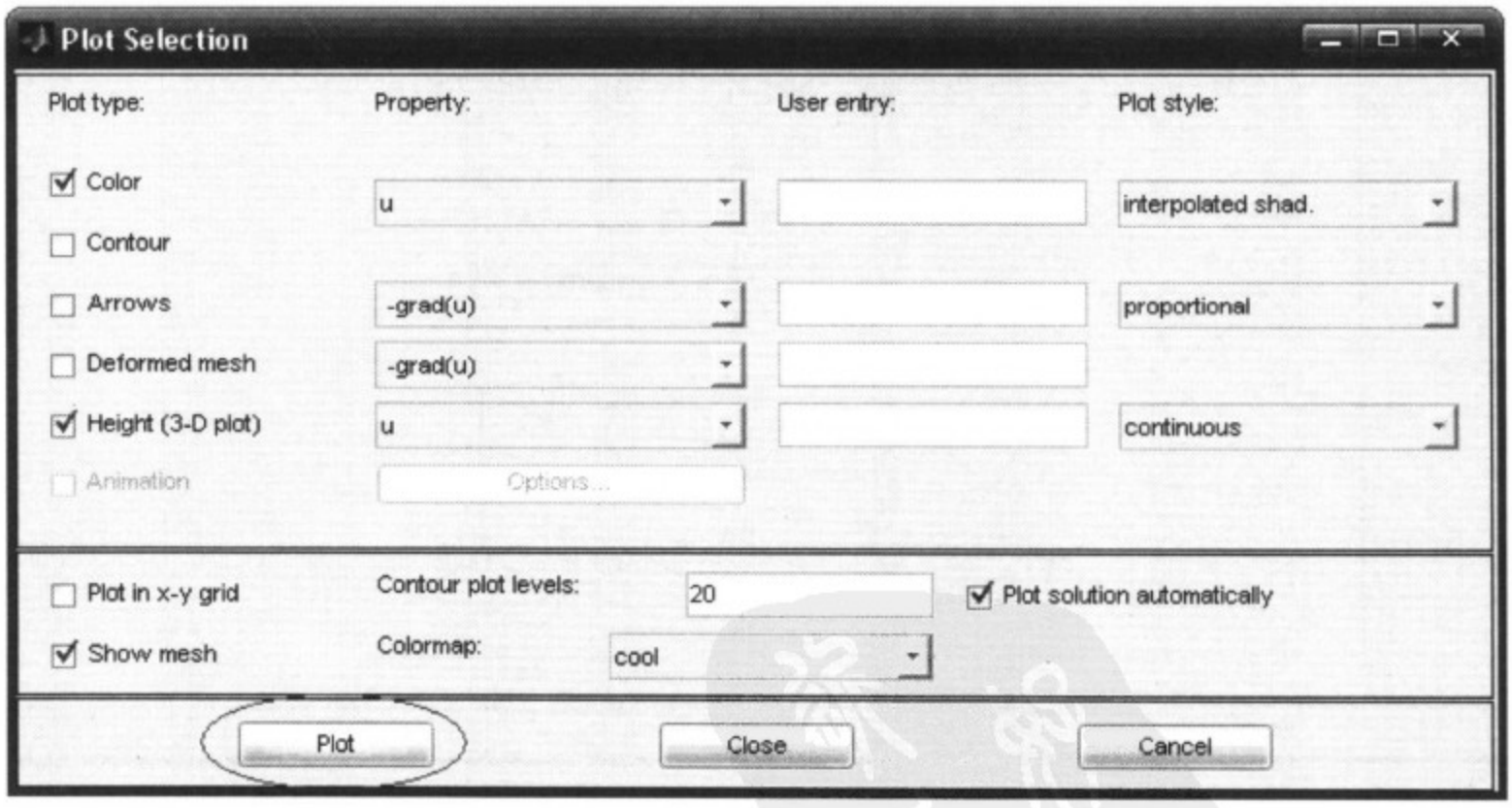

图 9-24　设置图形特性

第八步，单击图 9-24 中标出的“plot”按钮，或单击工具栏中的按钮“ ═ |”，可作出解的三维图，如 9-25 所示。

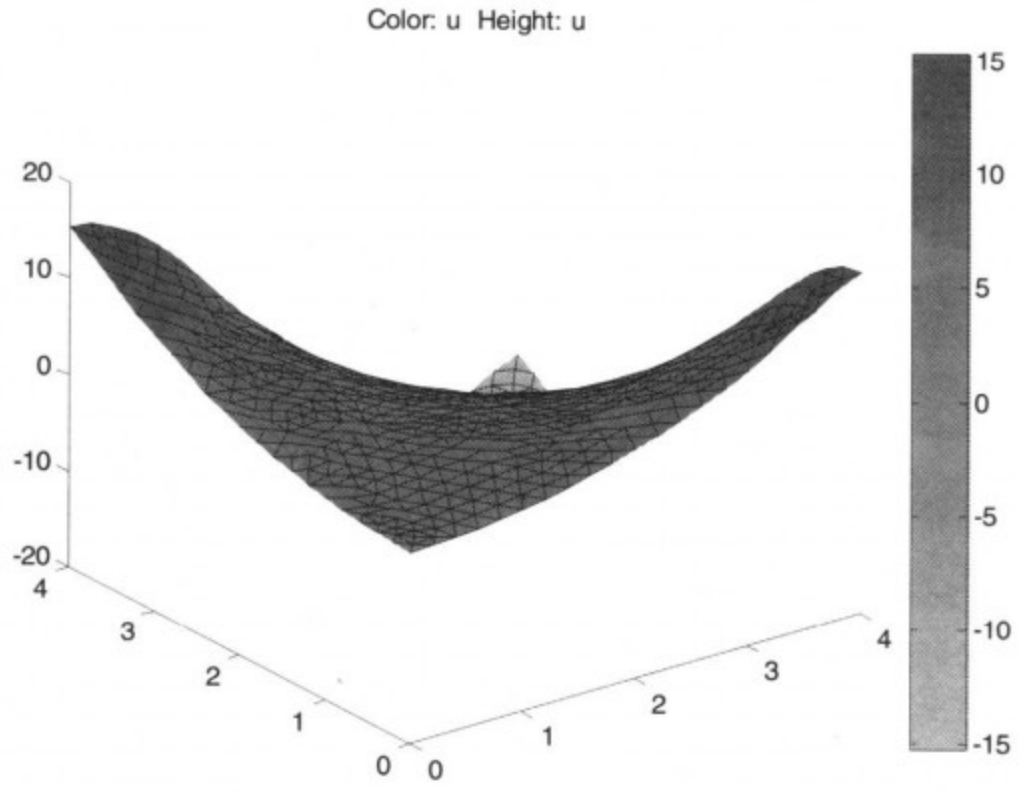

图 9-25　方程的数值解

与图 9-3 对比，可见两种解法得出的解近似相等。从而验证了【例 9-1】解法的正确性。

9.7　小结

本章讲述了椭圆型、抛物线型和双曲线型这三大类偏微分方程，并分别给出了 MATLAB 实现的求解算法。然后又介绍了解偏微分方程的一种有效方法：有限元法。最后介绍了采用 PDE 工具箱进行偏微分方程的求解方法。

第 10 章

复数与复变函数计算

复数，以及在其基础上发展起来的复变函数这一重要的数学分支，解决了许多实变函数无法解决的问题，有着广泛的应用。MATLAB 不仅支持实数运算，而且还支持在运算和函数中使用复数或复数矩阵。

通过本章学习，读者能掌握 MATLAB 中支持的复数基本运算等，还能掌握 MATLAB 在复变函数中的应用，包括留数运算、泰勒级数、傅里叶变换以及拉普拉斯变换等。

10.1 复数的表示法

10.1.1 复数的一般表示

MATLAB 是以 i 或 j 字元来代表虚部复数运算的。

一个复数可表示为：$x=a+bi$，其中 a 称为实部，b 称为虚部。

也可写成复指数形式：$x=re^{i\theta}$。其中 r 称为复数的模，又记为 $|x|$；θ 称为复数的幅角，又记为 $Arg(x)$，且满足如下关系：

$$r=\sqrt{a^2+b^2}\ ,\quad \tan\theta=\frac{b}{a}$$

具体采用哪种表示方法取决于实际问题。一般而言，第一种问题适合处理复数的代数运算，第二种方法适合处理复数旋转等涉及幅角改变的问题。

一个复数也可以看做是关于实部和虚部的符号函数。因此，既可以直接构造复数，也可以用符号函数构造复数。下面分别介绍这两种构造方法。

1．用直接法构造两种形式的复数

直接法就是利用符号 i 或 j 来表示复数单位，将复数看作完整的一个表达式输入。其具体形式也有两种，可以用实部虚部形式表示，也可以用复指数形式表示。

【例 10-1】 直接法构造复数实例之一。在 MATLAB 中用实部虚部形式表示复数 $x=-1+i$。

解：在 MATLAB 命令窗口中输入：

```
>> x=-1+i
```

输出结果为：

```
x =
  -1.0000 + 1.0000i
```

【例 10-2】 直接法构造复数实例之二。在 MATLAB 中用复指数形式表示复数 $x=-1+i$。

解：x 表示成复指数形式为：$x=\sqrt{2}e^{i\frac{3}{4}\pi}$，故可按如下形式表示。

在 MATLAB 命令窗口中输入：

```
>> x=sqrt(2)*exp(i*(3*pi/4))
```

输出结果为：

```
x =
  -1.0000 + 1.0000i
```

2．用符号函数法构造两种形式的复数

所谓符号函数法就是将复数看成是函数形式，其实部和虚部看作自变量，用 syms 来构造，用 subs 对符号函数中自变量进行赋值。

【例 10-3】 符号函数法构造复数实例。用 syms 构造复数 $1+i$，并表示为实部虚部形式。

解：在 MATLAB 命令窗口中输入：

```
>> syms a b real;                          %声明 a，b 为实数型
>> x=a+b*i
x =
a+i*b                                      %复数的符号表达
```

代入具体值：

```
>> subs(x,{a,b},{1,1})
ans =
   + 1.0000i
1.0000
```

【例 10-4】 符号函数法构造复数实例。构造【例 10-3】中的复数，表示为复指数形式。

解：在 MATLAB 命令窗口中输入：

```
>> syms r ct real;
>> y=r*exp(ct*i);
>> y=r*exp(ct*i)
```

输出结果为：

```
y =
r*exp(i*ct)
```

代入具体值如下：

```
>> subs(y,{r,ct},{sqrt(2),pi/4})
ans =
   1.0000 + 1.0000i
```

10.1.2 复数矩阵的表示

MATLAB 中矩阵运算贯穿始终，复数矩阵也是其中的一个重要方面。由于复数矩阵的每个元素都是复数，所以对应的复数创建矩阵也有两种，即直接创建和利用符号函数创建。

复数矩阵的直接创建有两种方法，一种是由复数元素构造复数矩阵，另一种是利用两个矩阵分别做实部和虚部构造新的复数矩阵。下面举例说明复数矩阵的直接创建，另一种方法类比很容易实现，在此不作赘述。

1. 由复数元素构造复数矩阵

复数矩阵的每个元素都是复数，将复数作为矩阵元素并按照矩阵格式进行填充就得到了复数矩阵。

【例 10-5】 由复数元素构造复数矩阵实例。构造复数矩阵 $\begin{bmatrix} 1+i & 1+2i & 1+3i \\ 1-i & 1-2i & 1-3i \end{bmatrix}$。

解：可直接输入各元素来构造矩阵，此处通过其复指数形式构造。

在 MATLAB 命令窗口中输入：

```
>> A=[sqrt(2)*exp((pi/4)*i) 1+2i 1+3i;sqrt(2)*exp((-pi/4)*i) 1-2i 1-3i]
```

输出结果为：

```
A =
   1.0000 + 1.0000i   1.0000 + 2.0000i   1.0000 + 3.0000i
   1.0000 - 1.0000i   1.0000 - 2.0000i   1.0000 - 3.0000i
```

2. 由实矩阵创造复数矩阵

由两个分别作实部和虚部的同维矩阵构造复数矩阵。

【例 10-6】 由实矩阵构造复数矩阵实例。通过复数的实部和虚部矩阵构造【例 10-5】中的矩阵。

解：在 MATLAB 命令窗口中输入：

```
>> Are=[1 1 1;1 1 1];
>> Aim=[1 2 3;-1 -2 -3];
>> A=Are+Aim*i
```

输出结果为：

```
A =
```

```
1.0000 + 1.0000i   1.0000 + 2.0000i   1.0000 + 3.0000i
1.0000 - 1.0000i   1.0000 - 2.0000i   1.0000 - 3.0000i
```

10.1.3　复数绘图

对于复数函数的绘图主要有两种形式。一种是直角坐标图，即分别以复数的实部和虚部为坐标作出复数的表示图；另一种为极坐标图，即分别以复数的模和幅角为坐标作图。

【例 10-7】　复数函数绘图实例。作出函数 $y=t+it\sin(t)$ 在两种坐标下的表示图。

解：作出直角坐标图——以实部为横坐标，以虚部为纵坐标，在 MATLAB 命令窗口中输入：

```
>> polar(delta,r)
>> t=0:0.01:2*pi;
>> y=t+i*t.*sin(t);
>> plot(y)
```

程序运行后，输出如图 10-1 所示的图形。

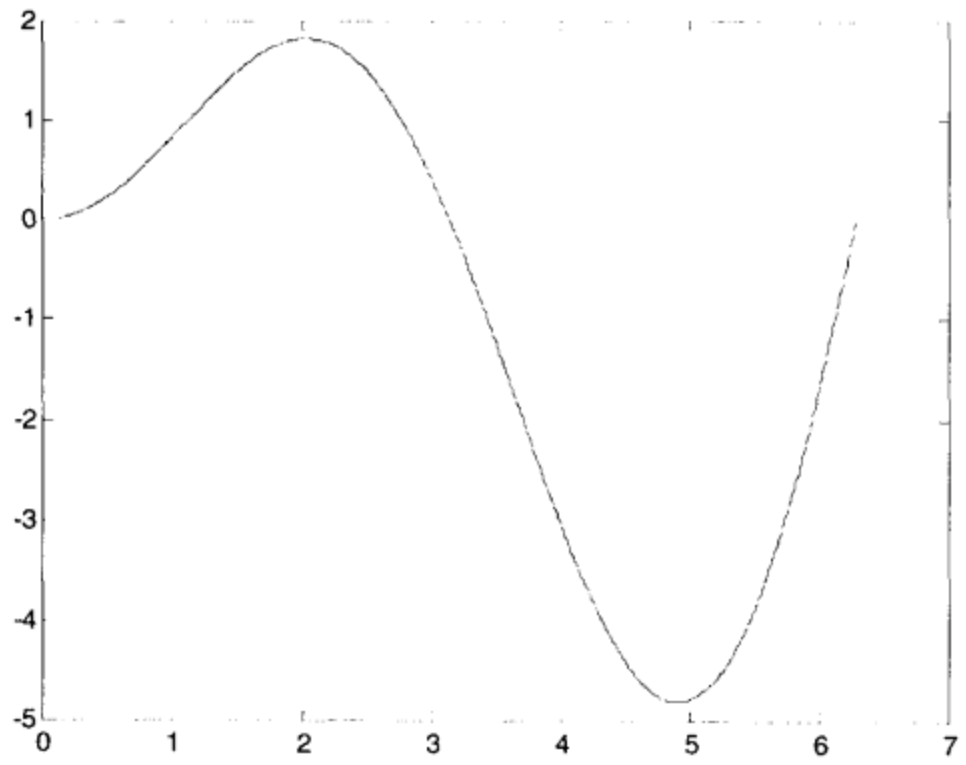

图 10-1　直角坐标系中复数的表示

作出极坐标图——以模为极半径，以幅角为极角，在 MATLAB 命令窗口中输入：

```
>> y=t+i*t.*sin(t);
>> plot(y)
>> r=abs(y);
>> delta=angle(y);
>> polar(delta,r)
```

程序运行后，输出如图 10-2 所示的图形。

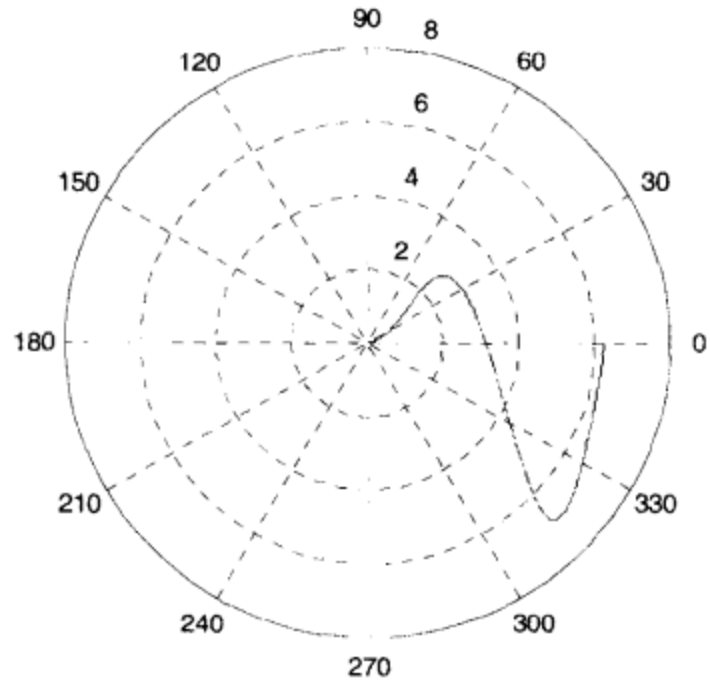

图 10-2　极坐标下复数的表示

10.2　复数的基本运算

10.2.1　复数的结构操作函数

对于一个形如 $x=a+bi$ 的复数，通常希望能够了解它自身的结构性质，包括实部，虚部、模和幅角等。MATLAB 提供了方便的操作函数。下面利用实例来说明各个函数的调用方法。

1．复数的实部和虚部

复数的实部和虚部的提取可由函数 real 和 imag 实现。

调用形式如下：

real(x)：返回复数 x 的实部；

imag(x)：返回复数 x 的虚部。

【例 10-8】　复数的实部和虚部计算的实例。求复数 $3+4i$ 的实部和虚部。

解：在 MATLAB 命令窗口中输入：

```
>> x=3+4i;
>> real(x)
ans =
     3
>> imag(x)
ans =
     4
```

2．复数的模和幅角

复数的模和辐角的求解由功能函数 abs 和 angle 实现。

调用形式如下：

abs(x)：求复数 x 的模；

angle(x)：求复数 x 的辐角。

【例 10-9】 复数的模和幅角计算的实例。求复数 $3+4i$ 的模和幅角。

解：在 MATLAB 命令窗口中输入：

```
>> x=3+4i;
>> abs(x)
ans =
     5
>> angle(x)
ans =
    0.9273
```

3. 复数的共轭

复数的共轭可由函数 conj 实现。

调用形式如下：

conj(x)：返回复数 x 的共轭复数。

【例 10-10】 共轭复数计算的实例。求复数 $3+4i$ 的共轭。

解：在 MATLAB 命令窗口中输入：

```
>> x=3+4i;
>> conj(x)
ans =
   3.0000 - 4.0000i
```

10.2.2　复数的基本数学运算

以如下两个复数为例来说明复数的基本数学运算，首先在 MATLAB 工作空间中生成复数 x 和 y，在 MATLAB 命令窗口中输入：

```
>> x=1+i;
>> y=1+sqrt(3)*i;
```

复数 x 和 y 就存在工作空间中，后面可对其进行运算。

1. 复数的乘除法

复数的乘除法运算通过运算符“*”和“/”实现。

【例 10-11】 复数乘除计算的实例。计算 $x*y$ 和 x/y。

解：在 MATLAB 命令窗口中输入：

```
>> x*y                                  %乘法
ans =
  -0.7321 + 2.7321i
>> x/y                                  %除法
ans =
   0.6830 - 0.1830i
```

2. 复数的平方根

复数的平方根运算由函数 *sprt* 实现。

调用形式如下：

sprt(x)　：返回复数 x 的平方根值。

【例 10-12】　复数平方根计算的实例。计算上述两复数的平方根。

解：在 MATLAB 命令窗口中输入：

```
>> sqrt(x)
ans =
   1.0987 + 0.4551i
>> sqrt(y)
ans =
   1.2247 + 0.7071i
```

3. 复数的幂运算

复数的幂运算的形式为 x ^ n，结果返回复数 x 的 n 次幂。

【例 10-13】　复数幂运算的实例。计算 x^2 和 y^3。

解：在 MATLAB 命令窗口中输入：

```
>> x^2
ans =
       0 + 2.0000i
>> y^3
ans =
  -8.0000 + 0.0000i
```

4. 复数的指数对数运算

复数的指数和对数运算分别由函数 exp 和 log 实现。

调用形式如下：

exp(x)：返回复数 x 的以 e 为底的指数值；

log(x)：返回复数 x 的以 e 为底的对数值。

【例 10-14】　复数指数和对数运算的实例。计算 e^x,e^{ix},log(x)。

解：在 MATLAB 命令窗口中输入：

```
>> exp(x)                          %指数形式
ans =
   1.4687 + 2.2874i
>> exp(i*x)                        %复指数形式
ans =
   0.1988 + 0.3096i
>> log(x)                          %对数形式
ans =
   0.3466 + 0.7854i
```

5. 复数的三角函数运算

复数的三角函数运算的函数如表 10-1 所示。

表 10-1　复数三角函数表

函 数 名	函数功能	函 数 名	函数功能
sin(x)	返回复数 x 的正弦函数值	$a$sin(x)	返回复数 x 的反正弦值
cos(x)	返回复数 x 的余弦函数值	$a$cos(x)	返回复数 x 的反余弦值
tan(x)	返回复数 x 的正切函数值	$a$tan(x)	返回复数 x 的反正切值
cot(x)	返回复数 x 的余切函数值	$a$cot(x)	返回复数 x 的反余切值
sec(x)	返回复数 x 的正割函数值	$a$sec(x)	返回复数 x 的反正割值
csc(x)	返回复数 x 的余割函数值	$a$csc(x)	返回复数 x 的反余割值
sinh(x)	返回复数 x 的双曲正弦值	coth(x)	返回复数 x 的双曲余切值
cosh(x)	返回复数 x 的双曲余弦值	sec h(x)	返回复数 x 的双曲正割值
tanh(x)	返回复数 x 的双曲正切值	csc h(x)	返回复数 x 的双曲余割值

下面通过实例讲述表 10-1 中的函数使用。

【例 10-15】 复数三角函数运算实例。计算 $\sin(x)$, $\cos(x)$, $\cos(x)+i\sin(x)$。

解：在 MATLAB 命令窗口中输入：

```
>> sin(x)                              %正弦值
ans =
   1.2985 + 0.6350i
>> cos(x)                              %余弦值
ans =
   0.8337 - 0.9889i
>> cos(x)+i*sin(x)                         %复指数形式的三角展开
ans =
   0.1988 + 0.3096i                    %与复指数形式计算结果相同
```

10.2.3　复数方程求根

复数方程求根或实方程的复数根求解也由函数 solve 实现，其调用格式为：

```
solve('原方程')
```

下面举例进行讲述。

【例 10-16】 复数方程求根实例。求解方程 $x^2+2x+2=0$。

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> y=x^2+2*x+2;
>> solve(y)
```

输出结果为：

```
ans =
 -1+i
 -1-i
```

由计算结果可知，方程 $x^2+2x+2=0$ 的根为 $-1\pm i$。

10.3 留数的基本运算

10.3.1 留数基础

留数的定义如下：

设 a 是 $f(z)$ 的孤立奇点，C 是 a 的充分小的邻域内一条把 a 点包含在其内部的闭路，积分 $\frac{1}{2\pi}\int_C f(z)\mathrm{d}z$ 称为 $f(z)$ 在 a 点的留数或残数，记作 $\mathrm{Re}\,s[f(z),a]$。

留数定理如下：

如果函数 $f(z)$ 在闭路 C 上解析，在 C 的内部除去 n 个孤立奇点 $a_1,a_2,\cdots a_n$ 外也解析，则闭路上的积分满足：

$$\int_C f(z)\mathrm{d}z = 2\pi i\sum_{k=1}^{n}\mathrm{Re}\,s[f(z),a_k]$$

如果能够求得函数 $f(z)$ 在各极点的留数的显式表达，则该闭路积分很容易获得。

由罗朗展开：若 a 是 $f(z)$ 的 m 重极点，则函数在该点的留数可以表示为：

$$\mathrm{Re}\,s[f(z),a] = \frac{1}{(m-1)!}\lim_{z\to a}\frac{\mathrm{d}^{m-1}}{\mathrm{d}z^{m-1}}[(z-a)^m f(z)]。$$

至此，通过留数定理可以把闭路积分转化为简单的代数计算。

10.3.2 MATLAB 中留数的求取

由于在工程中遇到的 $f(z)$ 多数情况下为有理分式，所以可表示为如下形式：

$$\frac{a_n z^n + a_{n-1}z^{n-1} + \ldots + a_1 z + a_0}{b_m z^m + b_{m-1}z^{m-1} + \ldots + b_m z + b_0}$$

函数 residue 可以求得该有理式的留数，residue 的返回参数有 3 个，分别对应留数向量、极点向量和高阶项。

调用格式：[r, p, k] = residue([a_n a_{n-1} ...a_0]，[b_n b_{n-1} ..b_0])

如果仅对留数感兴趣，后两个参数可省略。

【例 10-17】 留数计算实例。求函数：$\frac{2z+1}{z^2+3z+2}$ 的留数。

解：在 MATLAB 命令窗口中输入：

```
>> residue([2 1],[1 3 2])
```

输出结果为：

```
ans =
     3
    -1
```

若要求闭路积分，由留数定理，还需要知道闭路内函数的极点个数。

【例 10-18】　利用留数定理计算闭路积分实例。求闭路积分：

$$\int_C \frac{2z+1}{z^2+3z+2}\mathrm{d}x$$

其中 C 分别为：$|z|=1.5$，$|z|=3$。

解：在 MATLAB 中编写函数 ex1001 来实现，具体代码如下所示。

```
function S=ex1001(R)
%利用留数定理求取函数 2z+1/z^2+3z+2 在半径为 R 的闭路上的积分
% 闭路半径：x0
% 积分值： S
 [r,p,k]=residue([2 1],[1 3 2]);                    %r 是留数向量，p 为极点向量
sum=0;
for ii=1:length(p)
    if abs(p(ii))<R                          %闭路内的极点
        sum=sum+r(ii);                            %留数定理
    end
end
S=sum*2*pi*i;                                %积分值
```

在命令窗口中输入：

```
>> S=ex1001(3)
S =
        0 +12.5664i
>> S=ex1001(1.5)
S =
        0 - 6.2832i
```

由上述结果可知，在半径值为 1.5 的圆上的积分值为–6.2832i，在半径值为 3 的圆上的积分值为 12.5664i。

10.4　泰勒级数

泰勒级数在等式证明、近似估算、微分不等式证明及较为复杂的极限计算中有着广泛的应用。同时，泰勒级数展开在复变函数中有很重要的地位，如分析复变函数的解析性等。

设函数 $f(x)$ 在 x_0 处 n 阶可微，则 $f(x)$ 可展开为泰勒级数的形式：

$$f(x)=\sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}(x-x_0)^k+o((x-x_0)^n)$$

泰勒展开的一个特殊形式是 $x_0=0$，又称为麦克劳林展开：

$$f(x)=\sum_{k=0}^{n}\frac{f^{(k)}(0)}{k!}(x)^k+o((x)^n)$$

MATLAB 提供了泰勒展开的函数：

```
taylor (f,n,a)
```

其中，后面两个参数可以省略，因此有以下几种形式。

（1） taylor(f)：返回 f 函数的 5 次幂多项式近似。此功能函数可有 3 个附加参数，a 默认值为 0，即麦克劳林展开，返回函数为 5 次幂多项式近似；

（2） taylor(f, n)：返回（$n-1$）次幂多项式；

（3） taylor(f, a)：返回 a 点附近的幂多项式近似；

（4） taylor(r, x)：使用独立变量代替函数 findsym(f)。

下面通过实例进行讲述。

【例 10-19】 泰勒级数计算实例。求 sin(x) 的 5 阶泰勒展开。

解：在 MATLAB 命令窗口中输入：

```
>> syms x                                    %预先声明符号变量
>> taylor(sin(x))
```

输出结果为：

```
ans =
x-1/6*x^3+1/120*x^5
```

【例 10-20】 麦克劳林级数计算实例。求 sin(x) 的 7 阶麦克劳林展开。

解：在 MATLAB 命令窗口中输入：

```
>> taylor(sin(x),8)
```

输出结果为：

```
ans =
x-1/6*x^3+1/120*x^5-1/5040*x^7
```

【例 10-21】 泰勒级数计算实例之一。sin(x) 在 0.25π 处的 5 阶泰勒展开。

解：在 MATLAB 命令窗口中输入：

```
>> taylor(sin(x),pi*0.25)
```

输出结果为：

```
ans =
1/2*2^(1/2)+1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-1/4*pi)^2-1/12*2^(1/2)
*(x-1/4*pi)^3+1/48*2^(1/2)*(x-1/4*pi)^4+1/240*2^(1/2)*(x-1/4*pi)^5
```

【例 10-22】 泰勒级数计算实例之二。sin(x) 在 0.5π 处的 7 阶展开。

解：在 MATLAB 命令窗口中输入：

```
>> taylor(sin(x),8,pi*0.5)
```

输出结果为：

```
ans =
1-1/2*(x-1/2*pi)^2+1/24*(x-1/2*pi)^4-1/720*(x-1/2*pi)^6
```

若用如下命令：

```
>> taylor(sin(x),pi*0.5,8)
```

可得到相同的结果。

可见 taylor 后两个参数可交换，但两者的地位并不平等。第 2 个参数优先级更高。也就是说：当用户输入 taylor(*f*，*x*)时，如果 *x* 是整数，则按上述第二种情况处理，否则，按第三种情况处理。如果要得到整数点处的泰勒展开则需使用 3 个参数的形式。

【例 10-23】 泰勒级数计算实例之三。求 sin(*x*)的 3 阶泰勒展开以及 6 阶泰勒展开。比较不同的输入参数得到的不同结果。

解：在 MATLAB 命令窗口中输入：

```
>> taylor(sin(x),3)                          %3 作为阶数处理
ans =
x
>> taylor(sin(x),6,3)                        %3 是展开的点
ans =
sin(3)+cos(3)*(x-3)-1/2*sin(3)*(x-3)^2-1/6*cos(3)*(x-3)^3+1/24*sin(3)*(x
-3)^4+1/120*cos(3)*(x-3)^5
```

10.5　傅里叶变换

傅里叶变换是信号处理等学科的基础。通过傅里叶变换可以获取信号的频谱信息，使频域信号处理更为方便和直观。

傅里叶变换定义如下：

若函数 $f(t)$ 在 $(-\infty,\infty)$ 上满足下列条件：

（1） $f(t)$ 在任一有限区间上满足狄氏条件；

（2） $f(t)$ 在 $(-\infty,\infty)$ 上绝对可积。

则存在傅里叶变换对：

$$F(w)=\int_{-\infty}^{\infty} f(t)e^{-jwt}\mathrm{d}t$$

$$f(t)=\int_{-\infty}^{\infty} F(w)e^{jwt}\mathrm{d}w$$

10.5.1　基本傅里叶变换

MATLAB 提供求傅里叶变换的函数：

```
fourier(f,u,v)
```

其中后两个参数可以省略。下面分别举实例进行说明。

1．fourier(f)

返回以默认独立变量 *x* 对符号函数 *f* 的傅里叶（Fourier）变换，返回函数以 *w* 为默认变量。

【例 10-24】 傅里叶变换计算实例之一。求高斯分布函数 $f(t)=\frac{1}{\sqrt{2\pi}}e^{-t^2}$ 的傅里叶变换，并绘制变换前后的时域频域图。

解：在 MATLAB 命令窗口中输入：

```
>> syms f t w x F;
>> f=(1/sqrt(2*pi))*exp(-t^2);
>> F=fourier(f)                    %傅里叶变换
```

输出结果为：

```
F =
7186705221432913/18014398509481984*pi^(1/2)*exp(-1/4*w^2)
%作图
>> subplot(1,2,1),fplot('(1/sqrt(2*pi))*exp(-x^2)',[-3,3]),title('时域图');
>> hold on;
>>subplot(1,2,2),fplot('7186705221432913/18014398509481984*pi^(1/2)*exp(
-1/4*x^2)',[-3,3]),title('傅里叶变换后的频域图');
```

输出所得图形如图 10-3 所示。

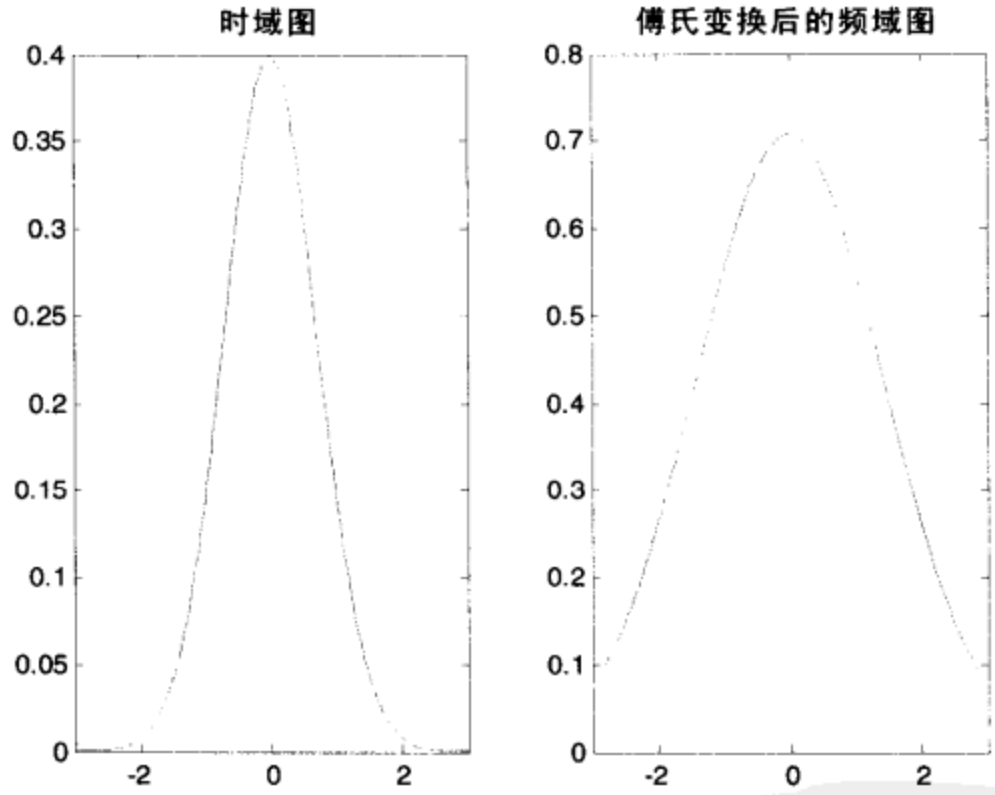

图 10-3 傅里叶变换前后的时域频域对应图

2. fourier(f,v)

变换结果为变量 v 的函数

【例 10-25】 傅里叶变换计算实例之二。求傅里叶核 $f(t)=\frac{\sin t}{\pi t}$ 的傅里叶变换，并绘制变换前后的时域频域图。

解：在 MATLAB 命令窗口中输入：

```
>>syms f t w x F;
```

```
>>f=sin(t)/(pi*t)
>>F=fourier(f,x)
```

输出结果为：

```
F =
-heaviside(x-1)+heaviside(x+1)
```

```
%作图
>> subplot(1,2,1),fplot('sin(x)/(pi*x)', [-3*pi,3*pi]),title('时域图');
>> hold on
>>    subplot(1,2,2),fplot('-heaviside(x-1)+heaviside(x+1)',[-3,3]),title
('傅里叶变换后的频域图'),axis([-3 3 0 1.5]);
```

所得图形如图 10-4 所示。

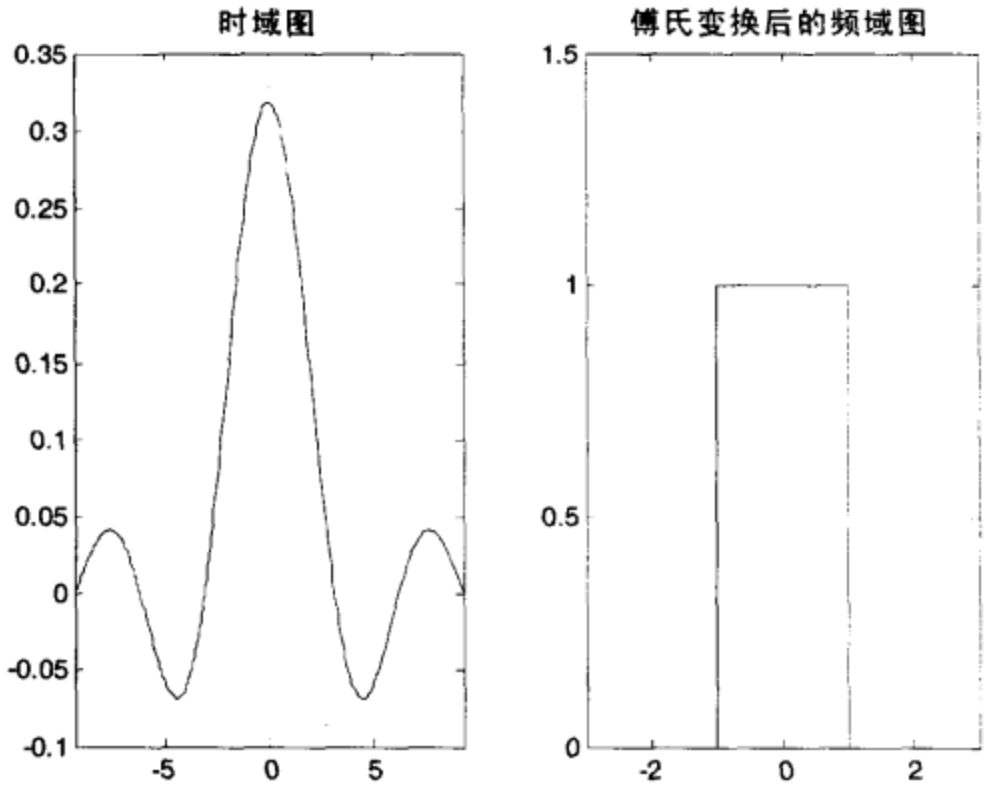

图 10-4　傅里叶变换前后的时域频域对应图

3. fourier(f,u,v)

对指定函数表达式作关于变量 u 的傅里叶变换，且变换结果为 v 的函数。

【例 10-26】　傅里叶变换计算实例之三。求下列函数关于变量 u 的傅里叶变换。

$$f(t)=\frac{\sin ut}{\pi t}$$

解：在 MATLAB 命令窗口中输入：

```
>>syms f t u v F;
>>f=sin(u*t)/(pi*t);
>>F=fourier(f,u,v)
```

输出结果为：

```
F=
1/2*i*(fourier(exp(-i*u*t),u,v)-fourier(exp(i*u*t),u,v))/pi/t
```

由于 exp(-*i**u*t)的傅里叶变换为单位冲激函数，而 fourier 函数无法求得此结果，故结

果中仍含有 fourier 的形式。最终结果应为 $F=\frac{1}{2i\pi t}[\delta(v-t)-\delta(v+t)]$。

10.5.2 傅里叶反变换

与傅里叶变换类似，傅氏反变换的 MATLAB 格式为：ifourier(*F*,*u*,*v*)，后两个参数可省略。下面举实例分别进行说明。

1. ifourier(*F*)

如果 *F*=*F*(*w*)，则返回参量为 *x* 的函数 *f*(*x*)，如果 *F*=*F*(*x*)，返回参量为 *t* 的函数 *f*(*t*)。

【例 10-27】 傅里叶反变换计算实例之一。求频谱函数 $\frac{-2}{w^2+1}$ 的傅氏反变换，并绘制变换前后的时域频域图。

解：在 MATLAB 命令窗口中输入：

```
>>syms f w u v t F;
>>F=-2/(w^2+1);
>>f=ifourier(F)
%作图
>>subplot(1,2,1),fplot('-2/(x^2+1)', [-3,3]),title('频谱函数');
>>hold on;
>>subplot(1,2,2),fplot('-2*pi*(exp(x)*heaviside(-x)+exp(-x)*heaviside
(x))',[-3,3]),title('傅氏反变换后的时域函数');
```

输出图形如图 10-5 所示。

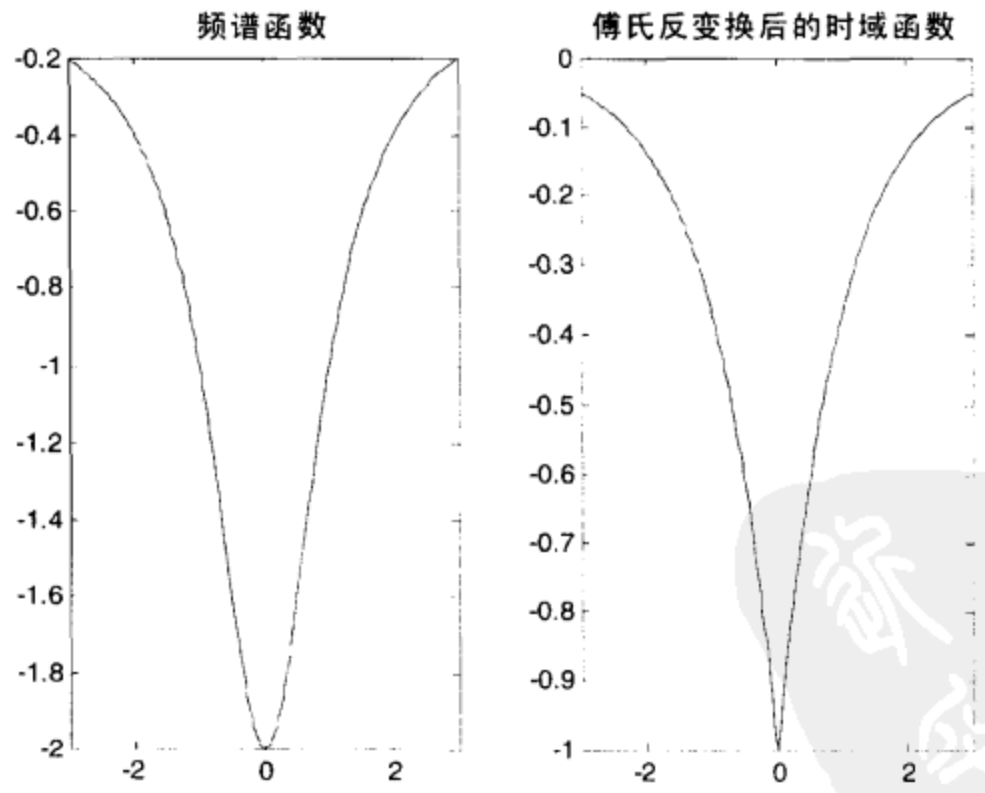

图 10-5 傅氏反变换前后的频域时域对应图

2. ifourier(*F*,*v*)

指定变换结果为 *v* 的函数。

【例 10-28】　傅里叶反变换计算实例之二。求频谱函数 $\frac{-2}{w^2}$ 的傅里叶反变换，结果返回为 v 的函数，并绘制变换前后的时域频域图。

解：在 MATLAB 命令窗口中输入：

```
>>syms f w u v t F;
>>F=-2/w^2;
>>f=ifourier(F,v)
>>f =
>>v*(2*heaviside(v)-1)
%作图
>>subplot(1,2,1),fplot('-2/(x*x)', [0.1,3]),title('频谱函数');
>>hold on;
>>subplot(1,2,2),fplot('x*(2*heaviside(x)-1)',[-3,3]),title('傅氏反变换后的时域函数');
```

输出图形如图 10-6 所示。

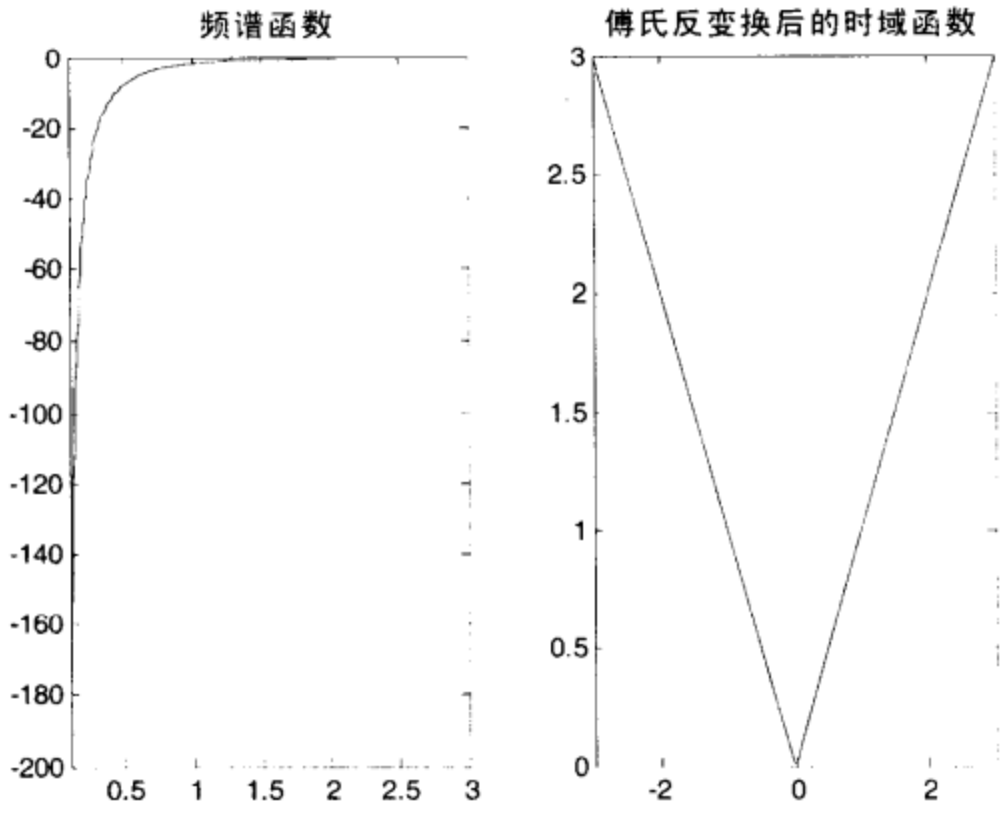

图 10-6　傅氏反变换前后的频域时域对应图

3. ifourier(*F*,*u*,*v*)

对参量 u 进行傅氏反变换，返回参量为 v 的函数。

【例 10-29】　参数傅氏反变换计算实例。求频谱为 $F=\frac{-2}{w^2+u^2}$ 的函数对 u 的傅里叶变换，返回参量为 v 的函数。

解：在 MATLAB 命令窗口中输入：

```
>>syms f w u v t F;
>>F=-2/(w^2+u^2);
>>f=ifourier(F,u,v)
>>f =
```

输出结果为：

```
1/w*(-signum(0,Re(w),0)*cosh(w*v)-sinh(w*v)+2*heaviside(v)*sinh(w*v))
```

10.6 拉普拉斯变换

拉普拉斯变换又称为拉氏变换，是一种常用的积分变换。它能够把分析运算化为代数运算。在物理、力学以及工程技术中都有广泛的应用。

拉氏变换可由傅里叶变换引出。

傅里叶变换的时间范围是：$(-\infty,\infty)$，而实际研究许多过程时，关心的是从某个时刻，例如：t=0 开始后的物理状态，而常常假设此时刻以前的状态为 0。用数学式表达即为：

$$f(t)=\begin{cases} f(t), & t\geqslant 0 \\ 0, & t<0 \end{cases}$$

此时傅立叶变换可表示为：

$$F(s)=\int_0^{\infty} f(t)\mathrm{e}^{-ist}\mathrm{d}t$$

令 $is=p$，即得到拉氏变换公式：

$$F(p)=\int_0^{\infty} f(t)\mathrm{e}^{-pt}\mathrm{d}t$$

相应地，其反变换为：

$$f(t)=\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} F(p)\mathrm{e}^{pt}\mathrm{d}p$$

拉氏变换及其反变换的求法包括部分分式法，以及留数定理法等。

相对傅里叶变换，拉氏变换的适用范围更广。因为傅里叶变换中的绝对可积概念比较强，很多简单的函数如：t^n,sin(t)， cos(t) 都不满足这个条件。而引入拉氏变换以后，这些函数的变换就存在了。

10.6.1 基本拉氏变换

MATLAB 提供求函数的拉氏变换函数 laplace(F,t,w)。

其中后两个参数可以省略，下面举实例说明具体用法。

1. laplace(F)

F 的拉氏变换，返回默认为 s 的函数。

【例 10-30】 拉氏变换计算实例之一。求函数 $f(x)=\sin(x)+x^2$ 的拉氏变换。

解：在 MATLAB 命令窗口中输入：

```
>> syms F x;
>> F=sin(x)+x^2;
```

```
>> laplace(F)
```

输出结果为：

```
ans =
1/(s^2+1)+2/s^3
```

编写.m 文件 ex1003 作出拉氏变换的网格图：

```
function ex1003()
x1=-0.2:0.03:0.2;
y1=-0.2:0.03:0.2;
[x,y]=meshgrid(x1,y1);
s=x+i*y;
f=1./(s^2+1)+2./s^3;
mesh(x,y,f);
surf(x,y,f);
colormap(hsv);
```

输出图形如图 10-7 所示。

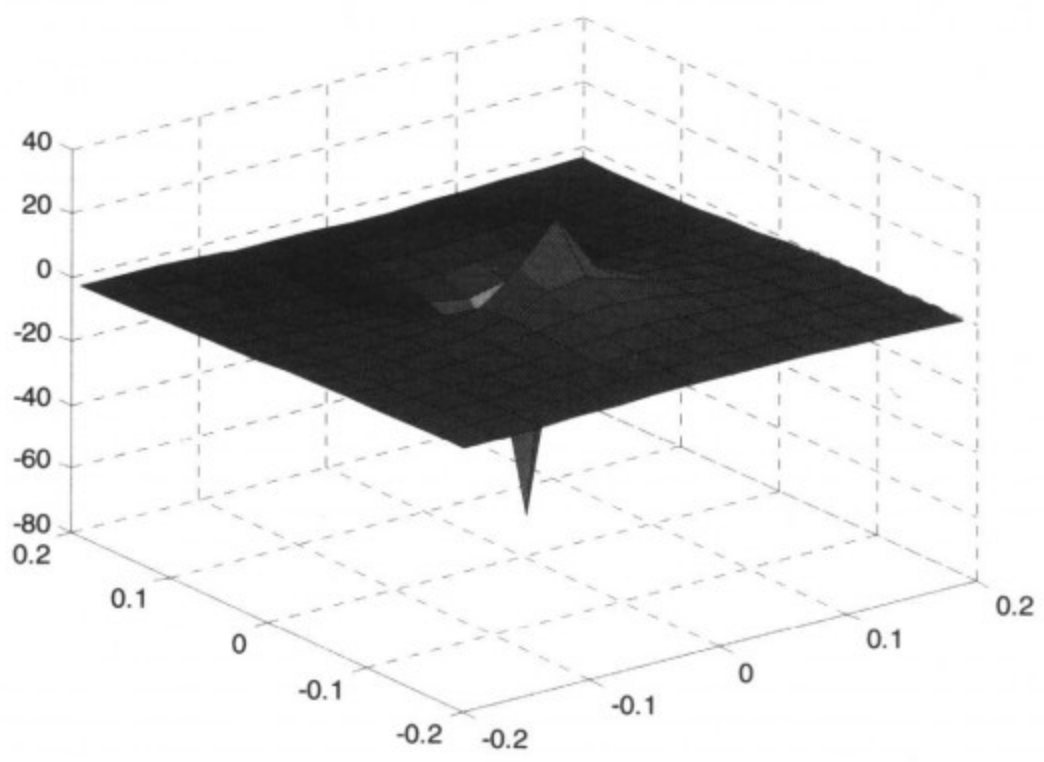

图 10-7　sin(*x*)+*x*^2 拉氏变换后的网格图

2. laplace(*F*,*t*)

拉氏变换结果为 t 的函数。

【例 10-31】　拉氏变换计算实例之二。将函数 $f(x)=\sin(x)+x^2$ 的拉氏变换表示成 t 的函数。

解：在 MATLAB 命令窗口中输入：

