

MATLAB  
开发案例系列图书



# MATLAB

## 高效编程技巧与应用： 25 个案例分析

吴 鹏(rocwoods) 编著



北京航空航天大学出版社

# MATLAB 高效编程技巧与应用： 25 个案例分析

吴 鹏(rocwoods) 编著

北京航空航天大学出版社

## 内 容 简 介

本书是作者八年 MATLAB 使用经验的总结,精心设计的所有案例均来自于国内各大 MATLAB 技术论坛网友的切身需求,其中不少案例涉及的内容和求解方法在国内现已出版的 MATLAB 书籍中鲜有介绍。

本书首先针对 MATLAB 新版本特有的一些编程思想、高效的编程方法、新技术进行了较为详细的讨论,在此基础上,以大量案例介绍了 MATLAB 在科学计算中的应用。内容包括:MATLAB 快速入门、重新认识矢量(向量)化编程、MATLAB 处理海量数据、匿名函数类型介绍、嵌套函数类型介绍、积分以及积分方程求解案例、优化及非线性方程(组)求解案例、人脸图像压缩与重建案例、有关预测分类的案例、常微分方程(组)求解案例、层次分析法及其 MATLAB 实现、定时器及其应用。

本书可作为高等院校本科生、研究生 MATLAB 课程的辅助读物,也可作为从事科学计算和算法研究的科研人员的参考用书。

### 图书在版编目(CIP)数据

MATLAB 高效编程技巧与应用:25 个案例分析/吴鹏  
编著. —北京:北京航空航天大学出版社,2010. 6

ISBN 978 - 7 - 5124 - 0083 - 2

I. ①M… II. ①吴… III. ①计算机辅助计算—软件  
包,MATLAB IV. ①TP391. 75

中国版本图书馆 CIP 数据核字(2010)第 079425 号

版权所有,侵权必究。

### MATLAB 高效编程技巧与应用:25 个案例分析

吴 鹏(rocwoods) 编著

责任编辑 陈守平

\*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191) <http://www.buaapress.com.cn>

发行部电话:(010)82317024 传真:(010)82328026

读者信箱:bhpress@263.net 邮购电话:(010)82316936

印刷有限公司印装 各地书店经销

\*

开本:787 mm×1 092 mm 1/16 印张:17 字数:435 千字

2010 年 6 月第 1 版 2010 年 6 月第 1 次印刷 印数:4 000 册

ISBN 978 - 7 - 5124 - 0083 - 2 定价:39.00 元

# 前言

当今社会,数学作为一切学科基础地位的特征越来越明显,其重要性不言而喻。MATLAB 从诞生那一天起,就为数学和实际应用之间架起了一座桥梁,如今经过 20 多年的发展,这座桥变得越来越雄伟、壮观。现如今,从国外高校到国内高校,从国外期刊到国内期刊,早已随处可见 MATLAB 应用的身影。更重要的是,近几年来,国内许多高新技术公司也开始普遍应用 MATLAB 来进行算法前期开发、验证。

如今,国内 MATLAB 相关书籍已经有很多,这些书籍极大地推动了 MATLAB 在国内的普及。但是 MATLAB 发展迅速,目前每年推出两个版本,现有书籍对 MATLAB 高版本一些特有的编程思想、高效的编程方法、新技术等,鲜有专门详细的讨论。

本书力图以一种全新的模式,从各个角度将 MATLAB 呈现给读者。全书共分两部分:第一部分(第 1~5 章)是有关 MATLAB 高效编程的一些方法、原则介绍;第二部分(第 6~12 章)是案例分析。关于高效编程,本书详细讨论了传统的向量化编程原则在新旧版本 MATLAB 下的异同,MATLAB 如何处理海量数据,匿名函数和嵌套函数灵活、强大、富有弹性的功能。在案例分析部分,介绍了 25 个案例,这些案例都来自我平时的研究积累以及长期以来帮助网友解决的典型问题。案例涉及复杂的多重积分、积分方程、非线性方程求解、全局优化、遗传算法、Benders 分解算法、人脸图像压缩与重建、灰色分析、距离判别法与 Bayes 判别法在分类中的应用、支持向量机、各类型的常微分方程(组)求解、层次分析法以及定时器的应用等。

互联网的兴起催生了很多专门讨论 MATLAB 相关技术的论坛、社区。这些社区集中了来自社会各行各业、高校各学科各专业的 MATLAB 使用者、爱好者。这些社区往往能够紧跟 MATLAB 的发展,并对其最新的技术及时作出反应。我从 2005 年开始就一直在国内一些成立较早的 MATLAB 论坛社区,如研学论坛、仿真科技论坛、振动论坛的 MATLAB 版面参与讨论问题,并发表了一系列技术精华帖。MATLAB 中文论坛成立后,我通过该平台更是经常与各种程度的 MATLAB 使用者打交道,了解 MATLAB 使用者最容易遇到的一些问题,以及一些 MATLAB 软件最新的技术。可以将本书看成是我对这些经验的提取与总结。

我在上大学期间一度十分痛恨“数值计算”这门课程,因为为了应付考试而不得不背一些算法流程、公式并手动计算结果,这是非常枯燥和烦琐的。后来竟 180 度转弯,喜欢上了数值计算,这完全是因为 MATLAB——转机就是大二下学期的数学建模,必须要借助 MATLAB 完成。当用 MATLAB 轻而易举地随意拟合了一个 20 多阶的多项式来近似一堆数据时,我被震撼了。这种震撼是忍受了长时间无比枯燥的手动计算后而发自肺腑的。当然现在看来,那时候的拟合毫无技术含量,毫无实际意义。但也正是因为这个开始,促使我不断去探索 MATLAB。一开始没有计算机,就去看书,在图书馆里看各种有关 MATLAB 的书。随着看的书的增多,MATLAB 在脑子里也越来越清晰了,对它的喜爱也越来越深。后来 2005 年在公司实习,查资料时偶然进入论坛这片新天地,蓦然发现居然有那么多相同爱好的人,从此一发不可收拾,便开始了和一帮志同道合的朋友切磋提高的过程。

针对本书,北京航空航天大学出版社和 MATLAB 中文论坛(<http://www.ilovematlab.>

cn/)特别提供了读者与作者在线交流的平台(<http://www.ilovematlab.cn/forum-182-1.html>),我希望借助这个平台实现与广大读者面对面交流,解决大家在阅读此书过程中遇到的问题,分享彼此的学习经验,从而达到共同进步。本书所有源程序以及测试数据将放到 MATLAB 中文论坛的读者与笔者在线交流平台及北京航空航天大学出版社的网站上,供读者自由下载。

特别感谢北京航空航天大学出版社陈守平编辑,在她一再支持与鼓励下,我决定接受挑战,完成本书创作。

在本书的写作过程中,我得到了天津科技大学谢中华老师、MATLAB 中文论坛创始人 math(张延亮)和仿真科技论坛前总版主 bainhome(马良)的支持与鼓励,在此,向他们表示最真诚的谢意。

最后,还要感谢我的家人和朋友,在他们无微不至的关心与支持下,我顺利完成本书的写作,在此,向他们表示最衷心的感谢。

由于作者水平有限,书中难免出现错误,恳请广大读者和同行批评指正。本书勘误网址:<http://www.ilovematlab.cn/thread-81886-1-1.html>。

吴 鹏

2010 年 3 月于北京市昌平区

# 目 录

## 第一部分 高效编程技巧

第 1 章 MATLAB 快速入门 .....	3
1.1 熟悉 MATLAB 环境 .....	3
1.1.1 MATLAB 的启动 .....	3
1.1.2 MATLAB desktop .....	3
1.1.3 MATLAB 程序编辑器(Editor) .....	4
1.2 MATLAB 牛刀小试 .....	4
1.2.1 Hello, MATLAB .....	4
1.2.2 万能计算器用法 .....	5
1.2.3 设计一个“囧”的动画 .....	6
1.2.4 用 MATLAB 编写的第一个函数 .....	6
1.2.5 用 MATLAB 运行 Windows 系统命令 .....	8
1.2.6 用 MATLAB 发送电子邮件 .....	8
1.3 M 语言介绍 .....	9
1.3.1 数值和变量 .....	9
1.3.2 MATLAB 程序流程控制 .....	10
1.4 学习 MATLAB 的方法 .....	12
第 2 章 重新认识向量化编程 .....	15
2.1 向量化编程流行的一些观点 .....	15
2.2 重新认识循环 .....	15
2.2.1 高版本 MATLAB 对循环结构的优化 .....	15
2.2.2 选择循环还是向量化 .....	20
2.3 提高代码效率的方法 .....	21
2.3.1 预分配内存 .....	21
2.3.2 选用恰当的函数类型 .....	24
2.3.3 选用恰当的数据类型 .....	27
2.3.4 减少无谓损耗——给一些函数“瘦身” .....	29
2.3.5 变“勤拿少取”为“少拿多取” .....	30
2.3.6 循环注意事项 .....	32
2.3.7 逻辑索引和逻辑运算的应用 .....	34
2.4 应用高版本向量化函数提高开发效率 .....	34
2.4.1 accumarray 函数 .....	34

2.4.2	arrayfun 函数	37
2.4.3	bsxfun 函数	38
2.4.4	cellfun 函数	39
2.4.5	spfun 函数	40
2.4.6	structfun 函数	41
<b>第 3 章</b>	<b>MATLAB 处理海量数据</b>	<b>42</b>
3.1	处理海量数据时遇到的问题	42
3.1.1	什么是海量数据	42
3.1.2	经常遇到的问题	42
3.2	有效设置增加可用内存	43
3.2.1	系统默认下内存分配情况	43
3.2.2	打开 Windows 3 GB 开关	45
3.3	减小内存消耗注意事项	46
3.3.1	读取数据文件	46
3.3.2	数据存储	47
3.3.3	减小内存其他注意事项	49
<b>第 4 章</b>	<b>匿名函数类型</b>	<b>50</b>
4.1	匿名函数	50
4.1.1	匿名函数的基本定义	50
4.1.2	匿名函数的种类	51
4.2	匿名函数应用实例	52
4.2.1	匿名函数在求解方程中的应用	52
4.2.2	匿名函数在显式表示隐函数方面的应用	54
4.2.3	匿名函数在求积分区域方面的应用	56
4.2.4	匿名函数在求数值积分方面的应用	56
4.2.5	匿名函数和符号计算的结合	56
4.2.6	匿名函数在优化中的应用	57
4.2.7	匿名函数在求积分区域方面的应用	57
4.2.8	匿名函数和 cell 数组的结合应用	58
<b>第 5 章</b>	<b>嵌套函数类型</b>	<b>59</b>
5.1	嵌套函数	59
5.1.1	嵌套函数的基本定义	59
5.1.2	嵌套函数种类	60
5.2	嵌套函数的变量作用域	60
5.3	嵌套函数彼此调用关系	63
5.3.1	主函数和嵌套函数之间	63
5.3.2	不同的嵌套函数之间	65
5.3.3	嵌套函数调用关系总结	67
5.4	嵌套函数应用实例	68

5.4.1	嵌套函数在求解积分上限中的应用	68
5.4.2	嵌套函数在 GUI 中的应用	68
5.4.3	嵌套函数在 3D 作图中的一个应用	70
5.4.4	嵌套函数表示待优化的目标函数	71
5.4.5	嵌套函数在表示微分方程方面的应用	71

## 第二部分 案例介绍

第 6 章	积分以及积分方程案例	75
6.1	案例 1:一般区域二重、三重积分 MATLAB 计算方法	75
6.1.1	概 要	75
6.1.2	一般区域二重积分的计算	75
6.1.3	一般区域三重积分的计算	78
6.2	案例 2:被积函数含有积分项的一类积分的一些求解方法	80
6.2.1	网格求解法	81
6.2.2	插值求解法	82
6.2.3	RBF 神经网络逼近法	83
6.2.4	dblquad 调用 RBF 神经网络法	86
6.2.5	dblquad+arrayfun 方法	87
6.2.6	quad2d+arrayfun 方法	88
6.3	案例 3:一般区域 $n$ 重积分	90
6.4	案例 4:蒙特卡洛法计算 $n$ 重积分	94
6.4.1	概 述	94
6.4.2	基本的蒙特卡洛积分法	94
6.4.3	等分布序列的蒙特卡洛法	96
6.5	案例 5:第二类 Fredholm 积分方程数值求解	98
6.5.1	概 述	98
6.5.2	具体解法	98
6.5.3	实 例	102
6.6	案例 6:第一类 Fredholm 积分方程数值求解	104
6.6.1	概 述	104
6.6.2	一类可以化为第二类 Fredholm 积分方程的第一类 Fredholm 积分方程求解方法	105
6.6.3	第一类 Fredholm 积分方程的直接数值积分解法讨论	108
6.7	案例 7:第二类 Volterra 积分方程数值求解	109
6.7.1	概 述	109
6.7.2	具体解法	109
6.7.3	实 例	113
6.8	案例 8:第一类 Volterra 积分方程数值求解	116
6.8.1	概 述	116



6.8.2	转化为第二类 Volterra 积分方程 .....	116
6.8.3	实 例 .....	117
<b>第 7 章</b>	<b>MATLAB 优化及非线性方程(组)求解案例 .....</b>	<b>120</b>
7.1	案例 9:全局最优化的讨论 .....	120
7.1.1	随机行走法寻优介绍 .....	120
7.1.2	改进的随机行走法寻优 .....	123
7.2	案例 10:fsolve 求非线性方程组的应用 .....	127
7.2.1	概 述 .....	127
7.2.2	四元非线性方程组的求解 .....	127
7.2.3	九元非线性方程组的求解 .....	128
7.2.4	非线性积分方程的求解 .....	130
7.3	案例 11:渐变光波导方程求解 .....	132
7.3.1	求解渐变光波导的模方程 .....	132
7.3.2	二维渐变光波导方程作图 .....	133
7.4	案例 12:遗传算法在复杂系统可靠度和冗余度分配优化中的应用 .....	134
7.4.1	问题提出 .....	134
7.4.2	数学模型 .....	135
7.4.3	遗传算法简介 .....	136
7.4.4	实例分析 .....	138
7.5	案例 13:遗传算法在车间设备布局优化中的应用 .....	143
7.5.1	问题提出 .....	143
7.5.2	数学模型 .....	143
7.5.3	算法步骤 .....	145
7.5.4	求解代码 .....	146
7.6	案例 14:应用 Benders 分解算法求解混合 0-1 规划 .....	151
7.6.1	概 述 .....	151
7.6.2	Benders 分解算法 .....	151
7.6.3	实例分析 .....	155
<b>第 8 章</b>	<b>案例 15:人脸图像压缩与重建 .....</b>	<b>160</b>
8.1	概 述 .....	160
8.2	基本的 PCA 方法实现人脸图像压缩与重建 .....	160
8.2.1	K-L 变换 .....	160
8.2.2	特征向量的选取 .....	162
8.3	2DPCA 方法实现人脸图像压缩与重建 .....	162
8.3.1	概 述 .....	162
8.3.2	2DPCA 算法介绍 .....	163
8.3.3	图像压缩(特征提取) .....	164
8.3.4	图像重建 .....	164
8.4	MatPCA 方法实现人脸图像压缩与重建 .....	165

