第3章 MATLAB核心——矩阵

MATLAB作为“矩阵的实验室”，其最核心的思想与特色就是“基于矩阵的数据结构与编程方法”，在第2章学习了MATLAB的基本操作与软件框架后，建议直接研究MATLAB语言的精髓——矩阵，掌握矩阵的概念、操作、运算及编程风格，从此进入区别于其他语言的极速编程境界。

本章首先从编程语言的基石——“数据类型与数据结构”入手，分别解析矩阵与它们之间的关系，再通过矩阵操作技术与运算方法进入矩阵化编程的殿堂，最后通过具体的编程案例讲解矩阵编程的核心精要。本章内容重要而基础，学习时不可仅停留在知识记忆层面，更要多加练习直至灵活应用。

3.1 矩阵与数据类型

数据类型是编程语言的基本，人类语言中的数据形式无非就是“数字”“文字”与“符号”，对应MATLAB中的3种核心数据类型即为“数值”“字符”与“符号”。

关于数据的结构，在数学中有几个常用概念——标量(Scalar)、向量(Vector)、矩阵(Matrix)、张量(Tensor)，在计算机学中把这一类数据结构统称为“数组”(Array)；它们在本质上其实都是统一的，在许多场合下并不加以区分。本书为了呼应MATLAB的名字“矩阵的实验室”，同时为了强化软件核心思想，使用“矩阵”这个词来涵盖以上所有概念，不同的概念只是不同维度及不同规模的矩阵而已，如图3-1所示。

(1) 空数据：规模0×0空矩阵。

(2) 标量：规模1×1的0维矩阵。

(3) 行向量：规模1×n的一维矩阵。

(4) 列向量：规模n×1的一维矩阵。

(5) 普通矩阵：规模n×m的二维矩阵。

(6) 多维数组：规模n×m×…的多维矩阵。

MATLAB的所有数据类型均是以矩阵为核心及基础的，矩阵中的

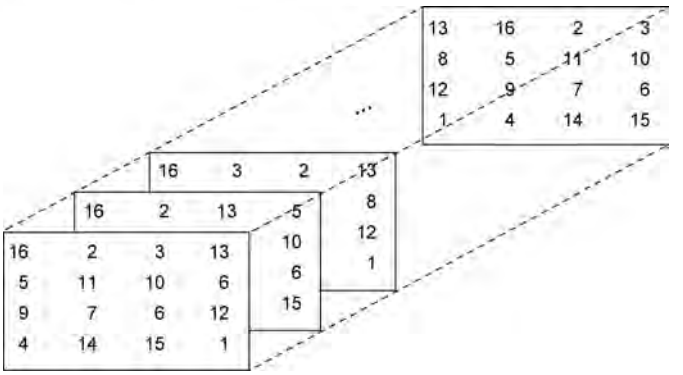

图 3-1　多维数组/矩阵

元素也可以是实数、复数、字符或符号变量。

3.1.1　数值矩阵

MATLAB 中的数值，拥有符合 IEEE 标准的存储格式与精度，包含浮点型与整型。浮点型包含双精度浮点(double)和单精度浮点(single)，整型包含 8、16、32、64 位带符号与不带符号整型。MATLAB 中默认的数值类型就是“双精度浮点型”，对于初学者来说，即可基本涵盖所有应用场合；次常用的是带符号 8 位整型(int8)。MATLAB 在复数计算领域也有很强的优势，因为它所有的运算都是定义在复数域上的，所以计算时不需要像其他程序语言那样将实部与虚部分开，如图 3-2 所示。

EX 3-1 数值类型矩阵

1. 浮点型与整型

```
% 赋值a为2的数值类型
a = 2
class(a)
% 定义变量b为2但不输出
b = int8(a)
class(b)
% 浮点型判断
c = a == 2*pi/3.14
d = abs(a-2*pi/3.14)<0.1
% 整数判断
e = b == 2*pi/3.14
f = b == int8(2*pi/3.14)
```

```
a = 2
ans = 'double'
b = int8
    2
ans = 'int8'
c = Logical
   0
d = Logical
   1
e = Logical
   0
f = Logical
   1
```

2. 复数矩阵

```
% 复数赋值
g = 3+4i
% 求复数g的模
h = abs(g)
% 复数矩阵
l = [g 2*g g*g]
```

```
g = 3.0000 + 4.0000i

h = 5
l = 1×3 complex
    3.0000 + 4.0000i   6.0000 + 8.0000i  -7.0000 +24.0000i
```