```
>> syms t x F;
>> F=sin(x)+x^2;
>> laplace(F,t)
```

输出结果为：

```
ans =
1/(t^2+1)+2/t^3
```

3. laplace(*F*,*t*,*w*)

对 *t* 进行拉氏变换，且变换结果为 *w* 的函数。

【例 10-32】 拉氏变换计算实例之三。求如下函数关于 *t* 的拉氏变换：

$$f(x,t)=(x-t)^2+2(x-t)$$

解：在 MATLAB 命令窗口中输入：

```
>> syms t x F w;
>> F=(x-t)^2+2*(x-t);
>> laplace(F,t,w)
```

输出结果为：

```
ans =
x^2/w-2*x/w^2+2/w^3+2*x/w-2/w^2
```

10.6.2 拉氏反变换

拉氏反变换可由 ilaplace(*L*, *y*, *x*)求得。与正变换类似，后两个参数也可以省略。为了对比说明，取正变换的结果作为输入验证反变换的结果。

1. ilaplace(*L*)

返回以默认独立变量 *s* 的数量符号 *L* 的 Laplace 变换，默认返回 *t* 的函数。

【例 10-33】 拉氏反变换计算实例之一。求【例 10-30】的结果并进行拉氏反变换。

解：在 MATLAB 命令窗口中输入：

```
>> syms s L;
>> L=1/(s^2+1)+2/s^3;
>> ilaplace(L)
```

输出结果为：

```
ans =
sin(t)+t^2
```

2. ilaplace(*L*,*y*)

以 *y* 代替默认的 *t* 的函数。

【例 10-34】 拉氏反变换计算实例之二。求 $F(s)=\frac{1}{s^2+1}+\frac{1}{s^3}$ 的拉氏反变换，并将结果表示为 *y* 的函数。

解：在 MATLAB 命令窗口中输入：

```
>> syms s L y;
>> L=1/(s^2+1)+2/s^3;
>> ilaplace(L,y)
```

输出结果为：

```
ans =
```

```
sin(y)+y^2
```

3. ilaplace(*L*,*y*,*x*)

以 *x* 代替 *t* 的函数。

【例 10-35】 拉氏反变换计算实例之三。求如下表达式关于 *w* 的拉氏反变换：

$$L=\frac{x^2}{w}-\frac{2x}{w^2}+\frac{2}{w^3}+\frac{2x}{w}-\frac{2}{w^2}$$

解：在 MATLAB 命令窗口中输入：

```
>> syms x L w t;
>> L=x^2/w-2*x/w^2+2/w^3+2*x/w-2/w^2;
>> ilaplace(L,w,t)
```

输出结果为：

```
ans =
x^2-2*x*t+t^2+2*x-2*tx  %等价于 x^2+2*x
```

10.7　小结

本章首先讲述了 MATLAB 中复数的基本运算，然后讲述了留数的基本运算、利用留数求闭路积分和泰勒级数计算，以及在复变函数领域中应用广泛的傅里叶积分变换和拉普拉斯积分变换。

第 11 章

概率统计计算

概率论与数理统计是一门研究随机现象并找出其统计规律的学科，广泛应用于社会、经济、科学等各个领域。MATLAB 中的统计工具箱（Statistics Toolbox）包含了 200 多个用于概率统计方面的功能函数，且具有简单的接口操作。

通过本章，读者能利用统计工具箱中相关函数进行统计量计算、参数估计、假设检验、方差分析和回归分析等计算，而且还能使用统计学中的一些特殊的绘图工具，直观地认识样本的结构和分布特征。

11.1 MATLAB 统计工具箱介绍

MATLAB 中的统计工具箱是一套建立在 MATLAB 数值计算环境下的统计分析工具，能够支持范围广泛的统计计算任务。

统计工具箱主要包括如下两类工具。

（1）概率统计的数值计算函数

该函数可以通过命令行或用户的应用程序调用，多数为 MATLAB 中的 M 文件。

（2）交互式图形工具函数

该函数是能够通过图形用户界面（GUI）来使用的交互式图形工具。

表 11-1 给出了统计工具箱中函数的主要分类。

表 11-1 统计工具箱中函数的主要分类

函数分类	具体描述
概率分布	参数估计
	累积分布函数（cdf）
	概率密度函数（pdf）

续表

函数分类	具体描述
概率分布	逆累积分布函数
	随机数产生器
	分配函数的矩
实验设计	构建统计模型
主成分分析	对数据进行降维处理
假设检验	假设的统计检验
描述统计学	针对数据抽样的描述统计
统计作图	统计的作图
统计的过程控制	统计的过程控制
聚集分析	有相似的特性的项目列入群体中
线性模型	对数据进行线性模型拟合
非线性回归	非线性回归模型的拟合
文件输入/输出	对操作系统文件读写数据
数据	示范的数据

11.2 随机变量的数字特征

随机变量的数字特征是与随机变量有关的某些数值，它们虽然不能完整地描述随机变量，但能描述随机变量在某些方面的重要特征。本节主要讲述用 MATLAB 求解随机变量的期望（均值）、方差、矩、协方差矩阵、相关系数等数字特征。该部分所用到的命令函数属于统计工具的描述统计学（Descriptive Statistics）部分。

11.2.1 期望

设离散性随机变量的分布律为：

$$P\{X=x_k\}=p_k, k=1,2,\cdots$$

则定义 X 的期望为：

$$E(X)=\sum_{k=1}^{\infty}x_k p_k$$

而对于来自总体 X 的一个样本，设其样本值为 $x=(x_1,x_2,\cdots,x_n)$，则定义样本均值为：

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$$

可以证明 $\bar{x}$ 依概率收敛于 X 的均值。因此通常用 $\bar{x}$ 来近似代替 X 的均值。MATLAB 中提供了函数 mean（）来计算这种较为简单的均值，其调用格式如下。

1. m = mean(X)

若 X 为向量形式，则 mean(X)为向量中各元素的平均值；

若 X 为矩阵，则 mean(X)为一个行向量，其中的每一个元素是矩阵 X 各列元素的平均值。

2. m = mean(X,dim)

函数返回 X 在某一维上的均值，其中维数由标量 dim 设置。

例如 X 为一个矩阵时，mean(X,2)是一个列向量，包含矩阵 X 每一行的均值。dim 的默认值为 1。

【例 11-1】 样本均值计算实例之一。下面给出一组样本值来自于某一个未知分布，计算该分布的样本均值。

[19.1,13.5,12.6,10,14.6,24,13.7,15,20,13.5]

解：通过在 MATLAB 中编写程序 ex1101.m 来实现。

```
%ex1101.m 用 mean () 计算样本均值
x=[19.1,13.5,12.6,10,14.6,24,13.7,15,20,13.5];
m=mean(x)
```

命令窗口中输出结果为：

```
>> ex1101
m =
   15.6000
```

由以上计算结果可知，该分布的样本均值为 15.6。

【例 11-2】 样本均值计算实例之二。如下为来自于总体的三个样本的样本值，分别计算其均值。

X_1：	2.1	3.5	5.0	1.5	3.0	3.2	4.2	2.5	3.0	1.8
X_2：	3.2	3.0	4.5	4.2	1.4	1.5	5.0	4.5	3.5	3.0
X_3：	2.4	2.6	4.2	3.5	3.2	3.3	4.0	4.2	3.6	2.8

解：用矩阵的均值形式计算，在 MATLAB 中编写程序 ex1102.m 来实现。

```
%ex1102.m 分别计算三个样本的均值
X1=[2.1  3.5  5.0  1.5  3.0  3.2  4.2  2.5  3.0  1.8];   %第一个样本
X2=[3.2  3.0  4.5  4.2  1.4  1.5  5.0  4.5  3.5  3.0];   %第二个样本
X3=[2.4  2.6  4.2  3.5  3.2  3.3  4.0  4.2  3.6  2.8];   %第三个样本
X=[X1;X2;X3];                                            %将三个样本作为 X 的各行
m1=mean(X,2)                                             %计算 X 各行的均值，即得各样本均值
X=[X1',X2',X3'];                                         %将三个样本作为 X 的各列
m2=mean(X)                                               %得各列均值
```

命令窗口输出为：

```
> ex1102
m1 =
    2.9800
    3.3800
    3.3800
```

```
m2 =
    2.9800    3.3800    3.3800
```

由以上计算结果可知，样本 *X*1 的均值为 2.98，样本 *X*2 的均值为 3.38，样本 *X*3 的均值为 3.38。

【例 11-3】 样本均值计算实例之三。设随机变量 *X*、*Y* 的联合分布律如表 11-2 所示，分别计算 *E*(*X*)，*E*(*Y*)，*E*(*X*+3)以及 *E*(*X*+*Y*)。

表 11-2　*X*、*Y* 联合分布律

Y / *X*	1	2	3	4	*P*(*Y*=*j*)
0	1/10	0	0	0	1/10
1	0	4/10	2/10	1/10	7/10
2	0	0	0	2/10	2/10
P(*X*=*i*)	1/10	4/10	2/10	3/10	1

解：用均值定义计算，通过在 MATLAB 中编写程序 ex1103.m 来实现，具体代码如下所示。

```
%ex1103.m 分别计算二维离散分布的均值
X=[0,1,2];
Y=[1,2,3,4];
Px=[0.1,0.7,0.2];                    %X 各点对应的概率
Py=[0.1,0.4,0.2,0.3];                %Y 各点对应概率
mX=sum(X.*Px)                        %E(X)  sum 为求和函数。
mY=sum(Y.*Py)
z=X+3;
mX3=sum(z.*Px)                       %E(X+3)
mXY=mX+mY                            %E(X+Y)
```

命令窗口输出为：

```
>> ex1103
mX =
    1.1000                           %E (x)
mY =
    2.7000                           %E(Y)
mX3 =
    4.1000                           %E(X+3)
mXY =
    3.8000                       %E(X+Y)
```

在 MATLAB 中还给出了求其他形式均值的一些函数，其命令形式与 mean 相同。各命令具体含义详见表 11-3。

表 11-3　几种均值函数

函 数 名	功　能
geomean	样本几何均值
harmmean	样本调和平均值
nanmean	可忽略样本中的非数值输入，并计算相应均值
trimmean	除去输入样本中一些数值变化太大的值，然后再计算样本均值

如需了解表 11-3 中函数的详细用法，可在 MATLAB 的 Help 中的相关部分检索查询，查看函数的详细使用说明。

11.2.2 方差、标准差、矩

方差是用来刻画随机变量 X 取值分散程度的一个量。其一般用下式表达：

$$D(X)=E\{[x-E(x)]^2\}$$

在应用上还引入与随机变量 X 具有相同量纲的量 $\sqrt{D(X)}$，记为 $\sigma(X)$，称为标准差或均方差。

X 的 k 阶中心矩应为：

$$E\{[X-E(X)]^k\},k=2,3,\cdots$$

可知方差即为二阶中心矩。

对于一个样本来说，样本方差通常有两种形式表达，如下所示。

无偏估计式：

$$S^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\overline{x})^2$$

有偏估计式：

$$S^2=\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2$$

样本标准差也对应有如下两种形式：

$$S=\sqrt{S^2}=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\overline{x})^2}$$

或

$$S=\sqrt{S^2}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2}$$

样本的 k 阶中心矩为：

$$B_k=\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^k,k=2,3\cdots$$

MATLAB 中用方差函数 var()来计算样本方差，用标准差函数 std()来计算样本的标准差，用矩函数 moment()计算样本的各阶中心矩，具体用法如下。

1．var()方差函数

$V=\mathrm{var}(X)$　　若 X 为向量则返回向量的样本方差值，若 X 为矩阵则返回矩阵各列向量方差组成的行向量。其采用无偏式计算方差。

$V=\mathrm{var}(X,1)$　　函数采用有偏估计式计算 X 的方差，即前置因子为 $1/n$。var(X,0)等同于 var(X)，其采用无偏式计算方差，前置因子为 $1/(n-1)$。

$V=\mathrm{var}(X,w)$　　函数返回 X 以 w 为权的方差。对于矩阵 X，w 的元素个数须等于 X 的

行数；对于向量 X，w 的元素个数与 X 元素个数相同。

V = var(X, flag, dim)函数返回 X 在特定维上的方差，dim 指定维数，flag 指定选择的计算式。即 flag=0，选择无偏式计算；flag=1，选择有偏式计算。

此外，nanvar()是忽略非数（NaNs）的方差计算函数，调用格式与 var()相同。

2．std()标准差函数

s = std(X)　　函数返回向量（矩阵）X 的标准差（前置因子 1/n-1）。

s = std(X, flag)　　flag=0，前置因子为 1/n-1；flag=1，前置因子为 1/n。

s = std(X, flag, dim)　　函数返回 X 在特定维上的标准差，dim 指定维数，flag 指定选择的计算式。

nanstd()函数用法同 std()，忽略非数的标准差计算函数。

【例 11-4】 样本方差、标准差计算实例。分别计算向量 x=[-1,-1,1,2]的方差和标准差以及加权后的方差和标准差，其中加权向量为 w= [1,2,3,4]。(加权方差和标准差值指的是计算样本方差或标准差时，各样本值要乘以相应的权值，然后再除以总的权值。)

解：通过在 MATLAB 中编写程序 ex1104.m 来实现，具体代码如下所示。

```
%ex1104.m 计算 x 的方差和标准差
x=[-1 -1 1 2];                          %输入向量 x
w=[1 2 3 4];                            %权向量 w
%用各种命令形式计算方差
v1=var(x)
v2=var(x,0)
v3=var(x,1)
v4=var(x,w)
v5=var(x,1,w)
%用各种命令形式计算向量标准差
s1=std(x)
s2=std(x,1)
s3=std(x,w)
```

在命令窗口中运行该文件，得如下结果：

```
>>  ex1104
v1 =
    2.2500    %var(X)
v2 =
    2.2500    %var(X,0)
v3 =
    1.6875    %var(X,1)
v4 =
    1.5600    %var(X,w)
v5 =
     0     0     0     0    %var(X,1,w)
s1 =
    1.5000    %std(X)
s2 =
    1.2990    %std(X,1)
s3 =
    1.2490     %std(X,w)
```

3. moment()矩函数

m=moment(*X*, order)　函数返回向量（矩阵）***X*** 的 *k* 阶中心矩。order 规定中心矩的阶数。

m=moment(*X*, order, dim)　函数返回 dim 维上的 ***X*** 的中心矩。

【例 11-5】　中心矩计算实例。随机产生正态分布的一组随机数，计算其 3 阶中心矩。

解：通过在 MATLAB 中编写程序 ex1105.m 来实现。

```
%ex1105.m 计算正态随机数的 3 阶矩
X=randn([5 4])                    %产生正态分布的随机数矩阵，函数为 5，列数为 4
m1=moment(X,3)                    %计算矩阵 x 各列的 3 阶矩
m2=moment(X,3,2)                  %计算矩阵 x 各行的 3 阶矩
>> ex1105
X =
   -0.4326    1.1909   -0.1867    0.1139
   -1.6656    1.1892    0.7258    1.0668
    0.1253   -0.0376   -0.5883    0.0593
    0.2877    0.3273    2.1832   -0.0956
   -1.1465    0.1746   -0.1364   -0.8323
m1 =
   -0.1135    0.0362    0.8780    0.0586
m2 =
    0.1983
   -1.7088
   -0.0227
    0.7167
   -0.0004
```

11.2.3　协方差、相关系数

随机变量 *X*、*Y* 的协方差和相关系数的定义式如下：

$$\mathrm{cov}(x,y)=E\{[x-E(x)][y-E(y)]\}$$

$$\mathrm{cof}(x,y)=\frac{\mathrm{cov}(x,y)}{\sqrt{D(x)}\sqrt{D(y)}}$$

对于 *n* 维随机变量，通常用协方差矩阵描述它的 2 阶中心矩。如对于二维随机变量(*X*,*Y*)，定义协方差矩阵形式为：

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

其中：

$$c_{11}=E\{[x-E(x)]^2\}$$
$$c_{12}=E\{[x-E(x)][y-E(y)]\}$$
$$c_{21}=E\{[y-E(y)][x-E(x)]\}$$
$$c_{22}=E\{[y-E(y)]^2\}$$

其相应的样本协方差形式此处不再给出。与样本方差形式类似，也具有两种形式。

MATLAB 中用 cov()和 corrcoef()计算协方差和相关系数，具体用法如下。

1．cov()协方差函数

C = cov(X)　X 为向量时，函数返回此向量的方差。X 为矩阵时，矩阵的每一行表示一组观察值，每一列代表一个变量。函数返回此矩阵的协方差矩阵，其中协方差矩阵的对角元素是 X 矩阵的列向量的方差值。

C = cov(X,Y)　返回 X、Y 的协方差矩阵，其中 X、Y 行数和列数相同。

C=cov(X,1)，C=cov(X,Y,1)　计算协方差矩阵时前置系数取 1/n。cov(X,0)与 cov(X)相同，都是取前置系数为 1/n-1，此用法可参考 var 函数。

2．corrcoef()相关系数函数

R=corrcoef(X)　返回矩阵 X 的相关系数矩阵，其各点值对应于相关矩阵的各点值除以相应的标准差。

R = corrcoef(x,y)　返回 x、y 的相关系数矩阵。若 x、y 分别为列向量，则该命令等同于 R=corrcoef([x y])。

[R,P] =corrcoef(...)　返回的 P 矩阵是不相关假设检验的 p 值。

[R, P, RLO, RUP] =corrcoef(...)　对于每一个 R 值，返回的 95%置信区间为[RLO,RUP]。

[...] = corrcoef(..., 'param1', val1, 'param2', val2,...)　设定特殊的可选项。

corrcoef 函数功能强大，本节仅讨论它的前两种用法。

【例 11-6】 协方差和相关系数计算实例。计算向量 x 和向量 y 的协方差和相关系数。向量 x 与向量 y 分别如下所示：

x= [1,2,2,2,1,2]

y= [2,3,2,2,3,2]

解：通过在 MATLAB 中编写程序 ex1106.m 来实现，具体代码如下所示。

```
%ex1106.m 计算向量之间的协方差和相关系数
x=[1,2,2,2,1,2];
y=[2,3,2,2,3,2];
cx=cov(x)                    %计算向量 x 的协方差，可知其等于 x 方差
vx=var(x)                    %x 方差
cxy=cov(x,y)                 %x、y 协方差
cor=corrcoef(x,y)            %x、y 相关系数
>> ex1106                    %运行 ex0806
cx =
    0.2667
vx =
    0.2667
cxy =
    0.2667   -0.0667
   -0.0667    0.2667
cor =
    1.0000   -0.2500
   -0.2500    1.0000
```

【例 11-7】 随机变量数字特征综合计算实例。根据表 11-4 中所示的数据计算各变量的均值、方差以及它们之间的协方差矩阵和相关系数。

表 11-4　几组变量数据

VOL	HP	MPG	SP
89	49	65. 4	96
92	55	55. 9	97
92	70	49. 0	105
92	53	46. 5	96
89	70	46. 2	105
92	55	45. 4	97
94	80	43. 4	107
89	73	39. 3	103
89	66	39. 6	100
91	78	38. 9	106

解：通过在 MATLAB 中编写程序 ex1107.m 来实现，具体代码如下所示。

```
%ex1107.m 计算各变量均值、方差以及它们之间的协方差矩阵和相关系数矩阵
VOL=[89 92 92 92 89 92 94 89 89 91]';
HP=[49 55 70 53 70 55 80 73 66 78]';
MPG=[65.4 55.9 49.0 46.5 46.2 45.4 43.4 39.3 39.6 38.9]';
SP=[96 97 105 96 105 97 107 103 100 106]';
X=[VOL,HP,MPG,SP];                    %将变量数据按列存到一个矩阵中
m=mean(X)                             %计算 X 均值，得各变量样本的均值
v=var(X)                              %计算 X 方差，得各个变量样本值之间方差
cX=cov(X)                             %计算变量之间的相关矩阵
cv=diag(cX)                           %取相关矩阵对角元素
corX=corrcoef(X)                      %计算变量之间的相关系数
```

输出计算结果为：

```
>> ex1107
m =
   90.9000   64.9000   46.9600  101.2000
v =
    3.2111  123.2111   68.6471   19.9556
cX =
    3.2111    2.4333   -0.7822    1.0222
    2.4333  123.2111  -64.8600   47.8000
   -0.7822  -64.8600   68.6471  -20.4356
    1.0222   47.8000  -20.4356   19.9556
cv =
    3.2111
  123.2111
   68.6471
   19.9556
corX =
    1.0000    0.1223   -0.0527    0.1277
    0.1223    1.0000   -0.7052    0.9640
   -0.0527   -0.7052    1.0000   -0.5521
    0.1277    0.9640   -0.5521    1.0000
```

11.2.4　偏斜度和峰度

偏斜度和峰度是用来描述随机变量分布的形状与对称形式，或正态分布型的偏离程度的统计量，其分别定义如下。

随机变量 x 的偏斜度：

$$y_1 = \frac{E(x-E(x))^3}{[D(x)]^{3/2}}$$

随机变量 x 的峰度：

$$y_1 = \frac{E(x-E(x))^4}{[D(x)]^4}$$

在 MATLAB 中分别用函数 skewness()、kurtosis()来实现上述计算。

1．skewness 函数

调用格式如下：

Y = skewness(X)，X 为向量时，函数返回此向量的倾斜度；X 为矩阵时，矩阵的每一行表示一组观察值，每一列代表一个变量，函数返回此矩阵的每一列的倾斜度。

y = skewness(X, flag)，flag 用于设定是否要修正偏差。若 flag=0，则计算倾斜度时修正系统偏差；flag=1（默认值）时不修正偏差。

y = skewness(X, flag, dim)，在矩阵 X 的 dim 维上计算倾斜度。通常对于二维矩阵，dim=1，函数计算各列的倾斜度；dim=2，函数计算各行的倾斜度。

2．kurtosis 函数

调用格式如下：

k = kurtosis(X)，函数返回 X 的峰度。

k = kurtosis(X, flag)，flag 意义同 skewness。

k = kurtosis(X, flag, dim)，参数意义同 skewness。

通过上面各式可以计算出向量（矩阵）X 的偏斜度和峰度。

式中 flag 默认值为 1 表示不进行偏移修正；flag 可以赋值为 0，表示进行偏移修正。dim 指定在矩阵 X 的特定维上求相应统计量。

下面是一段在 MATLAB 命令窗口中输入的计算偏斜度和峰度的例子：

```
>> x=[20.1,12.4,22,2,18.0,24.0,25.2,20.0,21.2];     %输入向量 x
>> y=[10.2,9.4,8.5;12.1,10.3,11.0;12.1,13.4,11.4]; %输入矩阵 y
>> skewness(x)                                      %求 x 偏斜度
ans =
   -1.4381
>> skewness(x,1)                                    %flag=1
ans =
   -1.4381
>> skewness(x,0)                                    %flag=0
ans =
```

```
   -1.7433
>> kurtosis(x)                                    %求 x 峰度
ans =
    4.0773
>> skewness(y)                                    %求 y 偏斜度
ans =
   -0.7071    0.5636   -0.6559
>> skewness(y,1,2)                                %flag=1,dim=2
ans =
   -0.0719
    0.2641
    0.3480
>> kurtosis(y)                                    %求 y 峰度，flag 默认 1，dim 默认 1
ans =
    1.5000    1.5000    1.5000
```

11.2.5 其他数字特征

除了上面介绍的几类重要的数字特征外，MATLAB 的统计工具箱中还有很多对数据性质进行描述的函数。下面通过一些表格对这些函数进行简单介绍。表 11-5 所示给出数据比较和简单计算类函数；表 11-6 给出了几种特殊的统计学函数。

表 11-5 数据比较和简单计算类函数

函 数 名	功 能
max	求样本中的最大值元素
nanmax	忽略样本中的非数求最大值元素
min	求样本中的最小值元素
nanmin	忽略样本中的非数求最小值元素
median	求随机变量的中值
nanmedian	忽略样本中的非数求中值
sum	求样本元素的累和
nansum	忽略样本中非数求和
cumtrapz	梯形累和函数
cumsum	求此样本值之前的元素和
mad	计算样本绝对偏差的均值或中值，由参数设定
sort	将样本值按从小到大排列
sortrows	将样本的行作为一个整体进行排序
range	求样本值中最大值和最小值之差
prctile	求样本的在不同分位数上的值
iqr	求四分位差值，即样本在 75%分位点与 25%分位点上取值之值

表 11-6 一些特殊函数

函 数 名	功 能
jackknife	计算 jackknife 样本的特定统计量
bootstrp	通过对数据进行 bootstrap 重采样计算相应的统计量
bootci	计算 bootstrap 统计量的置信区间
tabulate	以矩阵形式给出样本值的频数表

11.3　随机变量的数字特征

概率统计类课程中曾给出一些具有特殊分布的随机变量，例如均匀分布、二项分布、正态分布和泊松分布等。MATLAB 中也有专门的对这些特殊分布进行操作处理的函数。

MATLAB 7.0 的统计工具箱支持 20 多种特殊概率分布。对于每种分布，该工具箱都提供相应的用于计算其常用统计量的 5 种基本函数。它们分别是：概率密度函数、累积分布函数、逆累积函数、均值和方差函数以及随机数发生器，下面分别进行讲述。

11.3.1　概率密度函数（pdf）

表 11-7 给出了 MATLAB 7.0 支持的 20 多种分布以及它们名称的字母缩写。MATLAB 可以识别这些字母所代表的分布，并进行相应的计算或操作。

表 11-7　MATLAB 支持的分布类型

分布名称	字母缩写	分布名称	字母缩写
β 分布	beta	负二项分布	nbin
二项分布	bino	非中心 F 分布	ncf
χ^2 分布	chi2	非中心 t 分布	nct
指数分布	exp	非中心 χ^2 分布	ncx2
极值分布	ev	正态分布	norm
F 分布	f 或 F	泊松分布	poiss
γ 分布	gam	瑞利分布	rayl
几何分布	geo	t 分布	t
广义极值分布	gev	均匀分布	unif
广义帕累托分布	gp	离散均匀分布	unid
超几何分布	hyge	韦伯分布	wbl
对数正态分布	logn		

MATLAB 中求概率密度有一个通用的函数 pdf()，通过此函数可以求表 11-7 中所示的 23 种输入分布的概率密度函数，此函数的调用格式如下：

$$\boldsymbol{Y} = \text{pdf('name'}, \boldsymbol{X}, \boldsymbol{A}1, \boldsymbol{A}2, \boldsymbol{A}3)$$

式中，name 为表 11-7 中分布的字母缩写，$\boldsymbol{X}$ 为样本矩阵，$\boldsymbol{A}1$、$\boldsymbol{A}2$ 和 $\boldsymbol{A}3$ 是分布参数矩阵，$\boldsymbol{Y}$ 为概率密度矩阵。

对于某些分布，有些参数矩阵可以不必输入。$\boldsymbol{X}$、$\boldsymbol{A}1$、$\boldsymbol{A}2$ 和 $\boldsymbol{A}3$ 必须是具有同样的大小的矩阵，当其中之一输入为标量时，程序会将其调整为与其他输入同维的矩阵。

另外，MATLAB 对于每一种分布还有一个专用的求其概率密度的函数。它们的基本用法和含义见表 11-8。

表 11-8 概率密度函数

分布名称	概率密度函数	调用格式
β 分布	betapdf	*Y*=betapdf(*X*,*A*,*B*)
二项分布	binopdf	*Y*=binopdf(X,N,P)
χ^2 分布	chi2pdf	*Y*=chi2pdf(*X*,*V*)
混合分布	copulapdf	*Y*=copulapdf('Gaussian',U,rho) ;*Y*=copulapdf('*t*',*U*,rho,Nu) *Y*=copulapdf(family, *U*,alpha
极值分布	evpdf	*Y*=evpdf(*X*,mu,sigma)
指数分布	exppdf	*Y*=exppdf（*X*,mu）
F 分布	fpdf	*Y*=fpdf(*X*,*V*1,*V*2)
γ 分布	gampdf	*Y*=gampdf(*X*,*A*,*B*)
几何分布	geopdf	*Y*=geopdf(*X*,*P*)
广义极值分布	gevpdf	*Y*=gevpdf(*X*,*K*,sigma,mu)
广义帕累托分布	gppdf	*P*=gppdf(*X*,*K*,sigma,theta)
超几何分布	hygepdf	*Y*=hygepdf(*X*,*M*,*K*,*N*)
对数正态分布	lognpdf	*Y*=lognpdf(*X*,mu,sigma)
多项式分布	mnpdf	*Y*=mnpdf(*X*,PROB)
多元正态分布	mvnpdf	*Y*=mvnpdf(*X*) ;*Y*=mvnpdf(*X*,mu) ;*Y*=mvnpdf(*X*,mu,sigma)
多元 *t* 分布	mvtpdf	*Y*=mvtpdf(*X*,*C*,*df*)
负二项分布	nbinpdf	*Y*=nbinpdf(*X*,*R*,*P*)
非中心 *F* 分布	ncfpdf	*Y*=ncfpdf(*X*,NU1,NU2,DELTA)
非中心 *t* 分布	nctpdf	*Y*=nctpdf(*X*,*V*,DELTA)
非中心 χ^2 分布	ncx2pdf	*Y*=ncx2pdf(*X*,V,DELTA)
正态分布	normpdf	*Y*=normpdf(*X*,mu,sigma)
泊松分布	poisspdf	*Y*=poisspdf(*X*,LAMBDA)
瑞利分布	raylpdf	*Y*=raylpdf(*X*,*B*)
学生 *t* 分布	tpdf	*Y*=tpdf(*X*,*V*)
离散均匀分布	unidpdf	*Y*=unidpdf(*X*,*N*)
连续均匀分布	unifpdf	*Y*=unifpdf(*X*,*A*,*B*)
韦伯分布	wblpdf	*Y*=wblpdf(*X*,*A*,*B*)

上述函数的输出均为对应分布在 *X* 处的概率密度。后面的几个输入参数是描述分布的参数矩阵。要注意的是输入的参数的维数必须相等，否则会导致错误结果。

【例 11-8】 概率密度计算实例。计算坐标[0,2]上的均匀分布在 0.4，0.6，1.4，1.6 处的概率密度值。

解：可以分别用通用函数 pdf 和专用函数 unifpdf 求解。在 MATLAB 中编写程序 ex1108.m 来实现，具体代码如下所示。

```
%ex1108.m 求[0,2]均匀分布在某点的概率密度值
clear all
```

```
x=[0.4,0.6,1.4,1.6];
f1=pdf('unif',x,0,2)                    %用 pdf()计算
f2=unifpdf(x,0,2)                       %用 unifpdf()计算
```

结果为：

```
>> ex1108
f1 =
    0.5000    0.5000    0.5000    0.5000
f2 =
    0.5000    0.5000    0.5000    0.5000
```

【例 11-9】 概率密度图绘制实例。作出 N(0,1)，N(0,4)，N(1,4)的概率密度图。

解：通过在 MATLAB 中编写程序 ex1109.m 来实现，具体代码如下所示。

```
%ex1109.m 绘制正态分布的概率密度图
clear all
x=[-10:0.1:10];
f1=normpdf(x,0,1);                          %N(0,1)pdf
f2=normpdf(x,0,2);              %N(0,4)pdf
f3=normpdf(x,1,2);                          %N(1,4)pdf
plot(x,f1,'b*',x,f2,'ro',x,f3,'g+')
legend('X~N(0,1)','X~N(0,4)','X~N(1,4)')
```

计算结果如图 11-1 所示。

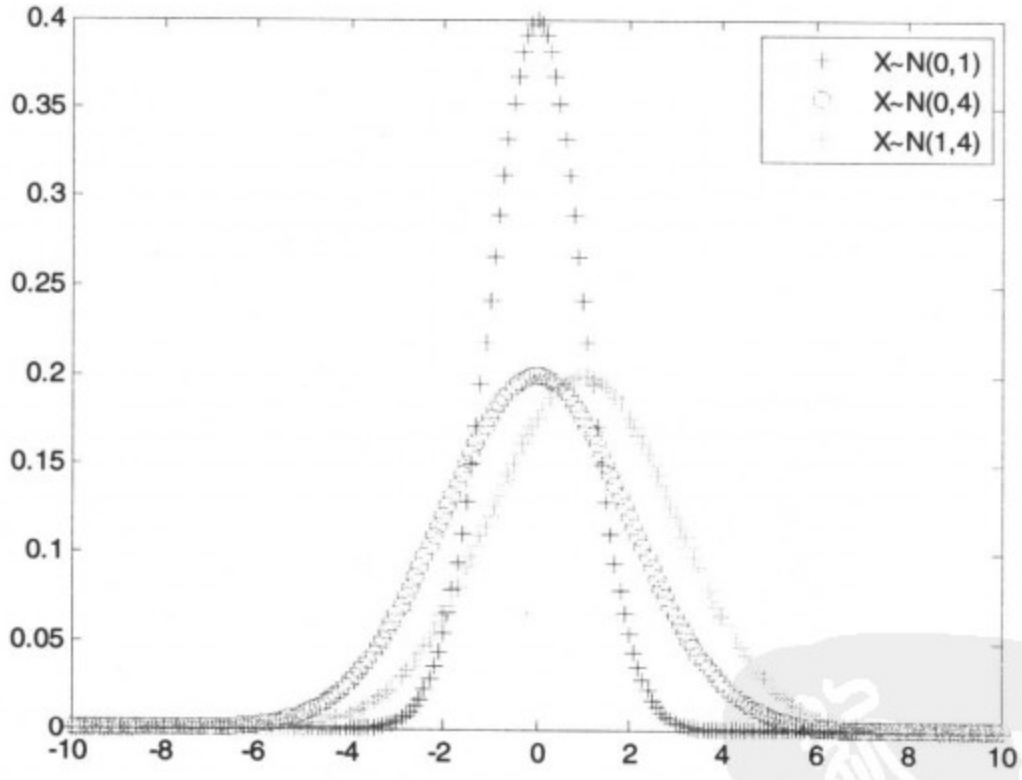

图 11-1　正态分布概率密度图

11.3.2　累积分布函数（cdf）与逆累积分布函数

1．累积分布函数

若随机变量 X 的概率密度函数为 $f(x)$，则累积分布函数的定义如下：

$$F(x)=P(X\leqslant x)=\int_{-\infty}^{x}f(t)\mathrm{d}t$$

与求 pdf 类似，MATLAB 中也有一个通用的函数和一些专用的函数来求解累积分布函数值。

MATLAB 中用 cdf('name',*X*,*A*1,*A*2,*A*3)可求解各种分布的累积概率分布值，其中各参数含义可参见 pdf。

表 11-9 给出了各分布的专用的累积分布函数。其具体参数含义和对应的 pdf 函数相同。

表 11-9　累积分布函数

分 布 名 称	累积分布函数	调 用 格 式
β 分布	betacdf	*Y*=betacdf(*X*,*A*,*B*)
二项分布	binocdf	*Y*=binocdf(*X*,*N*,*P*)
χ^2 分布	chi2cdf	*Y*=chi2cdf(*X*,*V*)
经验分布（kaplan-Meier 分布）	ecdf	[*f*, *x*]=ecdf(*y*);[*f*,*x*,flo,fup]=ecdf(*y*,'param1',value1,'param2',value2……)
极值分布	evcdf	*Y*=evcdf(*X*,mu,sigma)
指数分布	expcdf	*Y*=expcdf（*X*,mu）
F 分布	fcdf	*Y*=fcdf(*X*,*V*1,*V*2)
γ 分布	gamcdf	*Y*=gamcdf(*X*,*A*,*B*)
几何分布	geocdf	*Y*=geocdf(*X*,*P*)
广义极值分布	gevcdf	*Y*=gevcdf(*X*,*K*,sigma,mu)
广义帕累托分布	gpcdf	*P*=gpcdf(*X*,*K*,sigma,theta)
超几何分布	hygecdf	*Y*=hygecdf(*X*,*M*,*K*,*N*)
对数正态分布	logncdf	*Y*=logncdf(*X*,mu,sigma)
负二项分布	nbincdf	*Y*=nbincdf(*X*,*R*,*P*)
非中心 *F* 分布	ncfcdf	*Y*=ncfcdf(*X*,NU1,NU2,DELTA)
非中心 *t* 分布	nctcdf	*Y*=nctcdf(*X*,*V*,DELTA)
非中心 χ^2 分布	ncx2cdf	*Y*=ncx2cdf(*X*,*V*,DELTA)
正态分布	normcdf	*Y*=normcdf(*X*,mu,sigma)
泊松分布	poisscdf	*Y*=poisscdf(*X*,LAMBDA)
瑞利分布	raylcdf	*Y*=raylcdf(*X*,*B*)
学生 *t* 分布	tcdf	*Y*=tcdf(*X*,*V*)
离散均匀分布	unidcdf	*Y*=unidcdf(*X*,*N*)
连续均匀分布	unifcdf	*Y*=unifcdf(*X*,*A*,*B*)
韦伯分布	wblcdf	*Y*=wblcdf(*X*,*A*,*B*)

2. 逆累积分布函数

逆累积分布函数是累积分布函数的反函数，也称分值点函数。通用求解函数为 icdf，具体调用格式与 cdf 相同。表 11-10 给出了 MATLAB 中的专用的逆累积分布函数。

表 11-10 逆累积分布函数

分布名称	累积分布函数	调用格式
β 分布	betainv	X=betainv(Y,A,B)
二项分布	binoinv	X=binoinv(Y,N,P)
χ^2 分布	chi2inv	X=chi2inv(Y,V)
极值分布	evinv	X=evinv(Y,mu,sigma)
指数分布	eypinv	X=expinv（Y,mu）
F 分布	finv	X=finv(Y,V1,V2)
γ 分布	gaminv	X=gaminv(Y,A,B)
几何分布	geoinv	X=geoinv(Y,P)
广义极值分布	gevinv	X=gevinv(Y,K,sigma,mu)
广义帕累托分布	gpinv	X=gpinv(Y,K,sigma,theta)
超几何分布	hygeinv	X=hygeinv(Y,M,K,N)
对数正态分布	logninv	X=logninv(Y,mu,sigma)
负二项分布	nbininv	X=nbininv(Y,R,P)
非中心 F 分布	ncfinv	X=ncfinv(Y,NU1,NU2,DELTA)
非中心 t 分布	nctinv	X=nctinv(Y,V，DELTA)
非中心 χ^2 分布	ncx2inv	X=ncx2inv(Y,V,DELTA)
正态分布	norminv	X=norminv(Y,mu,sigma)
泊松分布	poissinv	X=poissinv(Y,LAMBDA)
瑞利分布	raylinv	X=raylinv(Y,B)
学生 t 分布	tinv	X=tinv(Y,V)
离散均匀分布	unidinv	X=unidinv(Y,N)
连续均匀分布	unifinv	X=unifinv(Y,A,B)
韦伯分布	wblinv	X=wblinv(Y,A,B)

【例 11-10】 累积分布函数和逆累积分布函数应用实例。设 $X\sim N$（1，9）分别求下面的值：

（1） $P(X\leqslant 4)$， $P(1<X\leqslant 5)$， $P(X>6)$， $P(|X|<4)$

（2）若 $P(X<c)=0.5$，求 c 值。

解：通过在 MATLAB 中编写程序 ex1110.m 来实现，具体代码如下所示。

```
%ex1110.m 累积分布函数和逆累积分布函数编程实现
clear all
%%%%用累积分布函数计算概率值
y1=normcdf(4,1,3) %P(X<4)
y2=normcdf(5,1,3)-normcdf(1,1,3) %P(1<X<5)
y3=1-normcdf(6,1,3)   %P(X>6)
y4=normcdf(4,1,3)-normcdf(-4,1,3) %P(|X|<4)
%%%%%用逆累积分布函数计算给定概率值的 x 值
c=norminv(0.5,1,3)  %P(X<c)=0.5
```

计算结果为：

```
> ex1110
```

```
y1 =
    0.8413
y2 =
    0.4088
y3 =
    0.0478
y4 =
    0.7936
c =
     1
```

由计算结果可知，$P(X \leqslant 4)=0.8413$，$P(1<X \leqslant 5)=0.4088$，$P(X>6)=0.0478$，$P(|X|<4)=0.7936$。

当$c=1$时，$P(X<c)=0.5$。

【例 11-11】 概率计算应用实例。设在一次实验中某事件发生的概率为 0.4，重复该实验 100 次，求出现该事件次数不小于 50 次的概率。若该事件刚好发生 30 次，求其概率。

解：设该事件为 X，则易知 X 服从（100，0.4）的二项分布。在 MATLAB 中编写程序 ex1111.m 来实现，具体代码如下所示。

```
%ex1111.m 计算二项分布的概率
clear all
a=binocdf(49,100,0.4);                    %P(X<50)
p1=1-a                                    %P(X>=50),事件出现不小于 50 次概率
p2=binopdf(30,100,0.4)                    %P(X=30)
```

输出结果为：

```
>> ex0811
p1 =
    0.0271
p2 =
    0.0100
```

由以上计算结果可知，出现该事件次数不小于 50 的概率为 0.0271，该事件刚好发生 30 次的概率为 0.01。

11.3.3 特殊分布的期望和方差

除了上节介绍的简单的均值函数 mean 以及方差函数 var 外，MATLAB 中还为每一种特殊分布提供了相应的计算其均值和方差的函数，详细函数说明见表 11-11。

表 11-11 特殊分布的期望和方差函数

函 数 名	调用格式	函 数 功 能
betastat	[M,V]=betastat(A,B)	β 分布的期望和方差
binostat	[M,V]=binostat(N,P)	二项分布的期望和方差
chi2stat	[M,V]=chi2stat(V)	χ^2 分布的期望和方差
evstat	[M,V]=evstat(mu,sigma)；	极值分布的期望和方差
eypstat	[M,V]=expstat（mu）；	指数分布的期望和方差

续表

函 数 名	调用格式	函 数 功 能
fstat	[M,V]=fstat(V1,V2)	F 分布的期望和方差
gamstat	[M,V]=gamstat(A,B);	γ 分布的期望和方差
geostat	[M,V]=geostat(P)	几何分布的期望和方差
gevstat	[M,V]=gevstat(K,sigma,mu)	广义极值分布的期望和方差
gpstat	[M,V]=gpstat(K,sigma,theta)	广义帕累托分布的期望和方差
hygestat	[M,V]=hygestat(M,K,N)	超几何分布的期望和方差
lognstat	[M,V]=lognstat(mu,sigma)	对数正态分布的期望和方差
nbinstat	[M,V]=nbinstat(R,P)	负二项分布的期望和方差
ncfstat	[M,V]=ncfstat(NU1,NU2,DELTA)	非中心 F 分布的期望和方差
nctstat	[M,V]=nctstat(V,DELTA)	非中心 t 分布的期望和方差
ncx2stat	[M,V]=ncx2stat(V,DELTA)	非中心 χ^2 分布的期望和方差
normstat	[M,V]=normstat(mu,sigma)	正态分布的期望和方差
poissstat	[M,V]=poissstat(LAMBDA)	泊松分布的期望和方差
raylstat	[M,V]=raylstat(B)	瑞利分布的期望和方差
tstat	[M,V]=tstat(V)	学生 t 分布的期望和方差
unidstat	[M,V]=unidstat(N)	离散均匀分布的期望和方差
unifstat	[M,V]=unifstat(A,B)	连续均匀分布的期望和方差
wblstat	[M,V]=wblstat(A,B)	韦伯分布的期望和方差

下面举例说明部分函数的用法。

【例 11-12】 特殊分布的期望和方差计算实例之一。求参数为 0.4 和 0.8 的 β 分布的期望和方差。

解：在 MATLAB 命令窗口中输入：

```
>> [m,v]=betastat(0.4,0.8)
```

输出结果为：

```
m =
    0.3333
v =
    0.1010
```

【例 11-13】 特殊分布的期望和方差计算实例之二。求参数为 5 的泊松分布的期望和方差。

解：在 MATLAB 命令窗口中输入：

```
>> [m,v]=poisstat(5)
```

输出结果为：

```
m =
     5
v =
     5
```

由此例可知泊松分布的均值和方差相同，且都等于参数 λ。

【例 11-14】 特殊分布的期望和方差计算实例之三。分别计算参数为 1～6 的离散均匀分布的均值和方差。

解：在 MATLAB 命令窗口中输入：

```
>> [m,v]=unidstat(1:6)
```

输出结果为：

```
m =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000
v =
         0    0.2500    0.6667    1.2500    2.0000    2.9167
```

由此例可知，表 11-11 中的函数还可以同时计算具有不同参数的相同类型分布的均值和方差。但是值得注意的是，输入的参数矩阵须具有相同的维数。

11.3.4 随机数生成器

MATLAB 的统计工具箱中，提供了通用的随机数产生函数 random 和特定分布的随机数产生函数（以 rnd 结尾）。可以直接调用这些函数来获得所需的随机数。

对于 MATLAB 7 支持的 20 多种特殊分布，每种分布的随机数都可以任意产生。函数可以产生单个随机数或随机数矩阵，这取决于函数调用时所采用的参数。表 11-12 给出了几种常用的随机数生成函数。

表 11-12 几种常用的随机数生成器

函 数 名	调 用 格 式	函 数 功 能
betarnd	*R*=betarnd(*A*,*B*)	β 分布的随机数
binornd	*R*=binornd(*N*,*P*)	二项分布的随机数
chi2rnd	*R*=chi2rnd(*V*)	χ^2 分布的随机数
eyprnd	*R*=exprnd（mu）;	指数分布的随机数
frnd	*R*=frnd(*V*1,*V*2)	*F* 分布的随机数
gamrnd	*R*=gamrnd(*A*,*B*);	γ 分布的随机数
geornd	*R*=geornd(*P*)	几何分布的随机数
normrnd	*R*=normrnd(mu,sigma)	正态分布的随机数
poissrnd	*R*=poissrnd(LAMBDA)	泊松分布的随机数
raylrnd	*R*=raylrnd(*B*)	瑞利分布的随机数
unidrnd	*R*=unidrnd(*N*)	离散均匀分布的随机数
unifrnd	*R*=unifrnd(*A*,*B*)	连续均匀分布的随机数

【例 11-15】 随机数生成实例。生成均值为 0，方差为 4 的 3 行 4 列的随机数。

解：在 MATLAB 命令窗口中输入：

```
>> n1=normrnd(0,2,3,4)          %生成 3 行 4 列的 N（0，4）的随机数
>> n2=normrnd(0,2,[3 4])        %生成 3 行 4 列的 N（0，4）随机数
>> n3=random('normal',0,2,3,4)  %用 random 函数生成
```

输出结果为：

```
n1 =
    2.3783    0.3493   -1.1766    0.2279
   -0.0753   -0.3734    4.3664    2.1335
    0.6546    1.4516   -0.2728    0.1186
n2 =
   -0.1913   -2.6724   -1.3836   -3.1875
   -1.6647    1.4286    1.7160   -2.8819
    0.5888    3.2471    2.5080    1.1423
n3 =
   -0.7998    1.4238    2.3817   -0.3134
    1.3800    2.5805   -2.4049   -3.2082
    1.6312    1.3372   -0.0396    0.5146
```

11.4 参数估计

参数估计主要包括点估计和区间估计。设总体 X 的分布函数的形式为已知，但它的一个或多个参数为未知，借助于总体 X 的一个样本来估计总体未知参数的值的问题称为参数的点估计问题。以区间形式给出包含参数真值的范围和此区间包含真值的可信程度的估计叫做区间估计，这样的区间叫做置信区间。点估计方法种类很多，如矩估计法、极大似然估计法和最小二乘估计法等。下面重点介绍极大似然估计以及它的区间估计。

MATLAB 的统计工具箱中对一些常用的特殊分布也提供了相应的极大似然估计函数（以 fit 结尾），具体见表 11-13。

表 11-13 参数点估计函数表

函 数 名	调用格式	函 数 功 能
betafit	phat=betafit(x) [phat,pci]=betafit(x,alpha)	phat 为β 分布的 a、b 参数的极大似然估计值；pci 为 alpha 水平的置信区间
binofit	phat=binofit(x,n) [phat,pci]=binofit(x,n,alpha)	phat 为二项分布中事件发生概率的极大似然估计；pci 为 alpha 水平的置信区间
evfit	phat=evfit(x)； [phat,pci]=evfit(x,alpha)	phat 为极值分布的均值和标准差极大似然估计；pci 为 alpha 水平置信区间
expfit	phat=expfit(x) [phat,pci]=expfit(x,alpha)	phat 为指数分布的参数 μ 的极大似然估计；pci 为 alpha 水平置信区间
gamfit	phat=gamfit(x); [phat,pci]=gamfit(x,alpha)	phat 为 γ 分布参数的极大似然估计；pci 为 alpha 水平置信区间
lognfit	phat=lognfit(x) [phat,pci] =lognfit(x,alpha)	phat 为对数正态分布的均值和标准差的极大似然估计；pci 为 alpha 水平的置信区间
nbinfit	phat=nbinfit(x) [phat,pci] =nbinfit(x)	phat 为负二项分布参数的极大似然估计；pci 同上。
normfit	[muhat,sigmahat]=normfit(x) [muhat,sigmahat,muci,sigmaci]=normfit(x,alpha)	muhat、sigmahat 分别为正态分布的均值和标准差的极大似然估计；muci，sigmaci 为相应的 alpha 水平的置信区间

续表

函 数 名	调用格式	函 数 功 能
poissfit	lambdshat=poissfit(x) [lambdshat,lambdaci]=poissfit(x,alpha)	lambdshat 为泊松分布的参数 λ 的极大似然估计；lambdaci 为 alpha 水平置信区间
raylfit	phat=raylfit(x) [phat,pci]=raylfit(x,alpha)	phat 为瑞利分布参数的极大似然估计；pci 为 alpha 水平的置信区间
uniffit	[ahat，bhat]=uniffit(x) [ahat,bhat,aci,bci]=unifit(x)	ahat,bhat 分别为连续均匀分布的 a、b 参数的极大似然估计；aci、bci 为 alpha 水平置信区间
wblfit	phat=wblfit(x) [phat,pci]=wblfit(x,alpha)	phat 为韦伯分布参数的极大似然估计；pci 为 alpha 水平置信区间

表中的各函数根据输入数据 x 计算各分布参数的极大似然估计值，并给出（1- a）的置信区间，其中 a 默认为 0.05。例如：

[pha, pci] = normfit(x)

其返回的置信区间 pci 为 95%的置信区间。

此外，工具箱中还包含一个通用的求极大似然估计值的函数 mle，其调用格式如下：

（1）phat = mle(x)

返回正态分布参数的极大似然估计值；

（2）[phat, pci] = mle(x)

返回正态分布参数的极大似然估计值和 95%置信区间；

（3）[phat, pci] = mle(x, 'distribution', dist)

返回由 dist 设定分布的极大似然估计和 95%置信区间。

下面通过实例说明该类函数的具体用法。

【例 11-16】 参数估计综合计算实例之一。参数设总体 X 在[a,b]上服从均匀分布，a，b 未知。现有如下的一组样本值：

1.2，1.4，1.8，2.1，3.4，4.5，2.6，5.0，1.8，3.8，4.2，3.4，2.6，4.8，4.0

试求 a，b 的估计量并给出 95%置信区间。

解：下面给出分别用专用函数 unifit 以及通用函数 mle 求解的过程，在 MATLAB 命令窗口中输入：