8.4.1	概 述 .....	165
8.4.2	MatPCA 算法 .....	165
8.5	ModulePCA 方法实现人脸图像压缩与重建 .....	166
8.5.1	概 述 .....	166
8.5.2	ModulePCA 算法 .....	166
8.6	算法在 MATLAB 平台上的实现 .....	167
8.6.1	概 述 .....	167
8.6.2	基本 PCA 与 2DPCA 和 MatPCA 方法 GUI .....	167
8.6.3	Module PCA 方法 GUI .....	176
<b>第 9 章</b>	<b>有关预测分类的案例 .....</b>	<b>183</b>
9.1	案例 16:北京市国民生产总值的灰色分析 .....	183
9.1.1	概 述 .....	183
9.1.2	引 言 .....	183
9.1.3	灰色数据融合预测算法与灰色关联度 .....	184
9.1.4	实例分析 .....	187
9.2	案例 17:距离判别法与 Bayes 判别法在分类中的应用 .....	190
9.2.1	概 述 .....	190
9.2.2	判别方法 GUI .....	190
9.2.3	判别方法 GUI 应用举例 .....	194
9.3	案例 18:支持向量机的应用 .....	195
9.3.1	概 述 .....	195
9.3.2	支持向量机介绍 .....	195
9.3.3	MATLAB 所依据的支持向量机模型 .....	196
9.3.4	支持向量机实现图像分割 .....	197
9.3.5	支持向量机实现手写体数字识别 .....	199
<b>第 10 章</b>	<b>常微分方程(组)求解案例 .....</b>	<b>203</b>
10.1	案例 19:常微分方程(组)解析求解案例 .....	203
10.1.1	概 述 .....	203
10.1.2	dsolve 函数 .....	203
10.1.3	dsolve 函数求解实例 .....	204
10.2	数值求解常微分方程函数 .....	207
10.2.1	概 述 .....	207
10.2.2	初值问题求解函数 .....	208
10.2.3	延迟问题以及边值问题求解函数 .....	209
10.2.4	求解前的准备工作 .....	209
10.3	案例 20:非刚性/刚性常微分方程初值问题求解 .....	210
10.3.1	概 述 .....	210
10.3.2	非刚性问题举例 .....	210
10.3.3	刚性问题举例 .....	212

10.4 案例 21:隐式微分方程(组)求解 .....	216
10.4.1 概 述 .....	216
10.4.2 利用 solve 函数 .....	216
10.4.3 利用 fzero/fsolve 函数 .....	217
10.4.4 利用 ode15i 函数 .....	220
10.5 案例 22:微分代数方程与延迟微分方程求解 .....	221
10.5.1 概 述 .....	221
10.5.2 微分代数方程举例 .....	221
10.5.3 延迟微分方程举例 .....	226
10.6 案例 23:边值问题求解 .....	230
10.6.1 概 述 .....	230
10.6.2 求解案例 .....	230
10.6.3 对 bvp4c 和 bvp5c 的改进 .....	234
<b>第 11 章 案例 24:层次分析法及其 MATLAB 实现 .....</b>	<b>236</b>
11.1 层次分析法概述 .....	236
11.2 层次分析法实现步骤 .....	236
11.2.1 层次分析法的主要实现步骤 .....	236
11.2.2 建立层次分析的结构模型 .....	236
11.2.3 构造成对比较矩阵 .....	238
11.2.4 单一准则下元素相对排序权重计算及比较矩阵一致性检验 .....	238
11.2.5 各元素对目标层合成权重的计算过程 .....	240
11.3 应用实例 .....	242
<b>第 12 章 案例 25:定时器及其应用 .....</b>	<b>247</b>
12.1 定时器介绍 .....	247
12.1.1 概 述 .....	247
12.1.2 定时器属性介绍 .....	247
12.2 定时器应用举例 .....	253
<b>参考文献 .....</b>	<b>258</b>

# 第一部分

## 高效编程技巧

北京航空航天大学出版社

北京航空航天大学出版社

本章旨在帮助从未接触过 MATLAB 的读者能快速入门,对 MATLAB 产生兴趣,能够用 MATLAB 完成简单任务,并告之其后续学习方法,确保其能有效的持续提高。

本着让初学者迅速上手的思想,本章由 MATLAB 界面入手,介绍基本的 MATLAB 操作环境,进而用一系列短小的程序帮助读者建立起对 MATLAB 的最初印象,激发学习兴趣。在这之后对 M 语言进行简明扼要介绍以帮助读者能自己编写 MATLAB 程序。最后一节是笔者长期使用、学习 MATLAB 并在国内众多 MATLAB 论坛与各层次 MATLAB 使用者交流的经验总结,相信无论读者以前用没用过 MATLAB,都能从中得到收获。

## 1.1 熟悉 MATLAB 环境

### 1.1.1 MATLAB 的启动

Windows 环境下, MATLAB 常用启动方法有以下几种:

- 1) 双击桌面上的 MATLAB 快捷方式图标(一般指向 MATLAB 安装目录下 bin 文件 matlab.exe 文件);
- 2) 依次单击 Windows 任务栏上“开始”按钮→“所有程序”→MATLAB,找到 MATLAB R2009a 选项(其他版本找到对应选项)进入 MATLAB 界面;
- 3) 单击 Windows 任务栏上“开始”按钮,在弹出的开始菜单中单击“运行”按钮,输入 MATLAB 并按 Enter 键进入 MATLAB 界面。

### 1.1.2 MATLAB desktop

第一次启动 MATLAB 完毕后,呈现给读者的是如图 1.1 所示的 MATLAB 界面(也即默认状态下的桌面)。对于初学者来说,最主要需要了解以下三个窗口,即命令窗口(Command Window)、工作空间(Workspace)和历史命令(Command History)窗口。

命令窗口是用来输入各种 MATLAB 命令,观察运算结果的窗口。

工作空间可以用来浏览程序运行过程中或完毕后得到的变量的值,当工作空间有变量后,可以选中要观察的变量,双击它, MATLAB 会新开一个窗口来展示这个变量的值或者属性。

历史命令窗口里面记录了曾经在 Command Window 中输入过的命令。当选中的是历史命令窗口或者命令窗口时,可以按键盘上的上、下方向键来选中或者调出上一条、下一条指令。双击历史命令窗口中某条指令,则会在命令窗口中重新运行这条指令。

由于本书的侧重点不在基础知识上,限于篇幅,关于这些窗口的详细介绍就略去了。需要说明的是, MATLAB 这些窗口操作非常人性化,人机交互性非常好。读者如果初次使用 MATLAB,可以按照自己的 Windows 系统使用经验来试探着操作这些窗口、按钮,观察结果,

很快就会熟悉 MATLAB 基本的操作环境。如果对某些图标的意义不清楚,可以把鼠标指针分别移到相应的图标上面(不要点击),过一两秒钟,就会显示当前图标的功能介绍。如果在操作过程中关闭了某些窗口,譬如命令窗口、工作空间等而改变了桌面结构,想要恢复默认桌面显示状态,只需单击菜单栏里 Desktop 菜单项,进入后单击 Desktop Layout,选择 Default 就恢复默认桌面结构了。

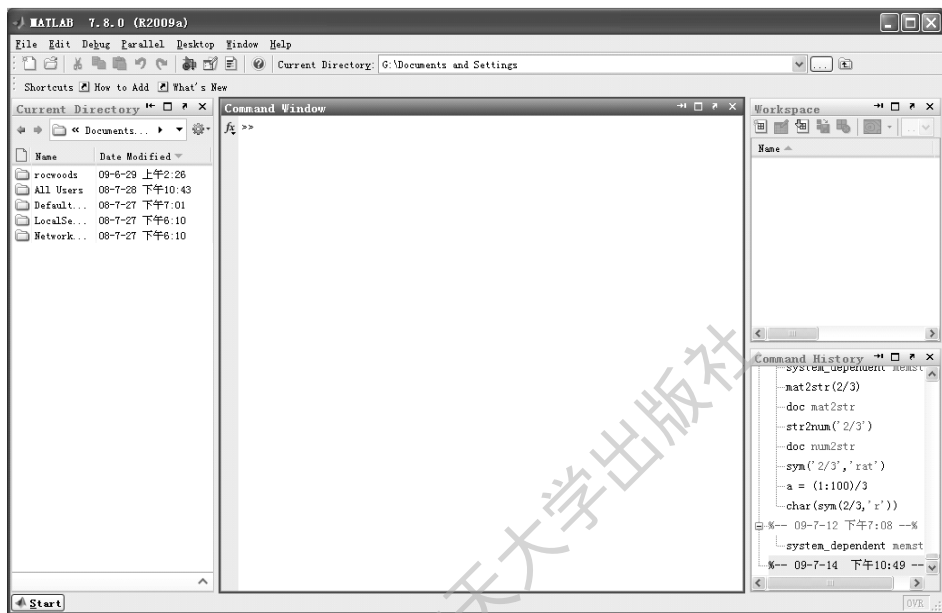



图 1.1 MATLAB R2009a 默认桌面

### 1.1.3 MATLAB 程序编辑器(Editor)

MATLAB 中 M 文件的编写都在程序编辑器里进行。一个完整的 MATLAB 项目或者工程可能由一个或者多个 M 文件构成,而一个 M 文件中可能包含一个或者多个各种类型的函数。

打开 Editor 的常用方法有如下几种:

- 1) 双击磁盘上现有的 M 文件;
- 2) 命令窗口中输入 edit 并按 Enter 键;
- 3) 单击 MATLAB 主窗口上方工具栏中  图标;
- 4) 依次选择 MATLAB 主窗口上方菜单栏:File→New→Function M-File。

## 1.2 MATLAB 牛刀小试

### 1.2.1 Hello, MATLAB

现在让我们来看 MATLAB 是如何实现“Hello, World!”这样简单的程序的。MATLAB 实现起来非常方便,一般常用的有两种方法:

方法 1 在 Command Window 窗口中输入下列代码并按 Enter 键：

```
disp('Hello,MATLAB!')
```

方法 2 在 Command Window 窗口中输入下列代码并按 Enter 键：

```
A = sprintf('Hello,MATLAB!')
```

以上两种方法的区别是：第一种方法是直接在屏幕上显示“Hello,MATLAB!”字样；第二种方法是将“Hello,MATLAB!”作为一个字符串常量赋值给变量 A，然后在屏幕上显示 A 的内容，即“Hello,MATLAB!”

## 1.2.2 万能计算器用法

相信没有用过 MATLAB 的朋友看了这一小节，肯定会对 MATLAB“电子演算纸”的称谓有较为形象的认识，因为对于简单的计算问题，在 MATLAB 里只是输入计算的表达式然后按下 Enter 键这么简单。

**【例 1.2-1】** 计算  $\sin(\ln(\pi))^{\exp(1+\sqrt{2\pi/3})}$ 。

要计算上述表达式的值，只需要在 Command Window 窗口中输入下列代码并按 Enter 键，即得到结果。

```
sin(log(pi))^exp(1+sqrt(2*pi/3))
```

命令窗口中运行结果如下：

```
ans =  
    0.3388  
>>
```

log 是 MATLAB 自带的求自然对数的函数，表达式中的四则运算、指数运算、开方运算等书写形式和大多数其他的编程语言都一致。

ans 是 MATLAB 中的表达式没有指定赋值给其他变量时，用来存放表达式结果的默认的变量，它是 answer 的缩写。

默认状态下，MATLAB 的显示精度是 short 型，如果想查看更多位的有效数字，可以通过运行 format long 来实现。本例中，上述操作的过程和结果如下：

```
>> format long  
>> ans  
ans =  
    0.338830950391107
```

如果想恢复到原来的显示精度，只需要运行 format short 即可。



**注意** 不要混淆显示精度和计算精度。MATLAB 默认的计算精度是 double 型，而显示精度是呈现结果时候的精度，与计算精度无关。很多初学者认为显示精度是 short 型时，计算精度也是 short 型，这是不对的。

**【例 1.2-2】** 计算福彩双色球一等奖的中奖概率。福彩双色球玩法如下：从编号 1~33 的红球里任选 6 个，另外在编号 1~16 的蓝球里再任选 1 个。如果选择的红球和蓝球和当期



的开奖结果完全一致(顺序可以不同)则中一等奖。

这是个组合问题,其中奖概率用数学表达式表示为 $\frac{1}{C_{33}^6 C_{16}^1}$ ,用 MATLAB 解,代码如下:

```
>> p = 1/nchoosek(33,6)/nchoosek(16,1)
p =
    5.6430e-008
```

分析结果可知,中奖概率只有亿分之五左右。由此可以看出,一等奖对绝大多数人来说是不可望而不可即的。

### 1.2.3 设计一个“囧”的动画

**【例 1.2-3】** 随机展示一下方向、大小、颜色各异的“囧”字。

运行如下代码,观察呈现的动画。

```
function RandDisplayJiong
axis off; % 隐去坐标轴
set(gcf,'menubar','none','toolbar','none'); % 不显示当前 figure 菜单栏和工具栏
for k = 1:100 % 循环 100 次
% 每次在(rand,rand)这个随机的位置,选择 20~50 之间随机分布的一个数作为其字体大小,以隶书的
% 形式,随机生成 RGB 颜色,并随机旋转一定的角度来显示“囧”字
h = text(rand,rand,...
['\fontsize{',num2str(unifrnd(20,50)),'}\fontname{隶书} 囧'],...
'color',rand(1,3),'Rotation',360 * rand);
pause(0.2); % 每显示完一次暂停 0.2s
end
```

图 1.2 是代码运行完毕最后得到的图片(由于随机,每次运行结果图都不完全一样)。



图 1.2 “囧”的动画效果图

### 1.2.4 用 MATLAB 编写的第一个函数

在本部分将编写一个结构相对完整的 MATLAB 函数,这个函数包含输入和输出。我们

用这个函数来解决下面这个有趣的问题。

**【例 1.2-4】**一只失明的小猫不幸掉进山洞里,山洞有三个门,一个门进去后走 2 h 可以回到地面,从第二个门进去后走 4 h 又回到原始出发点,不幸的是从第三个门进去后走 6 h 还是回到原始出发点。小猫由于眼睛失明,每次都是随机地选择其中一个门走。那么可怜的小猫走出山洞的期望时间是多少?