图 3-2　数值类型矩阵例程

说明：

（1）不建议两个浮点数之间直接判断是否相等，而应采用判断差的绝对值的方法，这是由于数值计算存在计算精度的考虑。

（2）复数中 i 不可与前面的数字有空格间隔，4i 是一个完整的虚数。

（3）在工作区中，数值矩阵类的变量图标都是“田”字形的图标。

3.1.2 字符矩阵

文字在 MATLAB 中的存储有两种形式：“字符”与“字符串”。字符矩阵是一个以字符为元素而组成的矩阵，使用单引号赋值可以直接生成字符行向量。字符串矩阵是以字符串为元素而组成的矩阵，每个字符串元素的长短不受约束，从 R2017a 开始可以使用双引号直接创建字符串，如图 3-3 所示。

EX 3-2 字符型矩阵

1. 字符矩阵

```
% 赋值字符矩阵
a = 'x'
b = 'yz'
class(a)
% 字符矩阵拼接
c = [a b;'uvw']
% 求字符矩阵的规模
d = size(c)
```

```
a = 'x'
b = 'yz'
ans = 'char'
c = 2×3 char 数组
    'xyz'
    'uvw'
d = 1×2
     2     3
```

2. 字符串矩阵

```
% 赋值字符串矩阵
e = "Hello, "
f = string('World!')
class(e)
% 字符串矩阵组合
g = [e,f]
h = strlength(g)
% 字符串矩阵拼接
l = join(g)
m = strlength(l)
```

```
e = "Hello, "
f = "World!"
ans = 'string'
g = 1×2 string 数组
    "Hello, "    "World!"
h = 1×2
     7     6
l = "Hello,  World!"
m = 14
```

图 3-3 字符型矩阵例程

说明：

（1）字符矩阵可以同时赋值一串字符，而且显示时也会将一行中的字符连续显示出，但这并不表示它是一个字符串，它的类型仍是字符类型。字符与字符串矩阵的操作，同数值矩阵操作原理一致。

（2）取规模函数 size()取的是矩阵中元素的个数；也就是说，对字符矩阵取规模时，取到的是矩阵中字符的个数，对字符串矩阵取规模时，取到的是字符串的个数，而无法得到字

符串内部有多少字符，如需计算字符串中字符的个数，可以使用函数 strlength()。

(3) 字符转换为字符串使用函数 string()，字符串转换为字符使用函数 char()。

(4) 在工作区中，字符变量的图标是 c|h，字符串变量的图标是 str，两者的图标不同。

3.1.3 符号矩阵

符号数学是数学中非常重要的部分之一，它引领人类从“算数学”进入“代数学”，从具象走向了抽象。因而，虽然 MATLAB 的诞生和崛起都依赖于它无与伦比的“数值计算”，但一直强力扩展它的符号计算功能，从 2008 年弃用 Maple 引擎而收购 MuPAD 以来的十余年间，MATLAB 的符号计算引擎早已今非昔比，成为业内最优秀的符号计算工具之一，并以符号数学工具箱(Symbolic Math Toolbox)的形式存在。

符号计算与数值计算都具有非常重要的实际意义，符号计算的优势在于可以无须在计算前对变量赋值，而直接以符号形式输出运算结果，在许多应用场景中其实更接近数学思维。符号变量需要声明定义，而由符号变量组成的表达式则会自动定义为符号类型的表达式；符号表达式是符号矩阵的基本元素。符号计算与数值计算在本质上是两种类型的独立计算引擎，但 MATLAB 实现了二者的深度融合，比如有许多函数都可以不限制输入类型，无论是数值还是符号都可以自由使用，如图 3-4 所示。

EX 3-3 符号型矩阵

1. 符号变量及矩阵

```
% 定义符号变量
syms x
% 符号表达式
y1 = x+5*x
classY1 = class(y1)
sizeY1 = size(y1)
% 符号表达式矩阵
y2 = x^2+3*x
y = [y1; y2]
sizeY = size(y)
% 简化符号表达式形式
simY = simplify(y)
```

2. 简单应用举例

```
syms a b c d
% 用符号代表矩阵中的元素
A = [a b; c d]
% 求矩阵的逆
invA = inv(A)
```

```
y1 = 6x
classY1 = 'sym'
sizeY1 = 1×2
     1     1
y2 = x^2+3x
y =
⎛  6x  ⎞
⎝x^2+3x⎠
sizeY = 1×2
     2     1
simY =
⎛  6x  ⎞
⎝x(x+3)⎠
A =
⎛a  b⎞
⎝c  d⎠
invA =
⎛   d/(ad-bc)   -b/(ad-bc)⎞
⎝  -c/(ad-bc)    a/(ad-bc)⎠
```