```
>>  x=[1.2,1.4,1.8,2.1,3.4,4.5,2.6,5.0,1.8,3.8,4.2,3.4,2.6,4.8,4.0];
>> [ahat,bhat,aci,bci]=unifit(x)  %用 unifit 求解
```

输出结果为：

```
ahat =
    1.2000
bhat =
     5
aci =
  -71.0000
    1.2000
bci =
```

```
    5.0000
   77.2000
```

在 MATLAB 命令窗口中继续输入：

```
>> x=[1.2,1.4,1.8,2.1,3.4,4.5,2.6,5.0,1.8,3.8,4.2,3.4,2.6,4.8,4.0];
>> [phat,pci]=mle(x,'distribution','uniform')   %用 mle 求解
```

输出结果为：

```
phat =
    1.2000    5.0000
pci =
  -71.0000    5.0000
    1.2000   77.2000
```

由此解可得 a 的估计值为 1.2，置信区间为[-71,1.2]；b 的估计值为 5，置信区间为[5,77.2]。

【例 11-17】 参数估计综合计算实例之二。一地质学家为研究密执根湖湖滩地区的岩石成分，随机地自该地区取 100 个样品，每个样品有 10 块石子，记录了每个样品中属石灰石的石子数。假设这 100 次观察相互独立，并且由过去经验知，它们都服从参数为 n=10，p 的二项分布。p 是这地区一块石子是石灰石的概率，求 p 的极大似然估计值。该地质学家所得数据如表 11-14 所列。

表 11-14　二项分布数据

样品中属石灰石的石子数	0	1	2	3	4	5	6	7	8	9	10
观察到石灰石的样品个数	0	1	6	7	23	26	21	12	3	1	0

解： 在 MATLAB 命令窗口中输入：

```
>>x=[1,2*ones(1,6),3*ones(1,7),4*ones(1,23),5*ones(1,25),6*ones(1,21),7*
ones(1,12),8*ones(1,3),9];          %输入 100 个样本值
>> n=10;                            %每一个样本服从 n=10,p 的二项分布
>> p=binofit(x,n);                  %每一个样本的 p 估计值
>> p1=mean(p)                       %总体的 p 的估计值为各样本均值
```

输出结果为：

```
p1 =
    0.4990
```

【例 11-18】 参数估计综合计算实例之三。已知下列数据为指数分布，求它们的点估计值和 97%的置信区间。数据为：1，6，8，25，28，15，1，3，8。

解： 在 MATLAB 命令窗口输入：

```
>> x=[1 6 8 25 28 15 1 3 8];
>> [parmhat,parmci]=expfit(x,0.03)
```

输出结果为：

```
parmhat =
   10.5556
parmci =
    5.6917
   25.2778
```

11.5 假设检验

假设检验是对总体的参数或总体的分布函数的形式的某种假设进行判断的过程。假设检验主要包括以下几个步骤：

（1）根据问题的要求，设置原假设 H_0 和备择假设 H_1；

（2）选择合适的检验统计量。在 H_0 成立的条件下，确定其概率分布；

（3）确定拒绝域。对给定的显著性水平 α，由统计量的分布查表或计算确定出临界值，进而得到 H_0 的拒绝域；

（4）根据样本观察值计算出统计量值；

（5）根据统计量的值是否落入拒绝域，做出拒绝或保留 H_0 的判断。

MATLAB 的统计工具箱提供了如表 11-15 所示的几种假设检验函数。

表 11-15 假设检验函数

函数名	调用格式	函数功能
chi2gof	*H*=chi2gof(*X*) [*H*,*P*]=chi2gof(*X*)	单样本分布服从正态分布的 χ^2 检验。返回检验结果 *H* 和 *p* 值 *P*
jbtest	*H*=jbtest(*X*) *H*=jbtest(*X*,alpha)	单样本分布服从正态分布的雅克比检验
kstest	*H*=kstest(*X*,cdf)	单样本服从任意特殊分布的检验，其中 cdf 规定特殊分布的累积分布函数
kstest2	*H*=kstest2(*X*1,*X*2,cdf)	两样本的同分布检验
lillietest	*H*=lillietest(*X*)	单样本分布服从正态分布的 Lilliefors 检验
ranksum	*H*=ranksum(*x*,*y*)	两独立样本来自具有相同中值的分布的检验
runstest	*H*=runstest(*X*)	原假设：样本值是按照随机顺序排列的
signtest	*p*=signtest(*x*,*y*)	成对数据样本的相应数据之差来自中值为 0 的分布的检验，返回 *p* 值
ttest	*H*=ttest(*x*，*m*) *H*=ttest(*x*,*y*)	*t* 检验假设 *x* 中数据来自均值为 *m* 的分布；检验两样本来自具有同均值的分布
ttest2	*H*=ttest2(*x*,*y*)	*t* 检验两方差未知的正态样本具有相同均值
vartest	*H*=vartest(*X*,*V*)	χ^2 检验正态样本方差为 *V*
vartest2	*H*=vartest2(*X*,*Y*)	*F* 检验两独立正态样本具有相同方差
vartestn	Vartestn(*X*)	原假设：矩阵 ***X*** 各列来自具有相同方差的正态分布。备择假设：矩阵 ***X*** 各列为方差不同的正态分布
ztest	*H*=ztest(*x*,*m*,sigma)	*z* 检验标准差为 sigma 的正态样本的均值为 *m*

表中 *X* 为输入样本；*H* 为检验结果，其值为 0 支持原假设，其值为 1 拒绝原假设；显著水平 *α* 默认值为 0.05，也可以在参数中进行设定。*p* 值是拒绝原假设的最小的 *α* 值。

下面介绍其中几种重要形式的假设检验。

11.5.1　单个总体 $N(\mu,\sigma^2)$ 均值 μ 的检验

1．σ^2 已知，关于 μ 的检验（Z 检验）

MATLAB 中 Z 检验用函数 ztest 来实现，其调用格式有如下几种。

h=ztest(*x*,*m*,sigma)在 α=0.05 水平上检验标准差已知为 sigma 的正态分布样本均值是否为 *m*。***x*** 为矩阵时，函数分别检验 ***x*** 的每一列，并返回一向量形式的结果。

h = ztest(*x*, *m*, sigma, alpha)对显著性水平加以限制，为 alpha 规定的值。

[*h*, sig, ci]=ztest(*x*, *m*, sigma, alpha, tail) tail（取值见表 11-16）对三种备择假设形式进行选择，sig 为 *p* 值，ci 为 1-alpha 置信区间。

表 11-16　单样本均值检验：tail 取值及选择的备择假设

tail 取值	备择假设形式
'both'（默认值）	样本均值不等于 *m*（$\bar{x}\neq m$）
'right'	样本均值大于 *m*（$\bar{x}>m$）
'left'	样本均值小于 *m*（$\bar{x}<m$）

【例 11-19】 假设检验综合实例之一。某糖厂有一台自动打包机做打包，额定标准每包质量为 100kg。设每包质量服从正态分布，且根据以往经验，其方差 $\sigma^2=(0.4)^2$。某天开工后，为检查打包机工作情况，随机地抽取 9 包，称得质量（单位：kg）如下：

99　98.5　102.5　101　98　99　102　102.1　100.5

问这天打包机工作是否正常？设显著性水平为 0.05。

解：由题意可提出如下假设：

原假设 H_0：$\mu=100$；

备择假设 H_1：$\mu\neq100$。

由于该样本服从正态分布且方差已知，因此可以用 *z* 统计量检验假设。具体通过下面的程序实现。

在 MATLAB 命令窗口中输入：

```
>> x=[99  98.5  102.5  101  98  99  102  102.1  100.5];
>> m=100;
>> h=ztest(x,m,0.4)
```

输出结果为：

```
h =
     1
```

在 MATLAB 命令窗口中继续输入：

```
>> [h,sig,ci]=ztest(x,m,0.4)
```

输出结果为：

```
h =
     1
sig =
    0.0303
ci =
100.276     100.5502
```

由 h=1 知拒绝原假设，即这天打包机打包的质量与规定质量有显著差异。另外可得 p 值为 0.0303 以及包含均值真值的 95%置信区间为[100.276,100.5502]。

2. σ^2 未知，关于 μ 的检验（t 检验）

在 MATLAB 中用 ttest 来实现。ttest 具体调用格式如下：

（1）h = ttest(x)

判断正态样本均值是否为零；

（2）h = ttest(x,m)

判断正态样本均值是否等于 m；

（3）[h, sig] = ttest(x, m, alpha, tail)

在显著性水平 alpha 下判断正态样本均值是否等于 m。各参数含义参见 ztest。

【例 11-20】 假设检验综合实例之二。在某砖厂生产的一批砖中，随机地抽取 6 块进行抗断强度试验，测得结果（单位：kg/cm^2）如下：32.56，29.66，32.64，30.00，31.87，32.03；设砖的抗断强度服从正态分布，问这批砖的平均抗断强度是否不大于32.50(kg/cm^2)？取 （α=0.05）。

解：由题意可得如下假设：

$$H_0: \ \mu \leqslant 32.50$$

$$H_1: \ \mu > 32.50$$

由于其为方差未知，检验均值大小的假设检验问题，故选择 t 统计量进行检验。具体用下列程序实现。

在 MATLAB 命令窗口中输入：

```
>> x=[32.56  29.66  32.64  30.00  31.87  32.03];
>> m=32.5;
>> h=ttest(x,m,0.05,'right')
```

输出结果为：

```
h =
     0
```

由 h=0 可知在显著性水平为 0.05 的情况下，不能拒绝原假设。即认为这批砖的平均抗断强度不大于 32.5 kg/cm^2。

11.5.2 两个正态总体均值差的检验（t 检验）

在 MATLAB 中用 ttest2 来检验两个正态样本的均值是否相同。这两个正态样本方差未知，但假定其方差相同。其调用格式如下：

h = ttest2(x, y)

[h, sig, ci] = ttest2(x, y, alpha, tail)

x、y 为两个输入样本，其长度可以不同。alpha 为显著性水平，tail 取值见表 11-17。

表 11-17 两样本均值检验：tail 取值及选择的备择假设

tail 取值	备择假设形式
'both'（默认值）	$\mu_x \neq \mu_y$
'right'	$\mu_x > \mu_y$
'left'	$\mu_x < \mu_y$

【例 11-21】 假设检验综合实例之三。表 11-18 分别给出两个文学家马克吐温的 8 篇小品文以及斯诺特格拉斯的 10 篇小品文中由 3 个字母组成的单词的比例。设两组数据分别来自正态总体，且两总体方差相等，但参数均未知。两样本相互独立。问两个作家所写的小品文中包含由 3 个字母组成的单词的比例是否有显著差异。

表 11-18 两文学家文章中 3 个字母的单词比例

马克吐温	0.225	0.262	0.217	0.240	0.230	0.229	0.235	0.217		
斯诺特格拉斯	0.209	0.205	0.196	0.210	0.202	0.207	0.224	0.223	0.220	0.201

解：由题意可得如下假设：

$$H_0: \ \mu_1 = \mu_2$$

$$H_1: \ \mu_1 \neq \mu_2$$

取 α=0.05。

在 MATLAB 命令窗口中输入：

```
>> x=[0.225 0.262   0.217   0.240   0.230   0.229   0.235   0.217];
>> y=[0.209 0.205   0.196   0.210   0.202   0.207   0.224   0.223   0.220
0.201];
>> h=ttest2(x,y)
```

输出结果为：

```
h =
     1
```

由 h=1 得，在显著性水平为 0.05 的情况下拒绝原假设。即两个作家的作品中包含 3 个字母的单词的比例有显著差异。

11.5.3 基于成对数据的检验（t 检验）

有时为了比较两种产品或两种仪器、两种方法等的差异，通常需要在相同条件下作对比试验，得到一批成对的观察值。然后分析观察数据作出判断。这种方法称为逐对比较法。

MATLAB 中的 ttest 的另一用法便是基于成对数据的检验的具体调用格式如下：

h = ttest(x, y), x、y 为成对数据，其长度应相同。该函数检验 x-y 构成的正态分布的均值是否为零。

【例 11-22】 假设检验综合实例之四。有两台光谱仪 I_x，I_y，用来测量材料中某种金属的含量，为鉴定它们的测量结果有无显著差异，制备了 9 件试块（它们的成分、金属含量、均匀性等均各不相同），现在分别用这两台仪器对每一试块测量一次，得到 9 对观察值，如表 11-19 所示。

·表 11-19 两台光谱仪测定材料中金属含量

x（%）	0. 20	0. 30	0. 40	0. 50	0. 60	0. 70	0. 80	0. 90	1. 00
y（%）	0. 10	0. 21	0. 52	0. 32	0. 78	0. 59	0. 68	0. 77	0. 89

问：能否认为这两台仪器的测量结果有显著的差异（取 α=0.01）？

解：由题意，设 μ_D 为两组数据差的均值，可得如下假设

$$H_0:\ \mu_D = 0$$

$$H_1:\ \mu_D \neq 0$$

采用基于成对数据的 t 检验，用 ttest 计算如下：

在 MATLAB 命令窗口输入：

```
>> x=[0.20  0.30    0.40    0.50    0.60    0.70    0.80    0.90    1.00];
>> y=[0.10  0.21    0.52    0.32    0.78    0.59    0.68    0.77    0.89];
>> h=ttest(x,y,0.01)
```

输出结果为：

```
h =
     0
```

故由 h=0 得，在显著性水平为 0.01 的情况下，不能拒绝原假设，即认为两台仪器的测量结果并无显著差异。

11.5.4 正态总体方差的假设检验

1. 单个总体的情况

样本来自单个服从正态分布总体，其均值、方差未知，检验其方差是否等于某一特定值。此类假设检验问题采用统计量 χ^2 进行检验。在 MATLAB 中用函数 vartest 实现，具体调用格式如下：

H = vartest(X, V, alpha)

H = vartest(X, V, alpha,tail)

X 为输入样本，V 为待检验方差值。alpha 为显著性水平，tail 取值见表 11-20 所示。

表 11-20 单样本方差检验：tail 取值及选择的备择假设

tail 取值	备择假设形式
'both'（默认值）	样本方差不等于 V
'right'	样本方差大于 V
'left'	样本方差小于 V

【例 11-23】 假设检验综合实例之五。某厂生产螺钉，生产一直比较稳定，长期以来，螺钉的直径服从方差为 0.0002(cm^2)的正态分布。今从产品中随机抽取 10 只进行测量，得螺钉直径的数据（单位：cm）如下：1.19，1.21，1.21，1.18，1.17，1.20，1.20，1.17，1.19，1.18。问是否可以认为该厂生产的螺钉直径的方差为 0.0002(cm^2) ?(取 a 为 0.05)

解：此题检验 H_0：$\sigma^2 = 0.0002$，用 vartest 计算如下。

在 MATLAB 命令窗口中输入：

```
>> x=[1.19  1.21  1.21  1.18  1.17  1.20  1.20  1.17  1.19  1.18];
>> v=0.0002;
>> h=vartest(x,v)
```

输出结果为：

```
h =
     0
```

故在显著性水平 0.05 情况下接受原假设，即认为生产的螺钉直径的方差为 0.0002 cm^2。

2. 两个总体的情况

通常采用 F 统计量检验两独立样本的方差是否相同。MATLAB 中采用 vartest2 实现，具体调用格式如下：

H = vartest2(X, Y)

H = vartest2(X, Y, alpha)

H = vartest2(X, Y, alpha, tail)

tail 取值如表 11-21 所示。

表 11-21 两样本方差检验：tail 取值

tail 取值	备择假设形式
'both'（默认值）	$V_x \neq V_y$
'right'	$V_x > V_y$
'left'	$V_x < V_y$

【例 11-24】 假设检验综合实例之六。有两台机床加工同一种零件，这两台机床生产的零件尺寸服从正态分布。今从两台机床生产的零件中分别抽取 11 个和 9 个零件进行测量，得数据（单位：mm）如下。

甲机床：6.2，5.7，6.5，6.0，6.3，5.8，5.7，6.0，6.0，5.8，6.0；

乙机床：5.6，5.9，5.6，5.7，5.8，6.0，5.5，5.7，5.5。

问甲机床的加工精度是否比乙机床的加工精度较差？（取 $a = 0.05$）

解：检验如下假设，

$$H_0: \sigma_1^2 = \sigma_2^2$$

$$H_0: \sigma_1^2 < \sigma_2^2$$

用 vartest2 计算，在 MATLAB 命令窗口中输入：

```
>> x=[6.2  5.7  6.5  6.0  6.3  5.8  5.7  6.0  6.0  5.8  6.0];
```

```
>> y=[5.6  5.9  5.6  5.7  5.8  6.0  5.5  5.7  5.5];
>> h=vartest2(x,y,0.05,'left')
```

输出结果为：

```
h =
     0
```

故在显著性水平为 0.05 的情况下接受原假设，即认为甲机床的加工精度与乙机床加工精度相当。

11.6 方差分析

方差分析（Analysis of Variance 缩写为 ANOVA）是数理统计中常用的数据处理方法之一，是工农业生产和科学研究中分析试验数据的一种有效的工具，也是开展试验设计、参数设计和容差设计的数学基础。

一个复杂的事物，其中往往有许多因素相互制约又相互依存。方差分析的目的是通过数据分析找出对该事物有显著影响的因素，各因素之间的交互作用，以及显著影响因素的最佳水平等。

方差分析是在可比较的数组中，把数据间的总的“变差”按各指定的变差来源进行分解的一种技术。对变差的度量，采用离差平方和。方差分析方法就是从总离差平方和分解出可追溯到指定来源的部分离差平方和。下面介绍几种简单情况的方差分析。

11.6.1 单因素试验的方差分析

单因素试验的方差分析是指试验中只有一个因素发生改变。MATLAB 中，单因素试验的方差分析可用函数 anova1 实现。

anova1 执行单因素试验的方差分析来比较两组或多组数据的均值，它返回原假设样本来自相同的总体或来自具有相同的均值的不同总体的 p 值，具体调用格式如下。

1. p=anova1(X)

将矩阵 X 中的每一列作为一个独立样本，函数判断这些样本的均值是否相同，并返回 p 值，ANOVA 表格和矩阵 X 各列的箱线图。

2. p=anova1(X,group)

用 group 中的值标记 X 样本的箱线图。group 的每一行包含 X 相应列的标记值。因此 group 的长度须等于 X 的列数。

3. p=anova1(X,group,'displayopt')

用 displayopt 确定是否要作出 ANOVA 的表格和样本的箱线图。当 displayopt='on' (默认值)时，给出表格和箱线图；displayept='off' 时不给出表格和图。

注意

1）ANOVA 表格共包含 6 列，每列的意义如下：

- 第一列为 source 项，即方差来源；
- 第二列给出每一项来源的平方和 SS；
- 第三列给出每一项来源的自由度 df，即包含的数据总数；
- 第四列给出每一项来源的均方 MS=SS/df；
- 第五列给出 F 比，也就是 MS 的比率。

2）该函数显示的第二幅图为矩阵 X 每一列的箱线图。

【例 11-25】 方差分析综合实例之一。将抗生素注入人体会产生抗生素与血浆蛋白结合的现象，以致减少了药效。表 11-22 列出 5 种常用的抗生素注入到牛的体内时，抗生素与血浆蛋白质结合的百分比。试检验这些百分比的均值有无显著的差异。

表 11-22　各抗生素与血浆蛋白结合的百分比

青霉素	四环素	链霉素	红霉素	氯霉素
29.6	27.3	5.8	21.6	29.2
24.3	32.6	6.2	17.4	32.8
28.5	30.8	11.0	18.3	25.0
32.0	34.8	8.3	19.0	24.2

解：在 MATLAB 命令窗口中输入：

```
>>x=[29.6,27.3,5.8,21.6,29.2;24.3,32.6,6.2,17.4,32.8;28.5,30.8,11.0,18.3
,25.0;32.0,3,4.8,8.3,19.0,24.2];
>> group=['青霉素';'四环素';'链霉素';'红霉素';'氯霉素'];
>> p=anova1(x,group)
```

输出结果为：

```
p =
    0.0020
```

由于 p=0.002，远小于 0.01，故可以判断这些抗生素的均值具有显著差异。

ANOVA 表格和 X 箱线图如图 11-2 和图 11-3 所示。

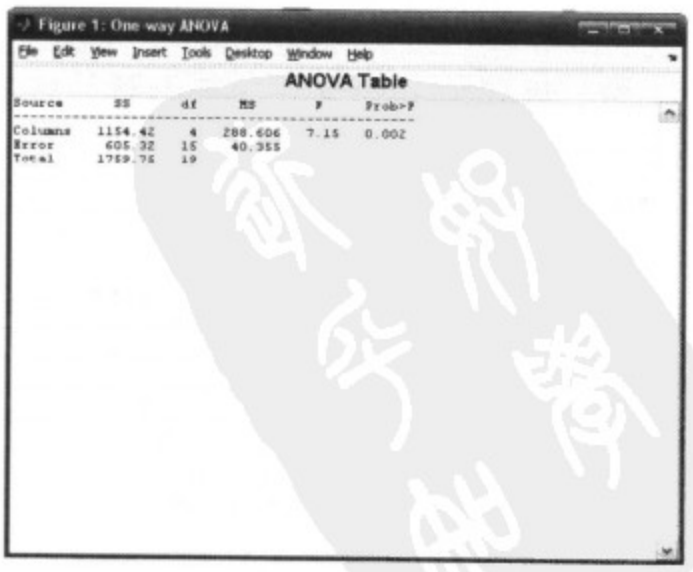

图 11-2　单因素方差分析的 ANOVA 表格

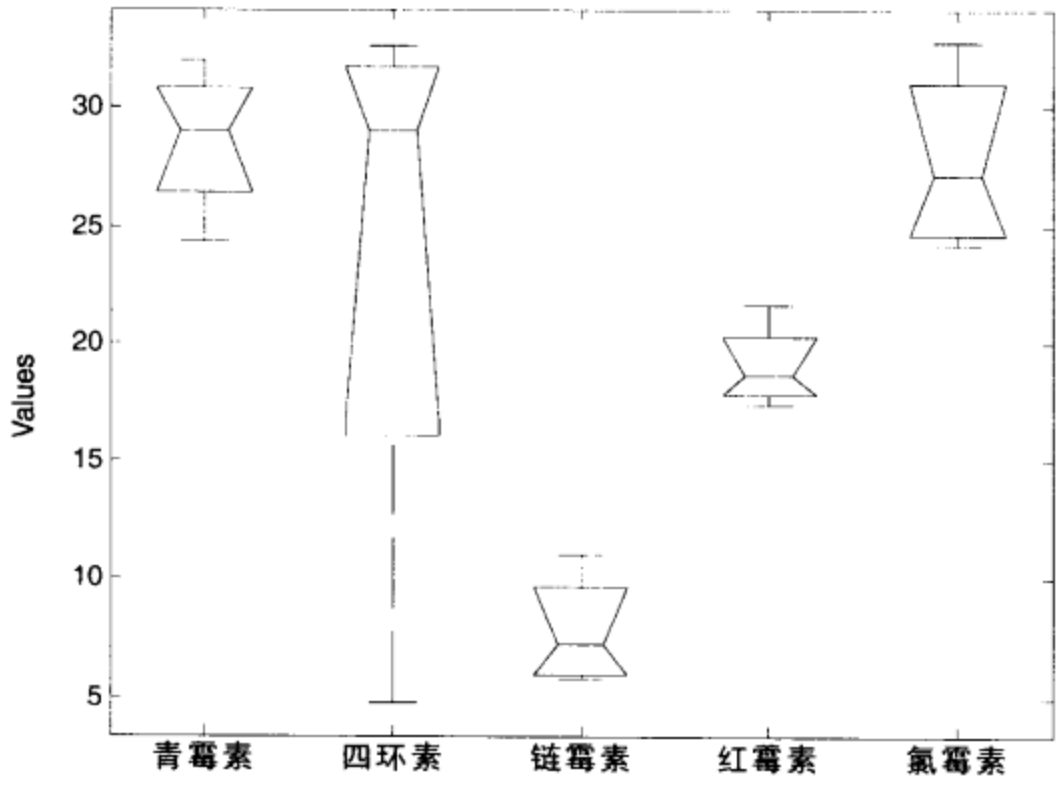

图 11-3　矩阵 X 箱线图

11.6.2　双因素试验的方差分析

MATLAB 中双因素试验的方差分析用函数 anova2 实现。其调用格式如下：

P=anova2(*X*, reps)

它执行平衡的双因素试验的方差分析，来比较 X 中两个或多个列或行的均值。不同列的数据代表某一因素的差异，不同行的数据代表另一因素的差异。如果每行列对有多于一个的观察点，则变量 reps 指出每一单元观察点的数目，每一单元包含 reps 行。

注意

（1）reps 默认值为 1，此时 anova2 返回的 p 值向量包含两个值，分别为原假设因素 A 作用的列向量来自一个总体的 p 值和因素 B 作用的行向量来自一个总体的 p 值。

（2）若 reps 大于 1 时，返回的 p 向量包含 3 个值。第 3 个值为原假设因素 A 和 B 共同作用的所有元素来自一个总体的 p 值。

（3）函数还给出 ANOVA 表格。

【例 11-26】 方差分析综合实例之二。在某种金属材料的生产过程中，对热处理温度（因素 B）与时间（因素 A）各取两个水平，产品强度的测定结果（相对值）如表 11-23 所示。在同一条件下每个试验重复两次。设各水平搭配下强度的总体服从正态分布且方差相同，各样本独立。问热处理温度、时间以及这两者的交互作用对产品强度是否有显著的影响（取 α=0.05）？

表 11-23　不同温度（*B*）、时间（*A*）下产品强度

	*B*1	*B*2
*A*1	38.0 38.6	47.0 44.8
*A*2	45.0 43.8	42.4 40.8

解：用 anova2 函数求解，在 MATLAB 命令窗口中输入：

```
>> x=[38.0,47.0;38.6,44.8;45.0,42.4;43.8,40.8];
>> p=anova2(x,2)
```

输出结果为：

```
p =
    0.0340    0.3009    0.0024
```

同时输出如图 11-4 所示的 ANOVA 表格。

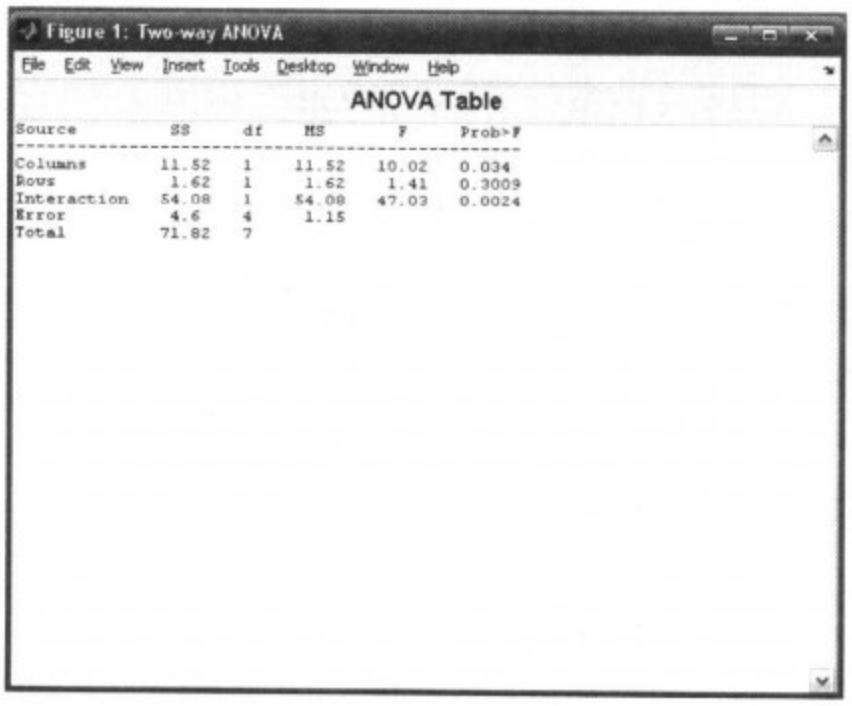

图 11-4　双因素方差分析的 ANOVA 表格

由结果知第一个 p 值代表列样本均值相同假设的 p 值，即反映了 B 因素（温度因素）的影响。由于 p（1）很小，故可得 B 因素对产品强度影响显著。

同理可得 A 因素（时间因素）对产品强度影响不显著（因 p（2）>0.05），A、B 因素的交互作用影响更为显著（$p(3)<0.01$）。

11.7　回归分析

回归分析（Regression Analysis）是研究一个变量 Y 与其他若干变量 X 之间相关关系的一种数学工具。它是在一组试验或观测数据的基础上，寻找被随机性掩盖了的变量之间的依存关系。粗略地讲，可以理解为用一种确定的函数关系去近似代替比较复杂的相关关系。

这个函数称为回归函数，在实际问题中称为经验公式。

回归分析所研究的主要问题就是如何利用变量 X、Y 的观察值（样本），对回归函数进行统计推断，包括对它进行估计及检验与它有关的假设等。

回归分析包含的内容广泛。此处将讨论一元多项式回归、多元线性回归、非线性回归以及逐步回归。

11.7.1 一元多项式回归

如果从数据的散点图上发现 y 与 x 呈现较明显的二次（或高次）函数关系，则可以选用多项式回归。

polyfit 是多项式系数估计函数，其中用参数 n 设定多项式的最高次幂。polyval 与 polyconf 分别用于计算预测输出及置信区间。

1．用函数 polyfit 估计模型参数，其具体调用格式如下：

p=polyfit(x,y,n)　p 返回多项式系数的估计值，n 设定多项式的最高次数。x，y 为对应数据点值。

[p,S]=polyfit(x,y,n)　S 包含 x 的范德蒙矩阵的 QR 分解的 R 的对角线、自由度以及残差，用来作为 polyval 函数的输入，从而估计误差。

2．输出预估值与残差的计算用函数 polyval 实现，其具体调用格式如下：

Y=polyval(p,X)　Y 为 X 在估计参数为 p 的线性模型作用下的输出。

[Y, DELTA] = polyval(p, X, S)　p，S 为 polyfit 的输出，DELTA 为误差估计。在线性回归模型中，Y±DELTA 以 50%的概率包含函数在 X 处的真值。

3．模型预测的置信区间用 polyconf 实现，具体调用格式如下：

[Y, DELTA] = polyconf(P, X, S) 返回预测的 95%置信区间为 Y ± DELTA；

[Y, DELTA] = polyconf(P, X, S, 'alpha', value1) 通过 value1 设定显著性水平 a 的值。

4．交互式画图工具 polytool，调用格式如下：

polytool(x, y, n)

polytool(x, y, n, alpha)

用 n 次多项式拟合 y、x 的值，默认值为 1，并作出显著性水平作为 alpha 的置信区间，默认值为 0.05。

【例 11-27】 回归分析综合实例。将 17 至 29 岁的运动员每两岁一组分为 7 组，每组两人测量其旋转定向能力，以考察年龄对这种运动能力的影响。先得到的一组数据如表 11-24 所示。

表 11-24　年龄与旋转定向能力数据

年龄	17	19	21	23	25	27	29
第一人	20.48	25.13	26.15	30.0	26.1	20.3	19.35
第二人	24.35	28.11	26.3	31.4	26.92	25.7	21.3

试建立两者之间的关系。

解：数据的散点图明显地呈现两端低中间高的形状，所以应拟合一条二次曲线。选用二次模型。通过在 MATLAB 中编写程序 ex1127.m 来实现，具体代码如下所示。

```
%ex1127.m 多项式拟合
%%%输入数据
x0=17:2:29;
x0=[x0,x0];
y0=[20.48,25.13,26.15,30.0,26.1,20.3,19.35...
24.35,28.11,26.3,31.4,26.92,25.7,21.3];
%%%%多项式系数拟合，得系数 p
[p,s]=polyfit(x0,y0,2);
p
%%%%%y 的拟合值及预测值 y 的置信半径 delta
[y,delta]=polyconf(p,x0,s);
y
%%%%%交互式画图
polytool(x0,y0,2)
```

所得系数 p 以及 $x0$ 的预测值如下：

```
>> ex1127
p =
   -0.2003    8.9782  -72.2150
y =
  Columns 1 through 10
    22.5243   26.0582   27.9896   28.3186   27.0450   24.1689   19.6904
22.5243   26.0582   27.9896
  Columns 11 through 14
   28.3186   27.0450   24.1689   19.6904
```

polytool 所得的交互式图形如图 11-5 所示。

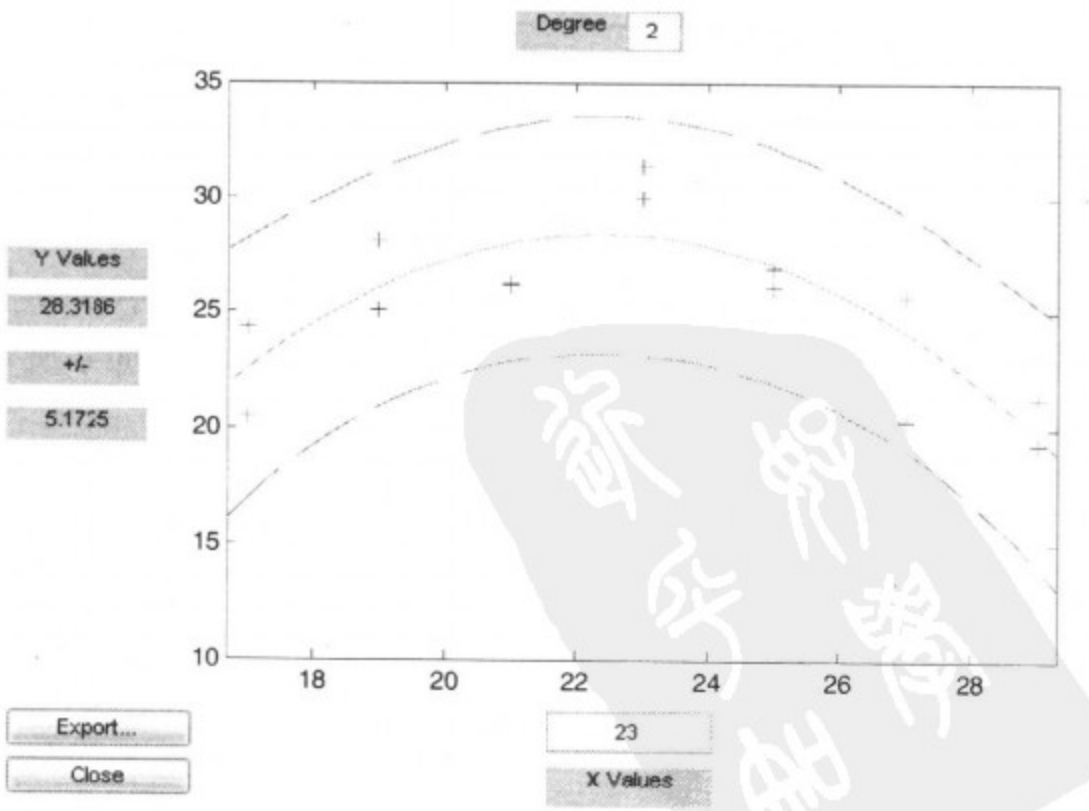

图 11-5　一元二次多项式拟合图

11.7.2 多元线性回归

在 MATLAB 统计工具箱中使用函数 regress 实现多元线性回归。具体调用格式为：

b = regress(y, x)

[b, bint, r, rint, stats] = regress(y, x, alpha)

注意

（1）其中因变量数据向量 $\boldsymbol{Y}$ 和自变量数据矩阵 $\boldsymbol{X}$ 按以下排列方式输入：

$$\boldsymbol{X}=\begin{bmatrix}1 & x_{11} & \cdots & x_{1k}\\ 1 & x_{21} & \cdots & x_{2k}\\ \vdots & \cdots & \cdots & \vdots\\ 1 & x_{n1} & \cdots & x_{nk}\end{bmatrix},\quad \boldsymbol{Y}=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix}$$

（2）对一元线性回归，取 k=1 即可。

（3）alpha 为显著性水平(默认时设定为 0.05)；输出向量为 $\boldsymbol{b}$，bint 为回归系数估计值和它们的置信区间；$\boldsymbol{r}$，rint 为残差及其置信区间；stats 是用于检验回归模型的统计量，有三个数值，第一个是 $\boldsymbol{R}^2$，其中 $\boldsymbol{R}$ 是相关系数，第二个是 $\boldsymbol{F}$ 统计量值，第三个是与统计量 F 对应的概率 P，当 $\boldsymbol{P}<\boldsymbol{\alpha}$ 时拒绝 $\boldsymbol{H}0$，回归模型成立。

（4）残差及其置信区间可以用命令 rcoplot（$\boldsymbol{r}$，rint）画出。

【例 11-28】 多元线性回归综合实例。已知某湖泊八年来湖水中 COD 浓度实测值(y)与影响因素，如湖区工业产值($x1$)、总人口数($x2$)、捕鱼量($x3$)、降水量($x4$)的资料，建立污染物 y 的水质分析模型。

表 11-25 湖水浓度和影响因素数据表

$x1$	1.376	1.375	1.387	1.401	1.412	1.428	1.445	1.477
$x2$	0.450	0.475	0.485	0.500	0.535	0.545	0.550	0.575
$x3$	2.170	2.554	2.676	2.713	2.823	3.088	3.122	3.262
$x4$	0.8922	1.1610	0.5346	0.9589	1.0239	1.0499	1.1065	1.1387
y	5.19	5.30	5.60	5.82	6.00	6.06	6.45	6.95

解：通过在 MATLAB 中编写程序 ex1128.m 来实现，具体代码如下所示。

```
%ex1128.m 多元线性回归
%%%%%输入数据
x1=[1.376, 1.375, 1.387, 1.401, 1.412, 1.428, 1.445, 1.477];
x2=[0.450, 0.475, 0.485, 0.500, 0.535, 0.545, 0.550, 0.575];
x3=[2.170 ,2.554, 2.676, 2.713, 2.823, 3.088, 3.122, 3.262];
x4=[0.8922, 1.1610 ,0.5346, 0.9589, 1.0239, 1.0499, 1.1065, 1.1387];
y=[5.19, 5.30, 5.60,5.82,6.00, 6.06,6.45, 6.95];
%%%%%%保存数据（以数据文件.mat 形式保存，便于以后调用）
save ex1128_data x1 x2 x3 x4 y
%%%%取出数据
load ex1128_data
```

```
%%%%执行回归命令
x=[ones(8,1),x1',x2',x3',x4'];
[b,bint,r,rint,stats]=regress(y',x);
b                          %输出参数
bin                        %各参数估计的置信区间
stats                      %几个特殊统计量
```

执行后，输出如下结果：

```
>> ex1128
b =
  -13.9849
   13.1920
    2.4228
    0.0754
   -0.1897
bint =
  -26.0019   -1.9679
    1.4130   24.9711
  -14.2808   19.1264
   -1.4859    1.6366
   -0.9638    0.5844
stats =
    0.9846   47.9654    0.0047    0.0123
```

所以，回归模型方程是：$\hat{y}=-13.9849+13.1920x_1+2.4228x_2+0.0754x_3-0.1897x_4$。

此外，由 stats 的值可知 R^2=0.9846，F=47.9654，P=0.0123。

11.7.3 非线性回归

非线线性回归包括如下几个用于计算回归参数、预测输出、置信区间以及输出交互图像的函数。

1. 非线性最小二乘参数估计

对于非线性方程的系数估计通常采用最小二乘估计，又叫做非线性最小二乘回归。在 MATLAB 中采用 nlinfit 实现，其调用格式如下。

（1）beta = nlinfit(x, y, fun, beta0)

返回非线性回归方程系数的最小二乘估计值。非线性方程由 fun 给定，fun 为用户提供形如 $y=f$(beta, x)的函数。beta0 为系数初值。

（2）[beta, r, J] = nlinfit(x, y, fun, beta0)

返回拟合系数 beta、残差 R 和 Jacobi 矩阵 $\boldsymbol{J}$。

2. 最小二乘估计参数的置信区间

用非线性最小二乘法估计的系数的置信区间用 nlparci 计算。其输入为 nlinfit 函数的输出 beta，r，$\boldsymbol{J}$。函数具体调用格式为：

（1）ci = nlparci(beta, r, $\boldsymbol{J}$)

返回系数 beta 的 95%置信区间。

（2）ci=nlparci(beta,*r*,*J*,alpha)

返回系数 beta 的 100(1-alpha)%置信区间。

3. 最小二乘估计模型的预测输出及其置信区间

非线性最小二乘估计模型的预测输出及其置信区间用 nlpredci 计算，具体调用格式如下：

（1）[ypred, delta]=nlpredci(FUN, inputs, beta, *r*, *J*)

返回模型在对应 inputs 处的输出预测值 ypred，并给出 95%的置信区间为[ypred-delta, ypred+delta]；

（2）[ypred, delta]=nlpredci(FUN, inputs, beta, *r*, *J*, alpha)

返回模型在对应 inputs 处输出预测值 ypred 以及其 100(1−alpha)%的置信区间[ypred−delta, ypred+delta]。

4. 非线性拟合和预测的交互图形工具

nlintool 是非线性拟合和预测的交互图形工具，其具体调用格式如下：

（1）nlintool(*x*, *y*, fun, beta0)

返回 *x*、*y* 的非线性最小二乘法的曲线拟合图，并画出 95%置信区间；

（2）nlintool(*x*, *y*, fun, beta0, alpha)

给出曲线拟合图及 100(1−alpha)%置信区间；

（3）nlintool(*x*, *y*, fun, beta0, alpha, 'xname', 'yname')

给出曲线拟合图和置信区间，标出 *x*、*y* 变量名称。

【例 11-29】 非线性回归综合实例。在化工生产中获得的氯气的级分 *y* 随生产时间 *x* 下降，假定在 x≥8 时，*y* 与 *x* 之间有如下形式的非线性模型：

$$y = a + (0.49 - a)e^{-b(x-8)}$$

现收集 44 组数据，如下所示。要求利用该数据求 *a*、*b* 的值以及它们 95%置信区间。并显示出拟合曲线。此处设 *a*、*b* 的初值分别为 0.30，0.02。

x：8.00，8.00，10.00，10.00，10.00，10.00，12.00，12.00，12.00，14.00，14.00，14.00，16.00，16.00，16.00，18.00，18.00，20.00，20.00，20.00，20.00，22.00，22.00，24.00，24.00，24.00，26.00，26.00，26.00，28.00，28.00，30.00，30.00，30.00，32.00，32.00，34.00，36.00，36.00，38.00，38.00，40.00，42.00

y：0.49，0.49，0.48，0.47，0.48，0.47，0.46，0.46，0.45，0.43，0.45，0.43，0.43，0.44，0.43，0.43，0.46，0.42，0.42，0.43，0.41，0.41，0.40，0.42，0.40，0.40，0.41，0.40，0.41，0.41，0.40，0.40，0.40，0.38，0.41，0.40，0.40，0.41，0.38，0.40，0.40，0.39，0.39，

解：首先编写函数 funex1129.m，表示出须拟合模型的函数形式。

```
%拟合模型的函数形式
function yhat=funex1129(beta0,x)
a=beta0(1);
```

```
    b=beta0(2);
    yhat=a+(0.49-a)*exp(-b*(x-8));
```

编写 M 文件 ex1129.m 对数据进行最小二乘拟合和处理：

```
    %ex1129.m 非线性最小二乘法拟合曲线
    clear all
    x=[8.00 8.00 10.00 10.00 10.00 10.00 12.00 12.00 12.00 14.00 14.00 14.00 16.00
16.00 16.00 18.00 18.00 20.00 20.00 20.00 20.00 22.00 22.00 24.00 24.00 24.00
26.00 26.00 26.00 28.00 28.00 30.00 30.00 30.00 32.00 32.00 34.00 36.00 36.00
38.00 38.00 40.00 42.00]';
    y=[0.49 0.49 0.48 0.47 0.48 0.47 0.46 0.46 0.45 0.43 0.45 0.43 0.43 0.44 0.43
0.43 0.46 0.42 0.42 0.43 0.41 0.41 0.40 0.42 0.40 0.40 0.41 0.40 0.41 0.41 0.40
0.40 0.40 0.38 0.41 0.40 0.40 0.41 0.38 0.40 0.40 0.39 0.39]';
    beta0=[0.30 0.02];                                    %系数初值
    [beta,r,j]=nlinfit(x,y,@funex0829,beta0);             %系数最小二乘估计值为 beta
    beta
    ci=nlparci(beta,r,j)                                  %求 95%置信区间
    nlintool(x,y,@funex0829,beta0,0.05,'生产时间 x','氯气级分 y')
                                                          %显示拟合结果和置信区间
```

```
    >> ex0829
    beta =
       0.3896    0.1011
    ci =
       0.3805    0.3988
       0.0771    0.1251
```

可知 a、b 的最小二乘估计值为 0.3896，0.1011，它们的 95%置信区间分别为[0.3805, 0.3988]和[0.0771, 0.1251]。所以用最小二乘法对该模型系数进行估计，最终的模型表达式为：

$$y = 0.3896 + 0.1004e^{-0.1011(x-8)}$$

最终得到的拟合图如图 11-6 所示。

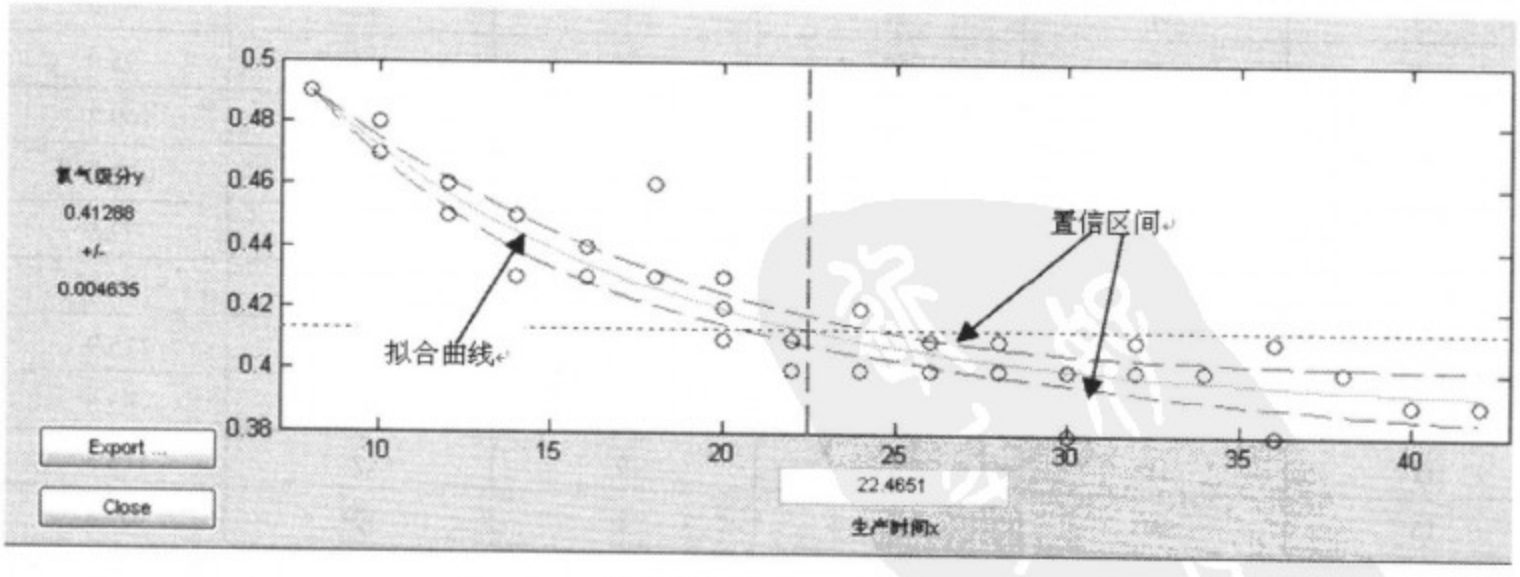

图 11-6　非线性最小二乘法拟合曲线图

图 11-6 中，中间的实线为拟合曲线，两侧的虚线标出其 95%置信区间。

11.7.4 逐步回归

实际问题中影响因变量的因素可能很多，希望从中挑选出影响显著的自变量来建立回归模型，这就涉及变量选择的问题。逐步回归是一种从众多变量中有效地选择重要变量的方法。

逐步回归的命令是 stepwise，它提供了一个交互式画面，通过此工具可以自由地选择变量，进行统计分析。其调用格式为：

stepwise(*x*, *y*, inmodel, alpha)

其中 *x* 是自变量数据，*y* 是因变量数据，分别为 *n*×*m* 和 *n*×1 矩阵，inmodel 是矩阵的列数指标（默认值为全部自变量），alpha 为显著性水平（默认值为 0.5）。

结果产生三个图形窗口，在“stepwise plot”窗口中，虚线表示该变量的拟合系数与 0 无显著差异，实线表示有显著差异；红色线表示从模型中移去的变量，蓝色线表明存在模型中的变量，点击一条会改变其状态。

在 stepwise Table 窗口中列出一个统计表，包括回归系数及其置信区间，以及模型的统计量剩余标准差（RMSE）、相关系数（*R*-square）、*F* 值和 *P* 值。

【例 11-30】 逐步回归综合实例。水泥凝固时放出的热量 *y* 与水泥中四种化学成分 *x*1、*x*2、*x*3、*x*4 有关，今测得一组数据如下，试用逐步回归法来确定一个线性模型。

表 11-26 逐步回归数据

序　号	*x*1	*x*2	*x*3	*x*4	*y*
1	7	26	6	60	78.5
2	1	29	15	52	74.3
3	11	56	8	20	104.3
4	11	31	8	47	87.6
5	7	52	6	33	95.9
6	11	55	9	22	109.2
7	3	71	17	6	102.7
8	1	31	22	44	72.5
9	2	54	18	22	93.1
10	21	47	4	26	115.9
11	1	40	23	34	83.8
12	11	66	9	12	113.3
13	10	68	8	12	109.4

解：通过在 MATLAB 中编写程序 ex1130.m 来实现，具体代码如下所示。