这个问题如果按常规思路,则需要求几个级数的和,很麻烦。不过可以这么想,设小猫走出山洞的期望时间为  $t$ ,如果小猫不幸进了第二个或第三个门,那么它过 4 h 或 6 h 后又和进门之前面临的状况一样了,只不过这两种不幸的情况发生的概率都为  $1/3$ 。而万幸一次性走出去的概率也是  $1/3$ 。于是可以得到下面的方程:

$$t = 2 \times (1/3) + (4 + t) \times (1/3) + (6 + t) \times 1/3$$

解得  $t = 12$ 。

为了验证结果的正确性,下面用 MATLAB 编写模拟小猫  $n$  次出洞时间的函数,该函数返回的结果  $T$  为  $n$  次出洞时间组成的数组,代码如下:

```
function T = cat_in_holl(n)
T = zeros(1,n);
for k = 1:n
    c = unidrnd(3,1);
    while c ~= 1
        if c == 2
            T(k) = T(k) + 4;
        else
            T(k) = T(k) + 6;
        end
        c = unidrnd(3,1);
    end
    T(k) = T(k) + 2;
end
```

下面来分析一下这个函数。“function”是函数声明符,其后定义了函数的输出变量,函数名以及输入变量,本例中,“T”为函数输出变量,“cat\_in\_holl”为函数名,“n”为输入变量。第一行是函数的声明,函数体必须另起一行开始写。 $T = \text{zeros}(1,n)$ ;表示生成一个 1 行  $n$  列的全 0 行向量,这步主要就是预分配内存,因为最后得到的  $T$  是小猫  $n$  次出洞的时间组成的数组,数组长度事先已知。给  $T$  预分配内存是基本的 MATLAB 高效编程准则之一,关于这一点将在后续章节中详细讨论。

接下来是一个 for 循环体( $k$  从 1 循环到  $n$ ),for 循环体内部还有一个 while 循环体(当  $c \neq 1$  时,运行 while 循环体内的代码),while 循环体内还有一个 if - else 选择分支结构。这些循环和选择分支的意义与其他语言都类似,不同的就是格式,每一个循环体或者选择分支结构结束时,MATLAB 用“end”来标志。

unidrnd 是 MATLAB 自带的函数,unidrnd(3,1)表示在 1,2,3 中等概率地随机生成一个数字。关于 unidrnd 函数的详细介绍,读者可以运行 doc unidrnd 查看详细说明。

打开 Editor,将上述代码粘贴过去,保存为 cat\_in\_holl.m 文件,并使之在 MATLAB 搜索路径上,就可以在命令窗口输入“ $Z = \text{cat\_in\_holl}(10000)$ ;”来观察模拟小猫 10 000 次出洞的时

间了。可以运行 `mean(T)` 观察平均时间,会发现随着 `n` 的增长,`mean(T)` 越来越接近 12。

### 1.2.5 用 MATLAB 运行 Windows 系统命令

在 MATLAB 环境里可以运行 Windows 系统命令,常用的方法有两种:一种调用格式是“! 系统命令”;另一种调用格式是“dos 系统命令”。下列实验基于 MATLAB 2009a 和 Windows XP sp2 操作系统。

读者可以在 MATLAB 命令窗口中输入“!calc”、“!mspaint”、“dos calc”代码,运行并观察结果。因为 MATLAB 可以运行系统命令,我们可以充分利用这个功能。譬如,一个程序要运行很长时间,而我们又不能一直守在计算机前,这时可以在程序运行完并保存好所需要的结果后在程序最后一行加上“!shutdown -s”,则程序运行完毕后会自动关闭计算机。

### 1.2.6 用 MATLAB 发送电子邮件

下面以一个有意思并且实用的问题来结束本节——用 MATLAB 发送电子邮件。

通过 MATLAB 发送电子邮件的重要意义在于:譬如正在运行一个程序,而这个程序运行时间很长,几小时、几天甚至更长,这期间我们不能一直守在计算机旁,比如要回家,但是又急于用这个程序的结果进行下一步的分析工作,这时候我们希望 MATLAB 能够在程序计算完成时,将需要的东西保存下来,然后以“附件”的形式发送到指定的邮箱,完成后, MATLAB 再运行系统命令关闭计算机。

MATLAB 自带了一个 `sendmail` 函数,可以用来发送电子邮件。如果用 `sendmail` 发邮件,只能发到不用 authentication 的地址。可是为了防止垃圾邮件蔓延,互联网上的公共邮箱几乎都要 authentication。那么怎么实现呢?那就是用 gmail 的邮箱,利用 Java 的方式发送到其他邮箱。实现过程举例如下(实验环境 MATLAB R2009a + Windows XP sp2 + Gmail 的电子邮箱):

```
function MySendMail
a = rand(100);
DataPath = [matlabroot,filesep,'mydata.mat'];
save(DataPath,'a');
MailAddress = 'Gmail 的邮箱地址';
password = '邮箱密码';
setpref('Internet','E_mail',MailAddress);
setpref('Internet','SMTP_Server','smtp.gmail.com');
setpref('Internet','SMTP_Username',MailAddress);
setpref('Internet','SMTP_Password',password);
props = java.lang.System.getProperties();
props.setProperty('mail.smtp.auth','true');
props.setProperty('mail.smtp.socketFactory.class',...
'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port','465');
subject = 'MATLAB 发的测试邮件';
content = '你好,这份邮件是我用 MATLAB 发的,数据见附件';
sendmail('收件人 Email 地址',subject,content,DataPath);
```

## 1.3 M 语言介绍

### 1.3.1 数值和变量

#### 1. 数 值

MATLAB 里的数值采用十进制表示,并可以用科学记数法表示,对于绝对值小于 1 的小数,“0”可以省略。以下记数都合法:

5, 0.5, .5, -1.5, 0.001, 6.02e23, 2.997 9e8, 6.626e-34, 6.67e-11

MATLAB 的数值计算默认是定义在复数域上的浮点计算,采用的标准是 IEEE754:1985《二进制浮点运算》,一个非零的浮点数可以表示为

$$x = \pm(1+f)2^e$$

其中: $f$  是尾数( $0 \leq f < 1$ ); $e$  是指数,默认情况下,MATLAB 用 double(双精度)型来表示一个浮点数,即 64 位来表示  $x$ ,其中  $f$  用 52 位表示, $e$  用 11 位表示, $-1022 \leq e \leq 1023$ ,剩下 1 位表示  $x$  的符号。由此可以看出, $f$  决定了数值的精度,这个精度是  $2^{-52}$ ,即 MATLAB 中预定义变量 eps 的值。 $e$  决定了数值的范围,其之所以没有取 1024 和 -1023,是因为 MATLAB 已经为两个特殊类型的数预留了。 $f$  取 0, $e$  取 1024 表示的是预定义变量 inf,非规范化数字(Denormal numbers)由  $e$  取 -1023 时表示。

了解了上述定义之后,就可以理解 MATLAB 中一些预定义变量的含义了,举例如下:

eps: $2^{-52}$

realmin: $2^{(-1022)} = 2.2251e-308$

realmax: $(2 - \text{eps}) * 2^{1023}$

读者如果想更深入地了解 MATLAB 浮点计算的有关机理,可以参考 MATLAB 软件创始人 Moler 教授的《Numerical Computing with MATLAB》一书的第一章。

#### 2. 变 量

很多 MATLAB 书籍将 MATLAB 变量分为 MATLAB 默认的预定义变量(如:pi,i,j,eps,nan,inf 等)和用户自己定义的变量。

其实这些所谓的预定义变量是 MATLAB 内部的一些函数,这些函数被调用后返回其函数名所表示的值,方便起见,人们仍旧称其为预定义变量。这些函数有的没有输入参数,如 pi,i,j 等,有的可以有也可以没有输入参数,如 realmin,realmax,eps,inf,nan 等。举例来说,我们直接在命令窗口输入 eps 并按 Enter 键,返回的是用 double 型来表示一个浮点数情况下的精度值,即  $2^{-52}$ 。如果输入 eps('single')按 Enter 键,那么返回的就是用单精度型表示浮点数情况下的精度值,即  $2^{-23}$ 。表 1-1 列出了常用的 MATLAB 预定义变量,大家注意在编写程序时不要使用这些变量名来表示自己定义的变量。

用户自定义的变量。MATLAB 中变量或者函数的命名遵循以下三条原则:

1) 变量名、函数名对字母大小写是敏感的。如变量 data 和 Data 表示两个不同的变量,cos 是 MATLAB 定义的余弦函数名,但 Cos,COS 都不是。

2) 变量名的第一个字符必须是英文字母,其余可以是英文字母、数字或者下画线,最多可包含 63 个字符(运行 namelengthmax 命令可以得到这个数)。如 a1,a\_b\_1 是合法的变量名,

\_a,1a,a\$ 则都不是合法的变量名。

3) 变量名中不得包含空格、标点、运算符。

表 1-1 MATLAB 中常用的预定义变量

预定义变量	含 义	预定义变量	含 义
ans	计算结果的默认变量名	pi	圆周率
eps	特定精度表示浮点数时的精度值	NaN 或 nan	非数(Not a Number),如 0/0, ∞/∞
Inf 或 inf	特定精度表示浮点数时候的无穷大	realmax	特定精度表示浮点数时的最大正实数
i 或 j	虚数单位	realmin	特定精度表示浮点数时的最小正实数

1.3.2 MATLAB 程序流程控制

和其他大多数编程语言一样,MATLAB 主要有以下几种程序流程控制方法,调用格式分别为:

1. if - else - end 结构

第一种:

```
if expr % expr 为表达式,如果成立,则执行到“end”为止的所有 commands,否则不执行 commands
commands;
end
```

第二种:

```
if expr1 % expr1 成立则执行 commands1;否则判断 expr2;
commands1;
elseif expr2 % 如果 expr2 为真,则执行 commands
commands2;
else % expr1, expr2 都不成立,执行 commands3
commands3;
end
```

其中根据程序分支的多少,“else - if”的个数可以有 0 个到多个。

2. switch - case - otherwise - end 结构

```
switch expr % expr 为表达式
case value1 % expr 取值为 value1 时,执行 commands1
(commands1)
case value2 % expr 取值为 value2 时,执行 commands2
(commands2)
case valuek
(commands k)
otherwise % expr 取值为 value1 到 valuek 之外的任何值时,执行 commands
(commands)
end
```

“otherwise”类似于“if - else - end”结构中的“else”,视程序逻辑需要,并不是一定要有,

但为了程序结构上的完整性以及某些情况下调试程序的需要,建议始终保留。commands 也可以为空语句。

### 3. for 循环

```
for ix = array
    commands;
end
```

在命令窗口中运行下列代码,体会 for 循环的用法。

```
for ix = 1:10
    a = ix;
end

for ix = (1:10)'
    a = ix;
end

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

运行上面代码可以发现,MATLAB 的 for 循环的机制是遍历 array 的列,无论这个 array 是向量还是矩阵。如果 array 是行向量,那么 for 就遍历它的每个元素;如果是列向量,for 循环就循环一次,即遍历列向量自身;如果 array 是矩阵,那么 for 循环就遍历它的每一列,循环  $n$  次, $n$  是 array 的列数。此外,如果 array 是三维矩阵,那么 for 循环先遍历第一页的所有列,之后是第二页的所有列,等等。

### 4. while 循环

```
while expr
    commands;
end
```

while 循环的机制是当 expr 为真的时候执行 commands 命令,直到 expr 为假。所以执行 commands 命令必须得在有限循环次数内使得 expr 为假,否则 while 循环会一直循环运行下去。

### 5. try - catch 结构

```
try
    commands1           % Try block
catch ME
    commands2           % Catch block
end
```

该结构意义是执行 commands1,如果不发生错误,则不用执行 commands2;如果执行 commands1 的过程中发生错误,那么 commands2 就会被执行,同时,ME 记录了发生错误的相关信息。

## 6. 其他中断、暂停语句

MATLAB 在循环体内还可以利用 `continue` 语句跳过位于它之后的循环体中的其他指令,而执行循环的下一个迭代。例如:

```
for ix = 1:5
    if ix == 3
        continue;
    end
    disp([' ix = ', num2str(ix)])
end
```

运行结果:

```
ix = 1
ix = 2
ix = 4
ix = 5
```

MATLAB 可以用 `break` 语句结束包含该指令的 `while` 或 `for` 循环体,还可以在 `if - end`, `switch - case`, `try - catch` 结构中导致程序中断。例如:

```
ix = 3;
jx = 6;
if ix == 3
    disp([' ix = ', num2str(ix)]);
    break
    disp([' jx = ', num2str(jx)]);
end
```

运行结果:

```
ix = 3
```

可见 `break` 中断程序执行了,导致“`jx=6`”没有被显示。

`pause(n)` 可以使程序暂停 `n` 秒后再继续执行;`pause` 指令使程序暂停执行,等待用户按任意键继续;`return` 指令可以结束“`return`”所在的函数的执行,如果“`return`”所在函数由函数 `fun` 来调用的,则程序将控制转至函数 `fun`,如果“`return`”所在函数是在命令窗口直接运行的,则程序将控制转至命令窗口。

## 1.4 学习 MATLAB 的方法

通过前面几节的介绍,相信刚接触 MATLAB 的读者对 MATLAB 已经有了初步的了解。MATLAB 功能非常强大,前面介绍的仅仅是其全部功能的非常小的一部分。MATLAB 本身自带有几个涉及多个专业领域的工具箱,每个工具箱都包含很多解决专业问题的函数,这些函数都是建立在一篇篇经典的论文基础上,而实际学习中没必要也不可能对每个工具箱中的每个函数的用法都掌握得很精通。那么我们怎么来学习 MATLAB 呢? 笔者给出的建议是:

用好 help,学会搜索,多读高手的程序,自己多练习。

**第一条:用好 help。**古语说:“授之以鱼不如授之以渔”。MATLAB 的帮助系统做得非常好,非常完善,帮助文档对 MATLAB 每个函数的用法都进行了详细的描述。学会使用 help 后,就好比学会了“打鱼”的本领,不仅可以大大减轻记忆的负担,而且可以帮助自身持续提高 MATLAB 应用水平。

这里建议大家牢记几个关键词,有了它们,相信很多时候就掌握了解决问题的方法。这些关键词即:help 或者 doc, see also, help navigator, Tab。不要小瞧这几个词,可以毫不夸张地说,这几个词带给读者的帮助是随着读者的 MATLAB 应用水平不断提高而越来越大的。

运行“help 函数名”,会在 command window 窗口里列出该函数的详细介绍,包括调用格式,输入输出参数的意义等。运行“doc 函数名”,会调出该函数在帮助文档里介绍的那一页,可读性比“help 函数名”要好些,具体使用哪个就看个人使用习惯了。

在当前查询的函数的最后,MATLAB 会在“see also”后面列出一些功能上和被查询的函数有相同或者相似地方的一些函数。不要小瞧这个“see also”,很有可能被查询的函数完成不了预期的任务,而实现相关功能的函数就在 see also 里。

help navigator,堪称 MATLAB 中的“百度”,打开帮助文档后其位于界面的左半部分。往往人们在对一个问题毫无头绪的时候,可以用它来搜一下相关的关键词,很多时候会有意想不到的结果。譬如,如果想知道 MATLAB 中有无求集合交集的函数,可以在 help navigator 里的搜索栏中输入集合交集的两个关键词“set intersect”(注:如果在关键词之间加“-”连接号,表示搜索时将 these 关键词作为一个短语整体来搜,而不是各自独立搜),按 Enter 键后发现,搜索结果第一项——intersect 函数的介绍:“Find set intersection of two vectors”即求集合交集的函数,而在 see also 里我们还一并得知了求集合并集、补集等函数,收获颇丰。

Tab 键也非常有用。当我们对一个函数记不清楚的时候(譬如只记得前半部分某几个字母),我们写下这几个字母并按 Tab 键,MATLAB 会列出所有刚写的那些字母开头的函数。我们找到需要的函数后可以进一步在帮助文档中查看其用法。

总之,建议读者经常翻看帮助文档,积累对 MATLAB 学习很重要,经常翻看帮助文档,会不知不觉地积累很多 MATLAB 相关技巧、知识。

**第二条:学会搜索。**当今社会的信息量极其丰富,如何获取我们需要的信息,搜索能力的重要性不言而喻。

对于 MATLAB 学习来说,搜索能力也是非常重要的。其实在第一条中已经对搜索能力有要求了,即在 help navigator 中搜索我们需要的函数。对于 MATLAB 初学者来说,面临的问题通常都早已被解决过很多次了,互联网上有很多现成的答案。可是在各个 MATLAB 社区中,笔者经常看到一些解决过好多次的问题一次又一次地被问起,这从一个侧面反映了有相当一部分 MATLAB 学习者缺乏搜索能力。将问题的解答寄希望于别人总没有通过自己搜索并努力钻研解决来得印象深刻。

搜索的关键在于搜索的关键词。一般说来,关键词不应过长,要有代表性,在选择搜索的关键词时,多想想别人问与自己同样的问题的时候可能会用到哪些关键词,并可以在网上看一些有关搜索技巧的文章。养成这样的习惯,久而久之,搜索经验越来越丰富,搜索能力就会有所提高。

**第三条:多读高手的程序。**俗话说得好,和臭棋篓子下棋,越下越臭。学习编程也是如此。



MATLAB 是一门灵活性非常高的语言,针对同一个问题,不同的人写出来的程序运行效率可能差别巨大。有些人不注意基本的 MATLAB 编程规范,写出来的代码运行效率低,还抱怨 MATLAB 运行太慢;有些人把 MATLAB 运用的得心应手,充分利用 MATLAB 向量化编程优点,对很多问题能写出不输于 C/C++ 语言,甚至比 C/C++ 语言还快的程序。

读高手的程序,能避免走很多弯路,而且还能享受 MATLAB 向量化编程带来的便利、高效以及乐趣。找到高手很简单,很多高手都在不止一个 MATLAB 技术论坛注册了同样的 ID,去各大 MATLAB 技术论坛翻看一下帖子,很容易就能发现哪个 ID 水平比较高,找到之后可以利用各论坛的搜索功能搜索一下他的帖子,搜出来帖子后就慢慢去读,从中可以学到很多东西。对于初学者,可能会发现很多问题都看不懂,这很正常,谁都有这个过程,重要的是现在能看懂的那部分,有收获的那部分。看完这部分可以再看些通过断点调试,查帮助文档能看明白的。实在看不懂的就先放下,过段时间自己感觉有进步了再回头看看,慢慢得会发现,每看一遍,自己就会有一些新的体会,当通过自己的努力看了很多遍的时候,也许突然就会有豁然开朗、峰回路转的感觉,这种感觉非常棒,会给自己增添不少学习信心以及乐趣。

建议初学者开始不要读比较长的程序,可以挑精简的程序去慢慢研究领会,这样容易保持旺盛的学习兴趣。

**最后,说一下亲自动手练习的重要性。**对于任何语言来说,写程序都是不断地自我完善的过程,上来就写出没有 bug 的代码,对于稍微复杂的问题来说都是不现实的。所以不要怕错误,一定要自己多动手编写程序,不要以为看懂书上的代码了,自己就能编出来,只有自己亲自写代码并运行成功,才能切身体会到很多只是看程序所体会不到的东西。如果觉得自己看明白了,可以试着不看书,自己重新写一下,然后和源程序对照一下。

在 MATLAB 相关技术论坛试着回答他人提出的问题,也是一种被证明了的比较有效的学习方法。在回答他人的问题过程中,自己会去查 Help 帮助文档,搜索,思考,然后自己再动手,还能参考别人给出的答案。这样一来,就把上面提到的几点学习方法有机结合在一起,久而久之,慢慢积累,学得东西越来越多,水平就会逐渐提高。在互助的过程中还能结识一些有着共同兴趣爱好的朋友,大家彼此交流,共同进步。

从本章开始,进入本书的主体部分,所讨论的话题很多时候都是围绕着“如何写出高效代码”这个主题,而对一些基本的 MATLAB 知识就不再做过多的陈述。

## 2.1 向量化编程流行的一些观点

MATLAB 向量(矢量)化编程,是一个伴随 MATLAB 发展的永恒的话题,是 MATLAB 语言的精髓所在,向量化编程运用好了,可以从代码运行效率明显改善中获得成功的快乐。传统的流行观点大致如下:

- 1) 尽量避免循环的使用,多使用 MATLAB 的内置函数。
- 2) 能用逻辑索引解决的就不用数值索引。
- 3) 使用变量前养成预分配内存的习惯。
- 4) 向量化计算代替逐点计算。
- 5) 能用普通数值数组完成的工作尽量不用 Cell 型数组。
- 6) 如果矩阵含有大量 0 元素,尽量采用稀疏矩阵来提高运算速度和减少存储空间。

MATLAB 在不断发展,随着版本的升级,有些向量化编程所描述的观点基本没变,但是有些观点就需要修正了。如果还按原来的观点为指导进行编程,那么很多情况下并不能得到效率明显的改善,甚至还不如按原来观点所不提倡的编程方式编写的代码效率高。

高版本的 MATLAB 向量化编程是一个比较复杂的问题,遵循的原则之间都有互相制约、相辅相成的关系。这些需要读者在实际应用中不断体会、感悟。下节开始,将讨论高版本的 MATLAB 高效编程注意事项。

## 2.2 重新认识循环

### 2.2.1 高版本 MATLAB 对循环结构的优化

从 MATLAB 6.5 版开始,MATLAB 引入了 JIT(just in time)技术和加速器(accelerator),并在后续版本中不断优化。到了 MATLAB R2009a,很多情况下,循环体本身已经不是程序性能提高的瓶颈了,瓶颈更多的来源于循环体内部的代码实现方式,以及使用循环的方式。循环就是多次重复做同一件事,如果这件事本身的代码写得不优化,放在循环体内多次实现后必然造成运行时间过长。

老版本的 MATLAB 对循环机制的支持不好,所以提倡避免循环,而高版本的 MATLAB 对循环机制的支持大大提高了,因此就不必再像过去那样谈“循环”色变了,不用千方百计避免循环了。当使用循环造成程序运行时间过长时,不要武断地将代码运行效率低归结到使用了循环。有的时候,我们千方百计地把一段代码向量化了,凭自己编程经验(很多是使用老版本积累下来

的经验)和常识(从老的教科书得到的常识)觉得程序很标准、很优化。殊不知,实际测量时会发现其性能不比采用很自然的想法实现的程序效率高多少,甚至还会降低。看下面这个例子:

**【例 2.2-1】** 运行下面的测试 JIT/ accelerator 的代码,体会高版本的 MATLAB 对循环的加速。

```
function JITAcceleratorTest
u = rand(1e6,1); % 随机生成一个 1 * 1000000 的向量
v = zeros(1e6,1);
tic
    u1 = u + 1;
time = toc;
disp(['用向量化方法的时间是:',num2str(time),'秒! ']);
tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['循环的时间是:',num2str(time),'秒! ']);

feature jit off;
tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['只关闭 jit 的时间是:',num2str(time),'秒! ']);

feature accel off;

tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['关闭 accel 和 JIT 的时间是:',num2str(time),'秒! ']);

feature accel on; % 测试完毕重新打开 accelerator 和 JIT
feature jit on;
end
```

运行结果如下:

```
用矢量化方法的时间是:0.0095308 秒!
循环的时间是:0.010176 秒!
只关闭 JIT 的时间是:0.084027 秒!
关闭 accel 和 JIT 的时间是:1.2673 秒!
```

自从引入 JIT 和 accelerator 后,MATLAB 对这两项功能默认都是打开的,这也是高版本 MATLAB 对循环支持好的原因。关闭 JIT 和 accelerator 需要用到 MATLAB 一个未公开

(undocumented)的函数:feature。feature accel on/off 即为打开/关闭 accelerator,类似的打开/关闭 JIT 是 feature jit on/off。

上面代码是计算一个长向量与一个标量的和,我们会发现,当 JIT 和 accelerator 都打开的状态下,循环和矢量化运算所需要的时间从统计意义上讲,已经没有显著差别了。关闭 JIT 后,运行时间变为原来的 8 倍左右,而再关闭 accelerator,运行时间立刻变为原来的 100 多倍。

再来看一个例子:

**【例 2.2-2】** 由一个  $m \times n$  的矩阵构造一个  $m \times (m+n-1)$  的矩阵。构造方式如下:以  $4 \times 4$  矩阵 **A** 为例,构造目的矩阵 **B**:

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

B =
     1     2     3     4     0     0     0
     0     3     4     5     6     0     0
     0     0     2     3     4     5     0
     0     0     0     1     3     4     6
```

例 2.2-2 中所涉及的问题用循环来解决非常容易实现。我们这里想要讨论的是,用向量化的方法解决该问题与用循环解决该问题两种运行时间的对比(还不考虑写出向量化的代码比用循环实现多花的时间)。

向量化思路:观察矩阵 **B**,发现,如果按行数的话,矩阵 **B** 的元素排列顺序是原来矩阵 **A** 中的每一行和下一行之间以四个 0 相隔。这样可以计算出矩阵 **A** 中的元素在矩阵 **B** 中相应的索引值(当然是按行)I,那么可以生成一个和矩阵 **A** 大小一样的全 0 矩阵 **B**,然后令  $B(I) = A(:)$ ;最后注意到 MATLAB 是按列的顺序遍历元素的,所以最后再将矩阵 **B** 转置。写成代码就是:

```
function B = rowmove(A)
    [m,n] = size(A);
    I = repmat(1:n,m,1) + repmat((0:m-1)' * (m+n),1,n);
    B = zeros(m+n-1,m);
    B(I(:)) = A(:);
    B = B';
```

循环的代码如下:

```
function C = LoopRowMove(A)
    [m,n] = size(A);
    C = zeros(m,m+n-1);
    for k = 1:m
        C(k,k:k+n-1) = A(k,:);
    end
```

可以看出,循环的思路以及操作非常简单,就是循环赋值操作。下面随机生成一个  $1000 \times 1000$  的矩阵,来用上述两种方法来比较一下运行速度:

```

A = rand(1000);
tic;B = rowmove(A);time = toc;
disp(['向量化求解时间是:',num2str(time),'秒!'])
tic;C = LoopRowMove(A);time = toc;
disp(['用循环求解时间是:',num2str(time),'秒!'])
if isequal(B,C)
    disp('两种方法结果完全一样')
end

```

运行结果如下：

```

向量化求解时间是:0.11389 秒!
用循环求解时间是:0.054231 秒!
两种方法结果完全一样

```

循环的时间反而不到向量化时间的一半,而且循环的代码要比向量化的代码容易写得多,算上“开发”时间,本例用循环比用向量化要高效的多。这是为什么呢?因为本例循环结构内部的代码仅仅为“赋值”这样简单的操作,不存在函数调用。目前在高版本的 MATLAB 中,循环本身往往不是程序瓶颈,反而函数调用产生的额外开销在很多情况下是构成程序瓶颈的因素之一,尤其是采用大量低效率的函数结构时(譬如 inline 对象,这个将在后续章节详细讨论)。

反观向量化操作,调用了两次 repmat 函数,至于 repmat 内部又做了些什么?有兴趣的话,可以运行 edit repmat 命令,查看 repmat 的源代码,会发现,里面有很多行代码都是这个问题没必要执行的。事实上,为了保证每个函数具有尽可能广的适应面,Mathworks 开发人员在编写函数时一般都在函数入口处做很多层判断,确保不同的输入能正确地执行相应的语句。

图 2.1 所示的窗口是用 MATLAB 的 Profiler 工具剖析“B=rowmove(A);”这条语句的结果。可以看出,两次调用 repmat 函数所占时间比例最高,其次是转置操作,这两项一共占去了整个程序的运行时间的 62%,而这两项对于本例来说完全都可以避免。而用循环实现时,只需要赋值 m 次即可,所用的系统开销相对要少。

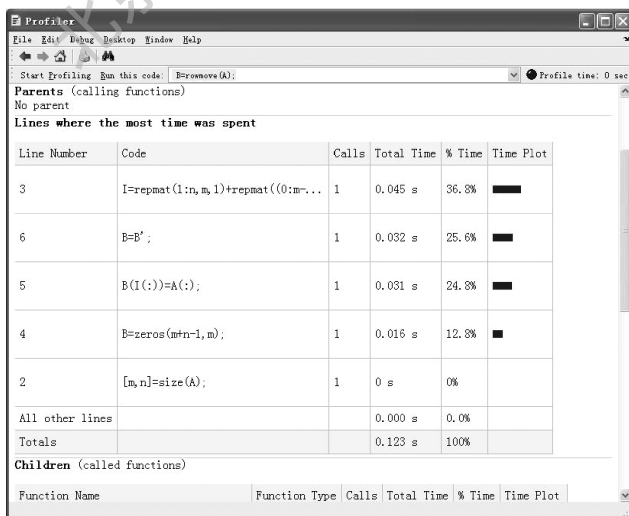


图 2.1 剖析例 2.2-2 所用的向量化方法性能

一些读者在使用 MATLAB 的过程中对向量化的高效率体会颇深,加上传统的书籍以及网络上流传的传统观点对循环的低效率的描述,导致其逐渐形成一种观点,就是只要是循环,效率就是低的。从上面的例子可以看到不能一味地否认循环,下节会讲到, MATLAB 效率低往往是由于发生大量函数调用以及算法本身运算次数多引起的。譬如计算多项式:

$$p_4(x) = a_1x^4 + a_2x^3 + a_3x^2 + a_4x + a_5$$

如果直接按照上述表达式按部就班进行的话,我们需要计算  $4+3+2+1=10$  次乘法以及 4 次加法,如果用下述表达式计算:

$$p_4(x) = (((a_1x + a_2)x + a_3)x + a_4)x + a_5$$

则只需要计算 4 次乘法和 4 次加法,随着多项式次数的增加,差距将越来越明显。请看下面的例子。

**【例 2.2-3】** 计算  $\sum_{k=1}^{10^6+1} k(1+10^{-6})^{10^6+1-k}$ 。

对于本例可以利用 sum 函数, polyval 函数以及根据前述将多项式拆开的算法自行写代码来实现:

```
clear
N = 1e6 + 1; k = [1:N]; x = 1 + 1e-6;
tic
p1 = 'sum(k.*x.^[N-1:-1:0]); % 方法 1: 用 sum 函数的方法
p1, toc
tic, p2 = polyval(k,x), toc % 方法 2: 调用 polyval 函数实现
tic, p3 = k(1);
for i = 2:N % nested multiplication
p3 = p3 * x + k(i);
end
p3, toc % 方法 3: 根据上述多项式求和算法实现
p1 =
    7.1828e+011
Elapsed time is 0.167499 seconds.
p2 =
    7.1828e+011
Elapsed time is 0.039696 seconds.
p3 =
    7.1828e+011
Elapsed time is 0.014916 seconds.
```