图 3-4 符号型矩阵例程

说明：

(1) 符号变量的定义有两种方式，一种是使用 syms，这是一个关键字，其后所跟变量会定义为符号变量；另一种是使用 sym()，这是一个函数，其代码写成 sym('x')也有同样的效果。

(2) 符号表达式计算当然有一些数值计算中没有的功能,比如简化表达式(simplify)等;但同时也有大量的共通功能,比如求矩阵的逆(inv)等,可以帮助推导公式,获得解析解。

(3) 符号计算尤其在高等数学的教学实践中举足轻重,本书将在第5章MATLAB数学计算中深入学习应用。

3.2 矩阵与数据结构

线性代数被称为"第二代数学模型",其中的"矩阵"概念,可以说是整个现代科学的基础,其中的底层逻辑就在于,矩阵中不仅包含每个元素的值,还通过"结构"包含了数据与数据之间的关系信息,正如亚里士多德所说"整体大于部分之和",就是这个道理,这正是"结构化语言"的好处。

矩阵是MATLAB中最核心的数据结构,然而矩阵也有它的不足,矩阵中的元素只可以是数值、字符、字符串、符号,而且只能使用数字进行索引,其实在许多程序设计场合下,都需要更复杂的存储模式,比如需要每个元素拥有不同的规模及不同的类型,或者需要使用名称而不是数字来索引元素,或者不同列拥有不同数据类型的表格类数据,这时就要对矩阵进行数据结构的拓展。在MATLAB中,还有3种核心数据结构:元胞数组、结构体和表,这三者可以归类为"数据存储结构",它们一般不直接参与计算,而是转移到矩阵中完成计算,再转存回去,三者与矩阵在存储元素、索引方式、结构形态上的异同如表3-1所示。

表3-1 MATLAB 4种数据结构比对

数据结构 / 比较项目	矩　阵	元胞数组	结构体	表
英文名称	matrix	cell array	structure	table
元素要求	同类元素	无要求	无要求	按列同类
元素是否可以是矩阵	不可以	可以	可以	不可以
索引方式	数字索引	数字索引	名称索引 (局部数字索引)	数字/名称索引
结构形态	阵状	阵状	树状	阵状

数据结构是编程语言的基础工具,心理学家马斯洛曾经说过"手里有把锤子,看什么都像钉子";如果仅掌握矩阵这一种数据结构,那么在编程实践中则难免舍近求远、事倍功半;元胞数组、结构体与表都是MATLAB程序设计中极为常用和重要的数据结构,对它们的使用不了解很可能造成程序复杂度的急剧攀升,可惜大多数教材与课堂并未给予足够的重视。

3.2.1 元胞数组

元胞数组(Cell Array)本质上是一种广义的矩阵,因此也可以翻译为"元胞阵",它与矩

阵的不同在于，元胞数组的每个元素，可以是矩阵中的数值、字符、字符串、符号等，也可以就是一个矩阵，甚至还可以是一个元胞数组，其灵活程度可想而知。元胞数组在许多场合下都非常实用，比如需要使用数字来进行索引，而每个元素（如矩阵）的规模都不能保证一致时（参见图 3-5），元胞数组往往是首选。