```
%ex1130.m 逐步回归
```

在 MATLAB 命令窗口中输入：

```
clc,clear
x0=[1 7 26 6 60 78.5
2 1 29 15 52 74.3
3 11 56 8 20 104.3
4 11 31 8 47 87.6
5 7 52 6 33 95.9
6 11 55 9 22 109.2
7 3 71 17 6 102.7
8 1 31 22 44 72.5
9 2 54 18 22 93.1
10 21 47 4 26 115.9
11 1 40 23 34 83.8
12 11 66 9 12 113.3
13 10 68 8 12 109.4];
x=x0(:,2:5);                    %x 数值初始化
y=x0(:,6);                      %y 数值初始化
stepwise(x,y,[1:4])             %所有变量都包括的模型
```

输出如图 11-7 所示的结果。

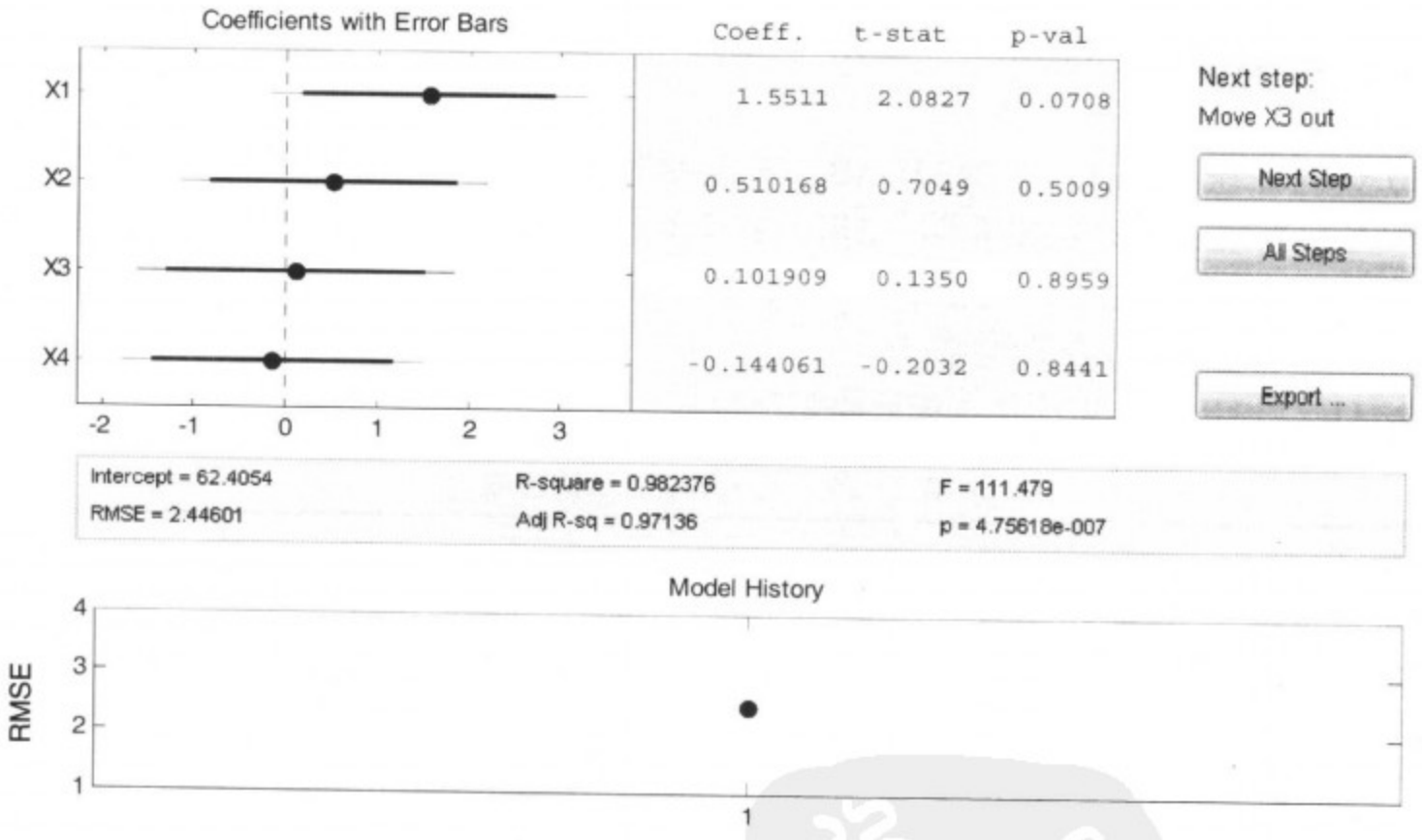

图 11-7　包含所有变量的模型图

由图 11-7 可以看出，$x3$、$x4$ 的 p 值较大，其对模型影响不显著。

若要移去 $x3$、$x4$ 两个变量，在 MATLAB 命令窗口中继续输入：

```
>> stepwise(x,y,[1 2])
```

输出如图 11-8 所示的结果，它就是移去变量 $x3$、$x4$ 后的模型图。

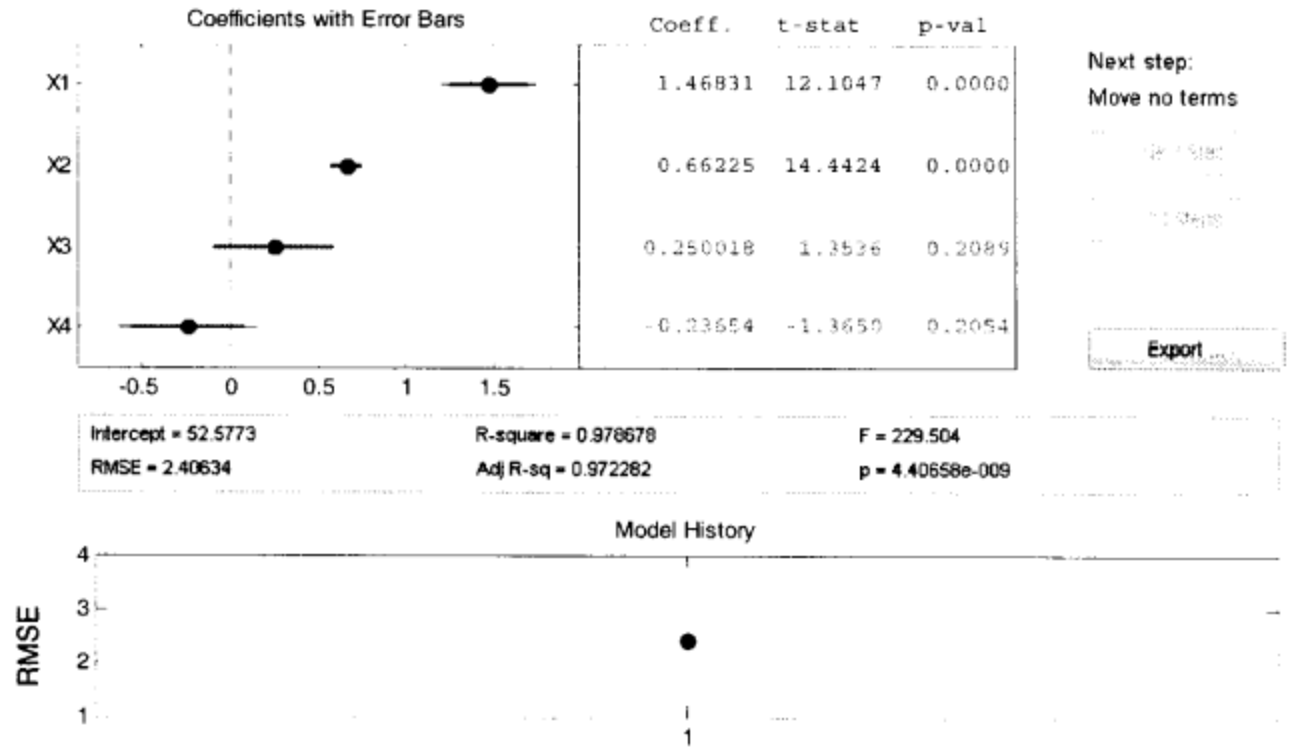

图 11-8　移去变量 $x3$、$x4$ 后的模型图

11.8　统计图绘制

MATLAB 统计工具箱在 MATLAB 丰富的绘图功能上又增添了一些特殊的图形表现函数。如 box 图用以描述数据样本，也用于通过图形来比较多个样本的均值；正态概率图可以从图形上检验样本是否为正态分布；分位数-分位数图用于比较两样本的分布；拟合曲线图给出当前数据点的拟合曲线等。

表 11-27 给出了常用的几种统计绘图函数。

表 11-27　常用的统计绘图函数

函 数 名	调用格式	函数用法	功　能
boxplot	boxplot(*X*,'param1',val1, 'param2',val2,...)	为 *X* 的每一列绘制一个 box 图。param1,param2 等设置图形参数 val1，val2 等为对应参数的取值	在矩形框内画样本数据
errorbar	errorbar(*X*,*Y*,*L*,*U*,symbol) errorbar(*X*,*Y*,*L*) errorbar(*Y*,*L*)	给出 *X*-*Y* 图以及由 *L* 和 *U* 规定误差界限的误差条。误差条与点（*X*，*Y*）上面的距离为 *U*(*i*),下面的距离为 *L*(*i*)。*X*、*Y*、*L*、*U* 长度需相同。symbol 为一字符串，规定线型颜色等	在曲线上画误差条
fsurfht	fsurfht('fun',xlims,ylims)	fun 为用户指定函数，xlims、ylims 分别给出 *x* 轴与 y 轴上的范围限制	画函数的交互轮廓线
gline	gline gline(fig) *h* = gline(fig)	通过鼠标确定两端点在当前图上绘制线段； 在图 fig 上画线段;返回线段的句柄 *h*	在图中交互式画线
gname	gname(cases) gname *h*=gname(cases,line_handle)	cases 为一个字符串矩阵。当移动鼠标确定需做标记点后，按“Enter”键或“Escape”键，就会在该点处输出 cases 包含的字符串	用指定的标志画点

续表

函 数 名	调用格式	函数用法	功　　能
lsline	lsline *h* = lsline	为当前坐标系的每一线性数据组绘制出其最小二乘拟合线。*h* 为线条句柄	画最小二乘拟合线
normplot	normplot(*X*) *h* = normplot(*X*)	显示数据 *X* 的正态概率图。若 *X* 为矩阵，则其每一列显示一条线。若数据服从正态分布，则图形为线性，其他分布对应曲线	画正态检验的正态概率图
pareto	pareto(*y*,names)	将矢量 *y* 中的每个元素按元素数值递减顺序绘成直方条，并以 names 中的名称对直方条进行标记	画统计过程控制的 Pareto 图
qqplot	qqplot(*X*,*Y*) qqplot(*X*,*Y*,pvec)	如果两样本来自同一分布，则绘出的线为直线。默认值时将位于第一分位数和第三分位数间的数据拟合绘制成一条线，矢量 pvec 可规定分位数	画两样本的分位数-分位数图
refcurve	*h* = refcurve(*p*)	*p* 表示多项式系数矩阵，在当前图中绘出 *p* 代表的多项式曲线。*h* 返回曲线句柄	在当前图中加一多项式曲线
rcoplot	rcoplot(r,rint)	r,rint 来自函数 regress 的输出，该函数最后按数据顺序给出各数据点的误差条	根据样本回归后的残差和置信区间作出误差条图
refline	refline(slope,intercept) refline(slope) refline	在图中给出斜率为 slope、截距为 intercept 的直线；后一个式中的 slope 为一个二元矢量，第一个元素为斜率，第二个元素为截距；无参数时，给出最小二乘拟合线	在当前坐标中画参考线
surfht	surfht(*z*) surfht(*x*,*y*,*z*)	根据给出的 *x*、*y*、*z* 数据，提供任意的 *x*、*y* 的坐标值上的 *z* 的内插值	画交互轮廓线
weibplot	wblplot(*x*)	将 *x* 中的数据显示为概率曲线。如果数据服从威布尔分布，则绘出的线为直线	画 Weibull 概率图

下面分别举例进行说明。

1. box 图绘制

在 MATLAB 命令窗口中输入：

```
>> x1 = normrnd(5,1,100,1);
>> x2 = normrnd(6,1,100,1);
>> boxplot([x1,x2],'notch','on') %画出带切口的 box 图
```

输出如图 11-9 所示的 box 图。

图形说明：盒子的上下两条线分别为样本的 25%和 75%分位数，中间的线表示样本中位数。从图中可知中位数在盒子中间，因而两组数据都大致关于它们的均值对称。

虚线表示样本的其余部分，位于盒子的上下侧。

切口表示样本中位数的置信区间，默认情况下没有切口。此外还可以用命令：boxplot([*x*1,*x*2]，'notch'，'on'，'whisker'，1)标出超出 1.0 倍的四分位数距离的样本奇异点。

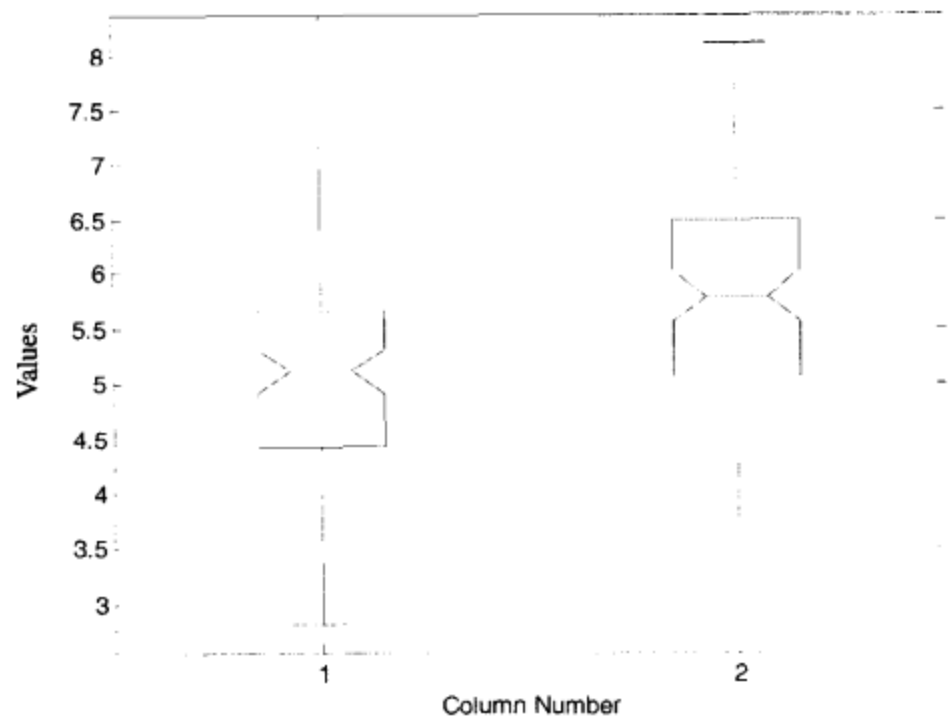

图 11-9　box 图

2. 正态概率图绘制

在 MATLAB 命令窗口中输入：

```
>> x = normrnd(0,1,50,1); %产生正态随机数
>> h = normplot(x);
```

输出如图 11-10 所示的正态概率图。

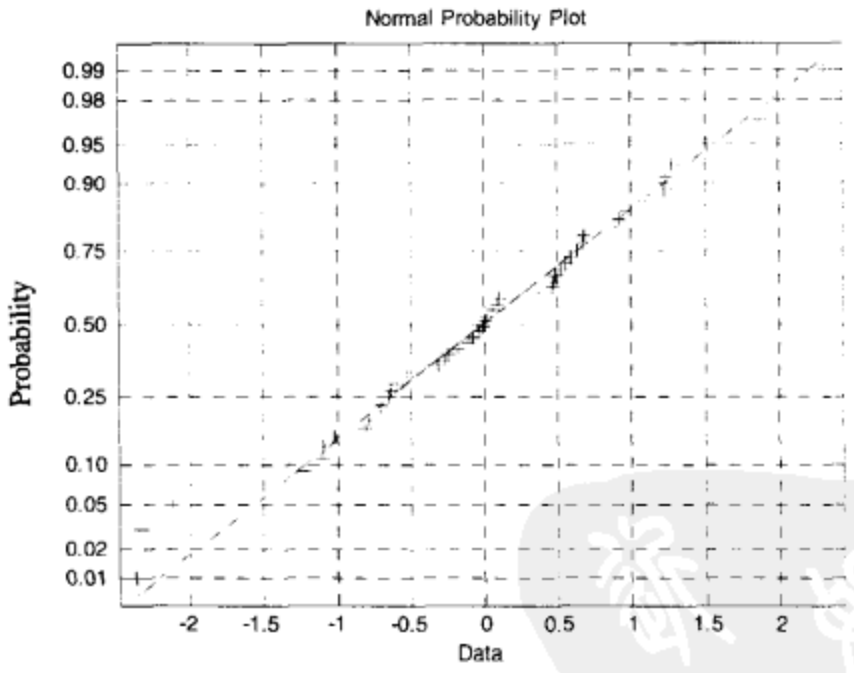

图 11-10　正态随机数的正态概率图

图形说明：由于样本是由正态随机数发生器产生的，因此其服从正态分布，故所得的概率图呈线性。叠加在数据上的实线为 x 中数据的第一和第三分位间的连线，有助于评估数据线性程度。

若样本不服从正态分布，则所得的概率图有所弯曲，如在 MATLAB 命令窗口中输入：

```
>> x=unifrnd(0,1,[30,1]);    %产生[0,1]上的均匀分布随机数
>> normplot(x)
```

输出如图 11-11 所示的正态概率图。

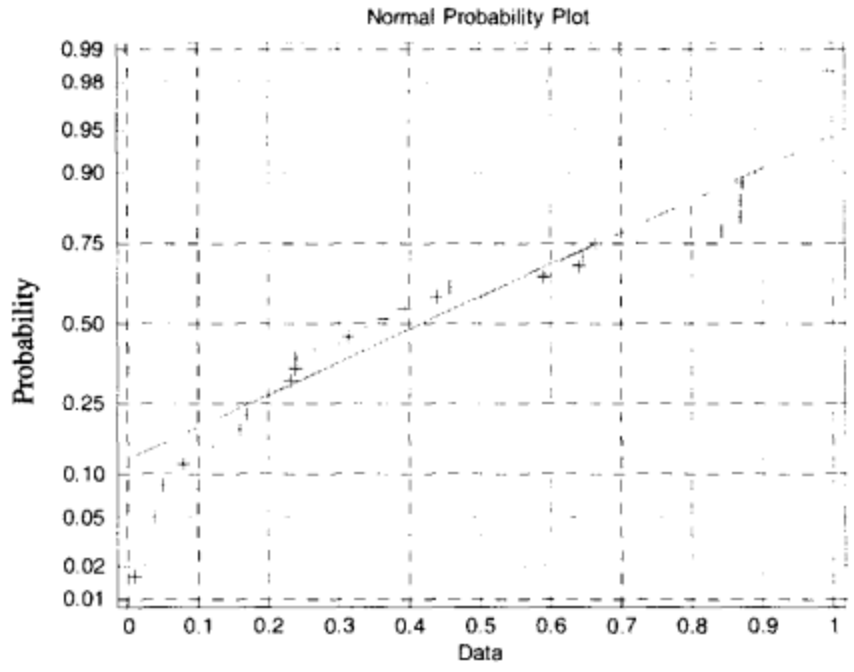

图 11-11　均匀分布的正态概率图

3. 分位数－分位数图的绘制

在 MATLAB 命令窗口中输入：

```
>>x = normrnd(0,1,100,1);          %产生均值为 0，方差为 1 的 100 个正态随机数
>>y = normrnd(0.5,2,50,1);         %产生均值为 0.5，方差为 2 的 50 个正态随机数
>>qqplot(x,y);                     %作出分位数－分位数图
```

输出如图 11-12 所示的分位数－分位数图。

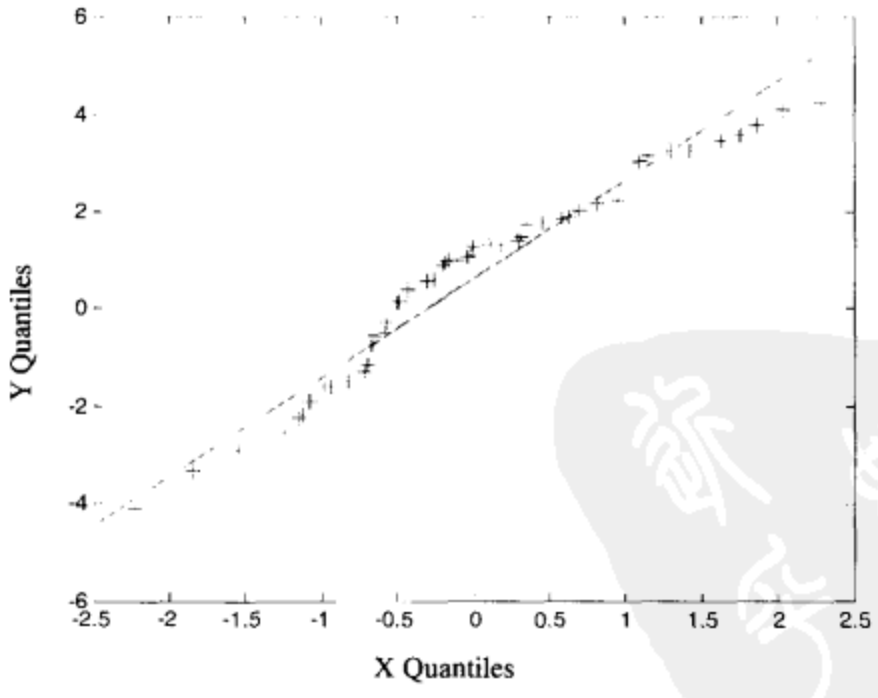

图 11-12　均值方差不同的两正态分布的分位数－分位数图

图形说明：由于 x、y 均值和方差均不同，即 x、y 数据不是来自同一分布，故所得的分位数－分位数图表现出一定的弯曲。

中间的直线是将位于第一分位数和第二分位数之间的数据拟合绘制而成的。

4. 最小二乘拟合线的绘制

在 MATLAB 命令窗口中输入：

```
>> y = [2 3.4 5.6 8 11 12.3 13.8 16 18.8 19.9]';    %输入一些数据点
>> plot(y,'+');                                      %绘出这些数据点
>> lsline;                                           %对这些数据进行最小二乘拟合
```

输出如图 11-13 所示的最小二乘拟合线图。

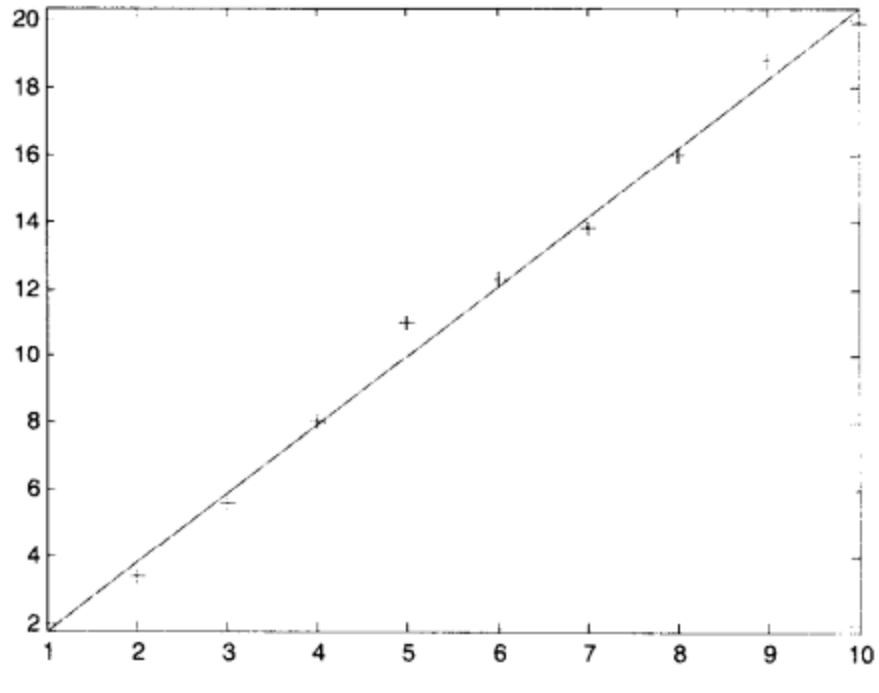

图 11-13　最小二乘拟合线

11.9　小结

本章主要讲述了 MATLAB 在概率统计中的应用。具体包括：常用统计量的计算，MATLAB 支持的 20 多种特殊分布的相关计算，参数估计，假设检验以及方差分析和回归分析。另外还讲述了统计学中绘图工具的使用。

本章中的函数均来自 MATLAB 的统计工具箱（Statistics Toolbox），该工具箱功能强大。在各小节中都以表格形式给出了该工具箱中的各类函数，并给出了其中常用函数的实例。对于其他不常用的一些函数，读者具体使用时可参考 MATLAB Help。熟练掌握和运用这些函数，无疑会给概率统计工作带来极大的方便。

第 12 章

最优化计算

最优化指的是在一定条件下，寻求使目标函数最大（小）的决策，最优化计算在实际中有着广泛的应用。可以利用 MATLAB 提供的最优化计算工具箱中的函数进行最优化计算，也可以编程实现相应的最优化算法来计算。

通过本章，读者不必掌握复杂的各种优化算法，便可轻松地利用 MATLAB 进行最优化计算，解决一些常见的最优化问题。

12.1 无约束最优化

常见的无约束最优化方法有：黄金搜索法、二次插值法、Nelder-Mead 方法（单纯型算法）、最速下降法、牛顿方法、模拟退火法（SA）和遗传算法（GA），利用 MATLAB，可以方便地实现这些算法，下面分别进行讲述。

12.1.1 黄金搜索法

1. 算法原理

黄金搜索法也叫 0.618 法，适用于求解如下形式的最优化问题：

$$\min f(x)，\quad x\in[a,b]$$

其中 a，b 已知，且 $f(x)$ 在区间内为单峰函数，也就是说 $f'(x)$ 在区间内最多改变一次，因此该类最优化问题可等价于求解 x^*，使 $f(x)$ 分别在区间 $[a,x^*]$ 上单调减，在 $[x^*,b]$ 上单调增。

2. 算法步骤

黄金搜索法的基本求解流程如下。

步骤 1：在区间[a,b]中取两点 $c=a+(1\text{-}r)h$ 和 $d=a+rh$。其中 $r=(\sqrt{5}-1)/2$，$h=b-a$；

步骤 2：如果 $f(x)$ 在这两点的值几乎相等并且区间的宽度充分小，则停止迭代跳出循环，并令 $x^*=c$（$f(c)\leqslant f(d)$）或 $x^*=d$（$f(c)>f(d)$），否则，跳到步骤 3；

步骤 3：如果 $f(c)<f(d)$，则令 $b=d$。否则，令 $a=c$，跳到步骤 1。

3. 算法的 MATLAB 实现

在 MATLAB 中编程实现的黄金搜索算法为：Opt_Golden。

功能：黄金搜索算法求区间上的无约束最优化解。

调用格式：[*xo*, *fo*]=Opt_Golden(*f*, *a*, *b*, TolX, TolFun, *k*)。

其中，*f* 为函数名；

a 为搜索区间起始点；

b 为搜索区间终止点；

TolX 为最优值点间的误差阈值；

TolFun 为函数的误差阈值；

K 为最大迭代次数；

xo 为最优化点值；

fo 为函数在点 *xo* 处的函数值。

黄金搜索法的 MATLAB 编程实现如下。

算法程序 Opt_Golden.m 的代码如下所示。

```
function [xo,fo]=Opt_Golden(f,a,b,TolX,TolFun,k)
%黄金搜索算法求在区间[a,b]上的最优化解
%f：函数名；
%a：搜索区间起始点；
%b：搜索区间终止点；
%TolX：最优值点间的误差阈值；
%TolFun：函数的误差阈值；
%k：最大迭代次数；
%xo：最优化点值；
%fo：函数在点 xo 处的函数值。

r =(sqrt(5)-1)/2;                    %r 为黄金分割点值，
h = b-a;                             %区间宽度
rh = r*h;
%%%取两点 c、d,并计算相应的函数值 fc 和 fd
c = b-rh;
d = a+rh;
fc = feval(f,c);
fd = feval(f,d);
%%%算法第二步判断是否停止迭代
if k <= 0 | (abs(h) < TolX & abs(fc - fd) < TolFun)
    if fc <= fd
        xo = c;
        fo = fc;
    else
        xo = d;
```

```
        fo = fd;
    end
    if k == 0
        fprintf('最好设定迭代次数大于 0');
    end
%%%%算法第三步，进行新一轮迭代
else
    if fc < fd
        [xo,fo] = Opt_Golden(f,a,d,TolX,TolFun,k-1);
    else
        [xo,fo] = Opt_Golden(f,c,b,TolX,TolFun,k-1);
    end
end
```

4. 算法举例

【例 12-1】 黄金搜索法求解无约束最优化问题实例。采用黄金搜索法求如下函数的最小值：

$$f(x)=x-(x^2-2)^3/2\text{，其中 } x\in[0,4]$$

解：用黄金搜索算法解此最优化问题。调用以上编写的 Opt_Golden()函数求解如下：

```
%ex1201.m 用 Opt_Golden()求解最优化
f1001 = inline('x-(x.*x-2).^3/2','x');
a = 0;
b = 4;
TolX = 1e-4;                %判断循环是否停止的 x 阈值
TolFun = 1e-4;              %函数阈值
MaxIter = 100;              %设定迭代次数
[xo,fo] = Opt_Golden(f1001,a,b,TolX,TolFun,MaxIter)
```

所得结果为：

```
>> ex1201
xo =
    4.0000
fo =
-1.3680e+003
```

由计算结果可知，在 $x\in[0,4]$ 中，当 $x=4$ 时，函数 $f(x)=x-(x^2-2)^3/2$ 取最小值−1368。

12.1.2　二次插值法

插值法是一种重要的线性搜索方法，其基本思想是在搜索区间中不断用低次（通常不超过三次）插值多项式来近似目标函数，并逐步用插值多项式的极小点来逼近目标函数的极小点。当函数具有比较好的解析性质时，插值方法比直接方法效果更好。此处主要介绍三点两次插值法。此算法也用来求区间上的无约束最优化解。

1. 算法原理

在包含 $f(x)$ 极小值 x^* 的区间$[a,b]$中，给定三点 x_0,x_1,x_2，其对应的函数值分别为 f_0,f_1,f_2，且满足 $x_0<x_1<x_2$。可以构造二次函数 $q(x)$，使其在三点处的值等于 $f(x)$ 对应

的函数值。可以根据 $q'(x)=0$，求得该二项式取最小值时的点 x_3 如下所示：

$$x_3=\frac{1}{2}\frac{(x_1^2-x_2^2)f_0+(x_2^2-x_0^2)f_1+(x_0^2-x_1^2)f_2}{(x_1-x_2)f_0+(x_2-x_0)f_1+(x_0-x_1)f_2}$$

根据求出的 x_3 的大小情况确定取代三点中的某一点。直到 $|x_2-x_0|<\varepsilon_1$，或 $|f_2-f_0|<\varepsilon_2$ 时停止迭代，并令 x_3 为对应小值的点。

2. 算法步骤

二次插值法的基本求解流程如下。

步骤 1：比较 x_3 与 x_1 大小，若 $x_3<x_1$，转步骤 2。否则，转到步骤 3；

步骤 2：如果 $f(x_3)<f(x_1)$，则 $x_2\Leftarrow x_1, x_1\Leftarrow x_3$，转到步骤 4；否则，$x_0\Leftarrow x_3$，转到步骤 4；

步骤 3：若 $f(x_3)\leqslant f(x_1)$，则 $x_0\Leftarrow x_1, x_1\Leftarrow x_3$，转到步骤 4；否则，$x_2\Leftarrow x_3$，转到步骤 4。

步骤 4：区间收缩完毕，在新的搜索区间[x_0, x_2]上计算插值函数的极小点 x_3。

3. 算法的 MATLAB 实现

在 MATLAB 中编程实现的二次插值算法为：Opt_Quadratic。

功能：二次插值算法求区间上的无约束最优化解。

调用格式：[*xo*,*fo*]=Opt_Quadratic(*f*, *x*, TolX, TolFun, MaxIter)。

其中，*f* 为函数名；

x 为给出搜索区间；

TolX 为最优值点间的误差阈值；

TolFun 为函数的误差阈值；

MaxIter 为最大迭代次数；

xo 为最优化点值；

fo 为函数在点 xo 处的函数值。

二次插值法的 MATLAB 编程实现如下：

算法程序 Opt_Quadratic.m 的具体代码如下所示。

```
function [xo,fo] = Opt_Quadratic(f,x,TolX,TolFun,MaxIter)
% 用二次插值求 f(x)最优解
%f：函数名；
%x：给出搜索区间；
%TolX：最优值点间的误差阈值；
%TolFun：函数的误差阈值；
%MaxIter：最大迭代次数；
%xo：最优化点值；
%fo：函数在点 xo 处的函数值。

%%%根据输入确定三点的初始值，并求出对应的函数值
if length(x) > 2
    x012 = x(1:3);
```

```
else
    if length(x) == 2
        a = x(1);
        b = x(2);
    else
        a = x-10; b = x+10;
    end
    x012 = [a (a + b)/2 b];
end
f012 = f(x012);
%%%%初始化三点值
x0 = x012(1);
x1 = x012(2);
x2 = x012(3);
f0 = f012(1);
f1 = f012(2);
f2 = f012(3);
%%%%%求出插值二项式的取最小值处的 x3 以及相应的最小值 f3
nd = [f0-f2 f1-f0 f2-f1]*[x1*x1 x2*x2 x0*x0; x1 x2 x0]';
x3 = nd(1)/2/nd(2);
f3 = feval(f,x3);
%%%%判断是否停止迭代
if MaxIter<= 0 | abs(x3 - x1) < TolX | abs(f3 - f1) < TolFun
    xo = x3;
    fo = f3;
else
%%%%%根据求出的多项式值与区间中三点值的大小比较确定相应的新的三点
    if x3 < x1
        if f3 < f1
            x012 = [x0 x3 x1];
            f012 = [f0 f3 f1];
        else
            x012 = [x3 x1 x2];
            f012 = [f3 f1 f2];
        end
    else
        if f3 <= f1
            x012 = [x1 x3 x2];
            f012 = [f1 f3 f2];
        else
            x012 = [x0 x1 x3];
            f012 = [f0 f1 f3];
        end
    end
%%%%%进行新一轮迭代
    [xo,fo] = Opt_Quadratic(f,x012,TolX,TolFun,MaxIter-1);
end
```

4. 算法举例

【例 12-2】 二次插值法求解无约束最优化问题实例。采用二次插值法求解如下表达式的最小值：

$$f(x)=(x^2-2)^2/2-1,\quad x\in[0,5]$$

解： 调用 Opt_Quadratic()程序求解如下：

```
%ex1202.m 用插值法求解最优化问题例 12-2
clear all
f1002=inline('(x.*x - 2).^2/2-1', 'x');;
%%%设定区间端点
a = 0;
b = 5;
%%%分别设定 x 阈值和函数 f 阈值
TolX = 1e-5;
TolFun = 1e-8;
%%%设定迭代次数
MaxIter = 100;
%%%用二次插值函数求解
[xoq,foq] = Opt_Quadratic(f1002,[a b],TolX,TolFun,MaxIter)
%用 MATLAB 内置函数求最小值以及它的函数值
[xob,fob] = fminbnd(f1002,a,b)      %MATLAB 内置函数求解
```

结果为：

```
>> ex1202
xoq =
    1.4142
foq =
   -1.0000
xob =
    1.4142
fob =
   -1.0000
```

由结果可知，在 $x\in[0,5]$ 中，当 $x=1.4142$ 时，函数 $f(x)=(x^2-2)^2/2-1$ 取最小值−1。且用二次插值法求得的解与用 MATLAB 内置函数 fminbnd 求得的解相同。

对于 MATLAB 中的几个求最优化的内置函数，将在最后一节中对其有具体说明。

12.1.3 Nelder-Mead 算法

Nelder-Mead 算法是求多维函数极值的一种算法，由 Nelder 和 Mead 提出，又叫单纯形算法。由于未利用任何求导运算，算法比较简单，但收敛速度较慢，适合变量数不是很多的方程求极值。

1. 算法原理

Nelder-Mead 法是利用多面体来逐步逼近最佳点 x^* 的。设函数变量为 n 维，则在 n 维空间里多面体有（n+1）个顶点。设 $x_1,x_2,\cdots,x_{n+1}$ 为多面体的顶点，且满足：

$$f(x_1)\leqslant f(x_2)\leqslant\cdots\leqslant f(x_{n+1})$$

Nelder-Mead 法试着将多面体中最差的顶点 $x_{n+1}^{(k)}$（也就是函数值最大的点）以新的最佳的点替代，来更新多面体，使之逼近至最佳解。更新的设定方式有四种，分别是：

- 反射（reflection）
- 扩展（expansion）
- 外收缩（outside contraction）
- 内收缩（inside contraction）

如果上面四种更新方式皆不适用，则进行变小（shrink）的步骤。

下面以二维空间为例，给出几种更新方式的示意图，如图 12-1 所示。

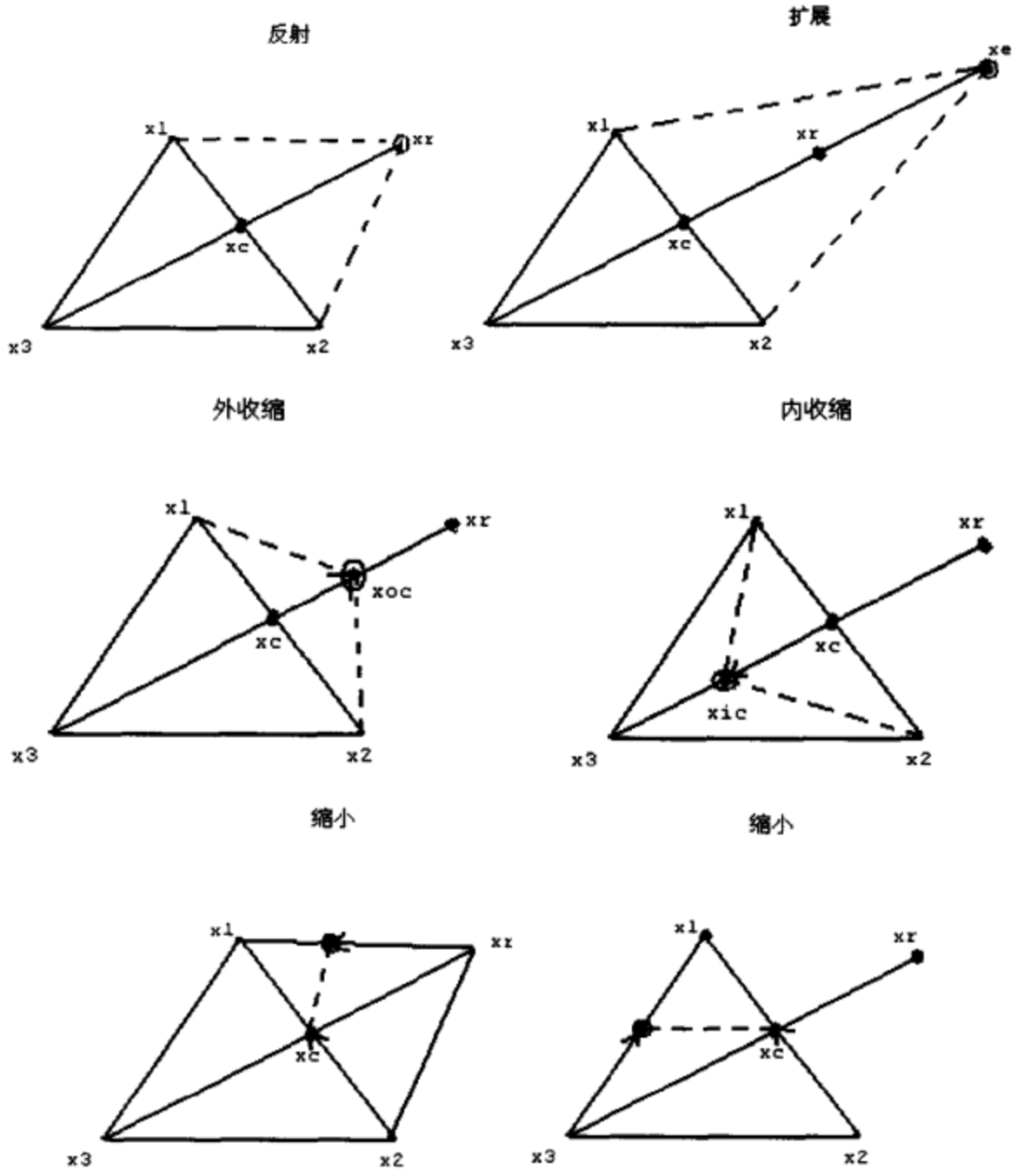

图 12-1　Nelder-Mead 方法示意图

图 12-1 中给出了 Nelder-Mead 方法的搜索全过程。

基本搜索过程为：首先，程序判断当前多面形顶点中是否有满足误差阈值内的最小值点，若存在，则终止程序并输出结果。若不满足则依据图中的几种方法依次更新多边形的顶点，并循环判断是否满足阈值条件。通过多边形点的不断更新，最终可得最小值点。

2. 算法的 MATLAB 实现

在 MATLAB 中编程实现的 Nelder-Mead 算法为：Opt_Nelder。

功能：Nelder-Mead 算法求无约束最优化解。

调用格式：[*xo*, *fo*] = Opt_Nelder (*f*, *x*0, TolX, TolFun, MaxIter)。

其中，*f* 为函数名；

*x*o 为搜索初始值；

TolX 为最优值点间的误差阈值；

TolFun 为函数的误差阈值；

MaxIter 为最大迭代次数；

xo 为最优化点值；

fo 为函数在点 *xo* 处的函数值。

Nelder-Mead 法的 MATLAB 编程实现如下。

算法程序为 Neldr0.m 和 Opt_Nelder.m。其中子程序 Nelder0.m 用于二维空间上的多边形最优化逼近。对于维数大于 2 的最优化问题，可以通过若干二维的迭代计算求出最优解。Opt_Nelder.m 可求解若干维变量的最优化问题。

（1）Nelder0.m 文件代码

```
function [xo,fo] = Nelder0(f,abc,fabc,TolX,TolFun,k)
%二维空间中的多边形逼近
%f:函数名
%abc:二维空间三个顶点值
%fabc:三个顶点处的函数值
%TolX:最优点值的误差阈值
%TolFun:最优点处函数值的误差阈值
%k:最大迭代次数
%%%%确定三个顶点 a，b，c 并且其函数值按从小到大排列
[fabc,I] = sort(fabc); %将二维空间中的多边形三个顶点的函数值按从小到大顺序排列
a = abc(I(1),:);
b = abc(I(2),:);
c = abc(I(3),:);
fa = fabc(1);
fb = fabc(2);
fc = fabc(3);
%%%判断三点或三点函数值的距离是否小于给定阈值.若小于阈值则停止循环，得最优解 x0=a
fba = fb - fa;                    %a,b 点函数值之差
fcb = fc - fb;                    %b，c 点函数值之差
if k <= 0 | abs(fba) + abs(fcb) < TolFun | abs(b - a) + abs(c - b) < TolX
    xo = a;
    fo = fa;          %小于阈值
else                  %大于阈值，更新多边形顶点
    m = (a + b)/2;
    e = 3*m - 2*c;     %扩展（expansion）
    fe = feval(f,e);
    if fe < fb
        c = e;
        fc = fe;
    else
        r = (m+e)/2;          %反射（reflection）
        fr = feval(f,r);
        if fr < fc
            c = r;
            fc = fr;
        end
        if fr >= fb
            s = (c + m)/2;        %内收缩（inside contraction）
            fs = feval(f,s);
```

```
            if fs < fc
                c = s;
                fc = fs;
            else
                b = m;
                c = (a + c)/2;   %变小（shrink）
                fb = feval(f,b);
                fc = feval(f,c);
            end
        end
    end
    [xo,fo] = Nelder0(f,[a;b;c],[fa fb fc],TolX,TolFun,k - 1);
end
```

（2）Opt_Nelder.m 文件代码

```
function [xo,fo] =Opt_Nelder(f,x0,TolX,TolFun,MaxIter)
%Nelder-Mead 法用于多维变量的最优化问题，维数>=2.
%f:函数名
%x0:多维空间中的搜索初值
%TolX:最优点值的误差阈值
%TolFun:最优点处函数值的误差阈值
%MaxIter:最大迭代次数

N = length(x0);
if N == 1                          %一维情况，用二次逼近计算
    [xo,fo] = Opt_Quadratic(f,x0,TolX,TolFun,MaxIter);
    return
end
S = eye(N);
for i = 1:N                        %自变量维数大于 2 时，重复计算每个子平面的情况
    i1 = i + 1;
    if i1 > N
        i1 = 1;
    end
    abc = [x0; x0 + S(i,:); x0 + S(i1,:)]; %每一个定向子平面
    fabc = [feval(f,abc(1,:)); feval(f,abc(2,:)); feval(f,abc(3,:))];
    [x0,fo] = Nelder0(f,abc,fabc,TolX,TolFun,MaxIter);
    if N < 3                       %二维情况不需重复
        break;
    end
end
xo = x0;
```

3. 算法举例

【例 12-3】 Nelder-Mead 算法求解无约束最优化问题实例。采用 Nelder-Mead 算法求如下函数的最小值问题：

$$f(x,y)=x(x-5-y)+y(y-4)$$

解： 在 MATLAB 中编写函数 ex1203.m 来进行求解，具体代码如下所示。

```
%ex1203.m 用 Nelder-Mead 方法求最优化解
f1003 = inline('x(1)*(x(1)-5-x(2))+x(2)*(x(2)-4)','x');
x0 = [0 4];
```

```
TolX = 1e-4;
TolFun = 1e-9;
MaxIter = 100;
[xN,fN] = Opt_Nelder(f1003,x0,TolX,TolFun,MaxIter)
%取最小值点以及此处的最小值
[xF,fF] = fminsearch(f1003,x0) %用 MATLAB 内置函数 fminsearch 求解
```

运行结果为：

```
>> ex1203
xN =
    4.6667    4.3333
fN =
  -20.3333
xF =
    4.6667    4.3333
fF =
  -20.3333
```

由计算结果可知，当 $x=4.6667$，$y=4.3333$ 时，函数 $f(x,y)=x(x-5-y)+y(y-4)$ 取得最小值为–20.3333。

该例题调用了 MATLAB 内置函数 fminsearch，来检验所编写的算法的精确性。fiminsearch 也是利用 Nelder-Mead 算法来求解多维变量的目标函数的最优化解的。

12.1.4 最速下降法

最速下降法是一种沿着 N 维目标函数的负梯度方向搜索最小值的方法。

1. 算法原理

函数的负梯度表示如下：

$$-g(x)=-\nabla f(x)=-\left[\frac{\partial f(x)}{\partial x_1}\quad\frac{\partial f(x)}{\partial x_2}\quad\cdots\quad\frac{\partial f(x)}{\partial x_N}\right]^T$$

搜索步长可调整，通常记为 α_k（第 k 次迭代中的步长）。该算法利用一维的线性搜索方法，如二次逼近法，沿着负梯度方向不断搜索函数的较小值，从而找出最优解。

2. 算法步骤

最速下降法的基本求解流程如下。

步骤 1：迭代次数初始化为 $k=0$，求出初始点 x_0 的函数值 $f_0=f(x_0)$。

步骤 2：迭代次数加 1，即 $k=k+1$，用一维线性搜索方法确定沿负梯度方向 $-g_{k-1}$ 的步长 α_{k-1}，其中 $\alpha_{k-1}=\mathrm{ArgMin}_\alpha f(x_{k-1}-\alpha g_{k-1}/\|g_{k-1}\|)$。

步骤 3：沿着负梯度的方向寻找下一个接近最小值的点，其中步长为 α_{k-1}，得下一点的坐标为：$x_k=x_{k-1}-\alpha_{k-1}g_{k-1}/\|g_{k-1}\|$。

步骤 4：如果 $x_k\approx x_{k-1}$，且 $f(x_k)\approx f(x_{k-1})$，那么就认为 x_k 为所求的最小值点，并结束循环；否则，跳到步骤 2。

3．算法的 MATLAB 实现

在 MATLAB 中编程实现的最速下降算法为：Opt_Steepest。

功能：最速下降算法求无约束最优化解。

调用格式：[*xo*, *fo*]=Opt_Steepest (*f*, grad，*x*0, TolX, TolFun, dist0, MaxIter)。

其中，*f* 为函数名；

grad 为梯度函数；

*x*0 为搜索初始值；

TolX 为最优值点间的误差阈值；

TolFun 为函数的误差阈值；

dist0 为初始步长；

MaxIter 为最大迭代次数；

xo 为最优化点值；

fo 为函数在点 xo 处的函数值。

最速下降法的 MATLAB 编程实现如下。

算法程序 Opt_Steepest.m

```
function [xo,fo] = Opt_Steepest(f,grad,x0,TolX,TolFun,dist0,MaxIter)
% 用最速下降法求最优化解
%f：函数名；
%grad：梯度函数
%x0：搜索初始值；
%TolX：最优值点间的误差阈值；
%TolFun：函数的误差阈值；
%dist0：初始步长；
%MaxIter：最大迭代次数；
%xo：最优化点值；
%fo：函数在点 xo 处的函数值。

%%%%%%判断输入的变量数，设定一些变量为默认值
if nargin < 7
    MaxIter = 100;                          %最大迭代次数默认为 100
end
if nargin < 6
    dist0 = 10;                             %初始步长默认为 10
end
if nargin < 5
    TolFun = 1e-8;                          %函数值误差为 1e-8
end
if nargin < 4
    TolX = 1e-6;                            %自变量距离误差
end
%%%%%第一步，求解的初值的函数值
x = x0;
fx0 = feval(f,x0);
fx = fx0;
dist = dist0;
```