从上述运行结果来看,根据多项式求和算法自己编写并且还用到了循环的代码,所用的时间最短。值得一提的是: polyval 函数核心部分也是采用上述方法 3 中的多项式求和算法,但是为什么时间要稍长一些呢? 这是因为 polyval 函数比较通用化,里面有一些额外的代码考虑的是各种可能出现的情况,感兴趣的读者可以在命令窗口中运行“type polyval”查看其源代码运行即可明了。关于这方面的代码运行效率问题 2.3 节还会涉及。

下面再看一个算法本身引起的效率差异的例子,该例也是被很多网友经常问到的问题。

**【例 2.2-4】** 如何从  $1, 2, \dots, n$  这  $n$  个数字中随机选出  $m$  个不重复的数字。

一个比较容易想到的办法就是利用 randperm 函数,随机排列  $1, 2, \dots, n$  这  $n$  个数字,然后

取前  $m$  个。可是这样做在  $n$  较大而  $m$  较小时效率不高,因为只需要  $m$  个不重复的数字,却把所有的数字都重新排列了一下。可以利用下面的算法:

令  $a=[1,2,\cdots,n]$ ,对于  $i$  从 1 循环到  $m$ ,随机等概率地生成一个属于  $i$  到  $n$  的随机整数  $ind$ ,交换  $a[i]$  和  $a[ind]$  的值,循环完毕后取  $a[1]\sim a[m]$ 。

上述算法写成代码如下:

```
function r = randnchoosek(n,m)
% n: 数组,需要从中随机选取 m 个不重复的元素
% r: 数组 n 中随机选取的 m 个不重复的元素
ln = length(n);
for i = 1:m
    ind = i - 1 + unidrnd(ln - i + 1);
    a = n(ind);
    n(ind) = n(i);
    n(i) = a;
end
r = n(1:m);
```

对于  $n=10^6$ ,  $m=1000$  的情况,我们可以比较两种方法的运行速度:

```
clear;toc
N = 1e6;m = 1000;
rnN = randperm(N);
r1 = rnN(1:m);toc
tic;r2 = randnchoosek(1:N,m);toc
Elapsed time is 0.233197 seconds.
Elapsed time is 0.038107 seconds.
```

可见第二种方法比第一种快得多。

## 2.2.2 选择循环还是向量化

关于该用循环还是向量化,这是一个比较复杂的问题。笔者结合多年使用 MATLAB 经验给出的建议如下:

(1) 凡是涉及矩阵运算的时候,则尽量用向量化

这是因为,向量化编程是 MATLAB 语言的精髓,如果不熟悉其向量化编程的方法,则相当于没掌握这门语言。MATLAB 以矩阵为核心,MATLAB 卓越的矩阵计算能力是建立在 LAPACK 算法包和 BLAS 线性代数算法包的基础之上的。这两个算法包里的程序都是由世界上多个顶尖专家编写并经过高度优化了的程序。譬如算矩阵乘法的时候,就不要用循环了,因此时循环方法不仅运行效率大大低于 MATLAB 的矩阵乘法,而且开发效率也非常低。

需要说明的是,本原则是建立在参与运算的矩阵不太大,能够适应系统物理内存的条件下。

(2) 如果向量化可能导致超大型矩阵的产生,使用前要慎重

向量化操作获得的高效率很多时候是以空间换时间实现的。

具体说来就是把数据以整体为单位,在内存中准备好,从而使得 MATLAB 内建的高效函数能批量处理。如果处理的数据量很大,譬如一些图像数据、地质数据、交通数据等等,那么可

能导致其准备过程以及存储这些矩阵耗费数百兆乃至千兆(吉)甚至更多的内存空间,如果超过系统物理内存空间,那么会造成效率的低下。原因如下:

第一,数据准备过程也是需要耗费资源的,事先存储大规模的矩阵,势必造成留给 MATLAB 计算引擎乃至操作系统的物理内存大大减少,如果系统内存不够大,势必大大影响计算效率。

第二,如果参与运算的矩阵尺寸过大,可能导致虚拟内存的使用,那么当对这块矩阵进行读写和计算时可能涉及频繁的内存与外存交换区的输入/输出,会造成效率的急剧下降。这时应该对程序进行重新思考和设计,后面章节会就 MATLAB 如何有效处理大规模矩阵进行专门讲解。

(3) 向量化的使用要灵活,多分析其运行机制,并与循环做对比

在网上经常看到一些 MATLAB 使用者视 MATLAB 的循环为“洪水猛兽”,千方百计避免循环,不管什么程序都要想尽办法将其向量化。好像只要向量化,速度就会神奇的上,可是,向量化有些时候会增加实际运算次数,这往往出现在那些不适合向量化的过程中。这样,即使绞尽脑汁、生搬硬套地利用一些向量化技巧、向量化的函数让操作变成矩阵运算,但是增加的无用计算使得即便是更高效的引擎也无法挽回损失。例 2.2-2 就很好地证明了这一点。

(4) MATLAB 初学者要尽量多用向量化思路编程

MATLAB 是一门灵活性与高效性结合的非常好的语言。我们不能过分强调、夸大向量化编程的优势,当然更不能把 MATLAB 当成 C/C++ 等语言来使用。MATLAB 初学者一定要提醒自己,处处想尽办法矢量化编程,而暂时先不管矢量化后会不会得到性能大幅提高或者应不应该向量化。

这是因为,只有当你充分熟悉 MATLAB 语言特点并且熟悉其最常用的内置函数之后才能灵活运用向量化和非向量化。向量化编程往往需要程序编写者熟悉 MATLAB 的常用内置函数、常用技巧,并且在向量化一个程序的过程中有效锻炼其观察、分析、概括、抽象、归纳的能力。

经过大量向量化编程的训练后, MATLAB 学习者能够更加深入、细腻地感受 MATLAB 语言的优缺点,能够积累大量的内置函数使用经验以及常用的向量化技巧。既能从高效、优美、简洁的向量化代码中获得意想不到提速带来的快乐,也能遇到经过向量化反而没有提高效率的困惑,从而迫使自己进一步加深对 MATLAB 语言的了解。

只有当对 MATLAB 向量化函数、技巧都了解的时候,才能为以后高效的利用 MATLAB 编程打下坚实基础。

(5) 高效的编程包括高效的开发和高效的运行

我们应该明白自己的程序目的是要干什么。如果仅仅是验证某个东西,程序最多运行几次以后就不会再用了。那么这个时候高效的编程就是怎么能够尽快实现程序的功能,让程序尽快在可接受的时间内跑出正确的结果。

如果我们采用自己容易想到的思路实现这个程序,程序运行几秒就可出结果。而我们为了优化这个程序多花的时间远远大于优化后的程序所节约的时间,这是得不偿失的。

## 2.3 提高代码效率的方法

### 2.3.1 预分配内存

当矩阵较大时,预分配内存(preallocation),是 MATLAB 高效编程中最应该注意,但其也



是很容易把握的一个技巧。看下面的例子：

**【例 2.3-1】** 比较预分配和不预分配内存对程序运行速度的影响。

```
function PreAllocMemVSnot
n = 30000;
tic;
for k = 1:n
    a(k) = 1;
end
time = toc;
disp(['未预分配内存下动态赋值长为',num2str(n),'的数组时间是:',num2str(time),'秒!'])
%%
tic
b = zeros(1,n,'double');
for k = 1:n
    b(k) = 1;
end
time = toc;
disp(['预分配内存下赋值长为',num2str(n),'的数组时间是:',num2str(time),'秒!'])
```

运行结果如下：

```
>> PreAllocMemVSnot
未预分配内存下动态赋值长为 30000 的数组时间是:0.58989 秒!
预分配内存下赋值长为 30000 的数组时间是:0.0004281 秒!
```

速度相差 1000 倍之巨！例子中仅仅是一个长为 30 000 的数组，数组的长度越大，这种差距越明显。造成上述速度巨大差距的原因是什么呢？原因是 MATLAB 动态扩充数组在内存中是按图 2.2 所示的过程进行的。

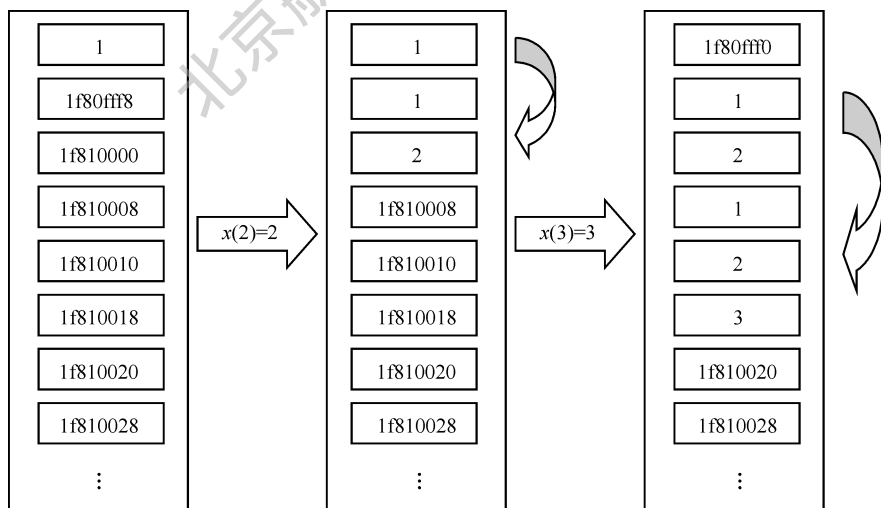


图 2.2 MATLAB 动态扩充数组内存实现方式

图 2.2 描述的是如下代码在内存中的实现方式：

```
>> x = 1;
>> x(2) = 2;
>> x(3) = 3;
```

图中的 16 进制数字代表内存地址,现在以 1,2,3 来依次给  $x$  动态扩充赋值,由于 MATLAB 默认是以双精度来表示数字的,因此每个数字占 8 字节(8B)。可以看出,每次赋值, MATLAB 都会在内存中重新开辟一块和当前赋完值之后的  $x$  的尺寸相等的连续区域来存储  $x$ ,并把原来  $x$  存储的元素赋值到新的相应区域。可以想象,当动态赋值一个大型数组时,会有大量的时间耗费在无谓的内存读写上,数组越长,耗费的时间增长得越快。而事先向量化呢? 即

```
>> x = zeros(3,1,'double');
>> x(1) = 1;
>> x(2) = 2;
>> x(3) = 3;
```

图 2.3 展示了这一过程。

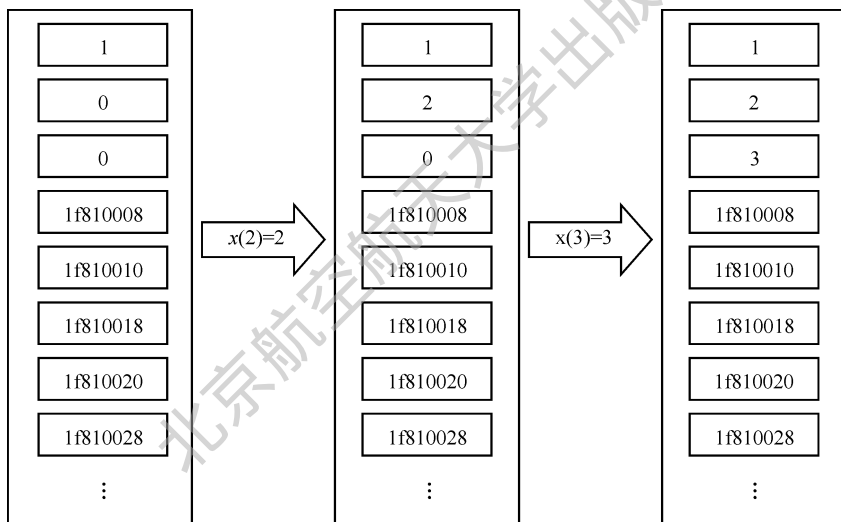


图 2.3 MATLAB 预分配内存后赋值实现方式

由图 2.3 可以看出,预分配内存后,不会有时间浪费在无谓的内存读写上,效率得到了大大提高。

细心的读者可能已经注意到了前面预分配内存的程序,譬如  $b = \text{zeros}(1,n,'double')$ ; 和我们常见的预分配内存  $\text{zeros}$  用法有些不同,多了一个数值类型输入参数,其实这是 MATLAB R2009a 又一个改进的地方。很多人没有注意这点,  $b = \text{zeros}(1,n,'double')$  或者  $b(1,n) = 0$  效率要远远高于  $b = \text{zeros}(1,n)$ , 这是因为前者并不在物理内存中真正写入这样一个矩阵,而是在 MATLAB 进程的虚拟内存空间中声明所需要的一片连续地址,当对生成的矩阵进行操作时,写入物理内存操作才真正发生;而运行后者  $b = \text{zeros}(1,n)$ , 在 MATLAB 的虚拟内存空间中声明以及写入物理内存都会发生,所以效率比较低。有兴趣的读者可以用 R2009a 版本试试下面的代码(多运行几次看时间差别):

```
clear;
tic;a(10000,5000) = 0;toc;clear
tic;a = zeros(10000,5000);toc;clear
tic;a = zeros(10000,5000,'double');toc;
```

经过笔者验证,包括 MATLAB 2008a 在内的以前的 MATLAB 版本都不具有这个性质,笔者没有验证 MATLAB R2008b 版本,有兴趣的读者可以自行去验证。

### 2.3.2 选用恰当的函数类型

MATLAB 有多种函数类型,不同的函数类型调用效率差别相当大,为了写出高效的 MATLAB 代码,我们要选用恰当的函数类型来完成我们的任务。下面将以一个实用的例子简要介绍一下 MATLAB 常用的一些函数类型,并比较其效率,相关函数类型的详细介绍将放在后面章节进行。

**【例 2.3-2】** 求  $f(k) = \int_0^5 \sin(kx)x^2 dx$ ,  $k$  取  $[0,5]$  区间不同值时的函数值,并画出函数图像。该例的意义在于其求解方法对于一些复杂的、无显式积分表达式的带参数积分问题具有通用性。

本例关键就是积分表达式中不同的  $k$ ,如何让 MATLAB 识别的问题。初学者对这个问题爱用符号积分,诚然本例函数关于  $x$  具有显式原函数表达式,符号积分可以奏效,但是现实中很多问题的积分表达式很复杂,是没有显示原函数表达式的,这时候符号积分就行不通了,只能用数值积分。下面的解法中就这个问题分别用 inline 函数类型、匿名函数类型、嵌套(nested)函数类型进行了求解。考虑到一般熟悉 MATLAB 的读者对普通子函数都比较了解,因此没有给出普通子函数的解法。读者完全可以参照嵌套函数的方法自己尝试用普通子函数实现。求解这个问题的代码如下:

```
function InlineSubAnonymousNestedDemo
%% 用 inline 解决
tic;
k = linspace(0,5);
y1 = zeros(size(k));
for i = 1:length(k)
    kk = k(i);
    fun = inline([' sin(',num2str(kk),' * x) . * x.^2 ']);
    y1(i) = quadl(fun,0,5);
end
time = toc;
disp(['用 inline 方法的时间是:',num2str(time),'秒! '])
%% 用 anonymous function 解决
tic;
f = @(k) quadl(@(x) sin(k.*x). * x.^2,0,5);
kk = linspace(0,5);
y2 = zeros(size(kk));
for ii = 1:length(kk)
    y2(ii) = f(kk(ii));
end
```

```

time = toc;
disp(['用 anonymous function 方法的时间是:',num2str(time),'秒!'])
%% 用 nested function 解决
function y = ParaInteg(k)
    y = quadl(@(x) sin(k. * x). * x.^2 ,0,5);
end
tic;
kk = linspace(0,5);
y3 = zeros(size(kk));
for ii = 1:length(kk)
    y3(ii) = ParaInteg(kk(ii));
end
time = toc;
disp(['用 nested function 方法的时间是:',num2str(time),'秒!'])
%% 用 arrayfun + anonymous function 解决
tic;y4 = arrayfun(@(k) quadl(@(x) sin(k. * x). * x.^2,0,5),linspace(0,5));time = toc;
disp(['用 arrayfun + anonymous function 方法的时间是:',num2str(time),'秒!'])
plot(kk,y2);
xlabel('k');
ylabel('f(k)')
end

```

运行这个程序,其结果如下:

```

用 inline 方法的时间是:0.96154 秒!
用 anonymous function 方法的时间是:0.13904 秒!
用 nested function 方法的时间是:0.1368 秒!
用 arrayfun + anonymous function 方法的时间是:0.13907 秒!