```
kmax1 = 25;                                    %线性搜索法确定步长的最大搜索次数
warning = 0;
%%%%%迭代计算求最优解

for k = 1: MaxIter
    g = feval(grad,x);
    g = g/norm(g);                             %求点 x 处的梯度方向
    %%线性搜索方法确定步长
    dist = dist*2;                             %令步长为原步长的二倍
    fx1 = feval(f,x-dist*2*g);
    for k1 = 1:kmax1
        fx2 = fx1;
        fx1 = feval(f,x-dist*g);
        if fx0 > fx1+TolFun & fx1 < fx2 - TolFun %fx0 > fx1 < fx2,
            den = 4*fx1 - 2*fx0 - 2*fx2;num = den - fx0 + fx2;  %二次逼近法
            dist = dist*num/den;
            x = x - dist*g; fx = feval(f,x);      %确定下一点
            break;
        else
            dist = dist/2;
        end
    end
    if k1 >= kmax1
        warning = warning + 1;                 %无法确定最优步长
    else
        warning = 0;
    end
    if warning >= 2|(norm(x - x0) < TolX&abs(fx - fx0) < TolFun)
        break;
    end
    x0 = x;
    fx0 = fx;
end
xo = x; fo = fx;
if k == MaxIter
    fprintf('Just best in %d iterations',MaxIter);
end
```

4. 算法举例

【例 12-4】 最速下降法求解无约束最优化问题实例。采用最速下降法求如下函数的最小值问题：

$$f(x,y)=x(x-5-y)+y(y-4)$$

即用最速下降法求解【例 12-3】所示函数的最小值问题。

解：须先求出该函数的梯度函数。可知其梯度函数为：$g(x)=(2x-5-y,-x+2y-4)$。

在 MATLAB 中编写函数 ex1204.m 来进行求解，具体代码如下所示。

```
%ex1204.m 用最速下降法求最优化解
f1004 = inline('x(1)*(x(1)-5-x(2))+x(2)*(x(2)-4)','x');     %目标函数
grad=inline('[2*x(1)-5-x(2),-x(1)+2*x(2)-4]','x');      %目标函数的梯度函数
x0 = [1 4];
TolX = 1e-4;
TolFun = 1e-9;
```

```
MaxIter = 100;
dist0=1;
[xo,fo] = Opt_Steepest(f1004,grad,x0,TolX,TolFun,dist0,MaxIter)
```

程序运行结果为：

```
>> ex1204
xo =
    4.6666    4.3333
fo =
  -20.3333
```

由计算结果可知，当 $x=4.6667$，$y=4.3333$ 时，函数 $f(x,y)=x(x-5-y)+y(y-4)$ 取得最小值-20.3333。此结果与【例 12-3】的 Nelder-Mead 方法所得结果相同。

12.1.5 牛顿法

类似于最速下降法，牛顿法也是利用梯度来寻找目标函数的最小值解的。这些基于梯度的最优化方法的最终目标都是要寻找一个梯度为零（或接近零）的点。

1. 算法原理

在牛顿方法中，目标函数 $f(x)$ 的最优化解等同于寻找其梯度函数 $g(x)$ 的零点，因此在求得目标函数 $f(x)$ 的梯度 $g(x)$ 的表达式后，就可以利用牛顿法求解非线性方程 $g(x)=0$，从而得到 $f(x)$ 取最小值时的点 x。

该方法以及上节所讲述的最速下降法的理论背景都是依据函数的二阶泰勒级数展开的。不同的是，最速下降法忽略了展开式中二阶的余项，而牛顿法则直接求函数二阶泰勒展开式导数的零点。

设函数为 $f(x)$，则其二阶泰勒展开式为：

$$f(x)\approx f(x_k)+\nabla f(x)^T|_{x_k}[x-x_k]+\frac{1}{2}[x-x_k]^T\nabla^2 f(x)|_{x_k}[x-x_k]$$

$$f(x)\approx f(x_k)+g_k^T[x-x_k]+\frac{1}{2}[x-x_k]^T H_k[x-x_k]$$

根据最速下降法可将上式变形为：

$$f(x_{k+1})\approx f(x_k)+g_k^T[x_{k+1}-x_k]=f(x_k)-\alpha_k g_k^T g_k/\|g_k\|$$

而由牛顿法可知：

$$f(x)'=0\Rightarrow g_k+H_k[x-x_k]=0\Rightarrow x_{k+1}=x_k-H_k^{-1}g_k$$

注意

上述式中 $g_k=\nabla f(x)\big|_{x_k}$，$H_k=\nabla^2 f(x)\big|_{x_k}$。

图 12-2 给出了牛顿法与最速下降法求最小值问题时的过程比较图。从图 12-2 中可以直观看出牛顿法的效率比最速下降法要高。

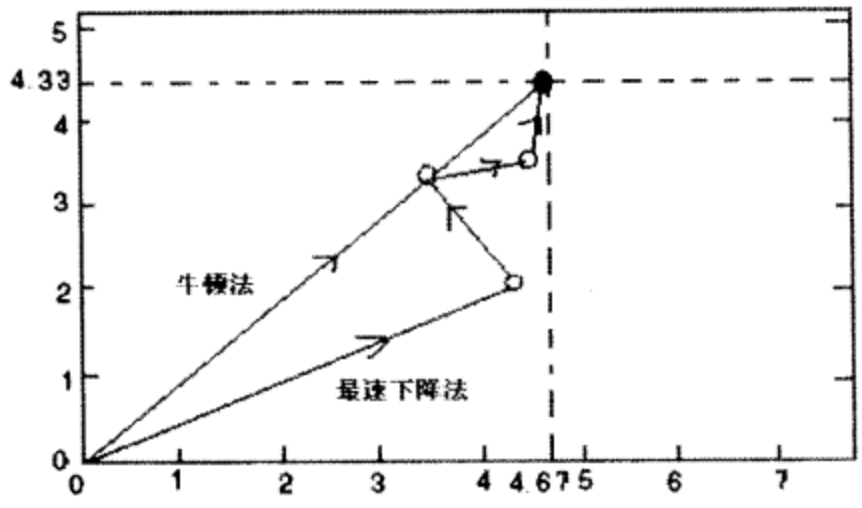

图 12-2　牛顿法与最速下降法过程比较

牛顿算法的核心是找到 $g(x)$ 的零点，所以任何求解向量非线性方程的解法都可以用来求函数的最优解。在使用牛顿方法编程求解时，首先要定义梯度函数，然后将定义的函数名做为非线性方程解答器的输入。但牛顿法的缺陷是，虽然牛顿法通常比最速下降法高效，但是牛顿法通常很难保证可以到达最小值点。牛顿法的致命性弱点是它得到的是梯度为零的点，而该点可能是局部最小点或最大值点。

2. 算法举例

【例 12-5】 牛顿法求解无约束最优化问题实例。采用牛顿法求如下函数的最小值问题：

$$f(x,y)=x(x-5-y)+y(y-4)$$

即采用牛顿法求解【例 12-3】所示函数的最小值问题。

解：原函数为 $f(x,y)=x(x-5-y)+y(y-4)$，可以求得其梯度函数为 $g(x)=(2x-5-y, -x+2y-4)$。

在 MATLAB 中编写函数 ex1205.m 来进行求解，具体代码如下所示。

```
%ex1205.m 用牛顿法求最优化解
f1005 = inline('x(1)*(x(1)-5-x(2))+x(2)*(x(2)-4)','x');%目标函数
grad=inline('[2*x(1)-5-x(2),-x(1)+2*x(2)-4]','x'); %目标函数的梯度函数
x0 = [-8;-8];
options=optimset('TolX',1e-4,'TolFun',1e-9,'MaxIter',100);
xo = fsolve(grad,x0,options)          %用 fsolve 求解非线性方程 g(x)=0
yo=f1005(xo)
```

运行结果为：

```
>> ex1205
Optimization terminated: first-order optimality is less than options.TolFun.
xo =
    4.6667
    4.3333
yo =
  -20.3333
```

由计算结果可知，当 $x=4.6667$，$y=4.3333$ 时，函数 $f(x,y)=x(x-5-y)+y(y-4)$ 取得最小值–20.3333。

12.1.6 模拟退火法

上述讨论的几个最优化方法在寻求最小值点时，除非选定的初值非常接近最小值点，一般效率都不是很高。并且，用上面的几种方法得到的点可能是某一个局部最小值点，而一般很难确定结果是否正确。

虽然可以通过设定不同的初值，计算出不同的局部最小值点，再比较它们的函数值，最终选出全局最小值点。但是初值的选取，以及计算量巨大，使得该过程几乎无法实现。

1. 算法原理

模拟退火法则可以完成这样的任务：通过在退火过程与求最小化的过程之间的模拟，并依概率决定局部最小点的取舍，从而可以跳过局部最小点，最终可得到全局的最小值点。

2. 算法步骤

模拟退火算法的基本求解流程如下。

步骤 1：选取初值点 x_0，估计最小值区间的下边界 l，上边界 u，选定最大迭代次数 $k_{\max}>0$，退火因子 $q>0$（退火速度较快时，因子较大），以及相应的函数值浮动误差容许值 ε_f。

步骤 2：令 $x=x_0$，$x^0=x$　$f^0=f\ x$ 。

步骤 3：k 循环，从 k=0 到循环到 $k=k_{\max}$

{生成一个在[-1,1]上服从均匀分布的 $N\times1$ 的随机向量 $\boldsymbol{y}$，$\boldsymbol{y}$ 与 x 大小相同。利用 μ^{-1} 定理计算 Δx：

$$\Delta x=(u-l)*g_\mu{}^{-1}(y),\quad g_\mu{}^{-1}(y)\frac{(1+\mu)^{|y|}-1}{\mu}sign(y),\quad \mu=10^{100}(k/k_{\max})^q。$$

然后在(l,u)区间内确定点　　$x_1=x+\Delta x$

如果 $\Delta f=f(x_1)-f(x)<0$ 或者在[0,1]上生成的均匀分布的随机数 z<p（步长为 $\Delta x)=\exp(-(k/k_{\max})^q\Delta f/|f(x)|/\varepsilon_f$ ），则令 x=x_1，$f(x)=f(x_1)$。

如果 $f(x)<f^0$ 则令 $x^0=x, f^0=f(x^0)$。

}

步骤 4：可知由以上步骤所得的 x^0 接近全局最小值点，从而可以设定 x^0 为初值，并运用任何一种求局部最优化的算法来寻找 $f(x)$ 的最小值点。

3. 算法的 MATLAB 实现

在 MATLAB 中编程实现的模拟退火算法为：Opt_Simu。

功能：模拟退火算法求无约束最优化解，可得全局最优解。

调用格式：[*xo*,*fo*]=Opt_Simu (*f*, *x*0, *l*, *u*, kmax, *q*, TolFun)。

其中，*f* 为函数名；

*x*0 为搜索初始值；

l 为搜索区间上限；

u 为搜索区间下限；

kmax 为最大迭代次数；

q 为退火因子；

TolFun 为函数的误差阈值；

xo 为最优化点值；

fo 为函数在点 xo 处的函数值。

模拟退火算法的 MATLAB 编程实现如下：

（1）μ^{-1} 定理的程序实现见 Mu_Inv.m

```
%模拟退火法中的 mu^（-1）定理
function x = Mu_Inv(y,mu)
x = (((1+mu).^abs(y)- 1)/mu).*sign(y);
```

（2）模拟退火算法主程序为 Opt_Simu.m

```
function [xo,fo] = Opt_Simu(f,x0,l,u,kmax,q,TolFun)
% 模拟退火算法求函数 f(x)的最小值点， 且 l <= x <= u
% f:待求函数
% x0:初值点
% l，u ：分别为搜索区间的上下限
% kmax:最大迭代次数
% q ：退火因子，
% TolFun ：函数容许误差
%%%%算法第一步根据输入变量数，将某些量设为默认值
if nargin < 7
    TolFun = 1e-8;
end
if nargin < 6
    q = 1;
end
if nargin < 5
    kmax = 100;
end
%%%%算法第二步，求解一些基本变量
N = length(x0);                    %自变量维数
x = x0;
fx = feval(f,x);                   %函数在初始点 x0 处的函数值
xo = x;
fo = fx;
%%%%%算法第三步，进行迭代计算，找出近似全局最小点
for k =0:kmax
    Ti = (k/kmax)^q;
    mu = 10^(Ti*100);              % 计算 mu
    dx = Mu_Inv(2*rand(size(x))-1,mu).*(u - l);% 步长 dx
    x1 = x + dx;                   %下一个估计点
%将 x1 限定在区间[l,u]上
    x1 = (x1 < l).*l +(l <= x1).*(x1 <= u).*x1 +(u < x1).*u;
    fx1 = feval(f,x1);
```

```
    df = fx1- fx;
%如果 fx1<fx 或者概率大于随机数 z
    if df < 0|rand < exp(-Ti*df/(abs(fx) + eps)/TolFun)
        x = x1;
        fx = fx1;
    end
    if fx < fo
        xo = x;
        fo = fx1;
    end
end
```

4. 算法举例

【例 12-6】 无约束最优化问题求解综合实例。采用多种方法计算下列函数的最小值，并比较结果。

$$f(x,y)=5\sin(xy)+x^2+y^2$$

解：在 MATLAB 中编写函数 ex1206.m 来进行求解，该程序中分别采用了函数 Opt_Nelder、fminsearch、fminunc，以及模拟退火函数 Opt_Simu 求解，具体代码如下所示。

```
%ex1206.m 用模拟退火法计算函数的最小值
clear all
f = inline('5*sin(x(1)*x(2))+x(1)^2+x(2)^2','x');
l = [-5 -5];                    %下限
u = [5 5];                      %上限
x0 = [1 1];
TolFun = 1e-9;
TolX=1e-5;
kmax = 800;
%%%%用 Nelder-Meat 法求
[xo_nd,fo] = Opt_Nelder(f,x0,TolX,TolFun,kmax)
 %%%%用 MATLAB 内置函数验证
[xos,fos] = fminsearch(f,x0)
[xou,fou] = fminunc(f,x0)
%%%%用模拟退火法求
 q =0.8;
[xo_sa,fo_sa] =Opt_Simu(f,x0,l,u,kmax,q,TolFun)
```

取不同的初值，可得不同运行结果如表 12-1 所示。

表 12-1 取不同初值时，不同算法所得的最优解

x0	Opt_Nelder	fminsearch	fminunc	Opt_Simu
[1,1]	xo=[1.0767,−1.0767] fo=-2.2640	xo=[1.0767,−1.0767] fo=-2.2640	xo=[0,0] fo=0	xo=[1.0767,−1.0752] fo=−2.2640
[5,5]	xo=[−1.0767,1.0767] fo=-2.2640	xo=[4.8115,4.8115] fo=41.7183	xo=[4.1070,4.1070] fo=29.1519	xo=[−1.0791,1.0744] fo=−2.2640
[−4,0]	xo=[−2.0738,−2.0739] fo= 4.0192	xo=[−1.0767,1.0767] fo=−2.2640	xo=[−1.0767,1.0767] fo=−2.2640	xo=[−1.0765,1.0767] fo=−2.2640

由表 12-1 可知，当取不同初值时，前三种方法有可能得到局部最小值点或最大值点，

而模拟退火算法结果相对较稳定，其所得的点为全局最小值点或非常靠近全局最小值点。

12.1.7 遗传算法

遗传算法（GA）是一种直接的随机搜索方法，它是在适者生存的自然进化过程中逐渐建立起来的。

1. 算法原理

类似于模拟退火算法，遗传算法也适用于寻找具有多个极值的目标函数的全局最小解。

2. 算法步骤

遗传算法的基本求解流程如图 12-3 所示

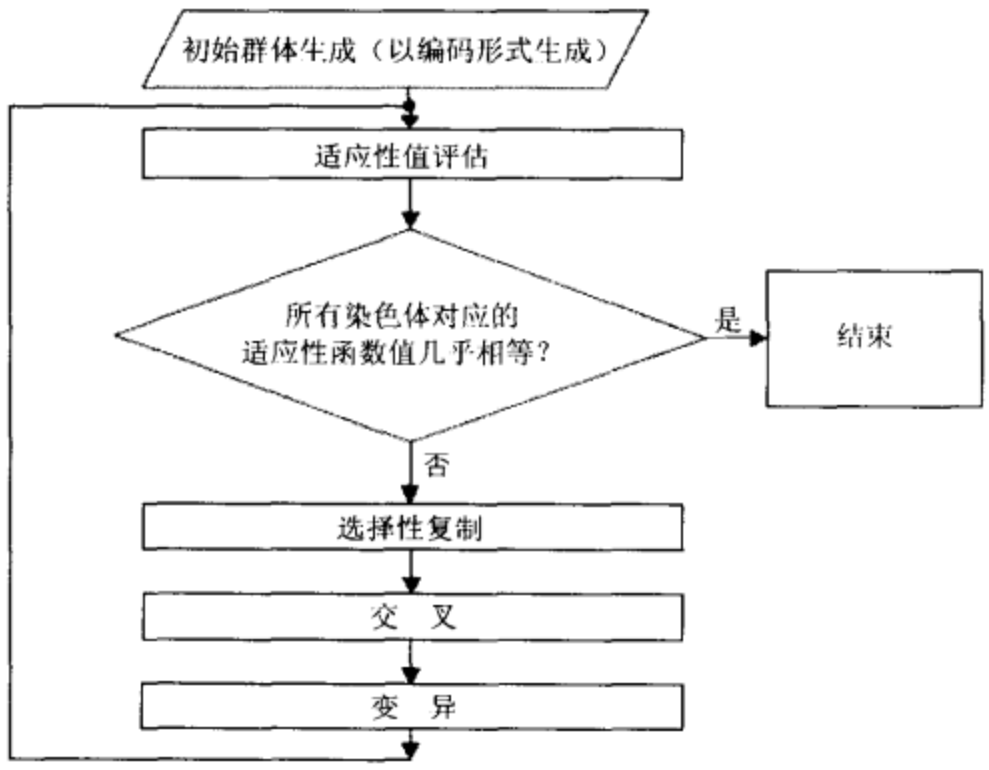

图 12-3 遗传算法流程图

流程图说明：

- 编码：GA 在进行搜索之前，先将解空间的解数据表示成遗传空间的遗传型串结构数据，这些串结构数据的不同组合便构成了不同的点。
- 初始群体的生成：随机产生 N 个初始串结构数据，每个串结构数据称为一个个体，N 个个体构成了一个群体。GA 以这 N 个串结构数据作为初始点开始迭代。
- 适应性值评估检测：适应性函数表明个体或解的优劣性。对于不同的问题，适应性函数的定义方式也不同。
- 选择性复制：选择的目的是为了从当前群体中选出优良的个体，使它们有机会作为父代为下一代繁殖子孙。

遗传算法通过选择过程体现这一思想，进行选择的原则是适应性强的个体为下一代贡献一个或多个后代的概率大。选择实现了达尔文的适者生存原则。

- 交叉：交叉操作是遗传算法中最主要的遗传操作。通过交叉操作可以得到新一代个体，新个体组合了其父辈个体的特性。交叉体现了信息交换的思想。
- 变异：变异首先在群体中随机选择一个个体，对于选中的个体以一定的概率随机地改变串结构数据中某个串的值。

同生物界一样，GA 中变异发生的概率很低，通常取值在 0.001~0.01 之间。变异为新个体的产中提供了机会。

3. 算法的 MATLAB 实现

在 MATLAB 中编程实现的遗传算法函数为：genetic。

功能：遗传算法求无约束最优化解。

调用格式：[*xo*,*fo*] =genetic(*f*, *x*0, *l*, *u*, Np, Nb, Pc, Pm, eta, kmax)。

其中，*f* 为函数名；

*x*0 为搜索初始值；

l, u 为搜索区间的上、下限；

Np 为群体大小；

Nb 为每一个变量的遗传值（二进制数）；

Pc 为交叉概率；

Pm 为变异概率；

eta 为学习率；

kmax 为最大迭代次数；

xo 为最优化点值；

fo 为函数在点 *xo* 处的函数值。

遗传算法的 MATLAB 编程包括如下几个程序文件：genetic.m（主程序文件），gen_encode.m，gen_decode.m，crossover.m，mutation.m，shuffle.m。

（1）genetic.m

```
function [xo,fo] = genetic(f,x0,l,u,Np,Nb,Pc,Pm,eta,kmax)
% 遗传算法求 f(x) 最小值 s.t. l <= x <= u
%f: 为待求函数,
%x0:初值,
%l, u:上下限,
%Np:群体大小,
%Nb:每一个变量的遗传值（二进制数）
%Pc:交叉概率,
%Pm:变异概率,
%eta:学习率,
%kmax:最大迭代次数
N = length(x0);
%%%%%确定各变量默认值
if nargin < 10,kmax = 100; end      %最大迭代次数默认值为 100
if nargin < 9|eta > 1|eta <= 0,eta = 1; end %学习率 eta,(0 < eta < 1)
if nargin < 8,Pm = 0.01;end         %变异概率默认值为 0.01
if nargin < 7,Pc = 0.5;end          %交叉概率默认值为 0.5
```

```
if nargin < 6,Nb = 8*ones(1,N);end %每一变量的遗传值（二进制数）
if nargin < 5,Np = 10;end          %群体大小（染色体数）
%%%%%生成初始群体
NNb = sum(Nb);
xo = x0(:)'; l = l(:)'; u = u(:)';
fo = feval(f,xo);
X(1,:) = xo;
for n = 2:Np,X(n,:) = l + rand(size(x0)).*(u - l);end  %初始群体随机数组
P = gen_encode(X,Nb,l,u);          %编码为 2 进制字串
for k = 1:kmax
    X = gen_decode(P,Nb,l,u);      %解码为 10 进制数
    for n = 1:Np,fX(n) = feval(f,X(n,:));end
    [fxb,nb] = min(fX);            %选择最适合的，函数值最小的
    if fxb < fo,fo = fxb;xo = X(nb,:);end
    fX1 = max(fxb) - fX;           %将函数值转化为非负的适合度值
    fXm = fX1(nb);
    if fXm < eps,return;end        %如果所有的染色体值相同，终止程序
    %%%%%复制下一代
    for n = 1:Np
        X(n,:)=X(n,:)+eta*(fXm-fX1(n))/fXm*(X(nb,:)-X(n,:));     %复制准则
    end
    P = gen_encode(X,Nb,l,u);      %对下一代染色体编码
    %%%%%%随机配对/交叉得新的染色体数组
    is = shuffle([1:Np]);
    for n = 1:2:Np - 1
        if rand < Pc
            P(is(n:n + 1),:) = crossover(P(is(n:n + 1),:),Nb);
        end
    end
    %%%%%%变异
    P = mutation(P,Nb,Pm);
end
```

（2）gen_encode.m

```
function P = gen_encode(X,Nb,l,u)
%将群体 X 的状态编码为二进制数组 P
Np=size(X,1);                          %群体大小
N = length(Nb);                        %变量（状态）维数
for n = 1:Np
    b2 = 0;
    for m = 1:N
        b1 = b2+1;
        b2 = b2 + Nb(m);
        Xnm =(2^Nb(m)- 1)*(X(n,m) - l(m))/(u(m) - l(m));   %编码方程
        P(n,b1:b2) = dec2bin(Xnm,Nb(m));                   %10 进制转换为 2
                                                           %进制
    end
end
```

（3）gen_decode.m

```
function X = gen_decode(P,Nb,l,u)
% 将二进制数组 P 解码为群体 X 的状态矩阵
```

```
Np = size(P,1);                                    %群体大小
N = length(Nb);                                    %变量维数
for n = 1:Np
    b2 = 0;
    for m = 1:N
        b1 = b2 + 1;
        b2 = b1 + Nb(m) - 1;
        X(n,m) = bin2dec(P(n,b1:b2))*(u(m)-l(m))/(2^Nb(m)-1)+l(m); %解码方程
    end
end
```

(4) crossover.m

```
function chrms2 = crossover(chrms2,Nb)
%两个染色体间的交叉
Nbb = length(Nb);
b2 = 0;
for m = 1:Nbb
    b1 = b2 + 1;
    bi = b1 + mod(floor(rand*Nb(m)),Nb(m));
    b2 = b2 + Nb(m);
    tmp = chrms2(1,bi:b2);
    chrms2(1,bi:b2) = chrms2(2,bi:b2);
    chrms2(2,bi:b2) = tmp;
end
```

(5) mutation.m

```
function P = mutation(P,Nb,Pm)
%变异
Nbb = length(Nb);
for n = 1:size(P,1)
    b2 = 0;
    for m = 1:Nbb
        if rand < Pm
            b1 = b2 + 1;
            bi = b1 + mod(floor(rand*Nb(m)),Nb(m));
            b2 = b2 + Nb(m);
            P(n,bi) = ~P(n,bi);
        end
    end
end
```

(6) shuffle.m

```
function is = shuffle(is)
%打乱染色体次序
N = length(is);
for n = N:-1:2
    in = ceil(rand*(n - 1));
    tmp = is(in);is(in) = is(n); is(n) = tmp; %将第 n 个元素与第 in 个元素交换
end
```

4. 算法举例

【例 12-7】 遗传算法求解无约束最优化问题实例。采用遗传算法求函数的最小值：

$$f(x,y)=x^4-16x^2-5xy+y^4-16y^2-5y$$

解：在 MATLAB 中编写函数 ex1207.m 来进行求解，具体代码如下所示。

```
%%%ex1207.m 遗传算法计算最优解
f = inline('x(1)^4-16*x(1)^2-5*x(1)*x(2)+x(2)^4-16*x(2)^2-5*x(2)','x');
l = [-5 -5];                          %下限
u = [5 5];                            %上限
x0 = [0 0];
Np = 30;                              %群体大小
Nb = [12 12];                         %代表每个变量的二进制位数
Pc = 0.5;                             %交叉概率
Pm = 0.01;                            %变异概率
eta = 0.8;                            %学习率
kmax = 200;                           %最大迭代次数
[xos,fos]=fminsearch(f,x0)
[xo_gen,fo_gen] = genetic(f,x0,l,u,Np,Nb,Pc,Pm,eta,kmax)
```

运行结果为：

```
xos =
    3.0453    3.1031
fos =
-186.4878
xo_gen =
    2.9805    3.3150
fo_gen =
-184.2604
```

由运算结果可知，用 MATLAB 内置函数 fminsearch 计算时，当 $x=3.0453$，$y=3.1031$ 时函数取最小值–186.4878，用遗传算法求解时，当 x=2.9805，y=3.3150 时函数取最小值 –184.2604。

12.2 约束最优化

这一节仅介绍约束最优化的基本算法思想，不提供具体的算法程序。对于本节所用到的相应的 MATLAB 的内置函数将在 12.3 节介绍。

12.2.1 拉格朗日乘子法

拉格朗日乘子法常用于求解一类常见的约束最优化问题。

1．算法原理

拉格朗日乘子法的约束条件为一系列等式，考虑下面的一个最优化问题，其包含 M 个等式约束：

$$\min\ f(x)$$

约束条件：

$$h(x)=\begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_M(x) \end{bmatrix}=0$$

依据拉格朗日乘子法，上面的约束最优化问题可以转化为下面的无约束最优化问题：

$$\min\ l(x,\lambda)=f(x)+\lambda^T h(x)=f(x)+\sum_{m=1}^{M}\lambda_m h_m(x)$$

对于该无约束最优化问题，如果解存在的话，可以通过函数对变量 x,λ 的偏导等于零来求得最优解。

注意

拉格朗日乘子法也可以求解约束条件包含不等式的最优化问题。求解时可以通过引入一个松弛变量使不等式约束变为等式约束。

例如，若原约束条件为 $g_j(x)\leqslant 0$，引入非负变量 y_j^2 后，约束条件变为 $g_j(x)+y_j^2=0$。此时满足等式约束条件，可用以上方法求解。

2. 算法举例

【例 12-8】 拉格朗日乘子法求解约束最优化问题实例。采用拉格朗日乘子法求如下最优化问题：

$$\min\ f(x_1,x_2)=x_1+x_2，\ 其中\ x_1^2+x_2^2=2。$$

解：将其转化为无约束最优化问题为：

$$\min\ l(x,\lambda)=x_1+x_2+\lambda(x_1^2+x_2^2)。$$

在 MATLAB 中编写函数 ex1208.m 来进行求解，具体代码如下所示。

```
%%%ex1208.m 拉格朗日乘子法求最优化解
x=zeros(1,2);
%用 syms 表示出转化后的无约束函数
syms x y lama
f=x+y+lama*(x^2+y^2-2);
%分别求函数关于 x、y、lama 的偏导
dx=diff(f,x);
dy=diff(f,y);
dlama=diff(f,lama);
%令偏导为零，求解 x、y
xx=solve(dx,x);                          %将 x 表示为 lama 函数
yy=solve(dy,y);                          %将 y 表示为 lama 函数
ff=subs(dlama,{x,y},{xx,yy});            %代入 dlama 得关于 lama 的一元函数
lamao=solve(ff);                         %求解得 lamao
xo=subs(xx,lama,lamao)                   %求得取极值处的 xo
yo=subs(yy,lama,lamao)                   %取极值处的 yo
fo=subs(f,{x,y,lama},{xo,yo,lamao})      %极值点函数值
```

程序运行结果为：

```
xo =
  1
 -1
yo =
  1
 -1
fo =
  2
 -2
```

可知极值点有两个，分别为（1,1）和（–1,–1），比较它们的函数值，最终可得函数在（–1,–1）处取极小值为–2。

12.2.2 惩罚函数法

惩罚函数法可以求解等式或不等式约束的最优化问题，比拉格朗日法适用性广。拉格朗日法的限制性条件为等式，对于限制条件为不等式的最优化问题，其虽可以近似求解，但不常用。

1. 算法原理

惩罚函数法所求解的最优化问题的限制条件可以是没有严格零约束的模糊的或宽松的限制。

考虑如下的最优化问题：

$$\min\ f(x)$$

约束条件：

$$\boldsymbol{H}(x)=\begin{bmatrix}\boldsymbol{H}_1(x)\\ \vdots \\ \boldsymbol{H}_M(x)\end{bmatrix}=0,\quad \boldsymbol{G}(x)=\begin{bmatrix}\boldsymbol{G}_1(x)\\ \vdots \\ \boldsymbol{G}_L(x)\end{bmatrix}\leqslant 0$$

惩罚函数法包含以下两个步骤。

步骤 1：构造一个新的目标函数，约束最优化问题转化为无约束最优化问题：

$$\min\ l(x)=f(x)+\sum_{m=1}^{M}\omega_m \boldsymbol{H}_m^2(x)+\sum_{m=1}^{L}v_m\psi(\boldsymbol{G}_m(x))$$

步骤 2：利用上一节介绍的求解无约束最优化问题的方法，求新的目标函数的最小值点。此处使用的无约束最优化方法不能采用基于梯度的最优化方法，如不可采用牛顿法、最速下降法等。Nelder-Mead 方法可用于此步。

2. 算法举例

【例 12-9】 惩罚函数法求解约束最优化问题实例。采用惩罚函数求解下面的最优化问题：

$$\min\ f(x)=[(x_1+1)^2+4(x_2-1.5)^2][(x_1-1.2)^2+0.4(x_2-0.5)^2]。$$

约束条件：

$$G(x)=\begin{bmatrix} -x_1 \\ -x_2 \\ 2x_1-x_1x_2+5x_2-6 \\ x_1-x_2+0.5 \\ x_1^2-4x_2^2+x_2 \end{bmatrix} \leqslant \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

解：将该约束最优化问题转化为无约束问题，构造新的目标函数：

$$\min\ l(x)=[(x_1+1)^2+4(x_2-1.5)^2][(x_1-1.2)^2+0.4(x_2-0.5)^2]+\sum_{m=1}^{5}v_m\psi_m(G_m(x))$$

$$\text{其中，}\ v_m=1,\ \psi_m(G_m(x))=\begin{cases} 0 & if\ G_m(x)\leqslant 0 \\ \exp(e_mG_m(x)) & if\ G_m(x)>0 \end{cases},\ e_m=1$$

首先编写函数 f1209.m，用来表示新的目标函数，具体代码如下所示。

```
function [fc,f,c] = f1209(x)
%%%将约束最优问题转化为无约束最优问题
f=((x(1)+ 1)^2 + 4*(x(2)- 1.5)^2)*((x(1)- 1.2)^2 + 0.4*(x(2)-0.5)^2);
%约束向量
c=[-x(1);-x(2);2*x(1)-x(1)*x(2)+5*x(2)-6;x(1)-x(2)+0.5;x(1)^2-4*x(2)^2+x
  (2)];
v=[1 1 1 1 1]; e = [1 1 1 1 1]';        %权系数向量
fc = f +v*((c > 0).*exp(e.*c));         %新的目标函数
```

选用不同的无约束最优化方法求解，并比较结果。非基于梯度的方法有 Nelder 方法以及 MATLAB 内置函数 fminsearch。基于梯度的方法有最速下降法，以及 MATLAB 内置函数 fminunc，编写文件 ex1209.m，代码如下所示。

```
%ex1209.m 惩罚函数法求约束最优化问题
clear
f ='f1009';
x0=[3 0];
TolX = 1e-4;
TolFun = 1e-9;
MaxIter=100;
alpha0 = 1;
%%%%选用不是基于梯度的无约束最优化方法求解，得正确结果
[xo_Nelder,fo_Nelder] = Opt_Nelder(f,x0,TolX,TolFun,MaxIter);%Nelder 方法
[fc_Nelder,fo_Nelder,co_Nelder] = f1009(xo_Nelder);%Nelder 方法结果
[xo_s,fo_s] = fminsearch(f,x0) ;%MATLAB 内置函数 fminsearch()
[fc_s,fo_s,co_s] = f1009(xo_s) ;%相应的结果
%%%基于梯度的方法最速下降法等，得到错误结果
grad=inline('[2*(x(1)+1)*((x(1)-1.2)^2+0.4*(x(2)-0.5)^2)+((x(1)+1)^2+4*
     (x(2)-1.5)^2)*2*(x(1)-1.2),8*(x(2)-1.5)*((x(1)-1.2)^2+0.4*(x(2)-0.5)
     ^2)+((x(1)+1)^2+4*(x(2)-1.5)^2)*0.8*(x(2)-0.5)]','x'); %目标函数的梯度函数
xo_steep = Opt_Steepest(f,grad,x0,TolX,TolFun,alpha0);%最速下降法
[fc_steep,fo_steep,co_steep] = f1009(xo_steep);          %相应结果
[xo_u,fo_u] = fminunc(f,x0);              % MATLAB 内置函数 fminunc()
[fc_u,fo_u,co_u] = f1009(xo_u);                    %相应结果
```

所得结果如表所示。

基于梯度算法所得结果	非基于梯度算法所得结果
最速下降法：不满足约束条件，结果错误 xo_steep = 0.2441 1.1824 fc_steep =3.2646 fo_steep =2.1463 %不是最小值 co_steep = -0.2441 -1.1824 0.1118 %>0，不满足约束条件 -0.4383 -4.3506 fminunc：不满足约束条件，结果错误 Warning: Gradient must be provided for .... fc_u = 8.2197e+003 fo_u =83.5000 %不是最小值 co_u = -3.0000 0 0 3.5000 % >0,不满足约束条件 9.0000 %不满足约束条件	Nelder 方法：满足约束条件，结果正确 xo_Nelder =0.5923 1.0924 fc_Nelder =1.6310 fo_Nelder =1.6310%最小值 co_Nelder = %四项约束值均小于 0，满足条件 -0.5923 -1.0924 -0.0003 -0.0002 -3.3305 fminsearch：满足约束条件，结果正确 xo_s = 0.5916 1.0916 fc_s =1.6326 fo_s =1.6326 %最小值 co_s = %四项约束值均小于 0，满足条件 -0.5916 -1.0916 -0.0045 -0.0000 -3.3248

由上表中所列结果可知，惩罚函数算法的第二步解无约束的最优化问题时，须采用非基于梯度的最优化算法求解，否则会得到错误结果。

12.3 MATLAB 的内置最优化函数

上两节中给出了处理最优化问题的一些算法原理及其对应的 MATLAB 程序。在这一节中将主要讲述 MATLAB 中内置的最优化函数的具体使用以及它们的用法比较。

12.3.1 最优化工具箱

MATLAB 中有专用的最优化工具箱（Optimization Toolbox），其包含处理各种最优化问题的函数。

Optimization Toolbox 中的最优化函数主要分为三类：求最小值的函数、等式求解函数和最小二乘函数。

表 12-2、12-3、12-4 分别列出了这三类函数以及其用法。

表 12-2　MATLAB 中求最小值的函数

函 数 名	最优化问题描述
bintprog	求解二进制整数规划问题，约束条件包含等式和不等式
fgolattain	解决多目标实现问题
fminbnd	在一个确定的区间上寻找单变量函数的最小值点
fmincon	确定有约束、非线性、多变量函数的最小值点
fminimax	求解最小最大值问题
fminsearch	确定无约束、多变量函数的最小值点
fminunc	确定无约束、多变量函数的最小值点
fseminf	确定半无限、有约束、多变量、非线性函数的最小值点
linprog	求解线性规划问题
quadprog	求解二次规划问题

表 12-3　MATLAB 中求解等式最优解的函数

函 数 名	用　　法
\	用于矩阵左除，可解线性方程
fsolve	求解非线性方程
fzero	单变量连续方程的零解

表 12-4　MATLAB 求最小二乘法拟合系数最优解函数

函 数 名	用　　法
lsqcurvefit	在最小二乘意义下求解非线性曲线拟合问题
lsqlin	求解有约束线性最小二乘问题
lsqnonlin	求解非线性最小二乘问题
lsqnonneg	求解非负最小二乘问题

12.3.2　无约束最优化函数

1．fminbnd 求定区间上单变量函数的最小值点

fminbnd 求解如下形式的最优化问题：

$$\min f(x)$$

约束条件：

$$x_1 < x < x_2$$

其调用形式如下：

x = fminbnd(fun,x1,x2);

x = fminbnd(fun,x1,x2,options)。

式中，fun 为一元函数，x1，x2 分别为区间的上下界。此函数用法简单，此处不举例进行说明。

2. fminsearch 与 fminunc 求多变量函数最小值点

MATLAB 工具箱中用于求解多变量无约束函数最小值问题的函数有 fminsearch 和 fminunc。其中 fminunc 是基于梯度的最优化算法，而 fminsearch 是根据 Nelder 算法（与自编函数 Opt_Nelder 在计算上有区别）编写的，该算法不涉及偏导的计算。

两函数的调用形式类似，下面以 fminsearch 为例对其说明。

$\boldsymbol{x}$ = fminsearch(fun, $\boldsymbol{x}$0)

$\boldsymbol{x}$ = fminsearch(fun, $\boldsymbol{x}$0, options)

[$\boldsymbol{x}$,fval] = fminsearch(...)

[$\boldsymbol{x}$,fval,exitflag] = fminsearch(...)

[$\boldsymbol{x}$,fval,exitflag,output] = fminsearch(...)

式中，x、x0 为向量形式。其中，x 为对应的最优解，x0 为初始值向量。

fun 为函数表达式，option 为选项，用来选择允许计算误差、迭代次数和算法等，详见 optimset 函数。

exitflag 为退出标志；exitflag>0 函数收敛于解 x；exitflag=0 迭代次数超过；exitflag<0 函数值不收敛。

output 输出解题信息，如有关的迭代次数、函数值计算次数、在 x 处的梯度范数和所用的算法等。

注意

optimset 函数是用来设置最优化选项（options）的。该函数的调用格式为：

options=optimset('param1',value1, 'param2',value2,...)

式中，param1，param2…为优化参数名，value1，value2…为优化参数对应的值，由 optimset 列表规定。表 12-5 给出了常用优化参数说明。

表 12-5 常用优化参数说明

参 数 名	取 值	含 义
TolX	正值，如 1e-5	变量误差容限
TolFun	正值，如 1e-9	函数值误差容限
MaxIter	正整数，如 100	最大迭代次数

【例 12-10】 无约束最优化函数应用实例之一。采用 MATLAB 中的无约束最优化函数求如下最小值问题：

$$\min\ f(x_1,x_2)=2x_1^3+4x_1x_2^2-8x_1x_2+x_2^3。$$

解：在 MATLAB 命令窗口中输入：

```
>> [xos,yos]=fminsearch('2*x(1)^3+4*x(1)*x(2)^2-8*x(1)*x(2)+x(2)^3',[0 0])
```

输出结果为：

```
xos =
```

```
    0.7883    0.7397
yos =
   -1.5551
```

用 fminunc 求解，取初值为[0,0]，在 MATLAB 命令窗口中继续输入：

```
>> [xom,yom]=fminunc('2*x(1)^3+4*x(1)*x(2)^2-8*x(1)*x(2)+x(2)^3',[0 0])
Warning: Gradient must be provided for trust-region method;
  using line-search method instead.
> In fminunc at 243
Optimization terminated at the initial point: the relative
 magnitude of the gradient at x0 less than options.TolFun.
```

输出结果为：

```
xom =
     0     0
yom =
     0
```

用 fminunc 求解，取初值为[0,1]，在 MATLAB 命令窗口中继续输入：

```
>> [xom,yom]=fminunc('2*x(1)^3+4*x(1)*x(2)^2-8*x(1)*x(2)+x(2)^3',[0 1])
Warning: Gradient must be provided for trust-region method;
  using line-search method instead.
> In fminunc at 243
Optimization terminated: relative infinity-norm of gradient less than
options.TolFun.
```

输出结果为：

```
xom =
    0.7884    0.7397
yom =
   -1.5551
```

可知 $f(x_1,x_2)=2x_1^3+4x_1x_2^2-8x_1x_2+x_2^3$ 在 $x_1=0.7884$，$x_2=0.7397$ 处取最小值为 −1.5551。

易知，当选择的初值不同时，会影响所得结果。这是由于 fminsearch 和 fminunc 在求得局部最小值后就会终止程序，而无法继续搜索以寻找全局最小值点。采用模拟退火法或遗传算法可避免此不足。

3. 最小二乘问题（平方和最小）

最小二乘问题也是求最小值问题的一种。其对应的最小值问题是：

$$\min \sum_{n=1}^{N} f_n^2(x)$$

MATLAB 内置函数 lsqnonlin 可以给出上述问题的非线性最小二乘解，其调用格式为：

x = lsqnonlin(fun, $x0$)

x = lsqnonlin(fun, $x0$, lb, ub)

x = lsqnonlin(fun, $x0$, lb, ub, options)

式中，fun 为一向量形式的函数或矩阵形式的函数，$x0$ 为初值。也可以限定解的区间，

即可用 *lb*，*ub* 设定解区间的上下限。options 用于设定误差容限、迭代次数、算法等。

【例 12-11】 无约束最优化函数应用实例之二。采用 lsqnonlin 函数求如下形式的最小值：

$$\min \sum_{k=1}^{10}(2+2k-e^{kx_1}-e^{kx_2})^2 \text{。}$$

解：首先编写函数文件 f1211.m：

```
function F=f1211(x)
%%例 12-11 函数
k=1:10;
F=2+2*k-exp(k*x(1))-exp(k*x(2));
```

求最小二乘解，编写.m 文件 ex1211.m：

```
%ex1211.m 用 lsqnonlin 求解最小二乘问题
clear all
x0 = [0.3 0.4];                          % 初值点
[x,resnorm] = lsqnonlin(@f1011,x0)       % 调用最优化函数求得 x 与平方和残差
```

运行结果为：

```
Optimization terminated: norm of the current step is less
 than OPTIONS.TolX.
x =
    0.2578    0.2578                     %最小二乘解
resnorm =
  124.3622                               %函数值
```

故 $f(x_1,x_2)=\sum_{k=1}^{10}(2+2k-e^{kx_1}-e^{kx_2})^2$ 在 $x_1=0.2578$，$x_2=0.2578$ 处取最小值为 124.3622。

12.3.3 约束最优化函数

通常，约束最优化问题更复杂也更难处理，所以此处不讨论函数的具体算法细节。首先引入 MATLAB 中求解约束最优化问题的功能强大的函数 fmincon。

此函数用于求解满足如下约束的最优化问题：

$$\min f(x)$$

约束条件：

$$Ax \leqslant b,\ A_{eq}x=b_{eq},\ c(x)\leqslant 0,\ c_{eq}(x)=0 \text{ 且 } l\leqslant x\leqslant u \text{。}$$

MATLAB 内置函数 fmincon 的具体用法如下所示：

[xo,fo] = fmincon('ftn', *x*0, *A*, *b*)

[xo,fo] = fmincon('ftn', *x*0, *A*, *b*, Aeq, beq)

[xo,fo] = fmincon('ftn', *x*0, *A*, b, Aeq, beq, l, u)

[xo,fo] = fmincon('ftn', *x*0, *A*, *b*, Aeq, beq, l, u, 'nlcon', options, *p*1, *p*2)

- 输入变量说明：

'ftn'表示目标函数 *f*(*x*)，通常用.m 文件定义，也可以用 inline 函数定义，此时去掉变量

上的引号。

$x0$ 为解的初始估计值。

A，b 为线性不等式约束 $Ax \leqslant b$；如果不需要此约束时，此变量用方括号“[]”代替。

Aeq，beq：线性等式约束 Aeqx = beq；如果不需要时用方括号“[]”代替。

l，u：下限/上限量，使得 $l \leqslant x \leqslant u$；无限制时为方括号[]。如果 $x(i)$无上限或无下限，可设置 $l(i)$ =inf 或 $u(i)$ =inf。

'nlcon'：用.m 文件定义的非线性约束函数。该函数返回两个输出值，一个为不等式约束 $c(x) \leqslant 0$，另一个为等式约束 $ceq(x)$ =0；如果不适用此类约束，则输入方括号“[]”。

options：用来设定显示参数、xo 或 fo 误差等。不考虑此参数时，输入方括号“[]”对于其详细说明，可以在 MATLAB 命令窗口中输入“help optimset”进行帮助查询。

p1,p2：需要传给目标函数 $f(x)$和非线性约束函数 $c(x)$，$ceq(x)$的和最优化问题有关的变量

- 输出变量说明：

xo 为满足约束条件的在指定区域内的最小值点。

fo 为函数在最小值处的值。

下面以实例对函数用法进行具体说明。

【例 12-12】 约束最优化函数应用实例之一。采用约束最优化函数求解如下形式的最优化问题：

$$\min f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)。$$

约束条件

$$1.5 + x_1x_2 - x_1 - x_2 \leqslant 0,\ x_1 + x_2 = 0,\ -x_1x_2 \leqslant 10$$

解： 可知此最优化问题的约束分别为线性等式约束和非线性不等式约束。故须先编写函数 fcon1212.m，用来输出非线性约束。

```
function [c,ceq]=fcon1212(x)
%例 12-12 的非线性约束
c=[1.5+x(1)*x(2)-x(1)-x(2);-x(1)*x(2)-10];      %非线性不等式约束 c<=0
ceq=x(1)+x(2);                                  %非线性等式约束，ceq=0
```

编写 ex1212.m 求解此最优化问题：

```
%ex1212.m 用函数 fmincon 求非线性约束最优化问题
clear all
f=inline('exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1)','x');
x0=[-1 1];
[x,fval]=fmincon(f,x0,[],[],[],[],[],[],'fcon1212')
```

输出结果为：

```
Warning: Large-scale (trust region) method does not currently solve this type
of problem,
    switching to medium-scale (line search).
    > In fmincon at 274
    In ex1212 at 5
```

```
    Optimization terminated: first-order optimality measure less than
options.TolFun
    and maximum constraint violation is less than options.TolCon.
    Active inequalities (to within options.TolCon = 1e-006):
    lower     upper    ineqlin   ineqnonlin
    1
    x =
    -1.2247    1.2247
    fval =
    1.8951
```

故 $f(x)=e^{x_1}(4x_1^2+2x_2^2+4x_1x_2+2x_2+1)$ 在约束条件为 $1.5+x_1x_2-x_1-x_2\leqslant 0$，$x_1+x_2=0,-x_1x_2\leqslant 10$ 的情况下，在 $x_1=-1.2247$，$x_2=1.2247$ 处取最小值 1.8951。

【例 12-13】 约束最优化函数应用实例之二。采用约束最优化函数求如下函数的最优化问题：

$$\min f(x)=-x_1x_2x_3。$$

约束条件：

$$-x_1-2x_2-2x_3\leqslant 0, x_1+2x_2+2x_3\leqslant 72$$

解：MATLAB 实现如下：

```
%ex1213.m 用 fmincon 解线性约束的最优化
clear all
f=inline('-x(1)*x(2)*x(3)','x');
A=[-1,-2,-2;1,2,2]; %A*x<=b
b=[0;72];
x0=[10;10;10];
[x,fval]=fmincon(f,x0,A,b)
```

运行结果：

```
    Warning: Large-scale (trust region) method does not currently solve this type
of problem,
     switching to medium-scale (line search).
    > In fmincon at 274
      In ex1213 at 7
    Optimization terminated: magnitude of directional derivative in search
     direction less than 2*options.TolFun and maximum constraint violation
      is less than options.TolCon.
    Active inequalities (to within options.TolCon = 1e-006):
      lower     upper    ineqlin   ineqnonlin
                          2
    x =
       24.0000
       12.0000
       12.0000
    fval =
    -3.4560e+003
```

故 $f(x)=-x_1x_2x_3$ 在满足约束条件 $-x_1-2x_2-2x_3\leqslant 0$，$x_1+2x_2+2x_3\leqslant 72$ 的情况下，在 $x_1=24$，$x_2=12$，$x_3=12$ 处取最小值-3.4560e+003。

另外还要介绍两个用于求解约束最优化问题的函数，它们分别是 fminimax 和 lsqlin。

1．fminimax——最小最大值函数

此函数主要是用来寻找一个点使得函数向量中的所有函数值的最大值最小。即解如下最优化问题：

$$\min_x\left\{\max_n\{f_n(x)\}\right\}$$

约束条件：

$$\boldsymbol{Ax}\leqslant\boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x}=\boldsymbol{b}_{eq},\ \boldsymbol{c}(x)\leqslant 0,\ \boldsymbol{c}_{eq}(x)=0,\ \text{且}\boldsymbol{l}\leqslant\boldsymbol{x}\leqslant\boldsymbol{u}\text{。}$$

其具体调用形式与 fmincon 相同。

2．Lsqlin——有约束线性最小二乘函数

此函数用于求解如下最优化问题：

$$\min_x\|\boldsymbol{Cx}-\boldsymbol{d}\|^2$$

约束条件：

$$\boldsymbol{Ax}\leqslant\boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x}=\boldsymbol{b}_{eq},\ \text{且}\boldsymbol{l}\leqslant\boldsymbol{x}\leqslant\boldsymbol{u}\text{。}$$

其调用形式为：

[xo,fo] = lsqlin(C, d, A, b)

[xo,fo] = lsqlin(C, d, A, b, Aeq, beq)

[xo,fo] = lsqlin(C, d, A, b, Aeq, beq, l, u)

[xo,fo] = lsqlin(C, d, A, b, Aeq, beq, l, u, $x0$)

[xo,fo] = lsqlin(C, d, A, b, Aeq, beq, l, u, $x0$, options, $p1$,...)

12.3.4　线性规划函数

MATLAB 内置函数 linprog 用于求解线性规划问题。所谓线性规划问题是指在满足一定的线性约束条件下寻求线性目标函数极值的问题。因此也属于约束最优化问题。

linprog 函数用于求解如下形式的最优化问题：

$$\min f(x)=f^Tx$$

约束条件：

$$Ax\leqslant b,\ A_{eq}x=b_{eq},\ \text{且}l\leqslant x\leqslant u\text{。}$$

该函数的调用形式如下（输入参数至少为 3 个）：

[xo,fo] = linprog(f, A, b, Aeq, Beq, l, u, $x0$, options)。

可知其与 fmincon 的主要区别是其对应的目标函数是线性的，约束条件也是线性的，且其可以不设定初值。该函数在解线性最优化问题时，较之 fmincon 更有效。

【例 12-14】 线性规划函数应用实例。采用线性规划函数求解如下形式的最优化问题：

$$\min f(x)=-5x_1+4x_2+2x_3$$

约束条件：

$$6x_1 - x_2 + x_3 \leqslant 8, x_1 + 2x_2 + 4x_3 \leqslant 10$$
$$-1 \leqslant x_1 \leqslant 3, 0 \leqslant x_2 \leqslant 2, x_3 \geqslant 0$$

解：在 MATLAB 中编写函数 ex1214.m 来进行求解。

```
%ex1214.m 解线性规划问题
%f(x)=-5x(1)+4x(2)+2x(3)
f=[-5,4,2];                          %函数系数
A=[6,-1,1;1,2,4];                    %不等式系数
b=[8;10];                            %不等式右边常数项
l=[-1,0,0];                          %下限
u=[3,2,inf];                         %上限
%%%%用 linprog 求解
[xol,fol]=linprog(f,A,b,[],[],l,u)
%%%%用 fmincon 求解
x0=[0,0,0];
f1014=inline('-5*x(1)+4*x(2)+2*x(3)','x');
[xoc,foc]=fmincon(f1014,x0,A,b,[],[],l,u)
```

所得结果为：

```
Optimization terminated.
xol =                                %linprog 结果
    1.3333
    0.0000
    0.0000
fol =
   -6.6667
    Warning: Large-scale (trust region) method does not currently solve this type
of problem,
     switching to medium-scale (line search).
    > In fmincon at 274
      In ex1214 at 13
    Optimization terminated: first-order optimality measure less
     than options.TolFun and maximum constraint violation is less
     than options.TolCon.
    Active inequalities (to within options.TolCon = 1e-006):
      lower     upper     ineqlin   ineqnonlin
        2                    1
        3
    xoc =                                %fmincon 结果
        1.3333         0         0
    foc =
       -6.6667
```

故 $f(x) = -5x_1 + 4x_2 + 2x_3$ 在满足一定约束条件下在 $x_1 = 1.3333$，$x_2 = x_3 = 0$ 处取最小值 −6.6667。

在此例中两种方法所得结果相同。

另外，最优化工具箱中还有一种用于处理二次规划问题的函数 quadprog。其用法类似于 linprog，不同的是其目标函数为二次函数。即 quadprog 用于处理如下最优化问题：

$$\min \frac{1}{2} \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x}$$

约束条件：

$$\boldsymbol{Ax} \leqslant \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \leqslant \boldsymbol{x} \leqslant \boldsymbol{u}$$

调用形式如下（输入参数至少为 4 个）：

[xo,fo] = quadprog(H, f, A, b, Aeq, beq, l, u, $x0$, options)

12.4　最优化问题应用综合实例

最优化方法在现实中被广泛应用。许多生产规划和管理问题都可以归纳为最优化问题。下面给出几个现实中遇到的最优化的例子，并给出用前几节讨论的最优化方法求解的过程。

12.4.1　无约束最优化问题综合实例

【例 12-15】　最大利润问题综合实例。设某公司生产两种产品分别为 x、y 千克，其相应的成本满足如下函数：

$$C(x, y) = x^2 + 2xy + 2y^2 + 2000$$

已知产品 I 的价格为 200 元每千克，产品 II 的价格为 300 元每千克，并假定两种产品全部售完，试求使公司获得最大利润的这两种产品的生产水平，公司获得的最大利润是多少？

解：公司利润可以用如下表达表达：

$$G(x, y) = 200x + 300y - C(x, y) = 200x + 300y - x^2 - 2xy - 2y^2 - 2000$$

故此最大利润问题等价于目标函数 $f(x, y) = -G(x, y)$ 在自变量大于零的区间上的最小值问题。

首先编写目标函数文件 f1215.m

```
function f=f1215(x)
%例 12-5 中利润函数的相反数
f=-200*x(1)-300*x(2)+x(1)^2+2*x(1)*x(2)+2*x(2)^2+2000;
```

分别用 MATLAB 内置函数 fminunc 以及 fminsearch 求解。

```
%ex1215.m 求解最大利润问题
x0=[0,0];
[xo_s,yo_s]=fminsearch('f1215',x0)
[xo_m,yo_m]=fminunc('f1215',x0)
```

所得结果为：

```
xo_s =
   50.0000   50.0000
yo_s =
  -1.0500e+004
Warning: Gradient must be provided for trust-region method;
  using line-search method instead.
> In fminunc at 243
  In ex1215 at 4
```

```
    Optimization terminated: relative infinity-norm of gradient less than
options.TolFun.
    xo_m =
       50.0000   50.0000
    yo_m =
    -1.0500e+004
```

由结果可知，当 $x=50$，$y=50$ 时，也就是两种产品分别生产 50 千克时，利润最大为 10500 元。

【例 12-16】 最大容积问题综合实例。对边长为 4m 的正方形铁板，在四个角处剪去相等的正方形以制成方形无盖水槽，问何种剪法使水槽的容积最大？

解：设减去的正方形边长为 x，则方形水槽的容积可用下式表示：

$$V(x)=(4-2x)^2x$$

求最大容积问题可等价于如下形式的最小值问题：

$$\min f(x)=-V(x)=-x(4-2x)^2,\quad x\in(0,2)$$

首先编写目标函数文件 f1216.m:

```
function f=f1216(x)
%例 12-16 目标函数
f=-x*(4-2*x)^2;
```

然后调用 fminbnd 函数求解，见 ex1216.m：

```
%ex1216.m 最大容积问题
[xo,fo]=fminbnd('f1216',0,2)
```

得到问题的解：

```
xo =
    0.6667
fo =
   -4.7407
```

由结果可知，当剪掉的正方形的边长为 0.5557m 时水槽的容积最大为 4.7407 m^3。

12.4.2 约束最优化问题综合实例

1. 线性规划问题（目标函数线性，约束线性）

【例 12-17】 最优生产决策问题综合实例。某厂生产甲乙两种产品，已知制成一吨产品甲需用资源 A 4 吨，资源 B 5m^3,资源 C 2 个单位；制成一吨产品乙需用资源 A 2 吨，资源 B 4m^3，资源 C 7 个单位。若一吨产品甲和乙的经济价值分别为 6 万元和 5 万元，三种资源的限制量分别为 90 吨、200m^3 和 210 个单位，试决定应生产这两种产品各多少吨才能使创造的总经济价值最高？

解：令生产产品甲的数量为 $x1$，生产产品乙的数量为 $x2$。由题意可以建立下面的模型：

$$C(x_1,x_2)=6x_1+5x_2$$

约束条件：

$$4x_1+2x_2\leqslant 90,\ 5x_1+4x_2\leqslant 200,\ 2x_1+7x_2\leqslant 210$$
$$6x_1+5x_2\geqslant 0,\ x_1\geqslant 0,\ x_2\geqslant 0$$

该模型中要求目标函数最大化，需要按照 MATLAB 的要求进行转换，即目标函数为：

$$f(x_1+x_2)=-C(x_1,x_2)=-6x_1-5x_2$$

此题属于线性规划问题，此处用 linprog 求解：

```
%ex1217.m 生产决策问题
f=[-5;-7];                              %线性方程系数
A=[4,2;5,4;2,7;-6,-5;];                 %线性不等式约束系数矩阵
b=[90;200;210;0];
l=[0 0];                                %变量上限
[xo,fo,exitflag]=linprog(f,A,b,[],[],l)
```

所得结果为：

```
Optimization terminated.
xo =
    8.7500
   27.5000
fo =
-236.2500
exitflag =
     1
```

由上可知，生产甲种产品 8.75 吨、乙种产品 27.5 吨可使创造的总经济价值最高。最高经济价值为 236.25 万元。exitflag=1 表示过程正常收敛于解 x 处。

【例 12-18】 投资问题综合实例。某单位有一批资金用于四个工程项目的投资，用于各工程项目时所得到得净收益（投入资金的百分比）如表 12-6 所示。

表 12-6 工程项目收益表

工 程 项 目	A	B	C	D
收 益（%）	20	12	9	14

由于某种原因，决定用于项目 A 的投资不大于其他各项投资之和；而用于项目 B 和 C 的投资要大于项目 D 的投资。试确定使该单位收益最大的投资分配方案。

解：用 $x1$、$x2$、$x3$ 和 $x4$ 分别代表用于项目 A、B、C 和 D 的投资百分数，由于各项目的投资百分数之和必须等于 100%，所以

$$x_1+x_2+x_3+x_4=1$$

据题意，可以建立下面的数学模型：

$$G(x_1,x_2,x_3)=0.2x_1+0.12x_2+0.09x_3+0.14x_4$$

约束条件：

$$x_1+x_2+x_3+x_4=1$$
$$x_1\leqslant x_2+x_3+x_4$$
$$x_2+x_3\geqslant x_4$$
$$0\leqslant x_1\leqslant 1,\ 0\leqslant x_2\leqslant 1,\ 0\leqslant x_3\leqslant 1,\ 0\leqslant x_4\leqslant 1$$

将它转换为最小值形式，并将约束变换为标准形式如下：

$$f(x_1,x_2,x_3)=-G(x_1,x_2,x_3)=-0.2x_1-0.12x_2-0.09x_3-0.14x_4$$

约束条件：

$$x_1+x_2+x_3+x_4=1$$

$$x_1-x_2-x_3-x_4\leqslant 0$$

$$x_4-x_2-x_3\leqslant 0$$

$$0\leqslant x_1\leqslant 1,\ 0\leqslant x_2\leqslant 1,\ 0\leqslant x_3\leqslant 1,\ 0\leqslant x_4\leqslant 1$$

用 MATLAB 求解如下所示：

```
%ex1218.m 投资问题
f=[-0.2;-0.12;-0.09;-0.14];
A=[1,-1,-1,-1;0,-1,-1,1];
b=[0;0];
Aeq=[1,1,1,1];
beq=1;
l=[0 0 0];
u=[1 1 1];
[xo,yo,exitflag]=linprog(f,A,b,Aeq,beq,l,u)
```

运行结果为：

```
Optimization terminated.
xo =
    0.5000
    0.2500
    0.0000
    0.2500
yo =
   -0.1650
exitflag =
     1
```

可见，四个项目的投资百分数分别为 0.50、0.25、0.00 和 0.25 时可使该单位获得最大的收益。最大收益为 15.5%，过程正常收敛。

2. 一般约束最优化问题（目标函数和约束非线性）

【例 12-19】 最小费用问题综合实例。假设需要制造这样一个长方体盒子：它的长不大于宽的 3 倍，三边之和不大于 30，盒子的上下两面的材料是每平方厘米 10 元，盒子的其余面的材料是每平方厘米 6 元，且盒子的体积需要 50 立方厘米。求使盒子造价最低的长、宽、高。

解：设盒子长、宽、高分别为 x、y、z，由题意可得如下最优化问题：

$$\min f(x,y,z)=20xy+12(xz+yz)$$

约束条件：

$$x\leqslant 3y,\ xyz=50,\ x+y+z\leqslant 30,\ x>0,y>0,z>0$$

首先编写目标函数 f1219.m：

```
function f=f1219(x)
%例 12－19 目标函数
```

```
f=20*x(1)*x(2)+12*x(1)*x(3)+12*x(2)*x(3);
```

用函数 fmincon 求解，由于其含有非线性约束，故需编写非线性约束函数文件 fcon1219.m：

```
function [c,ceq]=fcon1219(x)
%例 12-19 非线性约束条件
c=[x(1)+x(2)+x(3)-30;x(1)-3*x(2)];
ceq=x(1)*x(2)*x(3)-50;
```

程序文件 ex1219.m：

```
%ex1219.m 最小费用问题
x0=[1,1,1];
l=zeros(1,3);
[xo,yo]=fmincon('f1019',x0,[],[],[],[],l,[],'fcon1219')
```

运行结果为：

```
Warning: Large-scale (trust region) method does not currently solve this type
of problem,
 switching to medium-scale (line search).
> In fmincon at 274
  In ex1219 at 4
Optimization terminated: magnitude of directional derivative in search
 direction less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon.
No active inequalities.

xo =
   3.1072    3.1072    5.1787
yo =
  579.2936
```

由上述结果可知，当长方体盒子的长、宽、高分别为 3.1072cm，3.1072cm，5.1787cm 时，所需费用最少，为 579.2936 元。

3. 最小最大值问题

【例 12-20】 最佳定位问题综合实例。设某城市有某种物品的 10 个需求点，第 i 个需求点 P_i 的坐标为（a_i,b_i），道路网与坐标轴平行，彼此正交。现打算建一个该物品的供应中心，且由于受到城市某些条件的限制，该供应中心只能设在 x 界于[5,8]，y 界于[5,8]的范围内。问该中心应建在何处为好？

P_i 点的坐标为：

a_i: 1 4 3 5 9 12 6 20 17 8

b_i: 2 10 8 18 1 4 5 10 8 9

解：设供应中心的位置为（x,y），要求它到最远需求点的距离尽可能小。由于此处应采用沿道路行走的距离，可知用户 P_i 到该中心的距离为 $|x-a_i|+|y-b_i|$，从而可得目标函数如下：

$$\min\max(|x-a_i|+|y-b_i|)$$

约束条件：

$$5 \leqslant x \leqslant 8,\ 5 \leqslant y \leqslant 8$$

首先编写一个计算 x 处 10 个目标函数的 M 文件 f1220.m。

```
function f = f1220(x)
%输入各个点的坐标值,及目标函数值
a=[1  4  3  5  9  12  6  20  17  8];
b=[2  10 8  18 1  4   5  10  8   9];
f(1) = abs(x(1)-a(1))+abs(x(2)-b(1));
f(2) = abs(x(1)-a(2))+abs(x(2)-b(2));
f(3) = abs(x(1)-a(3))+abs(x(2)-b(3));
f(4) = abs(x(1)-a(4))+abs(x(2)-b(4));
f(5) = abs(x(1)-a(5))+abs(x(2)-b(5));
f(6) = abs(x(1)-a(6))+abs(x(2)-b(6));
f(7) = abs(x(1)-a(7))+abs(x(2)-b(7));
f(8) = abs(x(1)-a(8))+abs(x(2)-b(8));
f(9) = abs(x(1)-a(9))+abs(x(2)-b(9));
f(10) = abs(x(1)-a(10))+abs(x(2)-b(10));
```

然后输入初值、约束条件，并调用优化过程进行计算(M 文件为 ex1220.m)：

```
%ex1220.m 定位问题
x0 = [6; 6];        % 提供解的初值
l=[5,5]; %上界
u=[8,8]; %下界
[x,fval] = fminimax(@f1220,x0,[],[],[],[],l,u)
```

MATLAB 运行结果为：

```
Optimization terminated: magnitude of search direction less than 2*options.TolX
 and maximum constraint violation is less than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper     ineqlin   ineqnonlin
             1                    8
             2


x =
     8
     8
fval =

    13     6     5    13     8     8     5    14     9     1
```

可见，在限制区域内的东北角设置供应中心可以使该点到各需求点的最大距离最小。最小最大距离为 14 个距离单位。

12.5 小结

本章讲述了在 MATLAB 中实现常见的几类最优化计算的方法，包括无约束最优化方法：黄金搜索法、二次插值法、Nelder-Mead 方法（单纯型算法）、最速下降法、牛顿方法、模拟退火法（SA）和遗传算法（GA），以及约束最优化方法：拉格朗日乘子法和惩罚函数法。最后介绍了 MATLAB 最优化工具箱中的一些优化函数的使用。

附录 A

MATLAB 程序设计入门

MATLAB 语言是一种高级计算语言，也具有和 C、FORTRAN 等高级语言相类似的语言特征，能方便地实现程序控制，即编制一种以.m 为扩展名的 MATLAB 程序（简称 M 文件）。而且，由于 MATLAB 本身的一些特点，M 文件的编制同上述几种高级语言比较起来，有许多无法比拟的优点。

读者首先应掌握 MATLAB 程序设计的基本方法，不断实践，逐步将其强大的功能应用到科学计算及其他领域的学习和应用中去。

A.1 MATLAB 概述

A.1.1 MATLAB 启动和退出

以 Windows 操作系统为例，进入 Windows 后，选择“开始”→“程序”→“Matlab 7.0”，便可以进入如图 A-1 所示的 MATLAB 主窗口；如果安装时选择在桌面上生成快捷方式，也可以单击快捷方式直接启动。

在启动 MATLAB，命令编辑区显示帮助信息后，将显示符号“I”，符号“I”表示 MATLAB 已准备好，正等待用户输入命令，这时，就可以在提示符“I”后面输入命令，按下回车键后，MATLAB 就会解释执行所输入的命令，并在命令后面给出计算结果。如果在输入命令后再以分号结束，则不会显示结果。

退出 MATLAB 系统的方式有两种：

（1）在文件菜单(File)中选择“Exit”或“Quit”。

（2）用鼠标点击窗口右上角的关闭图标“×”。

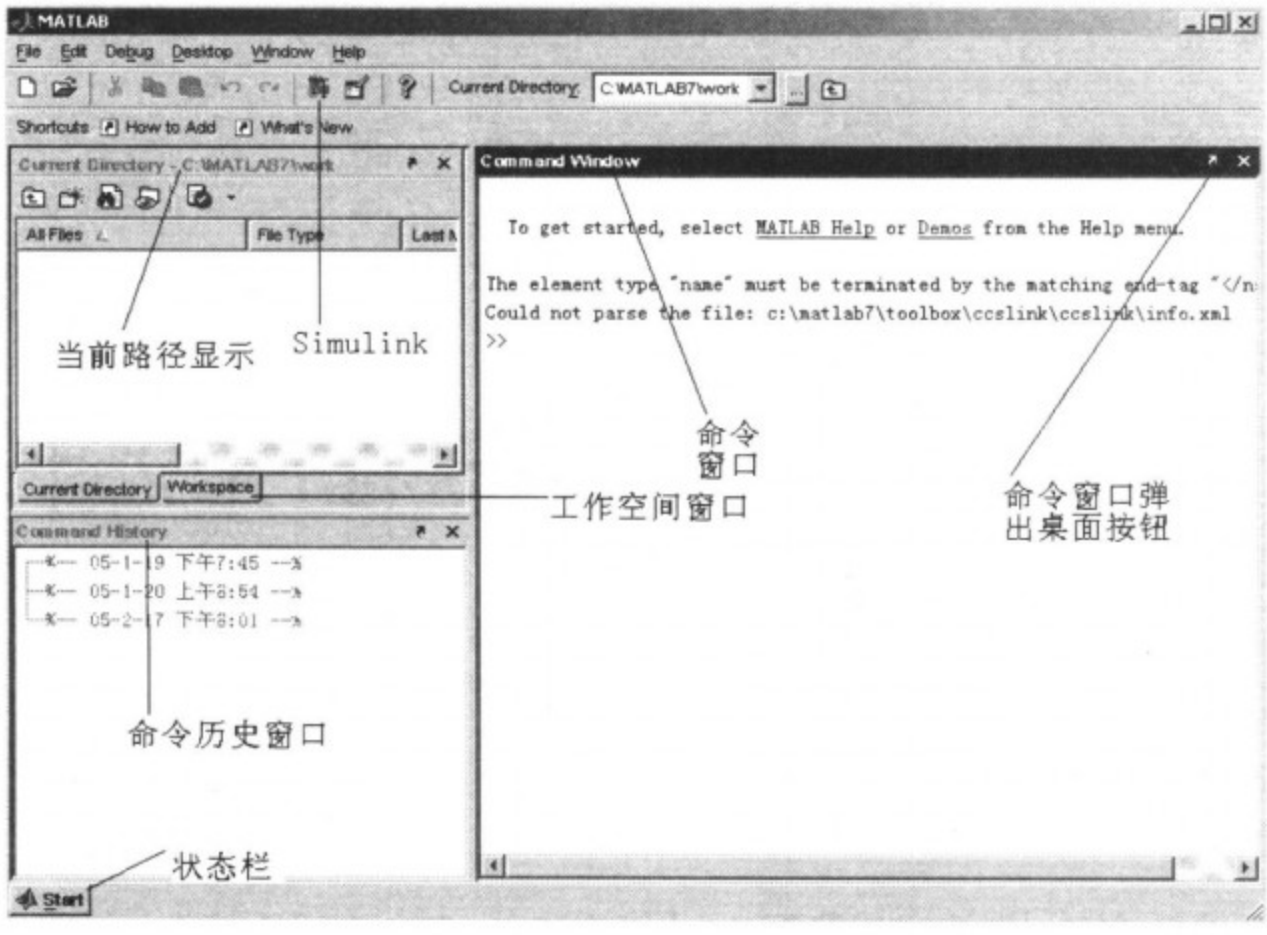

图 A-1　MATLAB 主窗口

A.1.2　MATLAB 系统的组成

MATLAB 系统由 MATLAB 开发环境、MATLAB 数学函数库、MATLAB 语言、MATLAB 图形处理系统和 MATLAB 应用程序接口（API）五大部分构成。

（1）MATLAB 开发环境

MATLAB 开发环境是一套方便用户使用 MATLAB 函数和文件的工具集，其中许多工具是图形化用户接口。它是一个集成化的工作空间，可以让用户输入输出数据，并提供了 M 文件的集成编译和调试环境。它包括 MATLAB 桌面、命令窗口、M 文件编辑调试器、MATLAB 工作空间和在线帮助文档。

（2）MATLAB 数学函数库

MATLAB 数学函数库包括大量的计算算法，包括从基本运算，如加法、正弦运算等，到复杂算法，如矩阵求逆、贝塞尔函数、快速傅里叶变换等。

（3）MATLAB 语言

MATLAB 语言是一个高级的基于矩阵/数组的语言，它具有程序流控制、函数、数据结构、输入/输出和面向对象编程等特色。用户既可以用它来快速编写简单的程序，也可以用来编写庞大复杂的应用程序。

（4）MATLAB 图形处理系统

图形处理系统使得 MATLAB 能方便地图形化显示向量和矩阵，而且能对图形添加标注和打印。它包括强大的二维、三维图形函数、图像处理和动画显示等函数。

（5）MATLAB 应用程序接口（API）

MATLAB 应用程序接口（API）是一个使 MATLAB 语言能与 C、FORTRAN 等其他高级编程语言进行交互的函数库，该函数库的函数通过调用动态链接库（DLL）实现与 MATLAB 文件的数据交换，其主要功能包括在 MATLAB 中调用 C 和 FORTRAN 程序，以及在 MATLAB 与其他应用程序间建立客户/服务器关系。

A.1.3　MATLAB 7 的最新特点

经过二十余年的补充与完善以及多个版本的升级换代，MATLAB 已发展至 2007 a 版本。MATLAB 7 在 MATLAB 6.5 的基础上进行了很多改善和提高。

MATLAB 7 在编程环境、代码效率、数据可视化、数学计算、文件 I/O 等方面进行了升级，其中包括如下一些内容。

1．开发环境

（1）重新设计的桌面环境，针对多文档界面应用提供了简便的管理和访问方法，允许用户自定义桌面外观，创建常用命令的快捷方式；

（2）增强数组编辑器（Array Editor）和工作空间浏览器（Workspace Browser）功能，用于数据的显示、编辑和处理；

（3）在当前目录浏览器（Current Directory Browser）工具中，增加代码效率分析和覆盖度分析等功能；

（4）M-Lint 编码分析，辅助用户完成程序性能分析，提高程序执行效率；

（5）增强 M 文件编辑器（M-Editor），支持多种格式源代码文件可视化编辑，如 C/C++、HTML 和 Java 等。

2．编程

（1）支持创建嵌套函数（Nested Function），提供更灵活的代码模块化方式；

（2）匿名函数（Anonymous Function）功能，支持在命令行或者脚本文件中创建单行函数（Single Line Function）；

（3）支持条件分支断点，可以在条件分支语句中进行程序中断调试；

（4）模块化注释，支持为代码段注释。

3．数学运算

（1）支持整数算术运算；

（2）支持单精度数据类型运算，包括基本算术运算、线性代数和 FFT 等；

（3）使用更强大的计算算法包 Qhull 2002.1，提供更丰富的算法支持；

（4）insolve 函数用于处理线性代数方程求解；

（5）ODE 求解器能够处理隐性微分方程组以及多点边界问题。

4．图形和 3-D 可视化

（1）新图形窗体界面；

（2）直接从图形窗体生成 M 代码，可以完成用户自定义绘图；

（3）增强图形窗体注释；

（4）数据浏览工具(Data Exploration Tools)，提供丰富的数据观测手段；

（5）自定义图形对象，提供丰富的图形显示能力；

（6）GUIDE 新增对用户界面面板和 ActiveX 控件支持；

（7）增强句柄图形对象支持完整的 TeX 和 LaTeX 字符集。

5．文件 I/O 和外部接口

（1）新增文件 I/O 函数，支持读取任意格式文本数据文件，并且支持写入 Excel 和 HDF5 格式数据文件；

（2）具有压缩功能的 MAT 文件格式，支持快速数据文件 I/O 能力；

（3）javaaddpath 函数，无需重新启动 MATLAB 即可完成 Java 类的加载、删除等功能；

（4）支持 COM、服务器事件以及 VBS；

（5）支持 SOAP，使用网络服务；

（6）支持 FTP 对象，直接访问 FTP 服务器；

（7）支持 Unicode 编码格式，增强 MAT 文件字符集。

6．性能与系统平台支持

（1）JIT 加速器支持所有数值数据类型；

（2）Windows XP 系统下支持 3GB 内存访问。

A.1.4 MATLAB 7 的工具箱

MATLAB 拥有一个专用的家族产品，用于解决不同领域的问题，称之为工具箱（Toolbox），工具箱用于 MATLAB 的计算和画图，通常是 M 文件和高级 MATLAB 语言集合，以使用户可以方便地修改函数和源代码，或增加新的函数。用户还可以很方便地结合使用不同的工具箱中的技术来设计针对某个问题的用户解决方案。

MATLAB 每年都会增加一些新的工具箱，所以，在一般情况下，工具箱的列表不是固定不变的。有关 MATLAB 工具箱的最新信息可以在 http://www.mathworks.com/products 中看到。

较为常见的 MATLAB 工具箱包括如下。

1．控制类工具箱

控制类工具箱包括：

（1）控制系统工具箱（Control Systems Toolbox）；

（2）系统辨识工具箱（System Identification Toolbox）；

（3）鲁棒控制工具箱（Robust Control Toolbox）；

（4）模糊逻辑工具箱（Fuzzy Logic Toolbox）；

（5）神经网络工具箱（Neural Network Toolbox）；

（6）频域系统辨识工具箱（Frequency Domain System Identification Toolbox）；

（7）模型预测控制工具箱（Model Predictive Control Toolbox）；
（8）多变量频率设计工具箱（Multivariable Frequency Design Toolbox）。

2．应用数学类工具箱

应用数学类工具箱包括：
（1）最优化工具箱（Optimization Toolbox）；
（2）样条工具箱（Spline Toolbox）；
（3）统计工具箱（Statistics Toolbox）；
（4）偏微分方程工具箱（Partial Differential Equation Toolbox）。

3．信号处理类工具箱

信号处理类工具箱包括：
（1）信号处理工具箱（Signal Processing Toolbox）；
（2）滤波器设计工具箱（Filter Design Toolbox）；
（3）通信工具箱（Communication Toolbox）；
（4）小波分析工具箱（Wavelet Toolbox）；
（5）高阶谱分析工具箱（Higher Order Spectral Analysis Toolbox）。

4．其他常用的工具箱

其他常用的工具箱包括：
（1）符号数学工具箱（Symbolic Math Toolbox）；
（2）虚拟现实工具箱（Virtual Reality Toolbox）。
MATLAB 7.0 对以前的版本进行了升级，主要新增了以下一些功能强大的工具箱：
（1）定点工具箱（Fixed-Point Toolbox）；
（2）遗传算法和直接搜寻工具箱（Genetic Algorithm and Direct Search Toolbox）；
（3）射频工具箱（RF Toolbox）；
（4）生物信息工具箱（Bioinformatics Toolbox）；
（5）OPC 服务器工具箱（OPC Toolbox）。

A.2　MATLAB 桌面环境

MATLAB 为用户提供了全新的桌面操作环境，了解并熟悉这些桌面操作环境是使用 MATLAB 的基础。下面介绍 MATLAB 的启动、主要功能菜单、命令窗口、工作空间、文件管理和帮助管理等。

A.2.1　MATLAB 主菜单及功能

打开 MATLAB 主窗口后，即弹出其主菜单栏。主菜单栏各菜单项及其下拉菜单的功能如下所述。

1. File 主菜单项

单击 File 主菜单项或同时按下“Alt+F”键，弹出如图 A-2 所示的 File 下拉菜单。其中，带下划线的字母表示快捷键，即按该字母键也可执行相应的功能。

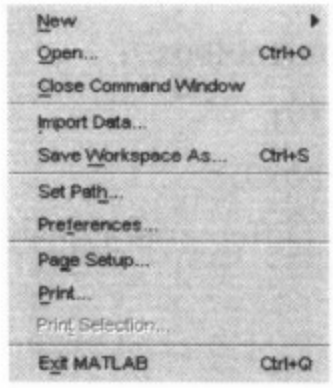

图 A-2　File 下拉菜单

（1）New：用于建立新的.m 文件、图形、模型和图形用户界面；

（2）Open：用于打开 MATLAB 的.m 文件、.fig 文件、.mat 文件、.mdl 文件、.cdr 文件等，也可通过快捷键“Ctrl+O”来实现此项操作；

（3）Close Command Window：关闭命令窗口；

（4）Import Data：用于从其他文件导入数据，单击后弹出对话框，选择导入文件的路径和位置；

（5）Save Workspace As：用于把工作空间的数据存放到相应的路径文件中；

（6）Set Path：设置工作路径；

（7）Preferences：用于设置命令窗的属性，单击该选项，弹出如图 A-3 所示属性对话框。

（8）Page Setup：用于页面设置；

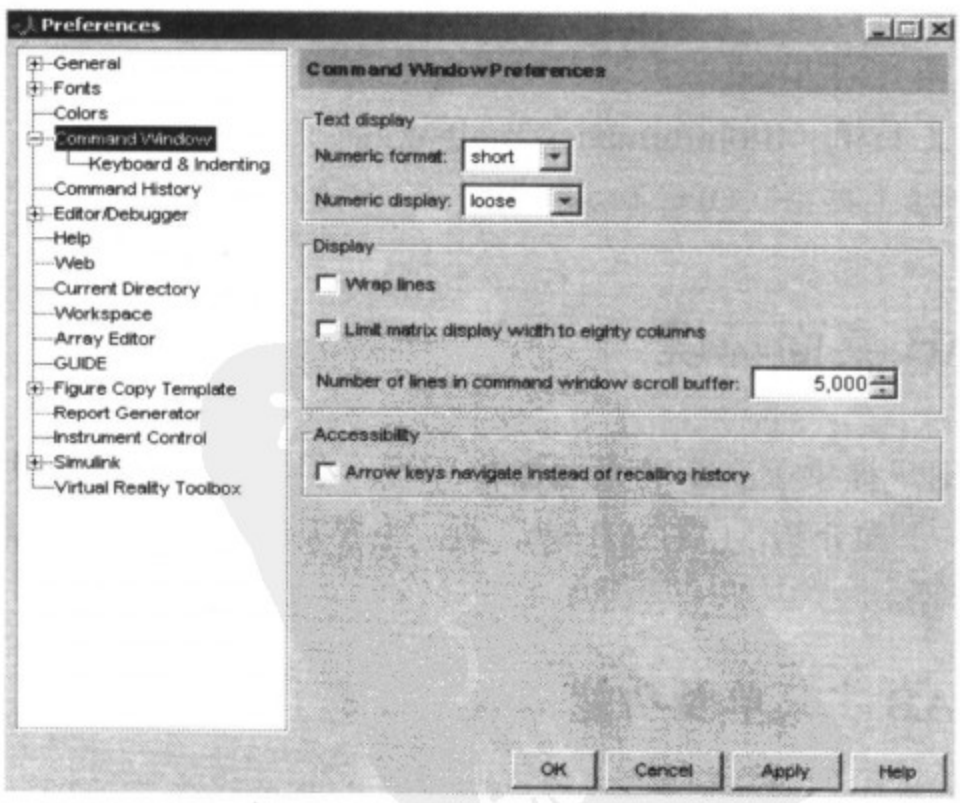

图 A-3　命令窗口属性对话框

（9）Print：用于设置打印属性；

（10）Print Selection：用于对选择的文件数据进行打印设置；

（11）Exit MATLAB：退出 MATLAB 桌面操作环境。

2. Edit 主菜单项

单击 Edit 主菜单项或同时按下“Alt+E”键，弹出如图 A-4 所示的下拉菜单。

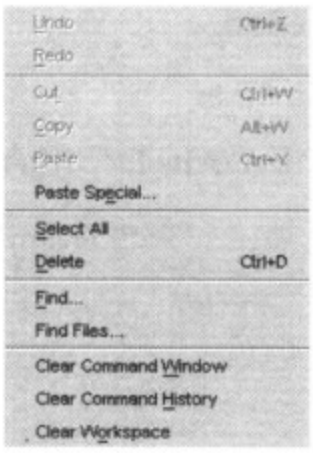

图 A-4　Edit 下拉菜单

（1）Undo：用于撤销上一步操作，也可通过快捷键“Ctrl+Z”来实现此项操作；

（2）Redo：用于重新执行上一步操作；

（3）Cut：用于剪切选中的对象，也可通过快捷键“Ctrl+W”来实现此项操作；

（4）Copy：用于复制选中的对象，也可通过快捷键“Alt+W”来实现此项操作；

（5）Paste：用于粘贴剪贴板上的内容，也可通过快捷键“Ctrl+Y”来实现此项操作；

（6）Paste Special：用于特定内容的粘贴；

（7）Select All：用于全部选择；

（8）Delete：用于删除所选的对象，也可通过快捷键“Ctrl+D”来实现此项操作；

（9）Find：用于查找所需选择的对象；

（10）Find Files：用于查找所需文件；

（11）Clear Command Window：用于清除命令窗口区的对象；

（12）Clear Command History：用于清除命令窗口区的历史记录；

（13）Clear Workspace：用于清除工作区的对象。

3. Debug 主菜单项

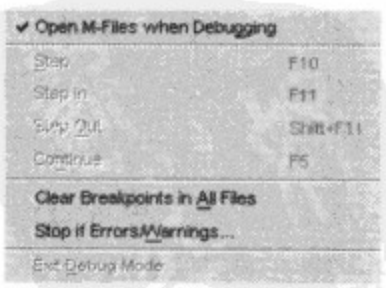

图 A-5　Debug 下拉菜单

单击 Debug 主菜单项或同时按下“Alt+B”组合键，弹出如图 A-5 所示的下拉菜单。

（1）Open M-Files when Debugging：用于调试时打开 M 文件；

（2）Step：用于单步调试程序，也可通过快捷键“F10”来实现此项操作；

（3）Step in：用于单步调试进入子函数，也可通过快捷键“F11”来实现此项操作；
（4）Step Out：用于单步调试从子函数跳出，也可通过快捷键“Shift+F11”来实现此项操作；
（5）Continue：程序执行到下一断点，也可通过快捷键“F5”来实现此项操作；
（6）Clear Breakpoints in All Files：清除所有打开文件中的断点；
（7）Stop if Errors/Warnings：在程序出错或报警处停止往下执行；
（8）Exit Debug Mode：退出调试模式。

4．Desktop 主菜单项

单击 Desktop 主菜单项或同时按下“Alt+D”组合键，弹出如图 A-6 所示的下拉菜单。

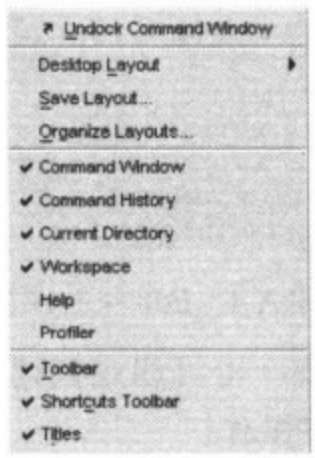

图 A-6　Desktop 下拉菜单

（1）Undock Command Window：将命令窗口变为全屏显示，并设为当前活动窗口；
（2）Desktop Layout：单击该项后，弹出如图 A-7 所示的子菜单；用于工作区的设置，其设置选项包括系统默认设置项（Default）、单独命令窗口项（Command Window Only）、命令历史窗口和命令窗口项（History and Command Window）、全部标签项显示（All Tabbed）；

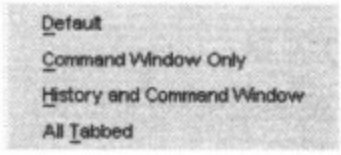

图 A-7　Desktop Layout 子菜单

（3）Save Layout：保存选定的工作区设置；
（4）Organize Layouts：管理保存的工作区设置；
（5）Command Window：命令窗口项，选择该项，屏幕上便会显示该窗口；
（6）Command History：命令历史窗口项，选择该项，屏幕上便会显示该窗口；
（7）Current Directory：当前路径窗口项，选择该项，屏幕上便会显示该窗口；
（8）Workspace：工作窗口项，选择该项，屏幕上便会显示该窗口；
（9）Help：帮助窗口项，选择该项，屏幕上便会显示该窗口；
（10）Profiler：轮廓图窗口项，选择该项，屏幕上便会显示该窗口；
（11）Toolbar：显示或隐藏工具栏选项；
（12）Shortcuts Toolbar：显示或隐藏快捷方式选项；
（13）Titles：显示或隐藏标题栏选项。

5. Window 主菜单项

单击 Window 主菜单项或同时按下“Alt+W”键，弹出如图 A-8 所示的下拉菜单。

（1）Close All Documents：关闭所有文档；

（2）0 Command Window：选定命令窗口为当前活动窗口，也可通过快捷键“Ctrl+0”来实现此项操作；

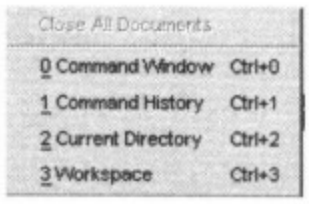

图 A-8　Window 下拉菜单

（3）1 Command History：选定命令历史窗口为当前活动窗口，也可通过快捷键“Ctrl+1”来实现此项操作；

（4）2 Current Directory：选定当前路径窗口为当前活动窗口，也可通过快捷键“Ctrl+2”来实现此项操作；

（5）3 Workspace：选定工作空间窗口为当前活动窗口，也可通过快捷键“Ctrl+3”来实现此项操作。

6. Help 主菜单项

单击 Help 主菜单项或同时按下“Alt+H”键，弹出如图 A-9 所示的下拉菜单。

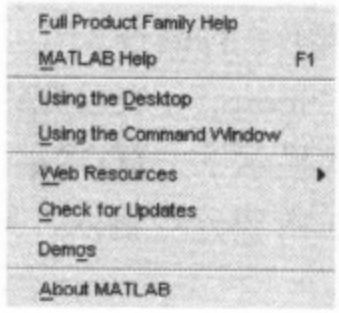

图 A-9　Help 下拉菜单

（1）Full Product Family Help：显示所有 MATLAB 产品的帮助信息；

（2）MATLAB Help：启动 MATLAB 帮助；

（3）Using the Desktop：启动 Desktop 的帮助；

（4）Using the Command Window：启动命令窗口的帮助；

（5）Web Resources：显示因特网上一些相关的资源网址；

（6）Check for Updates：检查软件是否更新；

（7）Demos：调用 MATLAB 所提供的范例程序；

（8）About MATLAB：显示有关 MATLAB 的信息。

A.2.2　MATLAB 命令窗口

MATLAB 的命令窗口，如图 A-10 所示。它用于 MATLAB 命令的交互操作，它具有两大主要功能：

（1）提供用户输入命令的操作平台，用户通过该窗口输入命令和数据；

（2）提供命令执行结果的显示平台，该窗口显示命令执行的结果。

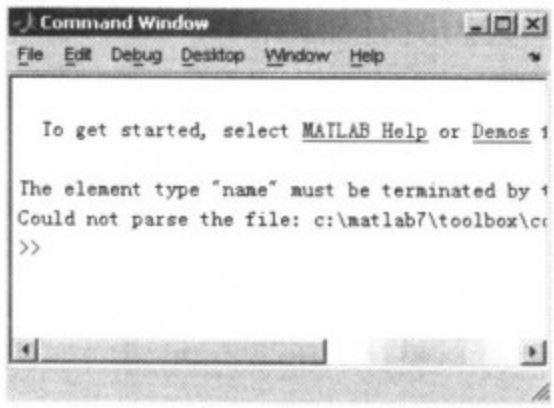

图 A-10　MATLAB 的命令窗口

计算机安装好 MATLAB 之后，双击 MATLAB 图标，就可以进入命令窗口，此时意味着系统处于准备接受命令的状态，可以在命令窗口中直接输入命令语句。

MATLAB 语句形式如下所示：

>>变量＝表达式；

通过等于号将表达式的值赋予变量。当按下回车键时，该语句被执行。语句执行之后，窗口自动显示出语句执行的结果。如果希望结果不被显示，则只要在语句之后加上一个分号（;）即可。此时尽管结果没有显示，但它依然被赋值并在 MATLAB 工作空间中分配了内存。

使用方向键和控制键可以编辑、修改已输入的命令，使用向上方向键“↑”回调上一行命令，使用向下方向键“↓”回调下一行命令。

使用 more off 表示不允许分页；more on 表示允许分页；more(*n*)表示指定每页输出的行数；回车键表示前进一行，空格键显示下一页；“q”键用于结束当前显示。

如果命令语句超过一行或者太长希望分行输入，则可以使用多行命令继续输入。如输入下列式子时，可以通过两行输入：

```
S=1-12+13+4+...
9-4-18;
```

A.2.3　MATLAB 工作空间

MATLAB 的工作空间（Workspace），如图 A-11 所示。

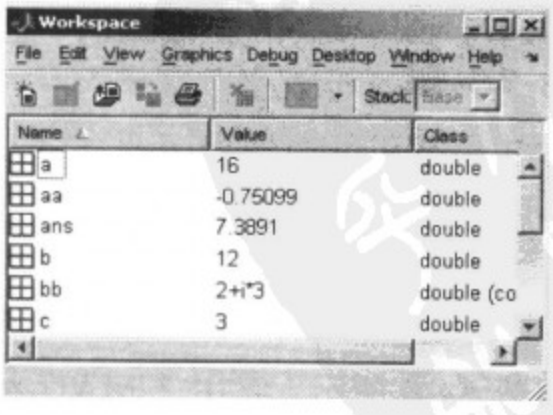

图 A-11　MATLAB 的工作空间

工作空间包含了一组可以在命令窗口中调整（调用）的参数，常用的工作空间操作的命令及功能如表 A-1 中所示。

表 A-1　工作空间常用操作命令及功能

命　令	功　能
clc	擦去一页命令窗口，光标回屏幕左上角
clear	清除工作空间中所有的变量
clear all	从工作空间清除所有变量和函数
clear　变量名	清除指定的变量
clf	清除图形窗口内容
delete <文件名>	从磁盘中删除指定文件
help <命令名>	查询所列命令的帮助信息
which <文件名>	查找指定文件的路径
who	显示当前工作空间中所有变量的一个简单列表
whos	列出变量的大小、数据格式等详细信息
what	列出当前目录下的 m 文件和 mat 文件
load name	下载'name'文件中的所有变量到工作空间
load name *x y*	下载'name'文件中的变量 *x*,*y* 到工作空间
save name	保存工作空间变量到文件 name.mat 中
save name *x y*	保存工作空间变量 *x y* 到文件 name.mat 中
pack	整理工作空间内存
size(变量名)	显示当前工作空间中变量的尺寸
length(变量名)	显示当前工作空间中变量的长度
disp(变量名)	显示当前工作空间中变量
“↑”或“Ctrl+P”	调用上一次的命令
“↓”或“Ctrl+N”	调用下一行的命令
“←”或“Ctrl+B”	退后一格
“→”或“Ctrl+F”	前移一格
Home 或“Ctrl+A”	光标移到行首
End 或“Ctrl+E”	光标移到行尾
Esc 或“Ctrl+U”	清除一行
Del 或“Ctrl+D”	清除光标后字符
Backspace 或“Ctrl+H”	清除光标前字符
“Ctrl+K”	清除光标至行尾字
“Ctrl+C”	中断程序运行

MATLAB 提供了以下保存和载入 workspace 的命令。

（1）save filename variables

将变量列表 variables 所列出的变量保存到磁盘文件 filename 中，variables 所表示的变量列表中不能用逗号，各个不同的变量之间只能用空格来分隔。未列出 variables 时，表示将当前工作空间中所有变量都保持到磁盘文件中。默认的磁盘文件扩展名为“.mat”，可以

使用连字符“-”定义不同的存储格式（ASCII、V4 等）。

（2）load filename variables

将以前用 save 命令保存的变量 variables 从磁盘文件中调入 MATLAB 工作空间。用 load 命令调入的变量，其名称为用 save 命令保存时的名称，取值也一样。

variables 所表示的变量列表中，不能用逗号，各个不同的变量之间只能用空格来分隔。未列出 variables 时，表示将磁盘文件中的所有变量都调入工作空间。

（3）退出工作空间

使用 quit 或 exit 命令可退出工作室间。

A.2.4 MATLAB 的文件管理

MATLAB 提供了一组文件管理命令，包括列文件名，显示或删除文件，显示或改变当前目录等，相关的命令及功能如表 A-2 所示：

表 A-2 MATLAB 常用文件管理命令

命　令	功　能
what	显示当前目录下所有与 MATLAB 相关的文件及它们的路径
dir	显示当前目录下所有的文件
which	显示某个文件的路径
cd path	由当前目录进入 path 目录
type filename	在命令窗口中显示文件 filename
delete filename	删除文件 filename
cd ..	返回上一级目录
cd	显示当前目录

A.2.5 使用 MATLAB 的帮助

MALTAB 的各个函数，不管是内建函数、M 文件函数，还是 MEX 文件函数等，一般都有 M 文件的使用帮助和函数功能说明，各个工具箱通常情况下也具有一个与工具箱名相同的 M 文件用来说明工具箱的构成内容等。在 MATLAB 命令窗口中，可以通过指令来获取这些纯文本的帮助信息。

常用的获取帮助的指令有 help、lookfor、which、doc、get、type，下面分别进行讲述。

1. help 指令

help 指令是 MATLAB 中最有用的指令之一，MATLBA 还提供了丰富的 help 命令，如表 A-3 所示。在命令窗口中输入相关命令就可以获得相关的帮助。

表 A-3 MATLAB 常用帮助命令

命　令	内　容
help matfun	矩阵函数－数值线性代数

续表

命　令	内　容
help general	通用命令
help graphics	通用图形函数
help elfun	基本的数学函数
help elmat	基本矩阵和矩阵操作
help control	控制系统工具箱函数
help datafun	数据分析和傅里叶变换函数
help ops	操作符和特殊字符
help polyfun	多项式和内插函数
help lang	语言结构和调试
help strfun	字符串函数

下面讲述 help 的几种常见用法。

（1）直接使用 help 指令

直接使用 help 指令，可以获取当前计算机上 MATLAB 的分类列表，即当前安装的工具箱名称以及其简要描述。例如，在命令窗口中输入 help，可以得到如下信息：

```
>> help
HELP topics

matlab\general        -  General purpose commands.
matlab\ops            -  Operators and special characters.
matlab\lang           -  Programming language constructs.
matlab\elmat          -  Elementary matrices and matrix manipulation.
matlab\elfun          -  Elementary math functions.
matlab\specfun        -  Specialized math functions.
matlab\matfun         -  Matrix functions - numerical linear algebra.
matlab\datafun        -  Data analysis and Fourier transforms.
matlab\polyfun        -  Interpolation and polynomials.
matlab\funfun         -  Function functions and ODE solvers.
matlab\sparfun        -  Sparse matrices.
matlab\scribe         -  Annotation and Plot Editing.
  ......          ......
reqmgt\rmidemos       -  (No table of contents file)
simulink\slvnv        -  Simulink Verification and Validation
simcoverage\simcovdemos -  (No table of contents file)
MATLAB71\work         -  (No table of contents file)
```

（2）使用 help 获得工具箱使用说明

使用 help 工具箱名，可以获取该工具箱的相关的函数、图形用户工具以及演示文件名等。

用 help 工具箱名，就可以得到该工具箱的函数列表，每个函数后面有简要的说明，可以根据其说明来确定可能需要的是哪个函数。例如，在命令窗口中，输入 help fuzzy 就可以获得该工具箱基本信息和分类函数列表：

```
>> help fuzzy
  Fuzzy Logic Toolbox
  Version 2.2.2 (R14SP3) 26-Jul-2005
```

```
GUI editors
  anfisedit  - ANFIS training and testing UI tool.
......          ......
  surfview   - Output surface viewer.

Membership functions.
  dsigmf     - Difference of two sigmoid membership functions.
......          ......
  zmf        - Z-shaped curve membership function.

Command line FIS functions
  addmf      - Add membership function to FIS
......          ......
  writefis   - Save FIS to disk.

Advanced techniques
  anfis      - Training routine for Sugeno-type FIS (MEX only).
 ......            ......
  sugmax     - Maximum output range for a Sugeno system.