```

得到的图形如图 2.4 所示。

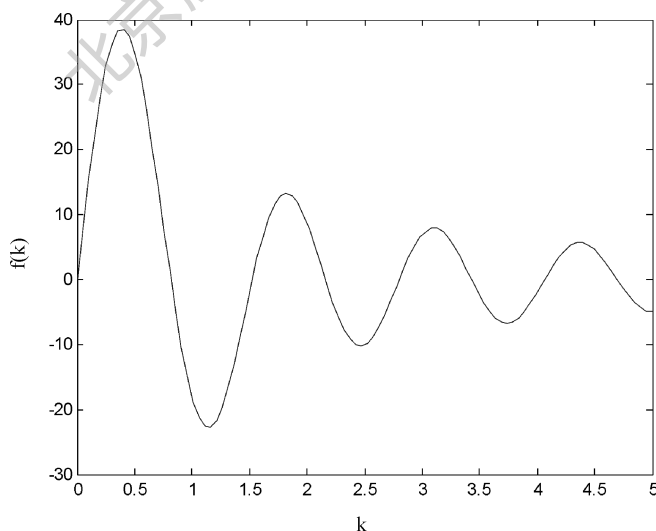


图 2.4  $f(k) = \int_0^5 \sin(kx)x^2 dx$  函数图像

从例 2.3-2 可以看出,匿名函数和嵌套函数的效率要明显高于 inline 函数,而且在参数传递方面要比 inline 函数方便高效。不熟悉匿名函数或者嵌套函数的读者可能对匿名函数或者嵌套函数的用法理解得不是很透彻,这没关系,对其的详细介绍将放在后续章节进行,通过本例读者只要能建立起对匿名函数和嵌套函数的初步印象即可。本例中还有用 arrayfun 的示例,arrayfun 最早出现在 MATLAB 7.1 版中,其功能是将一特定函数(用户指定函数句柄)应用到一数组的每个元素上,返回值是和输入数组尺寸一样大的数组,返回数组和输入数组对应元素是相应的函数输出。这里的数组不仅包括数值型的还可以是元胞(cell)型和结构(struct)型的。利用 arrayfun,可以简写很多循环的代码,后面章节有对其专门的介绍。类似的函数还有 structfun、cellfun、spfun 等。

这里需要强调的是,inline 对象作为 MATLAB 7.0 之前构造简短函数常用的结构,在 7.0 之后的版本中已经过时了,取而代之的是匿名函数(anonymous function),inline 函数能实现的,匿名函数完全可以更好地实现,并且调用效率要比 inline 函数高得多。而且用匿名函数很多可以方便实现的功能,用 inline 函数却很难方便实现。

这里先粗略比较一下 MATLAB 中各种函数结构的调用效率,说明选择好函数结构的重要性。下面以  $f(x)=x$  这个简单的函数为例,来比较内联(inline)函数、匿名(anonymous)函数、嵌套(nested)函数、普通子函数(sub-function)的调用效率。

**【例 2.3-3】** 编写  $f(x)=x$  相应的内联函数、匿名函数、嵌套函数、普通子函数形式代码,并分别调用 10 000 次,比较各种函数结构调用时间的差异。

比较的代码如下:

```
function InlineSubAnonymousNestedCallDemo
% 各种类型函数调用效率比较
n = 10000;
f1 = inline('x'); % f(x) = x 的 inline 形式
f2 = @(x) x; % f(x) = x 的 anonymous function 形式
function f3 = f3(x) % f(x) = x 的 nested function 形式
    f3 = x;
end
%% inline 的调用效率
tic
for k = 1:n
    f1(1);
end
time = toc;
disp(['f(x) = x 的 inline 形式调用',num2str(n),'次时间是:',num2str(time),'秒!'])
%% anonymous function 的调用效率
tic
for k = 1:n
    f2(1);
end
time = toc;
disp(['f(x) = x 的 anonymous function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒!'])
```

```

%% nested function 的调用效率
tic
for k = 1:n
    f3(1);
end
time = toc;
disp(['f(x) = x 的 nested function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒! '])
%% sub-function 的调用效率
tic
for k = 1:n
    f4(1);
end
time = toc;
disp(['f(x) = x 的 sub-function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒! '])
end

function f4 = f4(x) % f(x) = x 的 sub-function 形式
f4 = x;
end

```

运行结果如下：

```

f(x) = x 的 inline 形式调用 10000 次时间是:1.1487 秒!
f(x) = x 的 anonymous function 形式调用 10000 次时间是:0.0060664 秒!
f(x) = x 的 nested function 形式调用 10000 次时间是:0.0042187 秒!
f(x) = x 的 sub-function 形式调用 10000 次时间是:0.0034907 秒!

```

由此可以看出,匿名函数、嵌套函数和普通子函数的调用效率基本上相差不太大,而内联函数的调用效率是上述几种函数类型调用效率的几百分之一!

因此在使用高版本 MATLAB 编程的过程中,要坚决摒弃 inline 这种过时的函数结构,而要根据需要选用匿名函数或者嵌套函数以及子函数。一般说来,简短的数学表达式函数或者可以写成一行代码的函数可以选用匿名函数来实现;而复杂的,内部要做较多事情的函数可以选用子函数和嵌套函数来实现;其中在变量共享以及传递方面,嵌套函数具有较大的优势,尤其是用子函数实现并且用到全局变量的场合,用嵌套函数可以避免用全局变量,并且实现起来方便容易。后续专门讨论嵌套函数的章节会详细介绍嵌套函数的用法。

### 2.3.3 选用恰当的数据类型

在 2.3.2 节讨论了不同的函数结构实现相同功能之间的效率差异。MATLAB 同样有多种数据类型,不同的数据类型在存储和访问效率上也不尽相同,选用恰当的数据类型对写出高效的 MATLAB 代码同样具有重要的意义。下面就最常用的三种数据类型:double 型数组、cell 数组、struct 数组,讨论其访问效率。看下面的例子:

**【例 2.3-4】** 比较 double 型数组、cell 数组和 struct 数组的访问效率。

```
function VisitSpeedOfDifferentDataTypes
n = 30000;
a = 8;
b{1} = 8;
c.data = 8;
%%
tic;
for k = 1:n
    a;
end
time = toc;
disp(['访问',num2str(n),'次 double 型数组时间是:',num2str(time),'秒! '])
%%
tic;
for k = 1:n
    b{1};
end
time = toc;
disp(['访问',num2str(n),'次 cell 型数组时间是:',num2str(time),'秒! '])
%%
tic;
for k = 1:n
    c.data;
end
time = toc;
disp(['访问',num2str(n),'次结构数组时间是:',num2str(time),'秒! '])
```

运行结果如下：

```
访问 30000 次 double 型数组时间是:0.00010113 秒!
访问 30000 次 cell 型数组时间是:0.0069649 秒!
访问 30000 次结构数组时间是:0.00011091 秒!
```

由此可以看出,就访问效率来说,double 型数组和 struct 数组的访问效率相当,大约都是 cell 型数组的 70 倍。

就在内存中存储来说,即使空的 cell 型数组,每个 cell 单元也要占用 4 B 的内存空间。这是因为,cell 型的数组不需要连续内存空间来存储,因此,每个 cell 单元需要记录其在内存中位置等一些信息,每个 cell 单元无论空与否,都在内存中占有固定的 4 B 用来存储这些信息。而一个 1 \* 1 的 struct 数组,每一个域无论空与否,都会占用 124 B 的内存空间。

综合占用内存大小,以及访问效率等因素,在设计程序时应尽量采用 double 型数组。

选择好数据类型后,我们还要注意高效的访问方法。就 double 型数组来说,由于 MATLAB 是按列顺序存储数据的,因此按列访问、遍历数值数组要比按行访问、遍历快。看下面的例子:

```
function ColLoopVsRowLoop
n = 3000;
a = rand(n);
tic;
```

```

for ii = 1:n
    for jj = 1:n
        a(ii,jj);
    end
end
toc

tic;
for ii = 1:n
    for jj = 1:n
        a(jj,ii);
    end
end
toc

```

运行结果如下：

```
Elapsed time is 0.136464 seconds.
```

```
Elapsed time is 0.069589 seconds.
```

可见按列遍历比按行快一倍左右。

### 2.3.4 减少无谓损耗——给一些函数“瘦身”

编程者要清楚,在 MATLAB 程序完成一个特定任务时,哪些是完成这个特定任务必须执行的代码,哪些是不必须的。举例来说,MATLAB 自身以及工具箱里的很多函数具有较广的通用性,所谓通用性是添加必要的代码能顾及到一个个具体的情形,这样对于每个具体的情形,代码实际上都有冗余。如果我们的程序某些部分对通用性要求不高,那么在优化程序时要考虑将不必要的代码去掉,这样的结果就是重写一些简单的函数,轻装上阵,举例如下:

**【例 2.3-5】** median 函数的“瘦身”。

有兴趣的读者可以在命令窗口中运行“edit median”或者“type median”看一看 MATLAB 自带的“median”函数的源代码。

读者会发现,“median”函数源代码比较长,而且为了保证函数具有较广的通用性,源代码里各种分支判断和类型判断较多。这对于特定的某个求中位数的函数来说,可能很多代码都是不必要执行的,因为求中位数,最核心的是排序程序段,而且求中位数也不难自己实现,因此针对特定问题我们可以自己写“瘦身的 median 函数”代码。譬如程序中可能会频繁求一个维数为 1 000 左右的向量的中位数,可以写如下的函数:

```

function b = mymedian(a)
a = sort(a);
n = length(a);
half = floor(n/2); % 若非整数则向下取整,整数维持不变
b = a(half + 1);
if half * 2 == n % 若 n 为偶数,b 等于排好序的中间两个数平均值
    b = (b + a(half))/2;
end

```



做如下测试,测试和运行结果如下:

```
>> clear
a = rand(1000,1);
tic;for k = 1:10000;c = median(a);end;toc
tic;for k = 1:10000;d = mymedian(a);end;toc
Elapsed time is 0.859531 seconds.
Elapsed time is 0.690383 seconds.
>> isequal(c,d)
ans =
1
```

可见“瘦身”后的 mymedian 函数运行速度提高了 20% 左右。这是因为“瘦身”后的 mymedian 函数考虑的情况及其分支较 median 函数少。需要说明的是,当 a 的长度很大时,执行分支判断语句的时间远小于排序本身的时间,median 和 mymedian 的时间差距不会太明显。但是当 a 的长度较小时,执行排序的时间小于执行额外语句的时间时,这种差距就明显了。同样上述代码,不同的是 a 的长度是 100 了,两个函数运行时间差距就大了,测试和运行结果如下:

```
>> a = rand(100,1);
tic;for k = 1:10000;c = median(a);end;toc
tic;for k = 1:10000;d = mymedian(a);end;toc
Elapsed time is 0.228408 seconds.
Elapsed time is 0.060108 seconds.
>> isequal(c,d)
ans =
1
```

这时候瘦身的函数所用时间仅为原始 median 的 1/4 多点。

本例的启示是在通用性要求不高的场合,可以根据我们的需要来给 MATLAB 函数瘦身,当然要权衡瘦身的难易程度和获得的效率提升程度。

### 2.3.5 变“勤拿少取”为“少拿多取”

通过前面章节的介绍,我们可以了解如下事实:

在 MATLAB 中,函数调用都要有一定的开销,这种开销一般是 built-in (像 sin, cos, zeros, ones, find, all, any 等,但凡在命令窗口运行“type 文件名”,MATLAB 会显示“XXX is a built-in function.”这样的函数)为最低,其次是一般的 M 文件,即工具箱里能看到源代码的那些 M 函数以及用户自己写的 M 文件、子函数、nested function、匿名函数等。上面这些函数类型总的来说调用开销都比较小,而调用开销最大的是 inline 函数类型(在高版本的 MATLAB 中应当摒弃 inline 这种落后的函数类型)。函数调用开销相对于一些简单的计算来说,往往是相当可观的,即使是使用 built-in 函数类型。

另外我们还注意到为了使得函数具有通用性,很多函数刚开始都会有一些判断、准备工作,往往判断好几层才到了真正算法实现部分。在 2.3.4 节,讨论了“函数瘦身”所带来的运行速度上的提升。可是 MATLAB 作为方便易用的科学计算语言,其优势之一就是集中了大量现成的功能强大的函数,如果所用 MATLAB 自带函数的核心算法实现较复杂,那么为“函数瘦身”就可能得不偿失了。这时候为了尽可能多地降低函数调用而产生的无谓的损耗,就要想

尽办法把需要多次函数调用才能完成的工作用较少次数的函数调用来完成。我们形象地称之为：变“勤拿少取”为“少拿多取”。下面以一个例子来说明：

**【例 2.3-6】** 数组 **A** 是一个  $7 \times 1000000$  的矩阵，每行的形式类似下面所示：

```
A=[100 2 6 96 8 7 20;
15 69 7 6 4 20 11;
21 101 45 48 6 5 4;
...];
```