GUI helper files
  cmfdlg     - Add customized membership function dialog.
 ......            ......
  wsdlg      - Open from/save to workspace dialog.
```

（3）使用 help 获得函数使用说明

使用 help 函数名，可以获得该函数的纯文本的帮助信息，通常也带有少量的例子。在 MATLAB 命令窗口中用 help 指令就可获取该函数的具体信息。例如，help fminbnd 可以得到如下信息：

```
>> help fminbnd
 FMINBND Scalar bounded nonlinear function minimization.
  X = FMINBND(FUN,x1,x2) attempts to find  a local minimizer X of the function
  FUN in the interval x1 < X < x2.  FUN is a function handle.  FUN accepts
  scalar input X and returns a scalar function value F evaluated at X.
  ......          ......
%该函数的使用的语法及其各个输入输出参数意义描述<作者注>
  Examples
    FUN can be specified using @:
      X = fminbnd(@cos,3,4)
    ......          ......
% 这里是一些该函数的使用举例<作者注>
  See also optimset, fminsearch, fzero, function_handle.
%一些相关命令
  Reference page in Help browser
     doc fminbnd
```

在采用这种方法得到该函数帮助信息的时候，一个值得注意的问题是最后面的 See also 给出了与该函数相关的一些指令，有时候通过这些相关指令，可以查找到更广泛的有用信息。

说明

省略号由笔者所加，用来表示被删除的内容。这样做是出于节省篇幅的考虑。

2. lookfor 指令

lookfor 指令是在 MATLAB 的搜索路径所有 M 文件中的第一个注释行搜索特定关键字。

通常在不确定需要搜索的函数，但知道该函数的功能的时候，也可以通过 lookfor 搜索该功能的关键字。

例如，想查找一个绘制椭圆曲线的命令，可以用 lookfor ellipse（椭圆的英文单词），得到如下信息，然后再选取函数查找具体的信息：

```
>> lookfor ellipse
RECTANGLE   Create rectangle, rounded-rectangle, or ellipse.
PDEELLIP Draw ellipse, update Geometry Description matrix.
FITMODEL Fit an ellipse to data.
```

再例如，查找包含积分这个关键词的所有指令，应输入：

```
>> lookfor integral
ELLIPKE Complete elliptic integral.
EXPINT Exponential integral function.
   ......              ......
PEVINT Calculate integral of PEV
```

3. which 指令

which 指令可以用来定位该函数的位置，通过这个位置信息，可以获取该函数所属的类别。

通常，在创建一个 M 文件或者保存一个 M 文件的时候，为了避免与系统函数等同名，就应该先用 which 文件(函数)名“-all”搜索查找是否存在想要保存的文件名或者函数名。

另外，很好地利用得到的位置信息可以进行一些相关联的文件的帮助信息。

4. set/get 指令

set/get 指令可以获取图形对象的属性列表和被选属性值。

例如，在 GUI 编程和数据可视化的时候，有时想改变某些对象的属性，让它按照自己的想法实现，但是又想不起这些对象的属性名，更不知道如何设置它们。这时，可以用 get（objecthandles）得到此对象的所有的属性及其当前值，用 set(handles)可以得到对象所有可以设置的属性及其可能的取值。找到需要的属性名和可能的取值之后，就用 set(handles，propertyname，values)设置该对象此属性的值。

A.3　MATLAB 编程

A.3.1　MATLAB 编程概述

MATLAB 不仅是一种功能强大的高级语言，而且是一个集成的交互式开发环境，用户可以通过 MATLAB 提供的编辑调试器编写和调试 MATLAB 代码。

MATLAB 提供了代码书写和调试的集成开发环境，用户可以在 MATLAB 的代码编辑

调试器中完成书写和调试过程。单击 MATLAB 主界面的“新建”工具按钮或者单击文件菜单（File）“新建子菜单”（New）的“M-File”项，就可以打开 MATLAB 代码编辑-调试器，其空白界面如图 A-12 所示。

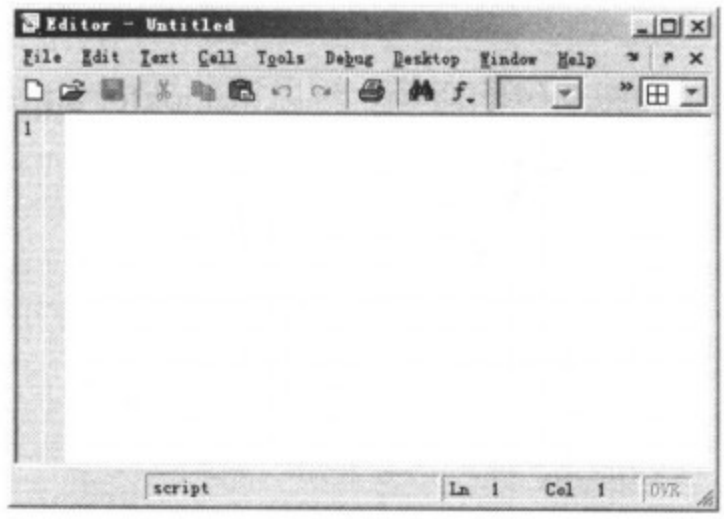

图 A-12　MATLAB 代码编辑-调试器

用户也可以在命令窗口通过 edit filename 命令打开已存在的 M 文件进行编辑调试。

从图 A-12 可见，MATLAB 能够根据 M 文件内容区别是脚本 M 文件还是函数 M 文件，并且在整个编辑过程中追踪光标位置（如图 A-12 底部的 Ln 1 Col 1 表示当前光标处在第一行的第一列），这对于准确快速定位当前编辑和修改位置是很方便有用的。

开发 MATLAB 程序一般需要经历代码编写、调试、优化几个阶段。

在编写代码时，要及时保存阶段性成果，可以通过 File 菜单的 Save 项或者保存工具按钮保存当前 M 文件。

完成代码书写之后，要试运行代码看看有没有运行错误，然后根据针对性的错误提示对程序进行修改。

运行脚本 M 文件，只需要在命令窗口中输入其文件名，然后按回车键，或通过 Debug 菜单的 Run 项，或按快捷键“F5”完成。

运行函数 M 文件，需要通过命令窗口传递输入参数来调用。除了一些很简单的代码，大部分情况下用户都可能遇到程序报错的问题，这就需要对代码进行调试纠错，一般需要通过 Debug 菜单下的子项辅助完成，包括设置断点、逐步运行等项。

当程序运行无误后，还要考虑程序性能是否可以改进。

MATLAB 提供了 M-Lint 和 Profiler 工具，能够辅助用户分析代码运行中时间消耗的细节和可能需要改变的编程细节，如循环赋值前没有预定义数组，用循环去实现可以用数组函数实现的运算等。这些工具都在 Tools 菜单下设了子菜单。

A.3.2　MATLAB 程序设计原则

MATLAB 程序的基本设计原则如下所述：

（1）百分号“%”后面的内容是程序的注解，要善于运用注解使程序更具可读性。

（2）养成在主程序开头用 clear 指令清除变量的习惯，以消除工作空间中其他变量对程

序运行的影响，但注意在子程序中不要用 clear。

（3）参数值要集中放在程序的开始部分，以便维护。要充分利用 MATLAB 工具箱提供的指令来执行所要进行的运算，在语句行之后输入分号使其及中间结果不在屏幕上显示，以提高执行速度。

（4）input 指令可以用来输入一些临时的数据；而对于大量参数，则通过建立一个存储参数的子程序，在主程序中通过子程序的名称来调用。

（5）程序尽量模块化，即采用主程序调用子程序的方法，将所有子程序合并在一起来执行全部的操作。

（6）充分利用 Debugger 来进行程序的调试（设置断点、单步执行、连续执行），并利用其他工具箱或图形用户界面（GUI）的设计技巧，将设计结果集成到一起。

（7）设置好 MATLAB 的工作路径，以便程序运行。

（8）MATLAB 程序的基本组成结构如下表所示。

MATLAB 程序的基本组成结构
%说明
清除命令：清除 workspace 中的变量和图形（clear,close）
定义变量：包括全局变量的声明及参数值的设定
逐行执行命令：指 MATLAB 提供的运算指令或工具箱提供的专用命令
…
…
控制循环：包含 for，if then，switch，while 等语句
逐行执行命令
…
…
end
绘图命令：将运算结果绘制出来

当然，更复杂的程序还需要调用子程序，或与其他应用程序相结合。

A.3.3　M 文件

M 文件是包含 MATLAB 代码的文件。

1．M 文件的类型

M 文件按其内容和功能可以分为脚本 M 文件和函数 M 文件两大类。

（1）脚本 M 文件

脚本 M 文件是许多 MATLAB 代码按顺序组成的命令序列集合，不接受参数的输入和输出，与 MATLAB 工作区共享变量空间。

它一般用来实现一个相对独立的功能，比如对某个数据集进行某种分析、绘图，求解

某个已知条件下的微分方程等。用户可以通过在命令窗口中直接输入文件名来运行脚本 M 文件。

通过脚本 M 文件，用户可以把为实现一个具体功能的一系列 MATLAB 代码书写在一个 M 文件中，每次只需要输入文件名即可运行脚本 M 文件中的所有代码。

（2）函数 M 文件

函数 M 文件也是为了实现一个单独功能的代码块，但与脚本 M 文件不同的是函数 M 文件需要接受参数输入和输出，函数 M 文件中的代码一般只处理输入参数传递的数据，并把处理结果作为函数输出参数返回给 MATLAB 工作区中的指定接收变量。

因此，函数 M 文件具有独立的内部变量空间。在执行函数 M 文件时，要指定输入参数的实际取值，而且一般要指定接收输出结果的工作区变量。

MATLAB 提供的许多函数就是用函数 M 文件编写的，尤其是各种工具箱中的函数，用户可以打开这些 M 文件来查看。实际上，对于特殊应用领域的用户，如果积累了充分的专业领域应用的函数，就可以组建自己的专业领域工具箱了。

通过函数 M 文件，用户则可以把实现一个抽象功能的 MATLAB 代码封装成一个函数接口，在以后的应用中重复调用。

2. M 文件的结构

图 A-13 显示的是 MATLAB 提供的一个函数 M 文件的全部内容，图中清楚地显示了一般的 M 文件包括的各部分结构。

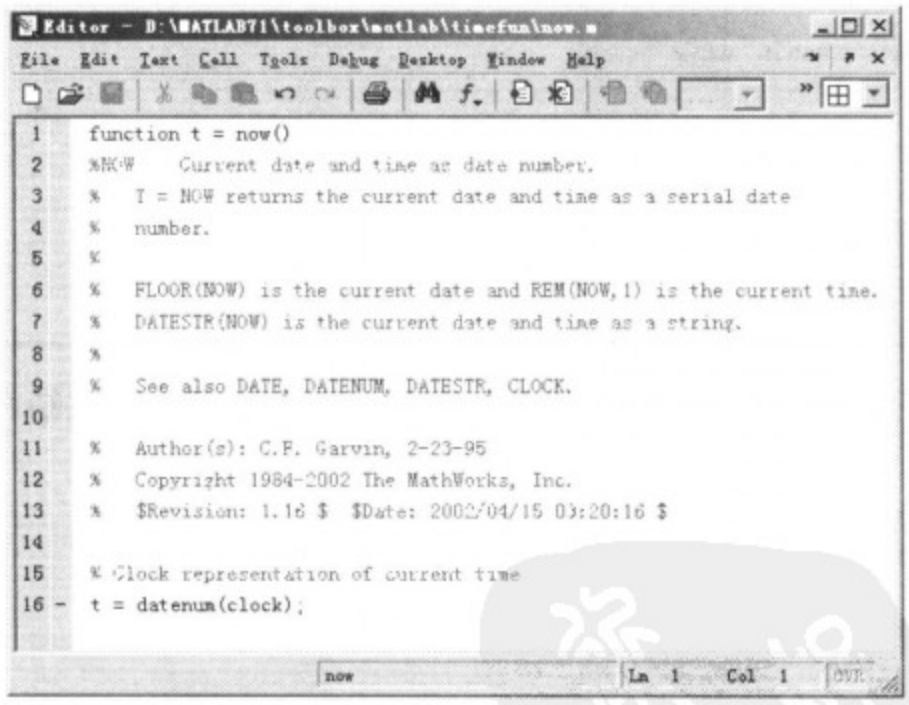

图 A-13　M 文件的一般结构

从图 A-13 可以看到，MATLAB 中 M 文件一般包括以下五部分结构：

（1）函数声明行（Function definition line），这一行只出现在函数 M 文件的第一行，通过 function 关键字表明此文件是一个函数 M 文件，并指定函数名、输入和输出参数，如图 A-13 中的第一行。

（2）H1 行，这是帮助文字的第一行（the first help text line），给出 M 文件帮助最关键

的信息。当用 lookfor 查找某个单词相关的函数时，lookfor 只在 H1 行中搜索是否出现指定单词，如图 A-13 中的第 2 行。

（3）帮助文字，这部分对 M 文件有更加详细的说明，经常解释 M 文件实现的功能，M 文件中出现的各变量、参数的意义，以及创作版权信息等。如图 A-13 中的第 13 行。当获取一个 M 文件的帮助时，H1 行和帮助文字部分同时显示。

（4）M 文件正文，这是 M 文件实现功能的 MATLAB 代码部分，通常包括运算、赋值等指令。图 A-13 的例子中只有第 16 行，但一般都由多行组成。

（5）注释部分，这部分出现的位置比较灵活，主要是用来注释 M 文件正文的具体运行过程，以方便阅读和修改，经常穿插在 M 文件正文中间。

图 A-13 的例子中的第 15 行就是注释说明正文第 16 行的意义。注释一般都是针对接下来的一段正文代码的，常见的 M 文件中也经常包括多行注释。

3. M 文件的创建

虽然一般脚本 M 文件可以包括上述五部分结构中除去“函数声明行”以外的四部分，但在实际应用中，脚本 M 文件经常仅仅由 M 文件正文和注释部分构成。正文部分实现功能，注释部分则给出每一块代码的功能说明。下面通过实例讲述脚本 M 文件的创建。

【例 A-1】 建立一个命令文件将变量 a，b 的值互换。

解：首先打开 M 文件编辑器，输入以下程序：

```
a=1:9;
b=[11,12,13;14,15,16;17,18,19];
c=a;
a=b;
b=c;
a
b
```

然后保存文件名为“A9.m”即完成了文件的建立。

在 MATLAB 的命令窗口中输入 A9，将会执行该命令文件。

```
>> A1
a =
    11    12    13
    14    15    16
    17    18    19
b =
    1     2     3     4     5     6     7     8     9
```

函数 M 文件的命名一般习惯和函数名一致，比如图 A-13 中函数声明行 function *t*=now()，表明函数名为 now，因此此函数 M 文件一般保存为 now.m，调用时就可以通过 now()语句；否则，如果函数名和文件名不一致时，函数调用就需要通过文件名和与函数声明中对应的参数列表。

下面的例 A-2 给出一个作者编写的判断一个点和一个三角形位置关系的函数 M 文件。

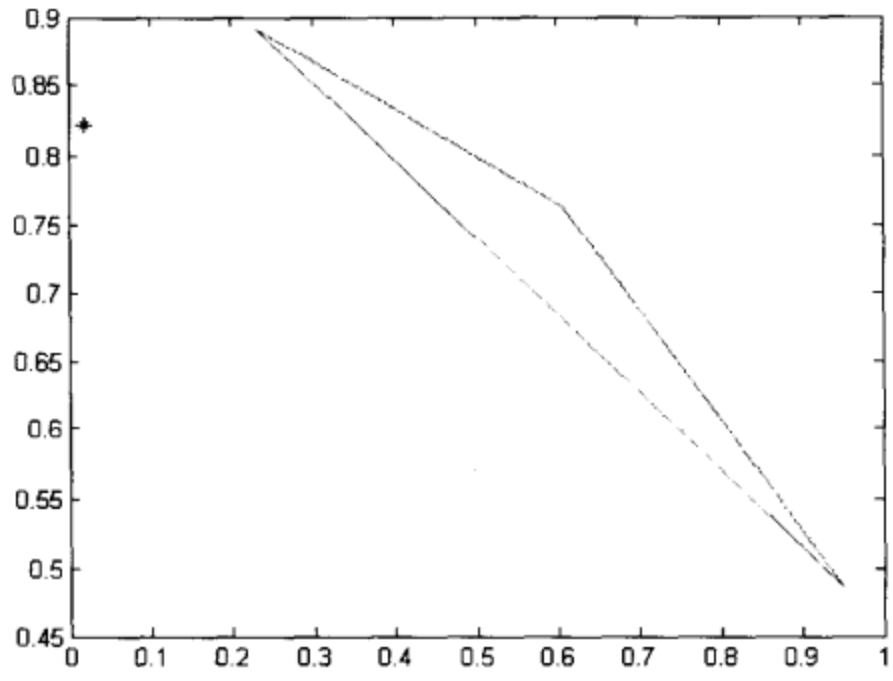

图 A-14　例 A-2 中 myisintri 函数调用后的绘图结果

【例 A-2】　函数 M 文件实例。

```
function y=myisintri(A,M)
%Determine whether point M is in triangle A or not.
%A makes the triangle,A is a 3*2 matrix
%M makes the point,M is a 1*2 vector

%y=1 means M lies in the triangle determined by A.
    %y=0 means M lies on the edge of triangle determined by A.
    %y=-1 means M lies out of the triangle determined by A.

    %assign the coordinates of the three points
    x1=A(1,1);y1=A(1,2);
    x2=A(2,1);y2=A(2,2);
    x3=A(3,1);y3=A(3,2);
    %assigh the coordinates of the target points
    x0=M(1);y0=M(2);
    %Determine the relationship between point M and triangle A
    if and(((M(1)-A(1,1))*(A(2,2)-A(1,2))-(A(2,1)-A(1,1))*(M(2)-A(1,2)))*((M(1)-A(1,1))
    *(A(3,2)-A(1,2))-(A(3,1)-A(1,1))*(M(2)-A(1,2)))<0,((M(1)-A(2,1))*(A(1,2)
-A(2,2))-(A(1,1)-A(2,1))*(M(2)-A(2,2)))*((M(1)-A(2,1))*(A(3,2)-A(2,2))-(A(3,1)
-A(2,1))*(M(2)-A(2,2)))<0)
        y=1;
    elseif or(((M(1)-A(1,1))*(A(2,2)-A(1,2))-(A(2,1)-A(1,1))*(M(2)-A(1,2)))*((M(1)-A(1,1))
    *(A(3,2)-A(1,2))-(A(3,1)-A(1,1))*(M(2)-A(1,2)))==0,((M(1)-A(2,1))*(A(1,2)
-A(2,2))-(A(1,1)-A(2,1))*(M(2)-A(2,2)))*((M(1)-A(2,1))*(A(3,2)-A(2,2))-(A(3,
1)-A(2,1))*(M(2)-A(2,2)))==0)
        y=0;
    else
        y=-1;
    end
    %plot the triangle A and the M point
    plot([A(:,1);A(1,1)],[A(:,2);A(1,2)],'r',M(1),M(2),'b*');
```

由于此函数名为 myisintri，因此该文件最好保存为 myisintri.m。函数调用时，必须指定输入参数的值，比如可以在命令窗口中用如下的指令调用 myisintri 函数：

```
>> A=rand(3,2)
A =
    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621
>> M=rand(1,2)
M =
    0.0185    0.8214
>> myisintri(A,M)    %调用后绘图结果如图 A-14
ans =
    -1
```

编写好的函数 M 文件，相当于 MATLAB 提供的命令，可以在命令行进行函数调用。但要注意，要求被调用的函数对应的.m 文件必须在 MATLAB 路径下。

A.3.4　程序控制语句

和各种常见的高级语言一样，MATLAB 也提供了多种经典的程序结构控制语句。MATLAB 中的程序流程控制语句有：分支控制语句（if 结构和 switch 结构）、循环控制语句（for 循环、while 循环、continue 语句和 break 语句）和程序终止语句（return 语句），下面分别进行介绍。

1．程序分支控制语句

分支语句可以使程序中的一段代码只在满足一定条件时才执行，因此也成为分支选择。MATLAB 中分支语句有两类：if 语句和 switch 语句。

（1）if 与 else 或 elseif 连用，偏向于是非选择，当某个逻辑条件满足时执行 if 后的语句，否则执行 else 语句；

（2）switch 和 case、otherwise 连用，偏向于情况的列举，当表达式结果为某个或某些值时，执行特定 case 指定的语句段，否则执行 otherwise 语句。

if 结构的语法形式如下：

```
if logical_expression
   statements
elseif logical_expression
       statements
else   logical_expression
       statements
end
```

其中 elseif 和 else 语句都是可选语句。if、elseif 和 else 构成的各项分支里面，哪个的条件满足（逻辑表达式 logical_expression 的结果为真），就执行哪个分支后面紧跟的程序语句。因此，各个分支条件满足的情况应该是互斥的和完全的，就是被选的条件在一个分支中成立，而且只能在一个分支中成立。

当然，省略了 elseif 和 else 分支的语句，就不必要求分支条件满足的情况具备完全性了。

if 结构中条件判断除了可以用逻辑表达式外，还可以用数组 A，这时判断相当于逻辑表达式 all(A)，即当数组 A 的所有元素都为非零值时，才执行该条件后的分支代码。

特别地，当数组 A 为空数组时，相当于该条件判断为假。

switch 结构的语法形式是：

```
switch expression (scalar or string)
   case value1
      statements      % Executes if expression is value1
   case value2
      statements      % Executes if expression is value2
    ...
   otherwise
      statements      % Executes if expression does not
                     % match any case
end
```

执行中，先计算表达式 expression 的值，当结果等于某个 case 的 value 时，就执行该 case 紧随的语句。如果所有 case 的 value 都和 expression 计算结果不相等，则执行 otherwise 后面紧随的语句。

otherwise 语句是可选的，当没有 otherwise 语句时，如果所有 case 的 value 都和 expression 计算结果不相等，则跳过 switch-case 语句段，直接执行后续代码。

相等的意义，对于数值类型来说，相当于判断“if result==value”，对于字符串类型来说，相当于判断“if strcmp(result,value)”。

由此可见，switch-case 语句实际上可以被 if-elseif-else 语句等效替换，不过两种结构各有更便利的地方，读者在以后的例子中会逐渐体会到。

学过 C 语言的读者需要注意，MATLAB 中的 switch-case 结构，只执行表达式结果匹配的第一个 case 分支，然后就跳出 switch-case 结构。因此，在每一个 case 语句中不需要用 break 语句跳出。

在一条 case 语句后可以列举多个值，只需要以元胞数组的形式列举多个值，就是用花括号把用逗号或空格分隔的多个值括起来即可。

2. 程序循环控制语句

循环控制语句能够使得某段程序代码多次重复执行，MATLAB 中提供了两类循环语句，分别是 for 循环和 while 循环：

- for 循环一般用在已知循环执行次数的情况；
- while 循环则用在已知循环退出条件的情况。

MATLAB 还提供了 continue 和 break 语句，用于更精细的控制循环结构：

- continue 语句使得当前次循环不向下执行，直接进入下一次循环；

- break 语句则是退出该循环。

（1）for 循环

for 循环用于知道循环次数的情况，其语法格式为：

```
for index = start:increment:end
    statements
end
```

index 为循环变量，increment 为增量，end 用于判断循环是否应该终止。增量 increment 默认值为 1，可以自由设置；当增量为正数时，循环开始先将 index 赋值为 start，然后判断 index 是否小于等于 end。若是，则执行循环语句，执行完后，对 index 累加一个增量 increment；再判断 index 是否小于等于 end，若是，则继续执行循环语句，继续对 index 累加，循环往复，直到 index 大于 end 时退出循环。

增量 increment 也可以设置为负整数，表示每次循环执行后对循环变量 index 进行递减，当 index 小于 end 时，退出循环。

MATLAB 中，循环的执行效率很低，提高程序效率的一个办法就是多采用数组结构和 MATLAB 内联函数。

for 循环中的循环变量 index 也可以赋值为数组 A，那么在第一次循环中 index 就被赋值为 A(:,1)，即 A 的第一列元素，第二次循环 index 被赋值为 A(:,2)，以此类推；若 A 有 n 列元素，则循环执行 n 次，第 n 次循环时，循环变量 index 被赋值为 A(:,n)。

（2）while 循环

while 循环用于已知循环退出条件的情况，其语法形式如下：

```
while expression
    statements
end
```

当表达式 expression 的结果为真时，就执行循环语句，直到表达式 expression 的结果为假，才退出循环。

如果表达式 expression 是一个数组 A，则相当于判断 all(A)。特别地，空数组则被当作逻辑假，循环不执行。

（3）continue 语句

continue 语句用在循环中，表示当前次循环不再继续向下执行，而是直接对循环变量进行递增，进入下一次循环。

（4）break 语句

break 语句用于退出循环。

3. 程序终止的 return 语句

一般程序代码都是按流程执行完毕后正常退出，但当遇到某些特殊情况，程序需要立即退出时，就可以用 return 语句提前终止程序运行。

【例 A-3】 return 语句

```
clear
```

```
clc
n=-2;
if n<0
    disp('negative number!');
    return;
end
disp('codon after return')
```

【例 A-3】中当变量 *n* 取值为负数时，通过 return 直接退出，不执行 if 后的代码。其运行结果是：

```
negative number!
```

若去掉其中的 return 语句，则运行结果变为：

```
negative number!
codon after return
```

return 语句更多地用在 MATLAB 函数 M 文件中。

A.3.5 MATLAB 中的函数及调用

1. 函数分类

MATLAB 中的函数可以分为：匿名函数、M 文件主函数、嵌套函数、子函数、私有函数和重载函数。

（1）匿名函数

匿名函数通常是很简单的函数，它是面向命令行代码的函数，通常只由一句很简单的声明语句组成。

匿名函数也可以接受多个输入和输出参数。使用匿名函数的优点是不需要维护一个 M 文件，而只需要一句非常简单的语句，就可以在命令窗口或 M 文件中调用函数，这对于那些函数内容非常简单的情况是很方便的。

创建匿名函数的标准格式是：

```
fhandle = @(arglist) expr
```

其中，

1）“expr”通常是一个简单的 MATLAB 变量表达式，实现函数的功能，比如 *x*+*x*.^2，sin(*x*).*cos(*x*)等；

2）“arglist”是参数列表，它指定函数的输入参数列表，对于多个输入参数的情况，通常要用逗号分隔各个参数；

3）符号“@”是 MATLAB 中创建函数句柄的操作符，表示对由输入参数列表 arglist 和表达式 expr 确定的函数创建句柄，并把这个函数句柄返回给变量 fhandle，这样，以后就可以通过 fhandle 来调用定义好的这个函数。

例如定义函数：

```
myfunhd=@(x)(x+x..^2)
```

表示创建了一个匿名函数，它有一个输入参数 *x*，它实现的功能是 *x*+*x*.^2，并把这个函

数句柄保存在变量“myfunhd”中，以后就可以通过“myfunhd(*a*)”来计算当“*x*=*a*”的时候的函数值。

需要注意的是，匿名函数的参数列表 arglist 中可以包含一个参数或多个参数，这样调用的时候就要按顺序给出这些参数的实际取值。

但 arglist 也可以不包含参数，即留空，这种情况下调用函数时还是需要通过 fhandle() 的形式来调用，即要在函数句柄后紧跟一个空的括号。否则，只显示 fhandle 句柄对应的函数形式。

匿名函数可以嵌套，即在 expr 表达式中可以用函数来调用一个匿名函数句柄。

【例 A-4】　匿名函数创建实例。

```
>> myfhd1=@(x)(x+x.^2)
myfhd1 =
    @(x)(x+x.^2)
>> myfhd1(2)
ans =
     6
>> myfhd2=@(x,y)(sin(x)+cos(y))
myfhd2 =
    @(x,y)(sin(x)+cos(y))
>> myfhd2(pi/2,pi/6)
ans =
    1.8660
>> myfhd3=@()(3+2)
myfhd3 =
    @()(3+2)
>> myfhd3()
ans =
     5
>> myfhd3
myfhd3 =
    @()(3+2)
>> myffhd=@(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1)) %匿名函数嵌套使用
myffhd =
    @(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1))
>> myffhd(0.5)
ans =
    5.1667
```

匿名函数可以保存在.mat 文件中，上例中就可以通过“save myfunquad.mat myffhd”把匿名函数句柄“myffhd”保存在“myfunquad.mat”文件中，以后需要用到匿名函数“myffhd”时，只需要使用语句“load myfunquad.mat myffhd”就可以了。

（2）M 文件主函数

每一个函数 M 文件第一行定义的函数就是 M 文件主函数，一个 M 文件只能包含一个主函数，并通常习惯上将 M 文件名和 M 文件主函数名设为一致。

M 文件主函数的说法是针对其内部嵌套函数和子函数而言的。一个 M 文件中除了一个主函数外，还可以编写多个嵌套函数或子函数，以便在主函数功能实现中进行调用。

（3）嵌套函数

在一个函数内部，可以定义一个或多个函数，这种定义在其他函数内部的函数就被称

为嵌套函数。嵌套可以多层发生，就是说一个函数内部可以嵌套多个函数，这些嵌套函数内部又可以继续嵌套其他函数。

嵌套函数的书写语法格式是：

```
function x = A(p1, p2)
...
   function y = B(p3)
   ...
   end
...
end
```

一般函数代码中结尾是不需要专门标明“end”的，但在使用嵌套函数时，无论嵌套函数还是嵌套函数的父函数（直接上一层次的函数）都要明确标出“end”表示函数结束。

嵌套函数的互相调用需要注意和嵌套的层次密切相关，如下面一段代码中：

```
function A(x, y)     %外层函数 A（例如主函数）
B(x, y);
D(y);

   function B(x, y) %A 的嵌套函数（以 A 为参照可以称为第一层嵌套函数），B 的父函数为 A
   C(x);
   D(y);

      function C(x)  %B 的嵌套函数（以 A 为参照可以称为第二层嵌套函数），C 的父函数为 B
      D(x);
      end
   end

   function D(x)     %A 的嵌套函数（以 A 为参照可以称为第一层嵌套函数），D 的父函数为 A
   E(x);

      function E(x)  %D 的嵌套函数（以 A 为参照可以称为第二层嵌套函数），E 的父函数为 D
      ...
      end
   end
end
```

1）外层的函数可以调用向内一层直接嵌套的函数（A 可以调用 B 和 D），而不能调用更深层的嵌套函数（A 不可以调用 C 或 E）；

2）嵌套函数可以调用与自己具有相同父函数的其他同层嵌套函数（B 和 D 可以互相调用）；

3）嵌套函数也可以调用其父函数或与父函数具有相同父函数的其他嵌套函数（C 可以调用 B 和 D），但不能调用与其父函数具有相同父函数的其他嵌套函数内深层嵌套的函数。

（4）子函数

一个 M 文件只能包含一个主函数，但一个 M 文件中可以包含多个函数，这些编写在主函数后的函数都称为子函数。所有子函数只能被其所在 M 文件中的主函数或其他子函数调用。

所有子函数都有自己独立的声明和帮助、注释等结构，只需要在位置上处在主函数之

后即可，而各个子函数的前后顺序都可以任意放置，和被调用的前后顺序无关。

M 文件内部发生函数调用时，MATLAB 首先检查该 M 文件中是否存在相应名称的子函数，然后检查这一 M 文件所在目录的子目录下是否存在同名的私有函数，然后按照 MATLAB 路径检查是否存在同名的 M 文件或内部函数。

根据这一顺序，函数调用时首先查找相应的子函数，因此，可以通过编写同名子函数的方法实现 M 文件内部的函数重载。

子函数的帮助文件也可以通过 help 命令显示，如 myfun.m 文件中有名为 myfun 的主函数和名为 mysubfun 的子函数，那么可以通过 help myfun>mysubfun 来获取子函数 mysubfun 的帮助。

（5）私有函数

私有函数是具有限制性访问权限的函数，它们对应的 M 文件需要保存在名为“private”的文件夹下，这些私有函数代码编写上和普通的函数没有什么区别，也可以在一个 M 文件中编写一个主函数和多个子函数，以及嵌套函数。

但私有函数只能被 private 目录的直接父目录下的脚本 M 文件或 M 文件主函数调用。

通过 help 命令获取私有函数的帮助，也需要声明其私有特点，例如要获取私有函数 myprifun 的帮助，就要通过 help private/myprivfun 命令。

（6）重载函数

“重载”是计算机编程中非常重要的概念，它经常地用在处理功能类似、但参数类型或个数不同的函数编写中。

例如现在要实现一个计算功能，一种情况下输入的几个参数都是双精度浮点类型，同时也有一种情况是，输入的几个参数都是整型变量，这时候，用户就可以编写两个同名函数，一个用来处理双精度浮点类型的输入参数，另一个用来处理整型的输入参数，这样，当用户实际调用函数时，MATLAB 就会根据实际传递的变量类型选择执行其中一个函数。

MATLAB 中重载函数通常放置在不同的文件夹下，通常文件夹名称以符号@开头，然后跟一个代表 MATLAB 数据类型的字符。

例如“@double”目录下的重载函数的输入参数应该是双精度浮点型，而“@int32”目录下的重载函数的输入参数应该是 32 位整型。

2. 函数参数传递

MATLAB 中通过 M 文件编写函数时，只需要指定输入和输出的形式参数列表，只是在函数实际被调用的时候，才需要把具体的数值提供给函数声明中给出的输入参数。

MATLAB 中参数传递过程是传值传递，也就是说，在函数调用过程中，MATLAB 将传入的实际变量值赋给形式参数指定的变量名，这些变量都存储在函数的变量空间中，这和工作区变量空间是独立的，每一个函数在调用中都有自己独立的函数空间。

例如编写函数：

```
function y=myfun(x,y,z)
```

在命令窗口通过 a = myfun(3, 2, 0.5)调用此函数，那么 MATLAB 首先会建立 myfun 函

数的变量空间，把 3 赋值给 x，把 2 赋值给 y，把 0.5 赋值给 z，然后执行函数实现的代码；在执行完毕后，把 myfun 函数返回的参数 y 的值传递给工作区变量 a，调用过程结束后，函数变量空间被清除。

（1）输入和输出参数的数目

MATLAB 的函数可以具有多个输入或输出参数。通常在调用时，需要给出和函数声明语句中一一对应的输入参数；而输出参数个数可以按参数列表对应指定，也可以不指定。不指定输出参数调用函数时，MATLAB 默认地把输出参数列表中的第一个参数的值返回给工作区变量“ans”。

MATLAB 中可以通过 nargin 和 nargout 函数确定函数调用时实际传递的输入和输出参数个数，结合条件分支语句，就可以处理函数调用中指定不同数目的输入输出参数的情况。

【例 A-5】 输入和输出参数的数目。

```
function [y1,y2]=mytestnio(x1,x2)
if nargin==1
    y1=x1;
    if nargout==2
        y2=x1;
    end
else
    if nargout==1
        y1=x1+x2;
    else
        y1=x1;
        y2=x2;
    end
end
```

这个函数可以处理 1 个或 2 个输入参数、1 个或两个输出参数的情况。当只有一个输入参数 $x1$ 和一个输出参数 $y1$ 时，把 $x1$ 赋值给 $y1$；当有 1 个输入参数 $x1$ 和两个输出参数 $y1$、$y2$ 时，把 $x1$ 赋值给 $y1$ 和 $y2$；当有两个输入参数 $x1$、$x2$ 和一个输出参数 $y1$ 时，把 $x1+x2$ 的计算结果赋值给 $y1$；当有两个输入参数 $x1$、$x2$ 和两个输出参数 $y1$，$y2$ 时，把 $x1$ 赋值给 $y1$、把 $x2$ 赋值给 $y2$。函数调用结果如下：

```
>> x=mytestnio(5)
x =
     5
>> [x,y]=mytestnio(5)
x =
     5
y =
     5
>> mytestnio(5)
ans =
     5
>> x=mytestnio(5,7)
x =
    12
>> [x,y]=mytestnio(5,7)
x =
```

```
     5
y =
     7
>> mytestnio(5,7)
ans =
     5
```

指定了输入和输出参数个数的情况比较好理解，只要对应函数 M 文件中对应的 if 分支项即可；而不指定输出参数个数的调用情况，MATLAB 是按照指定了所有输出参数的调用格式对函数进行调用的，不过在输出时只是把第一个输出参数对应的变量值赋给工作区变量 ans。

例如“mytestnio(5,7)”这句函数调用中，实际上是按照“[$y1$, $y2$]=mytestnio($x1$, $x2$)”这种形式调用的，在函数变量空间中 $x1$ 被赋值为 5，$x2$ 被赋值为 7，$y1$ 计算结果为 5，$y2$ 计算结果为 7，但函数只把输出参数列表中的第一个输出变量（即 $y1$）的取值返回给工作区变量 ans，因此，ans 取值为 5。

（2）可变数目的参数传递

函数 nargin 和 nargout 结合条件分支语句，可以处理可能具有不同数目的输入和输出参数的函数调用，但这要求对每一种输入参数数目和输出参数数目的组合分别进行代码编写。

有些情况下，用户可能并不能确定具体调用中传递的输入参数或输出参数的个数，即具有可变数目的传递参数，MATLAB 中可以通过 varargin 和 varargout 函数实现可变数目的参数传递，使用这两个函数对于处理具有复杂的输入输出参数个数组合的情况也是便利的。

函数 varargin 和 varargout 把实际的函数调用时传递的参数值封装成一个元胞数组，因此，在函数实现部分的代码编写中，就要用访问元胞数组的方法访问封装在 varargin 或 varargout 中的元胞或元胞内的变量。

【例 A-6】 可变数目的参数传递。

```
function y=mytestvario(varargin)
temp=0;
for i=1:length(varargin)
    temp=temp+mean(varargin{i}(:));
end
y=temp/length(varargin);
```

【例 A-6】的函数 mytestvario 以 varargin 为输入参数，从而可以接受可变数目的输入参数。函数实现部分首先计算了各个输入参数（可能是标量、一维数组或二维数组）的均值，然后计算这些均值的均值。调用结果如下：

```
>> mytestvario(4)
ans =
     4
>> mytestvario(4,[1 3])
ans =
     3
>> mytestvario(4,[1 3],[1 23;23 1],magic(4))
ans =
    6.6250
```

对于“mytestvario(4,[1 3],[1 23;23 1],magic(4))”这句函数调用，在函数变量区，varargin首先被赋值为一个元胞数组“{4,[1 3],[1 23;23 1],magic(4)}”，即 varargin 有 1 行 4 列个元胞，各个元胞中分别存储了一个标量数值、一维行数组、2 行 2 列的二维数组和 4 行 4 列的魔方数组；在函数实现部分，首先创建中间变量 temp，并初始化赋值为零（用来存储各个元胞中数据均值的总和），然后计算每一个元胞中所有数据的均值并将结果累加到 temp 上；最后通过“y=temp/length(varargin)”计算这些均值的均值。

函数 varargin 和 varargout 也可以放置在参数列表中某些必然出现的参数之后，其语法格式可以如下：

1）function [out1, out2] = example1(*a*, *b*, varargin)，表示函数 example1 可以接受大于等于两个输入参数，返回两个输出参数；两个必选的输入参数是 *a* 和 *b*，其他更多的输入参数被封装在 varargin 中。

2）function [i, j, varargout] = example2(*x*, *y*)，表示函数 example2 接受两个输入参数 *x* 和 *y*，返回大于等于两个输出参数，前两个输出参数为 *i* 和 *j*，其他更多的输出参数封装在 varargin 中。

函数 varargout 和 varargin 的用法类似，只需要注意访问时应按照访问元胞数组的方法，这里就不再举例了。

（3）返回被修改的输入参数

MATLAB 函数有独立于 MATLAB 工作区的自己的变量空间，因此，输入参数在函数内部的修改都只具有和函数变量空间相同的生命期，如果不指定将修改后的输入参数值返回到工作区空间，那么在函数调用结束后这些修改后的值将被自动清除。

【例 A-7】 函数内部的输入参数修改。

```
function y=mytest(x)
x=x+5;
y=x*2;
```

【例 A-7】中的 mytest 函数内部，首先修改了输入参数 *x* 的值（*x*=*x*+5），然后以修改后的 *x* 的值计算输出参数 *y* 的值（*y*=*x**2）。调用结果如下：

```
>> x=3
x =
     3
>> y=mytest(x)
y =
    16
>> x
x =
     3
```

由此结果可见，调用结束后，函数变量区中的 *x* 在函数调用中被修改，但此修改只在函数变量区有效，这并没有影响到 MATLAB 工作区变量空间中的变量 *x* 的值。函数调用前后，MATLAB 工作区中的变量 *x* 始终取值为 3。

那么，如果用户希望函数内部对输入参数的修改也对 MATLAB 工作区的变量有效，就需要在函数输出参数列表中返回此输入参数。

对于【例 A-7】的函数，则需要把函数修改为“function [y,x]=mytest(x)”，而在调用时也要通过“[y,x]= mytest(x)”这种形式。

【例 A-8】　将修改后的输入参数返回给 MATLAB 工作区。

```
function [y,x]=mynewtest(x)
x=x+5;
y=x*2;
```

MATLAB 工作区中的调用结果如下：

```
>> x=3
x =
     3
>> [y,x]=mynewtest(x)
y =
    16
x =
     8
>> x
x =
     8
```

通过【例 A-8】可见，函数调用后，MATLAB 工作区中的变量 x 取值从 3 变为 8（3+5），可见通过[y,x]=mynewtest(x)调用，实现了函数对 MATLAB 工作区变量的修改。

（4）全局变量

通过返回修改后的输入参数，可以实现函数内部对 MATLAB 工作区变量的修改。而另一种殊途同归的方法则是使用全局变量。声明全局变量需要用到 global 关键词，语法格式为“global variable”。

通过全局变量可以实现 MATLAB 工作区变量空间和多个函数的函数空间的共享，这样，多个使用全局变量的函数和 MATLAB 工作区共同维护这一全局变量，任何一处对全局变量的修改，都会直接改变此全局变量的取值。

在应用全局变量时，通常在各个函数内部通过 global variable 语句声明，在命令窗口或脚本 M 文件中也要先通过 global 声明，然后进行赋值和调用。

【例 A-9】　全局变量的使用。

```
function y=myprocess(x)
global T
T=T*2;
y=exp(T)*sin(x);
```

在命令窗口中声明全局变量然后赋值调用：

```
>> global T
>> T=0.3
T =
    0.3000
>> myprocess(pi/2)
ans =
    1.8221
>> exp(T)*sin(pi/2)
ans =
```

```
    1.8221
>> T
T =
    0.6000
```

通过【例 A-9】可见，用 global 将 T 声明为全局变量后，函数内部对 T 的修改也会直接作用到 MATLAB 工作区中。函数 myprocess 调用一次后，T 的值从 0.3 变为 0.6（0.3*2）。

A.3.6 程序调试

1. 调试方法

MATLAB 程序有直接调试法和工具调试法这两种调试方法。

（1）直接调试法

直接调试法就是在 M 文件中，将某些语句后面的分号去掉，迫使 M 文件输出一些中间计算结构，以便发现可能的错误。常用的做法有：

1）在适当位置，添加显示某些关键变量值的语句；

2）利用 echo 指令，使运行时在屏幕上逐行显示文件内容，echo on 能显示 M 脚本文件；echo FunName On 能显示名为 FunName 的 M 函数文件。

3）在原 M 脚本或函数文件的适当位置，添加指令 keyboard，keyboard 语句可以设置程序的断点；

4）通过将原 M 函数文件的函数声明行注释掉，可使一个中间变量难于观察的 M 函数文件变为一个所有变量都保存在基本工作空间中的 M 脚本文件。

（2）工具调试法

工具调试法就是在程序中设置一些断点，利用调试菜单（Debug）中的一些选项进行调试。

Debug 菜单用于程序调试，需要与 Breakpoints 菜单项配合使用。MATLAB 7 的 Debug 菜单中的菜单项有：

- Open M-Files when Debugging：用于调试时打开 M 文件；
- Step：用于单步调试程序；
- Step in：用于单步调试进入子函数；
- Step Out：用于单步调试从子函数跳出；
- Continue：程序执行到下一断点；
- Clear Breakpoints in All Files：清除所有打开文件中的断点；
- Stop if Errors/Warnings：在程序出错或报警处停止往下执行；
- Exit Debug Mode：退出调试模式。

除了采用调试器调试程序外，MATLAB 还提供了一些命令用于程序调试。命令的功能和调试器菜单命令类似。MATLAB 提供的调试命令有：

- 快捷键 F10：实现单步调试；
- 快捷键 F11：用于单步调试进入子函数；

- 快捷键 Shift+F11：用于单步调试从子函数跳出；
- 快捷键 F5：实现程序执行到下一断点。

MATLAB 提供了进行代码调试和代码分析优化的工具，这些工具，一般的 MATLAB 用户都应该有所了解。尤其是断点调试部分的内容，建议读者尽量以自己的程序代码为例，多加练习，熟练掌握。

2. 调试工具

当完成 MATLAB 代码编写后，用户就可以在命令窗口中运行代码（脚本或函数文件）。对于比较简单的代码，一般只要编程习惯较好，都可以一次通过。但对于很多比较复杂的情况，或者用户初学 MATLAB 编程，一些常见的错误还不能避免，就容易在运行时出现错误。这时候，就需要利用 MATLAB 的调试工具对出现错误的代码进行调试纠错。

MATLAB 的代码编辑调试器是一个综合了代码编写、调试的集成开发环境。MATLAB 代码调试过程，主要是通过 MATLAB 代码编辑-器的 Debug 菜单下的子项进行的，如图 A-15 所示。

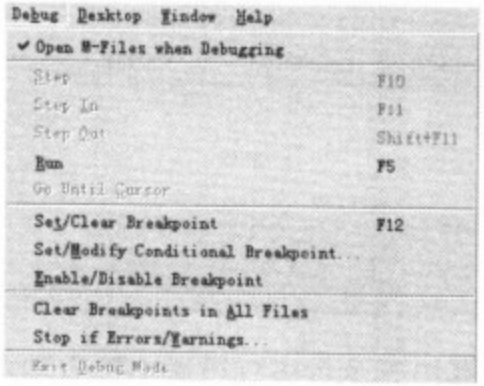

图 A-15　MATLAB 代码编辑-调试器的 Debug 菜单

Debug 菜单下各子项的含义如下：

1）Step：在调试模式下，执行 M 文件的当前行，对应的快捷键是 F10。

2）Step In：在调试模式下，执行 M 文件的当前行，如果 M 文件当前行调用了另一个函数，那么进入该函数内部，对应的快捷键是 F11。

3）Step Out：当在调试模式下执行 Step In 进入某个函数内部之后，执行 Step Out 可以完成函数剩余部分的所有代码，并退出函数，暂停在进入函数内部前的 M 文件所在行末尾。

4）Run：运行当前 M 文件，快捷键是 F5；当前 M 文件设置了断点时，运行到断点处暂停。

5）Go Until Cursor：运行当前 M 文件到在光标所在行尾。

需要注意，以上这些调试项，除了 Run（运行），都需要首先在 M 文件中设置断点，然后运行到断点位置后，这些调试项才可启用。

6）Set/Clear Breakpoint：在光标所在行开头设置或清除断点。

7）Set/Modify Conditional Breakpoint...：在光标所在行开头设置或修改条件断点，选择此子项，会打开“条件断点设置”对话框，如图 A-16 所示，用于设置在满足什么条件时，此处断点有效。

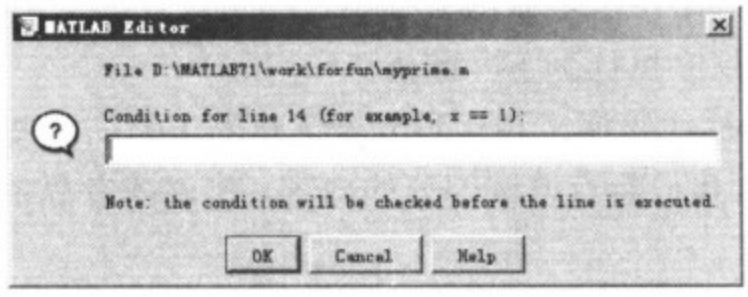

图 A-16 “条件断点设置”对话框

8）Enable/Disable Breakpoint：将当前行的断点设置为有效或无效。

9）Clear Breakpoints in All Files：清除所有 M 文件中的断点。

10）Stop if Errors/Warnings...：设置出现某种运行错误或警告时，停止程序运行，选择此子项，会打开“错误/警告设置”对话框，如图 A-17。

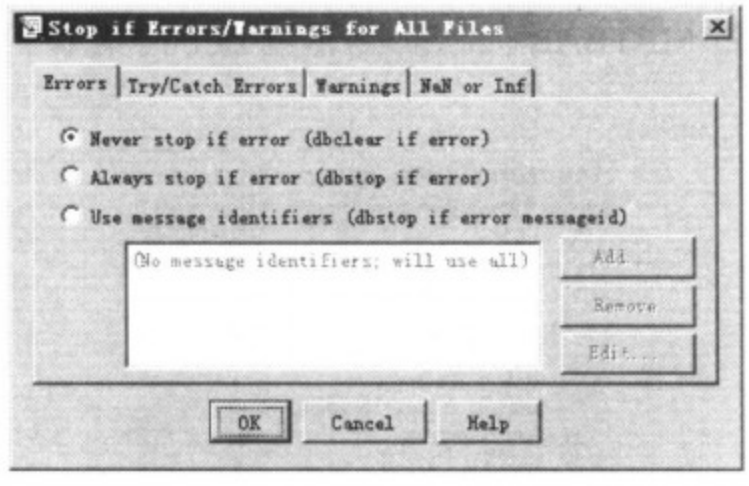

图 A-17 设置出现某种运行错误或警告则停止程序运行

11）Exit Debug Mode：退出调试模式。

上面逐项讲述了 Debug 菜单下每一个子项的意义，实际上，很多子项都有对应的快捷工具按钮。MATLAB 代码编辑-调试器中，如图 A-18 所示的部分工具按钮就是用于 M 文件调试的。



图 A-18 调试工具按钮

图 A-18 中的各个工具按钮，从左向右依次对应于 Set/Clear Breakpoint、Clear Breakpoints in All Files、Step、Step In、Step Out、Run、Exit Debug Mode 等菜单子项。

通常的调试过程是：先运行（Run）一遍 M 文件，针对具体的出错信息，在适当的地方设置断点或条件断点；再次运行（Run）到断点位置（如图 A-19 所示），此时 MATLAB 把运行控制权交给键盘，命令窗口出现“K>>”提示符（如图 A-20 所示）；此时可以在命令窗口中查询 M 文件运行过程中的所有变量，包括函数运行时的中间变量。运行到断点位置后，用户可以选择“Step/Step Into/Step Out”等调试运行方式，逐行运行并适时查询变量取值，从而逐渐找到错误所在并排除。

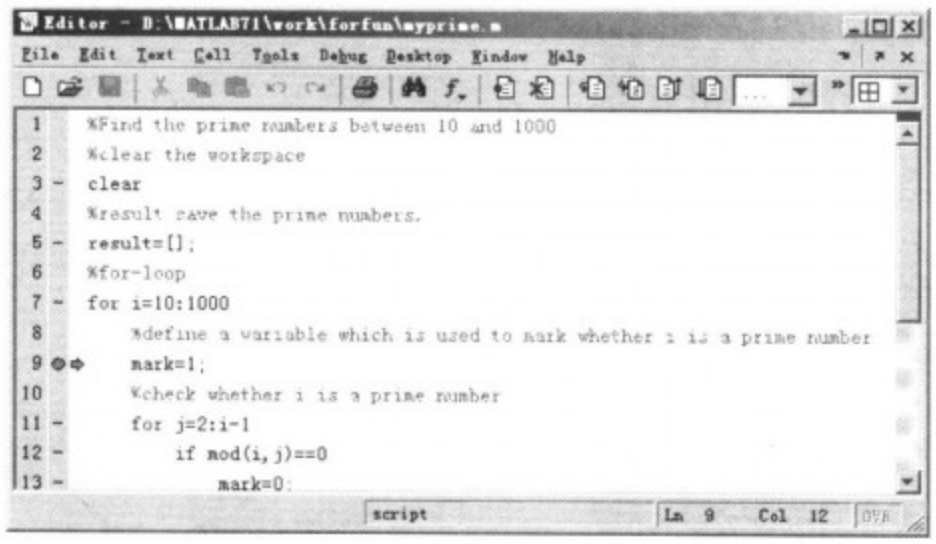

图 A-19　设置断点后运行（Run）到断点所在位置

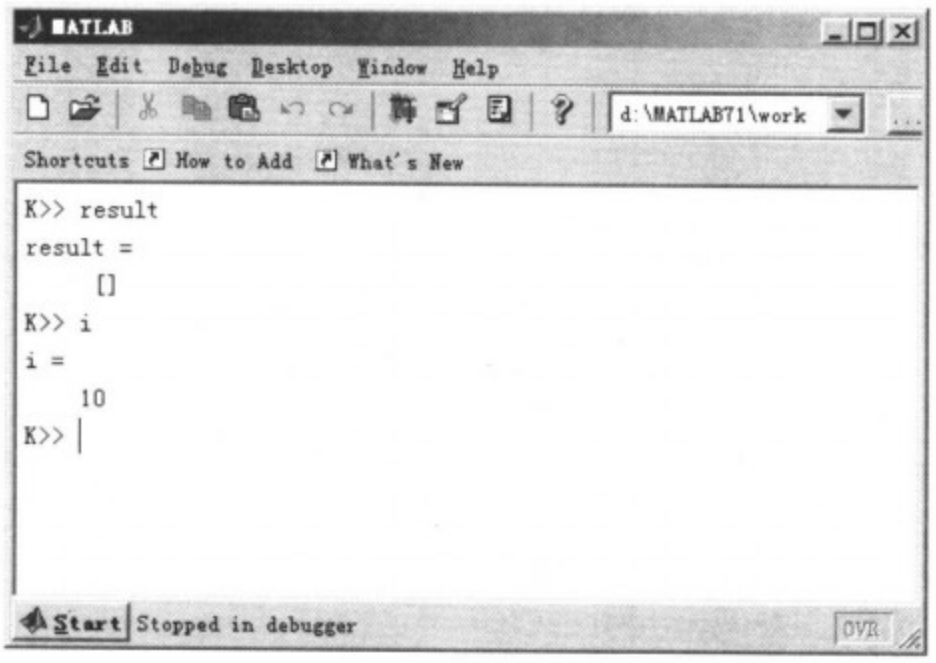

图 A-20　调试模式时 MATLAB 命令窗口把控制权交给键盘

3. M 文件分析工具

通过 Debug 项对 M 文件进行调试的过程，可以对文件中的编写错误和运行错误进行纠正。完成了调试过程后，用户编写的 M 文件就可以正确地运行了。但可能运行效率还不是最优，这就需要通过 MATLAB 提供的分析工具对代码进行分析，然后针对性地进行优化。

MATLAB 提供的 M 文件分析工具包括 M-Lint 工具和 Profiler 工具，他们都有图形操作界面，使用简单，方便，是 MATLAB 程序分析优化的必用工具。

（1）M-Lint 分析工具

M-Lint 工具可以分析用户 M 文件中的错误或性能问题。用户可以先在代码编辑—调试器中打开待分析的 M 文件，然后选择 Tools 菜单下的 Check Code with M-Lint 项，如图 A-21 所示。

图 A-21 所示中的代码是查找 10-1000 之内所有素数的 M 文件，它的脚本如下。