**b** 是一维数组 [4 6 8]；

试找出数组 **A** 中的每一行与数组 **b** 的交集，并将 **b** 中的交集元素置零，并将置零后的 **b** 存到 **B** 中。**B** 的形式为 [4 0 0; 0 0 8; 0 0 8; ...]。

本例如果按照常规思路做的话，可能会想到下面的办法：

```
function example2_3_6slow
A = unidrnd(100,1000000,7); % 随机生成 1000000 * 7 的 A 矩阵，A 的元素属于 1 到 100 的整数
B = zeros(1000000,3);
for m = 1:1000000
    a = A(m,:);
    b = [4 6 8];
    for ii = 1:3
        dd = a(a == b(ii)); % dd: a 中等于 b(ii) 的元素
        if isempty(dd) == 0 % dd 不为空
            b(ii) = 0;
        end
    end
    B(m,:) = b;
end
```

仔细分析上面的代码会发现，需要遍历 **A** 的每一行，判断 **A** 的每一行是否有元素等于 **b** 的某个元素，如果有的话，就把相应的 **b** 的这个元素置为 0。这样对本例来说，光 isempty 就调用了 300 万次，可以想象，效率比较低。如果采用“少拿多取”的思想，可以做如下改进：

```
function example2_3_6fast1
clear
A = unidrnd(100,1000000,7); % 这里先假设 A 是一个随机矩阵
B = repmat([4,6,8],1000000,1);
tic; C = [any(ismember(A,4),2) any(ismember(A,6),2) any(ismember(A,8),2)];
B(C) = 0;
toc
```

或者改进如下：

```
function example2_3_6fast2
clear
A = unidrnd(100,1000000,7);
B = repmat([4,6,8],1000000,1);
tic; C = [any(AA == 4,2) any(AA == 6,2) any(AA == 8,2)];
B(C) = 0;
toc
```

上述两个程序的共同之处在于：一次处理很多数据。这样，原本要调用 300 万次的“比较”操作，现在只调用三次，只不过每次需处理整个  $A$  矩阵。然后结合 any 函数给出最后要被置零的逻辑索引。

上面代码不难看懂，关键是上面叙述的编程理念。

### 2.3.6 循环注意事项

很多人编写循环时，往往忽略很重要的事实，那就是同样循环层数、次数，不同的写法可能造成速度巨大的差异。编写循环时应该遵循两个重要的原则：

- 1) 按列优先循环(因为 MATLAB 按列循序存储矩阵)；
- 2) 循环次数多的变量安排在内层。

我们以具体例子来说明上述原则的重要性。请看示例：

**【例 2.3-7】** 对于  $n \times n$  的矩阵  $A$ ，生成一个矩阵  $B$ ，使得其元素满足下列关系式：

$$B(i,j) = \left( \sum_{\substack{k=1 \leq k \leq i+1 \\ j-1 \leq l \leq j+1}} A(k,l) \right) / 9, \quad 2 \leq i \leq n-1, \quad 2 \leq j \leq n-1$$

对于本例，可以用多种方法来求解。由于现在是在讨论循环注意事项，因此用循环的方法来处理，而且是用四重循环。值得注意的是，用这么多层循环，在老版本时代，历来是被强烈反对的。我们看看在 MATLAB R2009a 平台下，不同的写法效率能有多大差距。读者可以用二重循环或者想尽办法避免循环试一试，并和效率高的四重循环做个对比(当然平台是 MATLAB R2009a)。下面以一个  $512 \times 512$  的随机矩阵  $A$  为例来说明。

先看方法一：

```
function y = ForLoopCompare1(x)
y(512,512) = 0;
tic;
for i = 2:511
    for j = 2:511
        for k1 = -1:1
            for k2 = -1:1
                y(i,j) = y(i,j) + x(i+k1,j+k2)/9;
            end
        end
    end
end
end
toc
```

本方法将循环次数少的放到了内循环，外循环依次是  $i, j$ ，并且是按行优先循环。

再看第二种方法：

```
function y = ForLoopCompare2(x)
y(512,512) = 0;
tic;
for j = 2:511
    for i = 2:511
        for k2 = -1:1
```

```

        for k1 = -1:1
            y(i,j) = y(i,j) + x(i + k1, j + k2)/9;
        end
    end
end
end
toc

```

本方法同样将循环次数少的放到了内循环,不同的是内外循环都是列优先。

第三种方法:

```

function y = ForLoopCompare3(x)
y(512,512) = 0;
tic;
for k1 = -1:1
    for k2 = -1:1
        for i = 2:511
            for j = 2:511
                y(i,j) = y(i,j) + x(i + k1, j + k2)/9;
            end
        end
    end
end
end
toc

```

本方法将循环次数少的放到了外循环,内外循环都是按行优先循环。

第四种方法:

```

function y = ForLoopCompare4(x)
y(512,512) = 0;
tic;
for k2 = -1:1
    for k1 = -1:1
        for j = 2:511
            for i = 2:511
                y(i,j) = y(i,j) + x(i + k1, j + k2)/9;
            end
        end
    end
end
end
end
toc

```

该方法不仅将循环次数少的放到了外循环,而且内外都是按照列优先的循环顺序。下面是这四种方法运行效率对比:

```

x = rand(512);
y1 = ForLoopCompare1(x);
y2 = ForLoopCompare2(x);
y3 = ForLoopCompare3(x);

```

```

y4 = ForLoopCompare4(x);
Elapsed time is 0.330648 seconds.
Elapsed time is 0.281970 seconds.
Elapsed time is 0.322625 seconds.
Elapsed time is 0.057423 seconds.

```

由此可见,同时遵循“按列优先循环”、“循环次数多的变量安排在内层”两个原则的方法耗时最少。

### 2.3.7 逻辑索引和逻辑运算的应用

我们知道,MATLAB的矩阵元素索引有两类:一类是数值索引,另一类是逻辑索引。前者又可分为线性索引(linear index)和下标索引(subscripts)。就运行效率来说,逻辑索引要高于数值索引,所以访问矩阵元素能用逻辑索引的就用逻辑索引。

这里需要强调的是 find 函数返回的是数值索引,因此,如果不需要对返回的索引值做进一步的操作,只是单纯寻找满足条件的一些元素的话,find 函数完全可以不用。看下面的例子:

**【例 2.3-8】** 随机生成一个  $1000 \times 1000$  的矩阵  $A$ ,  $A$  的元素服从  $[0,1]$  区间均匀分布,并找出  $(0.3,0.7)$  区间的所有元素。

请看用数值索引和逻辑索引的运行效率对比:

```

A = rand(1000);
tic;B = A(find(A > 0.3 & A < 0.7));toc
tic;C = A((A > 0.3 & A < 0.7));toc
Elapsed time is 0.056147 seconds.
Elapsed time is 0.047591 seconds.

```

关于逻辑运算,MATLAB 提供了一系列内置的逻辑运算函数,这些函数运行效率往往比较高,可以根据需要进行选用。这些函数通过各自函数帮助文档页面下面的 see also 选项里列出的函数彼此连接起来,这样可以减少使用时候的记忆负担。譬如我们运行 doc any,帮助文档里除了列出 any 函数的使用说明外,在下面的 see also 选项里还列出和 any 一类的其他函数,这些函数的帮助文档下面又列出和自身关系密切的其他函数。

此外,在命令窗口中运行“doc is”可以列出一系列有关判断的逻辑函数。这些函数的说明文档都比较容易明白,由于篇幅关系,这里对每个函数的功能就不做介绍了,读者可以看看这些函数的帮助文档。

## 2.4 应用高版本向量化函数提高开发效率

从 MATLAB 7.0 开始,陆续增加了一些向量化函数,使用这些函数可以减少很多循环的使用,在保证代码运行效率的前提下,使代码更加简洁。需要说明的是,很多情况下,使用这些函数其运行效率并不比恰当使用循环快多少,使用它们主要是为了提高开发效率,使代码更加简洁。下面逐一介绍这些函数。

### 2.4.1 accumarray 函数

accumarray 函数最早出现于 7.0 版(R14),在随后的 7.1 (R14SP3), 7.2 (R2006a)版里

又对其功能进行了增强。

常用的 `accumarray` 函数的调用格式有以下几种：

```
A = accumarray(subs,val)
A = accumarray(subs,val,sz)
A = accumarray(subs,val,sz,fun)
A = accumarray(subs,val,sz,fun,fillval)
A = accumarray(subs,val,sz,fun,fillval,issparse)
```

下面以 `accumarray` 函数最全的参数调用格式 `A = accumarray(subs, val, sz, fun, fillval, issparse)` 为例,说明一下 `accumarray` 函数最常用的使用方法。

给定一组索引值 `subs`,一般要求其是  $n$  行  $m$  列的正整数矩阵, `val` 是一长度为  $n$  的数组(行数组或者列数组), `subs` 的每一行表示一个索引值,与 `val` 相应的元素对应。`accumarray` 函数的功能是将索引值相同的 `val` 分别分组,然后用事先指定的函数 `fun` 作用于每一组,输出的结果是一个矩阵 **A**,矩阵 **A** 的维数等于 `subs` 的列数,

如果 **A** 的某个元素对应的索引值(即 **A** 通过该索引值可以访问该元素,譬如  $\mathbf{A} = \begin{bmatrix} 0.2 & -5; 7 & 8 \end{bmatrix}$ ,对于  $-5$  来说,其索引值就是  $[1, 2]$ ,表示第一行第二列, $\mathbf{A}(1, 2)$  可以访问到  $-5$  不曾在 `subs` 里出现过,那么 **A** 的这个元素只能是 0(默认),或者是我们事先指定的 `fillval`。如果出现过,则 **A** 的这个元素是相应的 `fun` 作用的 `val` 中对应元素的值。我们可以为 **A** 重新指定尺寸: `sz`,只是 `sz` 的每个分量都不小于对应的 `subs` 列中最大的值。 `issparse` 参数是来说明是否以稀疏矩阵的形式输出 **A**。

`accumarray` 函数用法比较抽象,很多读者包括一些熟练使用 MATLAB 的朋友即使看了英文帮助文档,还是对其用法不甚明白。所以看完上述解释后如果还是一头雾水,那么不要紧,可以结合下面的例子来看。

下面看几个关于 `accumarray` 函数的例子:

**【例 2.4-1】** 生成一个  $100\,000 \times 1$  的 **a**,其元素服从  $[1, 200\,000]$  之间离散均匀分布,找出都有哪些元素出现了。

先用下列语句生成 **a**:

```
a = unidrnd(200000,100000,1);
```

下面有几种办法找到 **a** 中都有哪些元素出现了。如下(环境:MATLAB R2009a+Windows XP sp2):

方法 1:

```
tic;b = union(a,a);toc;
Elapsed time is 0.033577 seconds.
```

方法 2:

```
tic;c = unique(a);toc;
Elapsed time is 0.015064 seconds.
```

## 方法 3:

```
tic,d=accumarray(a,1,[200000,1]);e=find(d);toc
Elapsed time is 0.009958 seconds.
```

由于 MATLAB 第一次运行函数会有编译等额外工作,因此第一次运行较慢,上述运行时间都是取的第三次运行的时间。可以看出使用 accumarray 函数的方法 3,不仅得到了哪些元素出现了(即向量  $e$ ),同时还给出了每个出现的元素出现的次数——向量  $d$ ,所用时间也是最少的。

**【例 2.4-2】** 在  $1000 \times 1000$  的正方形区域内随机生成 100 000 个点(坐标值是整数),统计每个坐标点上生成的点的个数。

在这个例子下,像例 2.4-1 一样简单应用 union 和 unique 就不行了。通常考虑用循环:

```
clear;
p=unidrnd(1000,100000,2);
tic;
A(1000,1000)=0;
for k=1:100000
A(p(k,1),p(k,2))=A(p(k,1),p(k,2))+1;
end
toc
Elapsed time is 0.020455 seconds.
```

而用 accumarray 函数可以这样:

```
tic;a=accumarray(p,1,[1000,1000]);toc
Elapsed time is 0.017606 seconds.
isequal(A,a)
ans =
1
```

可见,用 accumarray 函数,仅需一句代码,既简洁又高效。

**【例 2.4-3】** 1 000 人,身高分布在 170~180 cm,体重在 110~200 斤,年龄分布在 20~50 岁,计算身高体重都相等的人的年龄平均值。结果用矩阵来表示:行数表示身高,列数表示体重,矩阵元素表示年龄的平均值。

首先,生成数据:

```
rand('state',0)
height=unidrnd(10,1000,1)+170;%身高的数据
rand('state',0)
weight=unidrnd(90,1000,1)+110;%体重数据
rand('state',0)
old=unidrnd(30,1000,1)+20;
```

利用 accumarray 函数计算的语句如下:

```
tic;mo=accumarray([height,weight],old,[],@mean);toc
Elapsed time is 0.002271 seconds.
```

这个矩阵比较稀疏,因此也可以用稀疏矩阵来表示结果:

```
tic;mo = accumarray([height,weight],old,[],@mean,0,true);toc
Elapsed time is 0.002072 seconds.
```

维数大时,稀疏矩阵会有优势,当然这个例子还不明显。读者有兴趣可以试试传统方法的效果。

以上仅举了三个例子,实际上,accumarray 函数的应用方法非常灵活,尤其在对于很多要操作大矩阵的情况下。建议读者可以仔细研究下 accumarray 函数的用法,并在实际当中注意应用。

## 2.4.2 arrayfun 函数

相信很多读者都有这样的体会,看循环代码的时候比较累人,尤其是多重嵌套循环。很多时候我们编写的函数的输入变量是标量,而又要对很多组参数进行函数调用,而函数是很难或者不支持向量化的,这在 MATLAB7 以前的版本中我们只能通过老老实实通过循环来实现。虽说高版本下恰当使用 MATLAB 的循环,效率已不是大的瓶颈,但如果有效率和循环相当或者更好,并且很简洁的其他方式,想必绝大多数读者是不会拒绝的。arrayfun 函数和后面小节的 cellfun, spfun, structfun 等函数就是这样的一类函数。本小节先讨论 arrayfun 函数的用法。

arrayfun 这个 built-in 函数实现的是将指定的函数应用到给定数组包括结构数组在内的所有元素。这样很多以前不可避免的循环现在可以向量化了。看下面的几个例子:

**【例 2.4-4】** 生成一个这样的  $n \times n$  矩阵  $a$ :  $a(i,j) = \text{dblquad}(@ (u,v) \sin(u) * \sqrt{v}), 0,i,0,j)$ ,以  $n=10$  为例。

这个问题放在 MATLAB 7 以前,我们可能这样做:

```
a = zeros(10);
for ii = 1:10
    for jj = 1:10
        % 求 sin(u) * sqrt(v) 的二重积分
        a(ii,jj) = dblquad(@(u,v) sin(u) * sqrt(v),0,ii,0,jj);
    end
end
```

现在只需这样:

```
[J,I] = meshgrid(1:10);
a1 = arrayfun(@(ii,jj) dblquad(@(u,v) sin(u) * sqrt(v),0,ii,0,jj),I,J);
```