```
%Find the prime numbers between 10 and 1000
%clear the workspace
```

```
clear
%result save the prime numbers.
result=[];
%for-loop
for i=10:1000
    %define a variable which is used to mark whether i is a prime number
    mark=1;
    %check whether i is a prime number
    for j=2:i-1
        if mod(i,j)==0
            mark=0;
            break
        end
    end
    %add prime number to result
    if mark==1
        result=[result i];
    end
end
%output
result
```

运行 M-Lint 工具后结果如图 A-22 所示。



图 A-21　通过 Tools 菜单打开 M-Lint 工具

从图 A-22 可以看出，M-Lint 分析完成后，会返回一个浏览器界面下的分析报告（Check Report），报告中包括被分析的 M 文件的路径，以及若干个分析结果（如图 A-22 所示中的 1 message 表示只有一条分析结果）。分析结果的格式是“行号：错误或问题报告”。

M-Lint 分析结果中经常出现的错误或问题报告包括：没有以分号结束以阻止中间变量输出，变量在文件中从没有被其他语句调用，以及循环过程中数组尺寸会增加等。

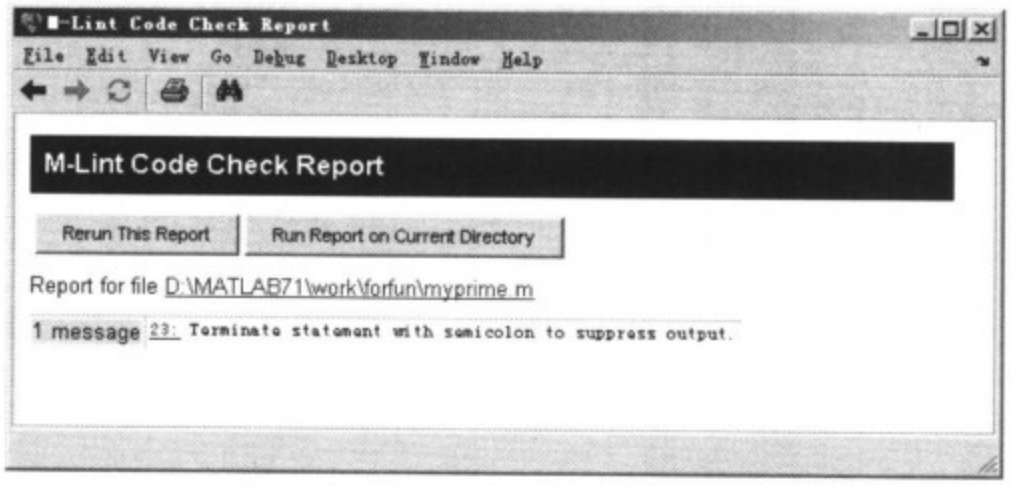

图 A-22　M-Lint 分析结果

实际上，M-Lint 分析得到的问题报告，并不一定必须消除该问题，而是要具体问题具体分析。当用户认可某一条分析结果时，可以点击分析结果中的行号，就可以快捷打开相应的 M 文件并定位到该行，用户就可以方便地修改代码了。

M-Lint 不仅可以分析单个 M 文件，还可以分析一个文件夹下的所有 M 文件。通常在 MATLAB 主界面下，选择“Desktop”菜单下的“Current Directory”，则可以显示文件夹面板，通过单击此面板顶部的“M-Lint”工具则可以分析相应文件夹下所有 M 文件，如图 A-23 所示。

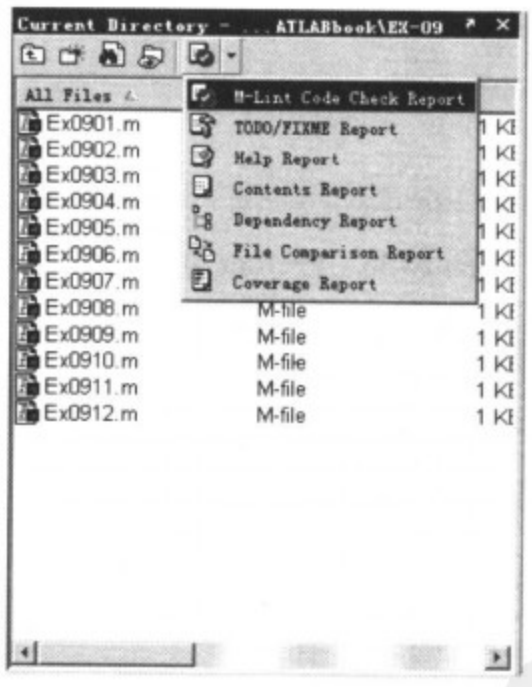

图 A-23　当前目录面板和 M-Lint 按钮

2. Profiler 分析工具

Profiler 工具是 MATLAB 提供的另一个功能强大的代码分析工具。使用时，用户可以提前在代码编辑调试器中打开 M 文件，然后选择“Tools”菜单的“Open Profiler”项，就可以运行 Profiler 分析工具，Profiler 图形界面如图 A-24 所示。

单击图 A-24 中所示的“Start Profiling”按钮，就可以分析此 M 文件，分析结果如图 A-25 所示。

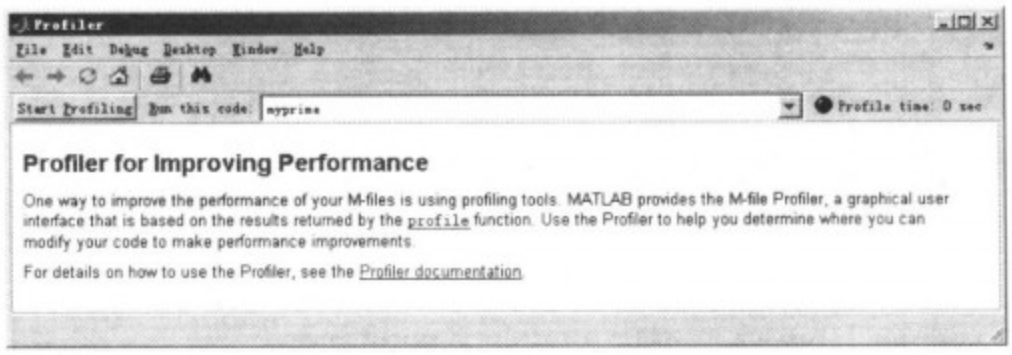

图 A-24　Profiler 工具图形界面

从图 A-25 可见，Profiler 分析结果给出了调用函数名称、调用次数、消耗总时间等信息。单击图 A-25 中蓝色的“myprime”，可以打开更加详细的关于此 M 文件的分析报告，如图 A-26 所示。

图 A-26 所示的更详细的 Profiler 分析结果中显示了 myprime.m 文件运行中最消耗时间的部分及其具体耗时信息。用户可以有针对性地修改那些最消耗时间的部分。

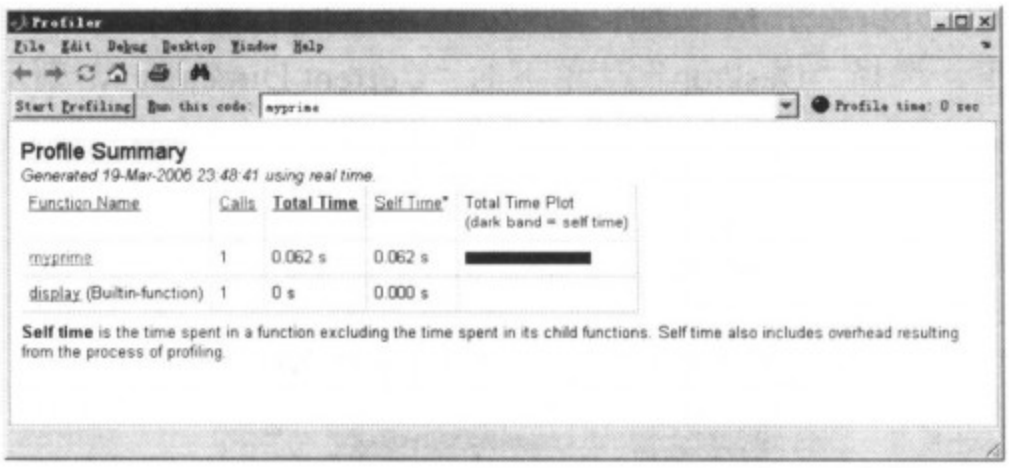

图 A-25　Profiler 分析结果

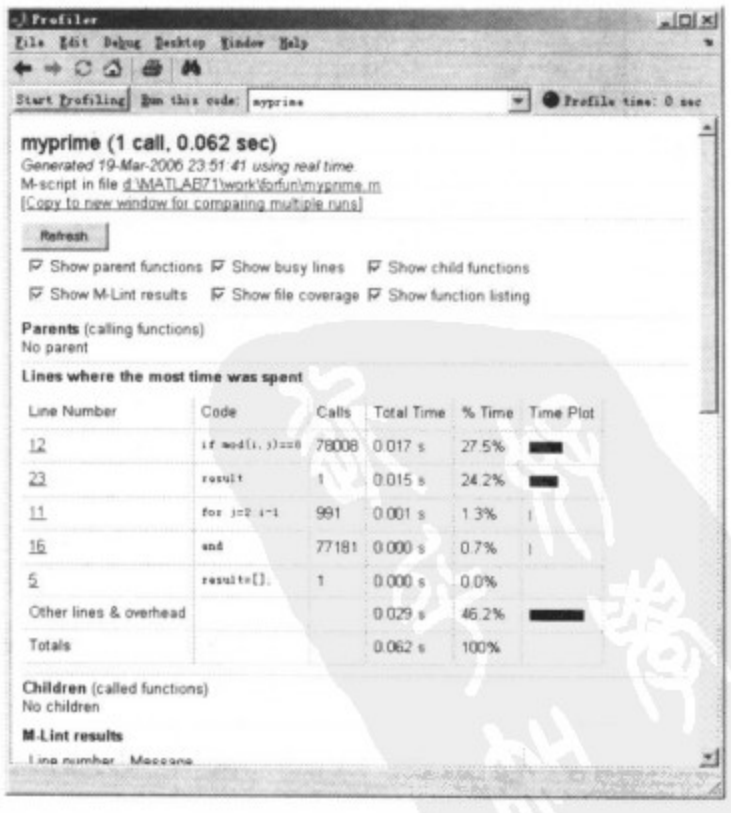

图 A-26　更详细的 Profiler 分析结果

一般来说，应该尽量避免不必要的变量输出、循环赋值前预定义数组尺寸，多采用向量化的 MATLAB 函数，少采用数组，这些都能够提高 MATLAB 程序的运行性能。

A.3.7　函数句柄

函数句柄实际上提供了一种间接调用函数的方法。创建函数句柄需要用到操作符@。前面已经讲过，匿名函数实际上就是一种函数句柄，而对 MATLAB 提供的各种 M 文件函数和内部函数，也都可以创建函数句柄，从而可以通过函数句柄对这些函数实现间接调用。

函数句柄的优点如下：

（1）方便地实现函数间互相调用；

（2）兼容函数加载的所有方式；

（3）拓宽子函数，包括局部函数的使用范围；

（4）提高函数调用的可靠性；

（5）减少程序设计中的冗余；

（6）提高重复执行的效率。

创建函数句柄的一般语法格式为：

```
fhandle=@function_fileame
```

其中，

“funciont_filename”是函数所对应的 M 文件的名称或 MATLAB 内部函数的名称；

“@”是句柄创建操作符；

“fhandle”变量保存这一函数句柄。

例如 fhandle=@sin 就创建了 MATLAB 内部函数 sin 的句柄，并将其保存在 fhandle 变量中，以后就可以通过 fhandle(x)来实现 sin(x)的功能。

通过函数句柄调用函数时，也需要指定函数的输入参数，比如可以通过 fhandle(arg1, arg2, ..., arg$n$)这样的调用格式来调用具有多个输入参数的函数。对于那些没有输入参数的函数，在使用句柄调用时，要在句柄变量后加上空的圆括号，即 fhandle()。

【例 A-10】　函数句柄的创建和调用。

```
>> fhd=@sin
fhd =
    @sin
>> x=0:0.25*pi:2*pi;
>> fhd(x)
ans =
         0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000
-0.7071   -0.0000
```

MATLAB 中提供了丰富的处理函数句柄的函数，如表 A-4 所示。

表 A-4　处理函数句柄的函数

函　数	说　明
functions(fhandle)	返回一个结构体，存储了函数的名称，函数类型（简单函数或重载函数），以及函数 M 文件的位置
func2str(fhandle)	将函数句柄转换为函数名称字符串
str2func(str)	将字符串代表的函数转换为函数句柄
save filename.mat fhandle	将函数句柄保存在.mat 文件中
load filename.mat fhandle	把.mat 文件中存储的函数句柄装载到工作区
isa(var, 'function_handle')	检测变量 var 是不是函数句柄
isequal(fhda,fhdb)	检测两个函数句柄是否对应于同一个函数
feval(fhandle)	调用函数句柄 fhandle

【例 A-11】　处理函数句柄的函数。

```
>> fhda=@exp
fhda =
    @exp
>> fhdb=@myprocess
fhdb =
    @myprocess
>> functions(fhdb)
ans =
    function: 'myprocess'
        type: 'simple'
        file: 'D:\MATLAB71\work\MATLABbook\EX-10\myprocess.m'
>> isa(fhda,'function_handle')
ans =
     1
>> isequal(fhda,fhdb)
ans =
     0
```

附录 B

本书所编写的算法程序索引

本书采用手把手的方式教读者如何编程实现相应的科学计算算法，并给出了这些算法的完整源程序，鼓励读者从一开始阅读本书就与算法编程、调试和应用联系起来。

这些程序代码在随书光盘中有序地列出，读者可方便地进行应用。同时，为了便于读者阅读、分析和学习这些算法程序，本附录给出了这些算法程序在书中的索引。

B.1 第 2 章“线性方程组求解”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
richason	里查森迭代法求解线性方程组	2.3.2	41
jacobi	雅可比迭代法求解线性方程组	2.3.3	43
gauseidel	高斯-赛德尔迭代法求解线性方程组	2.3.4	44
SOR	超松弛迭代法求解线性方程组	2.3.5	46
SSOR	对称逐次超松弛迭代法求解线性方程组	2.3.5	48
twostep	两步迭代法求解线性方程组	2.3.6	50
fastdown	最速下降法求解线性方程组	2.3.7	52
conjgrad	共轭梯度法求解线性方程组	2.3.7	54
followup	追赶法求解系数矩阵为三对角阵的线性方程组	2.4.1	58

B.2 第 3 章“数据插值与拟合”的算法程序索引

函 数 名	功 能	位于的章节	位于的页码
Language	求已知数据点的拉格朗日插值多项式	3.2	73
Atken	求已知数据点的艾特肯插值多项式	3.3	76
Newton	求已知数据点的均差形式的牛顿插值多项式	3.4	78
Newtonforward	求已知数据点的前向牛顿差分插值多项式	3.5.1	80
Newtonback	求已知数据点的后向牛顿差分插值多项式	3.5.1	81
Gauss	求已知数据点的高斯插值多项式	3.5.2	84
Hermite	求已知数据点的埃尔米特插值多项式	3.6	88
DCS	求已知数据点反差商形式的有理分式插值	3.7.1	90
Neville	求已知数据点 Neville 形式的有理分式插值	3.7.2	92
Chebyshev	函数的切比雪夫多项式逼近	3.8.1	94
Legendre	函数的勒让德多项式逼近	3.8.2	95
Pade	函数的帕德逼近	3.8.3	97
FZZ	函数的傅立叶三角逼近	3.8.4	99
DFF	离散数据的傅立叶变换	3.8.4	100
multifit	曲线的多项式拟合	3.8.5	101

B.3 第 4 章“矩阵特征值计算”的算法程序索引

函 数 名	功 能	位于的章节	位于的页码
Chapoly	特征多项式法求矩阵的所有特征值	4.4.1	109
pmethod	幂法求矩阵的主特征值及主特征向量	4.4.2	110
rpmethod	瑞利商加速幂法求矩阵主特征值及特征向量	4.4.3	112
spmethod	收缩法求矩阵的所有实特征值	4.4.4	114
ipmethod	逆幂法求矩阵按模的最小特征值及特征向量	4.4.5	116
dimethod	求矩阵离某个特定常数最近的特征值及特征向量	4.4.6	117
qrtz	QR 基本算法求矩阵的所有实特征值	4.4.7	119
hessqrtz	海森伯格矩阵的 QR 算法求矩阵的所有实特征值	4.4.7	122
rqrtz	瑞利商位移的 QR 算法求矩阵的所有实特征值	4.4.7	123
wilkqrtz	威尔金森位移的 QR 算法求矩阵的所有实特征值	4.4.7	124

B.4　第 5 章“求导与微分计算”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
MidPoint	中点公式法求函数在某点的数值导数	5.2.1	134
ThreePoint	三点公式法求函数在某点的数值导数	5.2.2	135
FivePoint	五点公式法求函数在某点的数值导数	5.2.3	136
CISimpson	辛普森法求已知函数在某点的数值导数	5.2.4	140
DISimpson	辛普森法求离散数据系列在某点的数值导数	5.2.4	141
Richason	理查森外推算法求已知函数在某点的数值导数	5.2.5	144

B.5　第 6 章“积分计算”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
CombineTraprl	复合梯形公式求函数的数值积分	6.3	148
IntSimpson	辛普森系列公式求函数的数值积分	6.4	150
NewtonCotes	牛顿-科茨系列公式求函数的数值积分	6.5	152
IntGauss	高斯公式求函数定积分的近似值	6.6.1	154
IntGaussLada	高斯-拉道公式求函数定积分的近似值	6.6.2	156
IntGaussLobato	高斯-洛巴托公式求函数定积分的近似值	6.6.3	159
DDTraprl	区间逐次分半梯形公式求函数的数值积分	6.7.1	161
DDSimpson	区间逐次分半辛普森公式求函数的数值积分	6.7.2	162
DDBuer	区间逐次分半布尔公式求函数的数值积分	6.7.3	164
Roberg	龙贝格积分法求函数的数值积分	6.8	166
SmartSimpson	自适应辛普森法求函数的数值积分	6.9	168
IntGaussLager	高斯-拉盖尔公式求无穷积分的近似值	6.11.1	170
IntGaussHermite	高斯-埃尔米特公式求无穷积分的近似值	6.11.2	172
DblTraprl	复合梯形公式求函数的二重数值积分	6.12.1	174
DblSimpson	复合辛普森公式求函数的二重数值积分	6.12.2	176

B.6　第 7 章“非线性方程求解”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
HalfInterval	二分法求方程在某个区间上的一个根	7.2.1	182
hj	黄金分割法法求方程在某个区间上的一个根	7.2.2	183
StablePoint	不动点迭代法求方程在某个区间上的一个根	7.2.3	185
AtkenStablePoint	艾肯特加速算法求方程在某个区间上的一个根	7.2.3	186
StevenStablePoint	史蒂芬森加速求方程在某个区间上的一个根	7.2.3	188

续表

函 数 名	功　　能	位于的章节	位于的页码
Secant	弦截法求方程在某个区间上的一个根	7.2.4	189
StevenSecant	史蒂芬森弦截法求方程在某个区间上的一个根	7.2.5	191
Parabola	抛物线法求方程在某个区间上的一个根	7.2.6	193
NewtonRoot	牛顿法求方程在某个区间上的一个根	7.2.7	195
SimpleNewton	简化牛顿法求方程在某个区间上的一个根	7.2.7	196
NewtonDown	牛顿下山法求方程在某个区间上的一个根	7.2.7	198
TwoStep	两步迭代法求方程在某个区间上的一个根	7.2.8	200
MultiRoot	求方程重根的迭代法	7.2.9	201
mulStablePoint	不动点迭代法求非线性方程组的一组根	7.3.1	203
mulNewton	牛顿法求非线性方程组的一组根	7.3.2	205
mulSimNewton	简化牛顿法求非线性方程组的一组根	7.3.2	205
mulDNewton	牛顿下山法求非线性方程组的一组根	7.3.3	207
mulVNewton	拟牛顿法求非线性方程组的一组根	7.3.4	209

B.7　第 8 章“常微分方程求解”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
MyEuler	欧拉法求解 y 常一阶微分方程	8.2.1	216
MyEulerpro	改进欧拉法求解一阶常微分方程	8.2.2	218
MyRunge_Kutta	四阶 Runge-Kutta 法求解常微分方程	8.3	220
MyHamming	Hamming 方法求解常微分方程	8.4.2	226

B.8　第 9 章“偏微分方程求解”的算法程序索引

函 数 名	函数功能	位于的章节	位于的页码
Helmholtz	迭代法求解常规 Helmholtz 方程	9.2.1	239
Helmholtz_Newton	求解满足牛顿边界的 Helmholtz 方程	9.2.2	243
EF_Euler	利用显式前向欧拉法求解一维抛物线型偏微分方程	9.3.1	246
IB_Euler	利用隐式后向欧拉法求解一维抛物线型偏微分方程	9.3.2	250
Grank_Nicholson	利用 Grank-Nicholso 方法求解一维抛物线型偏微分方程	9.3.3	253
TDE	利用差分法求解二维抛物线型偏微分方程	9.3.4	255
ECD_Wave	利用显式中心差分法求解一维波动方程	9.4.1	258
Wave2	利用差分法求解二维波动方程	9.4.2	261
产生基函数：fem_basis_ftn 求解系数向量：fem_coef 显示基函数图形：show_Basis	利用有限元法求解偏微分方程	9.5	265

B.9　第 12 章“最优化计算”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
Opt_Golden	利用黄金搜索算法求无约束最优化问题	12.1.1	345
Opt_Quadratic	利用二次插值法求无约束最优化问题	12.1.2	347
Opt_Nelder	利用 Nelder-Mead 算法求无约束最优化问题	12.1.3	350
Opt_Steepest	利用最速下降法求无约束最优化问题	12.1.4	354
Opt_Simu	利用模拟退火法求无约束最优化问题	12.1.6	358
genetic	利用遗传算法求无约束最优化问题	12.1.7	362

附录 C

MATLAB 科学计算常用函数注释

本书讲述的是 MATLAB 的科学计算，对于科学计算部分涉及的主要函数以及工具箱，采用附录的形式给出。通过表格的形式进行说明。

本附录给出了 MATLAB 提供的曲线拟合工具箱、样条工具箱、偏微分方程工具箱、统计工具箱和最优化工具箱中每个函数的功能，以及线性代数部分常用函数的功能，便于读者进行查询和参考。

C.1 线性代数

C.1.1 矩阵分析

函　数	功　能
cond	矩阵条件数
condeig	已知特征值求条件数
det	矩阵行列式的值
norm	矩阵或向量的范数
normest	估计矩阵的 2 范数
null	零空间
orth	正交化
rank	矩阵的秩
rcond	矩阵逆条件数估计
rref	简化矩阵为梯形形式
subspace	两个空间的夹角
trace	矩阵的迹

C.1.2　线性方程

函　数	功　能
\ 和 /	线性方程求解
chol	Cholesky 分解
cholinc	不完全的 Cholesky 分解
cond	矩阵条件数
condest	1 范条件数估计
funm	一般矩阵函数计算
inv	矩阵的逆
linsolve	带特殊控制的线性方程求解
lscov	已知协方差的最小二乘
lsqnonneg	非负线性最小二乘
lu	LU 分解
luinc	不完全的 LU 分解
pinv	伪逆
qr	QR 分解
rcond	矩阵逆条件数估计

C.1.3　特征值和奇异值

函　数	功　能
balance	改善特征值精度的对角变换
cdf2rdf	复对角块变为实对角块
condeig	已知特征值求条件数
eig	特征值和特征向量
eigs	特征值
gsvd	广义奇异值分解
hess	Hessenberg 型
poly	特征多项式
polyeig	多项式特征值问题
qz	广义特征值的 QZ 分解
rsf2csf	实对角块变为复对角块
schur	Schur 分解
svd	奇异值分解
svds	稀疏矩阵的奇异值和向量

C.1.4 矩阵对数和指数

函　数	功　能
expm	矩阵指数
logm	矩阵对数
sqrtm	矩阵平方根

C.1.5 矩阵分解功能函数

函　数	功　能
balance	改善特征值精度的对角变换
cdf2rdf	复对角块变为实对角块
chol	Cholesky 分解
cholinc	不完全的 Cholesky 分解
cholupdate	Cholesky 分解的秩 1 修正
lu	LU 分解
luinc	不完全的 LU 分解
planerot	Givens 平面旋转
qr	QR 分解
qrdelete	从 QR 分解出发消去列
qrinsert	从 QR 分解出发插入列
qrupdate	QR 分解的秩 1 修正
qz	广义特征值的 QZ 分解
rsf2csf	实对角块变为复对角块

C.2 曲线拟合工具箱函数

C.2.1 数据拟合

函　数	功　能
cfit	创建一个曲线拟合对象
fit	用库中已有的模型（光滑样条模型或内插模型）或自定义模型来拟合数据点
fitoptions	创建或修改一个拟合属性对象
fittype	创建一个曲线拟合类型（或函数）对象

C.2.2 信息获取与帮助

函　数	功　能
cflibhelp	显示库中已有的曲线拟合模型，即光滑样条模型或内插模型的相关帮助信息
disp	显示曲线拟合工具箱中关于对象的相关信息

C.2.3　属性的获取与设定

函　数	功　能
get	获取一个拟合属性对象的所有属性
set	修改或显示一个拟合属性对象的属性值

C.2.4　数据预处理

函　数	功　能
excludedata	指定拟合数据中要剔除的数据
smooth	对原始数据进行平滑处理

C.2.5　数据后处理

函　数	功　能
confint	计算拟合系数的置信区间
differentiate	对拟合结果微分
integrate	对拟合结果积分
predint	计算本模型对新的数据点或新的函数的拟合预期误差限

C.2.6　通用工具

函　数	功　能
cftool	开启曲线拟合工具箱
datastats	获得原始数据的描述统计信息
feval	执行表达式获得一个拟合结果对象或拟合类型对象
plot	作图，根据选项可作出原始数据、拟合曲线、预期误差、奇异点和误差的曲线图

C.3　样条工具箱函数

C.3.1　样条 GUI 函数

函　数	功　能
bspligui	在节点处生成 B 样条曲线
splinetool	用一系列方法生成各种样条曲线

C.3.2 样条构建函数

函　数	功　能
csape	生成给定约束条件下的三次样条函数
csapi	插值生成三次样条函数
csaps	平滑生成三次样条函数
cscvn	生成一条内插参数的三次样条曲线
getcurve	动态生成三次样条曲线
ppmak	生成分段多项式样条函数
rpmak	生成有理样条函数
rsmak	生成有理样条函数
spapi	插值生成 B 样条函数
spaps	对生成的 B 样条曲线进行光滑处理
spap2	用最小二乘法拟合生成 B 样条函数
spcrv	生成均匀划分的 B 样条函数
spmak	生成 B 样条函数
stmak	将函数整理成 st 形式
tpaps	薄板平滑样条函数

C.3.3 操作样条函数

函　数	功　能
fmbrk	返回样条函数的某一部分（如断点或系数等）
fnchg	一种形式的样条函数的改变部分
fncmb	对样条函数进行算术运算
fnder	求样条函数的微分（即求导数）
fndir	求样条函数的方向导数
fnint	求样条函数的积分
fnjmp	在间断点处求函数值
fnmin	求取函数的最小值
fnplt	画样条曲线图
fnrfn	在样条曲线中插入断点
fntlr	生成 tarylor 系数或 taylor 多项式
fnval	计算在给定点处的样条函数值
fnzeros	计算函数的零点
fn2fm	把一种形式的样条函数转化成另一种形式的样条函数

C.3.4　分段多项式样条函数

函　数	功　能
pplst	显示关于生成分段多项式样条曲线的 M 文件
ppmak	生成分段多项式样条函数
ppual	计算在给定点处的分段多项式样条函数值

C.3.5　B 样条函数

函　数	功　能
splst	显示生成 B 样条函数的 M 文件
spmak	生成 B 样条函数
spcrv	生成均匀划分的 B 样条函数
spapi	插值生成 B 样条函数
spap2	用最小二乘法拟合生成 B 样条函数
spaps	对生成的 B 样条曲线进行光滑处理
spcol	生成 B 样条函数的配置矩阵

C.3.6　有理样条函数

函　数	功　能
rpmak	生成有理样条函数
rsmak	生成有理样条函数

C.3.7　样条曲线端点和节点处理函数

函　数	功　能
aptknt	求出用于生成样条曲线的节点数组
augknt	在已知节点数组中添加一个或多个节点
aveknt	求出节点数组元素的平均值
brk2knt	增加断点数组中元素的重次
chbpnt	求出用于生成样条曲线的合适节点数组
knt2brk	从节点数组中求得节点及其重次
knt2mlt	从节点数组中求得节点及其重次
newknt	对分段多项式样条函数进行重分布
optknt	求出用于内插的最优节点数组
sorted	求出节点数组 points 的元素在节点数组 meshpoints 中属于第几个分量

C.3.8 解线性方程组的函数

函　数	功　能
bkbrk	描述分块对角矩阵的详细情况
slvblk	解对角占优的线性方程组

C.3.9 演示函数表

函　数	功　能
bspline	显示一条 B 样条曲线和它的分段形式

C.3.10 实用函数表

函　数	功　能
spcol	生成 B 样条函数的配置矩阵
splpp	0 左面的节点序列从 B 形式转换为 pp 形式
sprpp	0 右面的节点序列从 B 形式转换为 pp 形式
stcol	配置矩阵的离散变换

C.4 偏微分方程工具箱函数

C.4.1 偏微分方程求解算法函数

函　数	功　能
adaptmesh	生成自适应网格并求解 PDE 问题
assema	组合面积的整体贡献
assemb	组合边界条件的贡献
assempde	组合刚度矩阵和 PDE 问题的右端项
hyperbolic	求解双曲线 PDE 问题
parabolic	求解抛物线型 PDE 问题
pdeeig	求解特征值 PDE 问题
pdenonlin	求解非线性 PDE 问题
poisolv	在矩形网格上对泊松方程进行快速求解

C.4.2 用户界面算法函数

函　数	功　能
pdecirc	绘圆
pdeellip	绘椭圆
pdemdlcv	将 PDE 工具箱 1.0 模型的 M 文件转换为 PDE 工具箱 1.0.2 版本的格式

续表

函　数	功　能
pdepoly	绘多边形
pderect	绘矩形
pdetool	PDE工具箱图形用户集成界面（GUI）

C.4.3　几何算法函数

函　数	功　能
csgchk	核对几何描述矩阵的有效性
csgdel	删除最小子域之间的界线
decsg	将建设性实体几何模型分解为最小子域
initmesh	创建初始三角形网格
jigglemesh	微调三角形网格的内部点
pdearcl	在参数表示和圆弧长度之间进行内插
poimesh	在矩形几何图形上生成规则网格
refinemesh	加密一个三角形网格
wbound	写边界条件指定文件
wgeom	写几何指定函数

C.4.4　绘图函数

函　数	功　能
pdecont	绘等值线图
pdegplot	绘制PDE几何图
pdemesh	绘PDE三角形网格
pdeplot	一般PDE工具箱绘图函数
pdesurf	绘三维表面图

C.4.5　实用函数

函　数	功　能
dst	离散化sin转换
idst	逆离散化sin转换
pdeadgsc	使用相对容限临界值选择三角形
pdeadworst	选择相对于最坏值的三角形
pdecgrad	PDE解的变动

续表

函　数	功　能
pdeent	与给定三角形集合相邻的三角形的指数
pdegrad	PDE 解的梯度
pdeintrp	从节点数据至三角形中点数据进行内插
pdejmps	对于自适应网格进行误差估计
pdeprtni	从三角形中点数据向节点数据进行内插
pdesde	子域集合中点的指数
pdesdp	子域集合边缘的指数
pdesdt	子域集合三角形的指数
pdesmech	计算结构力学张量函数
pdetrg	三角形几何数据
pdetriq	三角形质量度量
Poiasma	用于泊松方程快速求解器的边界点矩阵
poicalc	矩形网格上泊松方程的快速求解器
poiindex	经过规范排序的矩形网格的点的指数
sptarn	求解广义稀疏特征值问题
tri2grid	从 PDE 三角形网格到矩形网格进行内插

C.4.6　自定义算法函数

函　数	功　能
pdebound	边界条件 M 文件
pdegeom	几何模型 M 文件

C.4.7　演示函数

函　数	功　能
pdedemo1	单位圆盘上泊松方程的精确解
pdedemo2	求解 Helmholtz 方程，研究反射波
pdedemo3	求解最小表面问题
pdedemo4	用子域分解求解 PDE 问题
pdedemo5	求抛物线型问题（热传导方程）
pdedemo6	求双曲线型 PDE 问题（波动方程）
pdedemo7	点源的自适应求解
pdedemo8	在矩形网格上求解泊松方程

C.5　统计工具箱函数

C.5.1　参数估计函数

函 数 名	功　　能
betafit	β分布的参数估计
betalike	β对数似然函数的参数估计
binofit	二项分布的参数估计
expfit	指数分布的参数估计
gamfit	伽玛分布的参数估计
gamlike	伽玛似然函数的参数估计
mle	极大似然估计的参数估计
normlike	正态对数似然函数的参数估计
normfit	正态分布的参数估计
poissfit	泊松分布的参数估计
unifit	均匀分布的参数估计
weibfit	威布尔分布的参数估计
weiblike	威布尔对数似然函数的参数估计

C.5.2　累加分布函数

函 数 名	功　　能
betacdf	β分布的累加函数
binocdf	二项分布的累加函数
chi2cdf	χ^2 分布的累加函数
expcdf	指数分布的累加函数
fcdf	f 分布的累加函数
gamcdf	伽玛分布的累加函数
geocdf	几何分布的累加函数
hygecdf	超几何分布的累加函数
logncdf	对数正态分布的累加函数
nbincdf	负二项分布的累加函数
ncfcdf	非中心 f 分布的累加函数
nctcdf	非中心 t 分布的累加函数
ncx2cdf	非中心χ^2 分布的累加函数
normcdf	正态（高斯）分布的累加函数
poisscdf	泊松分布的累加函数
raylcdf	端利分布的累加函数
tcdf	学生氏 t 分布的累加函数
unidcdf	离散均匀分布的累加函数
unifcdf	连续均匀分布的累加函数
weibcdf	威布尔分布的累加函数

C.5.3 概率密度函数

函 数 名	功　　能
betapdf	β分布的概率密度函数
binopdf	二项分布的概率密度函数
chi2pdf	χ^2 分布的概率密度函数
exppdf	指数分布的概率密度函数
fpdf	f 分布的概率密度函数
gampdf	伽玛分布的概率密度函数
geopdf	几何分布的概率密度函数
hygepdf	超几何分布的概率密度函数
normpdf	正态（高斯）分布的概率密度函数
lognpdf	对数正态分布的概率密度函数
nbinpdf	负二项分布的概率密度函数
ncfpdf	非中心 f 分布的概率密度函数
nctpdf	非中心 t 分布的概率密度函数
ncx2pdf	非中心χ^2 分布的概率密度函数
poisspdf	泊松分布的概率密度函数
raylpdf	瑞利分布的概率密度函数
tpdf	学生氏 t 分布的概率密度函数
unidpdf	离散均匀分布的概率密度函数
unifpdf	连续均匀分布的概率密度函数
weibpdf	威布尔分布的概率密度函数

C.5.4 累加分布函数的逆函数

函 数 名	功　　能
betainv	β分布的累加分布函数逆函数
binoinv	二项分布的累加分布函数逆函数
chi2inv	χ^2 分布的累加分布函数逆函数
expinv	指数分布的累加分布函数逆函数
finv	f 分布的累加分布函数逆函数
gaminv	伽玛分布的累加分布函数逆函数
geoinv	几何分布的累加分布函数逆函数
hygeinv	超几何分布的累加分布函数逆函数
logninv	对数正态分布的累加分布函数逆函数
nbininv	负二项分布的累加分布函数逆函数
ncfinv	非中心 f 分布的累加分布函数逆函数
nctinv	非中心 t 分布的累加分布函数逆函数
ncx2inv	非中心χ^2 分布的累加分布函数逆函数

续表

函 数 名	功　　能
icdf	逆累计分布函数(icdf)
norminv	正态（高斯）分布的累加分布函数逆函数
poissinv	泊松分布的累加分布函数逆函数
raylinv	瑞利分布的累加分布函数逆函数
tinv	学生氏 t 分布的累加分布函数逆函数
unidinv	离散均匀分布的累加分布函数逆函数
unifinv	连续均匀分布的累加分布函数逆函数
weibinv	威布尔分布的累加分布函数逆函数

C.5.5　随机数生成器函数

函　　数	功　　能
betarnd	β分布的随机数生成器
binornd	二项分布的随机数生成器
chi2rnd	χ^2 分布的随机数生成器
exprnd	指数分布的随机数生成器
frnd	f 分布的随机数生成器
gamrnd	伽玛分布的随机数生成器
geornd	几何分布的随机数生成器
hygernd	超几何分布的随机数生成器
lognrnd	对数正态分布的随机数生成器
nbinrnd	负二项分布的随机数生成器
ncfrnd	非中心 f 分布的随机数生成器
nctrnd	非中心 t 分布的随机数生成器
ncx2rnd	非中心χ^2 分布的随机数生成器
normrnd	正态（高斯）分布的随机数生成器
poissrnd	泊松分布的随机数生成器
raylrnd	瑞利分布的随机数生成器
trnd	学生氏 t 分布的随机数生成器
unidrnd	离散均匀分布的随机数生成器
unifrnd	连续均匀分布的随机数生成器
weibrnd	威布尔分布的随机数生成器

C.5.6　分布函数的统计量函数

函　　数	功　　能
betastat	β分布函数的统计量
binostat	二项分布函数的统计量
chi2stat	χ^2 分布函数的统计量

续表

函　数	功　能
expstat	指数分布函数的统计量
fstat	f 分布函数的统计量
gamstat	伽玛分布函数的统计量
geostat	几何分布函数的统计量
hygestat	超几何分布函数的统计量
logstat	对数正态分布函数的统计量
nbinstat	负二项分布函数的统计量
ncfstat	非中心 f 分布函数的统计量
nctstat	非中心 t 分布函数的统计量
ncx2stat	非中心χ^2分布函数的统计量
normstat	正态（高斯）分布函数的统计量
poisstat	泊松分布函数的统计量
raylstat	瑞利分布函数的统计量
tstat	学生氏 t 分布函数的统计量
unidstat	离散均匀分布函数的统计量
unifstat	连续均匀分布函数的统计量
weibstat	威布尔分布函数的统计量

C.5.7　统计量描述函数

函　数	功　能
bootstrap	任何函数的自助统计量
corrcoef	相关系数
cov	协方差
crosstab	列联表
geomean	几何均值
grpstats	分组统计量
harmmean	调和均值
iqr	内四分极值
kurtosis	峰度
mad	中值绝对差
mean	均值
median	中值
moment	样本模量
nanmax	包含缺失值的样本的最大值
Nanmean	包含缺失值的样本的均值
nanmedian	包含缺失值的样本的中值
nanmin	包含缺失值的样本的最小值
nanstd	包含缺失值的样本的标准差

续表

函　　数	功　　能
nansum	包含缺失值的样本的和
prctile	百分位数
range	极值
skewness	偏度
std	标准差
tabulate	频数表
trimmean	截尾均值
var	方差

C.5.8　统计图形函数

函　　数	功　　能
boxplot	箱形图
cdfplot	指数累加分布函数图
errorbar	误差条图
fsurfht	函数的交互等值线图
gline	画线
gname	交互标注图中的点
gplotmatrix	散点图矩阵
gscatter	由第三个变量分组的两个变量的散点图
lsline	在散点图中添加最小二乘拟合线
normplot	正态概率图
pareto	帕雷托图
qqplot	Q-Q 图
rcoplot	残差个案次序图
refcurve	参考多项式曲线
refline	参考线
surfht	数据网格的交互等值线图
weibplot	威布尔图

C.5.9　统计过程控制函数

函　　数	功　　能
capable	性能指标
capaplot	性能图
ewmaplot	指数加权移动平均图
histfit	添加正态曲线的直方图
normspec	在指定的区间上绘正态密度
schart	S 图
xbarplot	x 条图

C.5.10 线性模型函数

函　数	功　能
anova1	单因子方差分析
anova2	双因子方差分析
anovan	多因子方差分析
aoctool	协方差分析交互工具
dummyvar	拟变量编码
friedman	Friedman 检验
glmfit	一般线性模型拟合
kruskalwallis	Kruskalwallis 检验
leverage	中心化杠杆值
lscov	已知协方差矩阵的最小二乘估计
manova1	单因素多元方差分析
manovacluster	多元聚类并用冰柱图表示
multcompare	多元比较多项式评价及误差区间估计
polyfit	最小二乘多项式拟合
polyval	多项式函数的预测值
polyconf	残差个案次序图
regress	多元线性回归
regstats	回归统计量诊断
Ridge	岭回归
rstool	多维响应面可视化
robustfit	稳健回归模型拟合
stepwise	逐步回归
x2fx	用于设计矩阵的因子设置矩阵

C.5.11 非线性回归函数

函　数	功　能
nlinfit	非线性最小二乘数据拟合（牛顿法）
nlintool	非线性模型拟合的交互式图形工具
nlparci	参数的置信区间
nlpredci	预测值的置信区间
lsqnonneg	非负最小方差 (in MATLAB)
treefit	拟合一个分类或回归的树状模型
treeprune	通过处理树状模型产生子树
treedisp	用图形显示分类或回归的树状模型
treetest	计算树的误差率
treeval	计算适用于数据的决策树的拟合值

C.5.12　试验设计函数

函　数	功　能
bbdesign	Behnken 工具箱设计
candgen	D-最优设计的可选设置
candexch	使用行变换实现 D-最优设计的可选设置
ccdesign	中央复合设计
cordexch	D-优化设计（列交换算法）
daugment	递增 D-优化设计
dcovary	固定协方差的 D-优化设计
ff2n	二水平完全析因设计
fracfact	二水平部分析因设计
fullfact	混合水平的完全析因设计
hadamard	Hadamard 矩阵（正交数组）
rowexch	D-优化设计（行交换算法）

C.5.13　聚类分析函数

函　数	功　能
cluster	根据 linkage 函数的输出创建聚类
clusterdata	根据给定数据创建聚类
cophenet	Cophenet 相关系数
dendrogram	创建冰柱图
inconsistent	聚类树的不连续值
kmeans	K-均值 聚类
linkage	系统聚类信息
pdist	观测量之间的配对距离
silhouette	聚类数据的轮廓图
squareform	距离平方矩阵

C.5.14　降维方法

函　数	功　能
factoran	最大似然一般因子分析
pcacov	源于协方差矩阵的主成分
pcares	源于主成分的方差
princomp	根据原始数据进行主成分分析

C.5.15 其他多元方法

函　数	功　能
barttest	Barttest 检验
canoncorr	典型相关性分析
classify	聚类分析
cmdscale	经典多维标定
mahal	马氏距离
manova1	单因素多元方差分析
manovacluster	多元聚类分析
procrustes	Procrustes 分析
zscore	Z 分数

C.5.16 决策树方法

函　数	功　能
treefit	拟合一个分类或回归的树状模型
treeprune	通过处理树状模型产生子树
treedisp	用图形显示分类或回归的树状模型
treetest	计算树的误差率
treeval	计算适用于数据的决策树的拟合值

C.5.17 假设检验函数

函　数	功　能
ranksum	秩和检验
signrank	符号秩检验
signtest	符号检验
ttest	单样本 t 检验
ttest2	双样本 t 检验
ztest	z 检验

C.5.18 分布检验函数

函　数	功　能
jbtest	正态性的 Jarque-Bera 检验
kstest	单样本 Kolmogorov-Smirnov 检验
kstest2	双样本 Kolmogorov-Smirnov 检验
lillietest	正态性的 Lilliefors 检验

C.5.19 非参数函数

函　数	功　能
friedman	Friedman 检验
kruskalwallis	Kruskalwallis 检验
ranksum	秩和检验
signrank	符号秩检验
signtest	符号检验

C.5.20 隐马尔可夫模型

函　数	功　能
hmmdecode	计算序列状态的后验概率
hmmestimate	估计隐马尔可夫模型的参数
hmmgenerate	为隐马尔可夫模型产生一个序列
hmmtrain	计算隐马尔可夫模型的最大似然估计
hmmviterbi	计算马尔可夫模型序列到达最可能状态的路径

C.5.21 文件输入输出函数

函　数	功　能
caseread	读取个案名
casewrite	写个案名到文件
tblread	以表格形式读数据
tblwrite	以表格形式写数据到文件
tdfread	从表格间隔形式的文件中读取文本或数值数据

C.5.22 演示函数

函　数	功　能
aoctool	协方差分析的交互式图形工具
disttool	探察概率分布函数的 GUI 工具
glmdemo	一般线性模型演示
randtool	随机数生成工具
polytool	多项式拟合工具
rsmdemo	响应拟合工具
robustdemo	稳健回归拟合工具

C.5.23 实用函数

函　数	功　能
statget	从一个统计可选结构获得参数值
statset	创建或者编辑一个统计可选结构

C.6 最优化工具箱函数

C.6.1 最小化函数表

函　数	功　能
fgoalattain	多目标达到问题
fminbnd	有边界的标量非线性最小化
fmincon	有约束的非线性最小化
fminimax	最大最小化
fminsearch, fminunc	无约束非线性最小化
fseminf	半无限问题
linprog	线性课题
quadprog	二次课题

C.6.2 方程求解函数表

函　数	功　能
\	线性方程求解
fsolve	非线性方程求解
fzero	标量非线性方程求解

C.6.3 最小二乘函数表

函　数	功　能
\	线性最小二乘
lsqlin	有约束线性最小二乘
lsqcurvefit	非线性曲线拟合
lsqnonlin	非线性最小二乘
lsqnonneg	非负线性最小二乘

C.6.4 实用函数表

函　数	功　能
optimset	创建或编辑优化选项结构
optimget	获取优化选项结构的参数值

C.6.5　大型方法的演示函数表

函　数	功　能
circustent	马戏团帐篷问题——二次课题
molecule	用无约束非线性最小化进行分子组成求解
optdeblur	用有边界线性最小二乘法进行图形处理

C.6.6　中型方法的演示函数表

函　数	功　能
bandemo	香蕉函数的最小化
dfildemo	过滤器设计的有限精度
goaldemo	目标达到举例
optdemo	演示过程菜单
tutdemo	教程演示



参考文献

1. 《现代数学手册》编纂委员会. 现代数学手册——计算与数值分析卷. 武汉：华中科技大学出版社，2001
2. 王正林，刘明. 精通 MATLAB 7. 北京：电子工业出版社，2006
3. John H.Mathews, Kurtis D.Fink 著，周璐，陈渝，钱方等译. 数值方法（MATLAB 版）第四版. 北京：电子工业出版社，2005
4. 王沫然. MATLAB 与科学计算（第二版）. 北京：电子工业出版社，2003
5. Duane Hanselman and Bruce Littlefield，Mastering MATLAB 7. Prentice Hall PTR，2004
6. 郑惠娆，陈绍林，莫忠息，黄象鼎. 数值计算方法. 武汉：武汉大学出版社，2002
7. 张志涌. 精通 MATLAB 6.5 版. 北京：北京航空航天大学出版社，2003
8. 李庆扬，关治，白峰杉. 数值计算原理. 北京：清华大学出版社，2000