**【例 2.4-5】** 验证角谷猜想,一个正整数  $n$ ,如果是偶数除以 2,如果是奇数乘以 3 加 1,得到的新数继续按上述规则运算,最后结果都为 1。验证 1~100 000 内的正整数。先编写单个数的验证函数:

```
function f = SizuoKakutani (n)
if n < 1 || mod(n,1) ~ = 0
    error('n 必须是正整数')
end
f = n;
return
end
while n > 1
```



```

        if mod(n,2) == 1
            n = n * 3 + 1;
        elseif mod(n,2) == 0
            n = n/2;
        end
    end
end
f = n;

```

复制代码验证 1:100000，以前我们可能这样做：

```

a = zeros(1,100000);
for k = 1:100000
    a(k) = SizuoKakutani(k);
end
all(a)

```

现在只需这样：

```
all(arrayfun(@SizuoKakutani,1:100000))
```

可见，arrayfun 函数可以使代码简化很多。举这两个例子目的是为了说明很多以前不能简写的很多循环的程序利用这个函数可以在保证效率的前提下简化代码。关于 arrayfun 函数其他使用方法，有兴趣的读者可以运行 doc arrayfun，看其帮助文档加以了解。

### 2.4.3 bsxfun 函数

以前，当我们想对一个矩阵  $A$  的每一列或者每一行与同一个向量  $a$  进行某些操作（比较大小，乘除等）时，只能用循环方法或者利用 repmat 函数将要操作的向量  $a$  复制成和  $A$  一样尺寸的矩阵，进而进行操作。从 MATLAB R2007a 版本开始，再遇到类似的问题时，有了简洁高效的方法，即利用 bsxfun 函数。

为了节省篇幅，bsxfun 的调用格式就不列在这里了，读者可以参考帮助文档，下面给出两个 bsxfun 的例子，读者可以仔细揣摩，相信会从中得到使用 bsxfun 的很多启发。

**【例 2.4-6】** 有如下矩阵：

$$A = \begin{pmatrix} 1 & 2 & 3 & 1 \\ 2 & 3 & 4 & 2 \\ 3 & 3 & 8 & 3 \end{pmatrix}$$

向量为  $b = [1 \ 2 \ 3]^T$ ，请找出  $b$  在  $A$  矩阵列中的位置 loc = [1,4]。

下面给出五种方法，这五种方法每种方法都只有一句话，包括 bsxfun 的使用和前面 arrayfun 函数的使用。读者可以仔细体会其用法，相信会受益匪浅。

方法 1：

```
loc = find(all(bsxfun(@eq,A,b))) % 把 A 的每一列和 b 用 == (@eq)来判断，找出全 1 的列
```

方法 2:

```
loc = find(arrayfun(@(n) all(A(:,n) == b),1:4)) % 用 arrayfun 对 n 进行 1:4 的遍历
```

方法 3:

```
loc = find(all(~bsxfun(@minus,A,b))) % 把 A 的每一列和 b 来相减(@minus),求反后找出全 1 的列
```

方法 4:

```
loc = find(arrayfun(@(n) isequal(A(:,n),b),1:4)) % 用 arrayfun 对 n 进行 1:4 的遍历后用 isequal 函数来判断 A 的每列和 b 是否相等
```

方法 5:

```
loc = find(b'*A == sum(b.^2)) % b 的转置和 A 相乘,然后和 b.^2 每列的和进行比较,找到相等的
```

上述几种方法都可以达到目的,但是思路各不相同。读者可以用较大规模的矩阵比较上述各方法的效率。

再看一个例子:

**【例 2.4-7】** 如何将一个矩阵的每行或每列元素分别扩大不同的倍数? 如 $\begin{bmatrix} 1 & 2 & 3; \\ 4 & 5 & 6; \\ 7 & 8 & 9 \end{bmatrix}$ ,第一列元素乘以 1,第二列元素乘以 2,第三列元素乘以 4。

利用 bsxfun 函数,可以给出下列代码:

```
a = [1,2,3;4,5,6;7,8,9];
acol = bsxfun(@times,a,[1 2 4])
```

事实上,bsxfun 的第一个输入参数是函数句柄,其支持的函数类型有多种,帮助文档里面都列了出来,大家在使用的时候可以参考帮助文档。

从 MATLAB R2009b 开始,bsxfun 函数开始更好地支持多线程了,即不需要用户干预,其内部已经较之前的版本更好的支持多线程了。更多的较之以前更好地支持多线程的函数还有:sort (有其对于大矩阵),mldivide (对于稀疏矩阵),qr (对于稀疏矩阵),filter,gamma,gammain,erf,erfc,erfcx,erfinv 等函数。

## 2.4.4 cellfun 函数

类似 arrayfun 函数,cellfun 函数是将指定的函数应用到元胞数组内每个元胞存储的内容上。看下面的简单例子:

**【例 2.4-8】**  $A = \{ 'Hello', 'MATLAB', 'I love MATLAB', 'MATLAB is powerful', 'MATLAB is the language of technical computing' \}$ ;试统计 A 中每个元胞单元存放的字符串的长度。

本例用 cellfun 的代码非常简单,只一句话即可完成任务。

```
len = cellfun(@length,A)
len =
    5     6    13    18    45
```

**【例 2.4-8 续】** 试提取  $A$  中每个元胞内容前三个字符,并全部转化成大写。  
用 `cellfun` 做同样很简单:

```
Upp = cellfun(@(x) upper(x(1:3)),A,'UniformOutput',false)
Upp =
    'HEL'    'MAT'    'I L'    'MAT'    'MAT'
```

这里要说明一点。`cellfun` 和 `arrayfun` 函数都可作用于一个 cell 数组(`arrayfun` 函数还可以用于数值,结构等数组),那它们有什么区别呢?在回答这个问题前,我们先打个比方:

cell 数组好比一栋“楼房”,不同的行数好比不同的“楼房”层数,每个 cell 单元好比楼层上的“房间”,而每个 cell 单元里面具体存放的东西好比是放在“房间”里的东西。

`cellfun` 的输入函数操作的是放在“房间”里的东西。而 `arrayfun` 的输入函数操作的是具体的每个“房间”整体。读者可以体会下列代码的运行结果。

```
a = {12,34,55,66,77}
a =
    [12]    [34]    [55]    [66]    [77]
>> b = cellfun(@(x) x,a)
b =
    12     34     55     66     77
>> c = arrayfun(@(x) x,a)
c =
    [12]    [34]    [55]    [66]    [77]
>> isequal(c,a)
ans =
    1
```

`cellfun` 和 `arrayfun` 函数的指定函数都相同,都是  $f(x)=x$  这个简单的函数,可以清楚看出, `cellfun` 函数把每个“房间”里的数都取了出来,由于指定函数是  $f(x)=x$ ,所以没有对其进行任何操作,只是将它们放到一个和  $a$  行列数相同的数值数组  $b$  里。而 `arrayfun` 函数则是将每个“房间”连同里面的东西都取了出来,所有都取完后再拼成一个整体  $c$ ,因此  $c$  还是等于  $a$ 。

如果 `cellfun` 函数的指定函数操作每个单元返回的值无法用数值数组来统一表示,那么必须用 cell 型数组来表示。譬如当  $a=\{1,2,'abc'\}$  时,  $b=\text{cellfun}(@(x) x, a)$  这么写就会报错了,为什么?因为其指定函数  $f(x)=x$  操作每个单元返回的值不是同一类型,无法用数值矩阵统一表示,只能用 cell 数组表示。于是应该写为  $b=\text{cellfun}(@(x) x, a, 'UniformOutput', false)$ ;这时候  $b$  和  $c=\text{arrayfun}(@(x) x, a)$  是相同的。

## 2.4.5 spfun 函数

`spfun` 函数的使用方法比较简单,即将指定的函数 `fun` 应用到稀疏矩阵  $a$  每个不为 0 的元素上,返回一个和  $a$  尺度相同的,且具有稀疏结构的矩阵。看下面的例子。

**【例 2.4-9】** 演示 `spfun` 函数的简单使用方法:

```
a = sparse([1 3 20 60 100],[2 20 30 60 80],[1 2 3 4 5])
```

```
a =
```

```
(1,2)      1
(3,20)     2
(20,30)    3
(60,60)    4
(100,80)   5
```

```
>> sa = spfun(@(x) x.^2 + 1,a)
```

```
sa =
```

```
(1,2)      2
(3,20)     5
(20,30)   10
(60,60)   17
(100,80)  26
```

## 2.4.6 structfun 函数

structfun 函数用法和上述介绍的函数类似,不过 structfun 函数要求操作的结构数组是标量结构数组,所谓标量结构数组是指在 MATLAB 工作空间的“变量 value”那一栏显示为  $1 \times 1$  struct 的结构数组。

structfun 所接收的函数将作用到所接收的结构各个域上。看下面的例子:

```
s.f1 = magic(3);
s.f2 = magic(5);
s.f3 = magic(8);
SumFields = structfun(@(x) sum(x) , s,'UniformOutput', false)
SumFields =
    f1: [15 15 15]
    f2: [65 65 65 65 65]
    f3: [260 260 260 260 260 260 260 260]
```

从上面的例子可以看出,新的结构数组 SumFields 的域和 structfun 接收的结构数组 s 的域相同,每个域的内容是 sum 函数作用于 s 的相应域的内容的结果。

相信很多使用过 MATLAB 的读者都遇到“out of memory”的错误提示。出现这样的提示说明 MATLAB 的工作空间中的变量占据的内存空间总和超出了当前可用内存空间的总和。在不增加物理内存条件下,该怎么办?有的时候需要处理大量的数据,怎样才能高效处理?

我们还经常听到 MATLAB 使用者抱怨 MATLAB 的内存管理机制很差,本来机器 4 GB 的物理内存,可是最多只能开 1 GB 大小的矩阵。怎样才能有效利用 4 GB 的内存呢?

类似的这些问题是这章的主要讨论问题。

### 3.1 处理海量数据时遇到的问题

#### 3.1.1 什么是海量数据

海量数据,顾名思义,数据量一般大得多。海量数据一般是相对当前计算机处理能力而言的,今天看来是海量数据,也许随着计算机硬件能力的提升在不久的将来就不算海量数据了。即使是同一时期,不同的应用领域,不同的学科以及不同的硬件条件下,海量数据所呈现的数据量也不尽相同。

对于用户个人来说,凡是在个人计算机上处理数据时经常出现“out of memory”提示的数据量,都可称之为海量数据。

#### 3.1.2 经常遇到的问题

在有限的计算机处理能力下,我们处理海量数据经常遇到一些问题。这些问题根据表象可以分为两大类:一类是内存溢出(out of memory),一类是运行极其缓慢。

##### (1) 导致内存溢出的主要原因

1) 变量需要的存储空间超过了可用的内存空间。这是最简单的情况。原因是将要生成的矩阵太大,所需存储空间超过了所有可用的内存总和。但是很多时候矩阵需要的存储空间大小并没有超过可用内存总和,但同样有内存溢出的错误提示,其原因见下面第 2) 条。

2) 数值矩阵所需要的存储空间,超过了内存中最大的可用连续存储空间。MATLAB 里默认用 double 型来表示数值变量,这样一个数字就需要 8 B 的存储空间。而 MATLAB 中的数值矩阵存储需要内存中连续的存储空间。这样如果内存中碎片过多,虽然可用的内存总和较大,但是最大的连续内存空间不大,这样当矩阵的元素较多时,就可能没有足够的连续内存空间来存储,从而发生内存溢出的错误提示。

譬如  $a = \text{rand}(10000, 10000)$ ; 生成  $10^8$  个  $(0, 1)$  之间随机分布的随机数,需要的存储空间是  $8 \times 10^8 \text{ B}$ , 大约需要 800 MB 的连续内存空间。如果内存中最大的连续空间都小于  $8 \times$

$10^8$  B,那么无论所有可用的内存空间之和是否大于  $8 \times 10^8$  B, MATLAB 都会给出“out of memory”错误提示的。

3) 程序设计方面考虑的不周,导致内存溢出。这方面的原因一般表现得比较隐蔽。我们在设计一些变量较多的复杂程序时,可能没有及时释放掉一些不再会用到的变量所占用的内存。这样随着程序的运行,变量越来越多,内存中碎片也越来越多,再要产生新的较大的矩阵,就可能发生内存溢出的问题。

4) 问题求解方法考虑欠周,导致内存溢出。同样的问题,可能由多种求解方法。不同方法的效率以及所占用的资源相差很大。问题规模小的时候,不用考虑这方面的问题,可是问题规模较大的时候就必须考虑具体的求解方法了,否则就可能会由于内存溢出而无法求解问题。

譬如符号计算。符号计算由于其语法简单,并且和传统教科书解决问题思路一致等特点,使得许多初学者很爱使用符号计算。但是符号计算由于精确性,使得其在计算过程中要保留大量的中间结果,这样导致复杂的计算问题用符号计算很容易内存溢出,即使不溢出,对于很多复杂的计算问题,符号计算也无法给出解析解。

还有当求解涉及较大规模的稀疏型矩阵的问题时,没有采用稀疏矩阵数据结构也容易导致内存溢出。

## (2) 运行极其缓慢的原因

这是因为大数据量的处理总是伴随着大量的计算还有频繁的对内存各地址进行访问,这些开销往往比较占用资源。如果物理内存不够,系统开始调用页面文件来供程序使用,会使得程序运行速度有很大程度的下降。

本章后半部分将介绍不增加物理内存的情况下,可供 MATLAB 高效利用内存的方法。

## 3.2 有效设置增加可用内存

随着计算机硬件的发展,如今的内存价格较之以前低廉了很多,主流 PC 包括笔记本式计算机的内存配置基本都不低于 4 GB 了,这已经达到 32 位操作系统的寻址极限了。虽然 64 位操作系统可以支持更大的内存,但由于诸多因素导致 64 位操作系统完全取代 32 位操作系统还需要一段时间,在可预见的一段时间内,使用 32 位操作系统的读者还会很多,因此在 32 位操作系统下如何有效利用内存是这节主要讨论的内容。

本小节以 Windows XP 32 位操作系统为例来说明如何设置 boot. ini,使得 MATLAB 程序可以利用更多的内存。

### 3.2.1 系统默认下内存分配情况

首先需要澄清一个概念——虚拟内存(virtual memory)。有很多朋友习惯把硬盘上预留出来来做内存交换和扩展内存寻址空间的“交换文件”(也叫页面文件 Pagefile. sys)当做“虚拟内存”。因为它不是内存芯片,却在必要的时候和物理内存进行内存页交换,所以认为是“虚拟”的内存。当然这样讲有一定道理,也可以认为是人们通常说的“虚拟内存”代表的意思。但是有的时候,“虚拟内存”还代表另一种含义。简单说来,这种含义就是特定操作系统下多路程序进程共享的计算机物理内存以及页面文件。

Windows 的虚拟内存系统可以让所有的应用程序都运行在自己独占的私有所谓“虚拟地