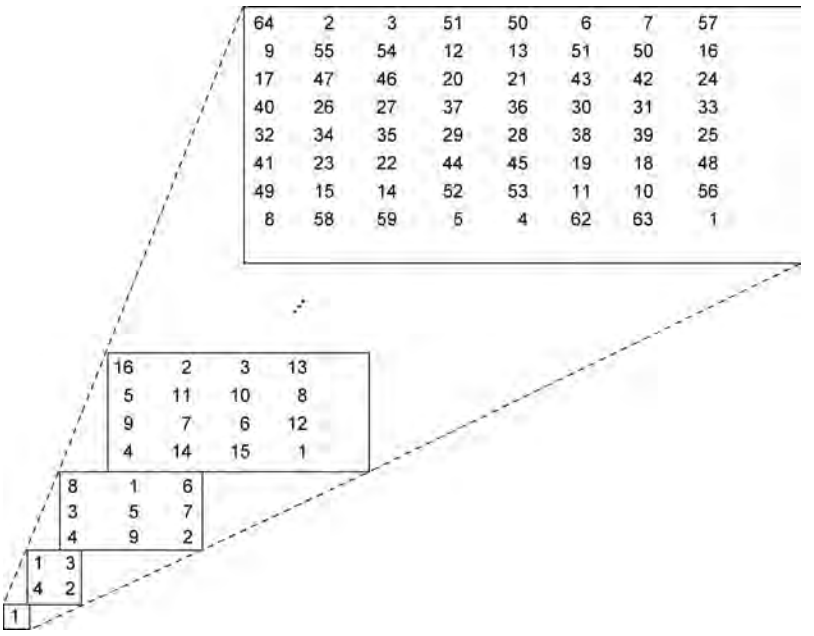

图 3-5　元胞数组的一种存储形式

元胞数组在程序设计中，一般作为存储介质，将获得的数据灵活地保存在元胞中，需要时再利用数字索引快速提取。元胞数组的常用创建与访问操作如图 3-6 所示。

说明：

（1）元胞数组采用大括号赋值，其余格式与矩阵相同，每个元素会默认形成一个单元素元胞。

（2）celldisp()函数可以用于显示元胞数组中每个元素的具体内容。

（3）元胞数组本质是一个矩阵，因此也要符合阵形结构，比如仅对某一个位置赋值后，软件会自动用空元胞将其他位置补齐。

（4）cell()函数实现创建一个空元胞数组，多用于预分配内存，与矩阵赋值中的 zeros()函数同理。

（5）注意，元胞数组中每个元素默认即为一个 1×1 元胞，因此直接使用小括号索引，得到的是元胞元素而不是其中的数据内容；其实，可以使用大括号索引方式，就能直接突破元胞，以矩阵形式取得其中的数据元素。

（6）cell2mat()函数用于将元胞转换为矩阵，但前提是准备转换的数据本身就符合矩阵的格式要求，同样的函数还有 cell2struct()和 cell2table()。

EX 3-4 元胞数组

1. 创建元胞数组

```
% 元胞数组整体输入
cellA = {1, 'text'; zeros(2,3), {11; 22}}
celldisp(cellA)
% 元胞数组按元素赋值
cellB(2,2) = {5}
% 快速创建空元胞数组
cellC = cell(2,3)
```

2. 访问元胞数组中的数据

```
% 使用小括号，访问的是元素，即元胞
a = cellA(1,1)
% 使用大括号，访问的元素中的数据
b = cellA{1,1}
c = cellA{1,2}
```

3. 元胞数组转化为矩阵

```
cellD = {[1 2]; [3 4]}
matD = cell2mat(cellD)
```

```
cellA = 2×2 cell 数组
    {[     1]}    {'text' }
    {2×3 double}    {2×1 cell}
cellA{1,1} =
     1
cellA{2,1} =
     0     0     0
     0     0     0
cellA{1,2} =
text
cellA{2,2}{1} =
    11
cellA{2,2}{2} =
    22
cellB = 2×2 cell 数组
    {0×0 double}    {0×0 double}
    {0×0 double}    {[     5]}
cellC = 2×3 cell 数组
    {0×0 double}    {0×0 double}    {0×0 double}
    {0×0 double}    {0×0 double}    {0×0 double}
a = 1×1 cell 数组
    {[1]}
b = 1
c = 'text'
cellD = 2×1 cell 数组
    {1×2 double}
    {1×2 double}
matD = 2×2
     1     2
     3     4
```

图 3-6　元胞数组例程

3.2.2　结构体

结构体(Structure)在许多编程语言中都有非常重要的应用，因为它代表着一种通过名称(field，翻译为字段)索引来存储的“树形结构”。在 MATLAB 中，结构体中可以存储的元素同元胞数组一样，都没有什么特定的要求，可以是矩阵中的数值、字符、字符串、符号等，也可以就是一个矩阵，甚至还可以是一个元胞数组。结构体中的元素还可以是另一个结构体，只不过在表示时，结构体下的结构体可以直接使用多级结构来定义；结构体也可以形成数组，此时可以局部使用数字索引来完成。结构体是树形结构，在程序设计中拥有非常重要的应用，它可以把杂乱的数据很好地按层级与名称存储归类(参见图 3-7)，灵活使用结构体是 MATLAB 程序设计中非常重要的技能之一。

说明：

(1) 英文句点“.”是用来定义结构体层级的，多层级设置也同理，如 patient. name. firstName。

(2) 结构体作为树状结构，既可以使用名称(字段)来进行分支，也可以使用数字来分支，此时形成“结构体数组”，结构体数组中的所有结构体都具有相同的分支，因为毕竟没有脱离数组(矩阵)的本质。

(3) 程序设计中，常用结构体主名称为一个“对象”，使用结构体分支字段来存储该对象的一些“属性”，通过存取和修改对象的属性来完成一些程序功能，这种思想虽然与真正的

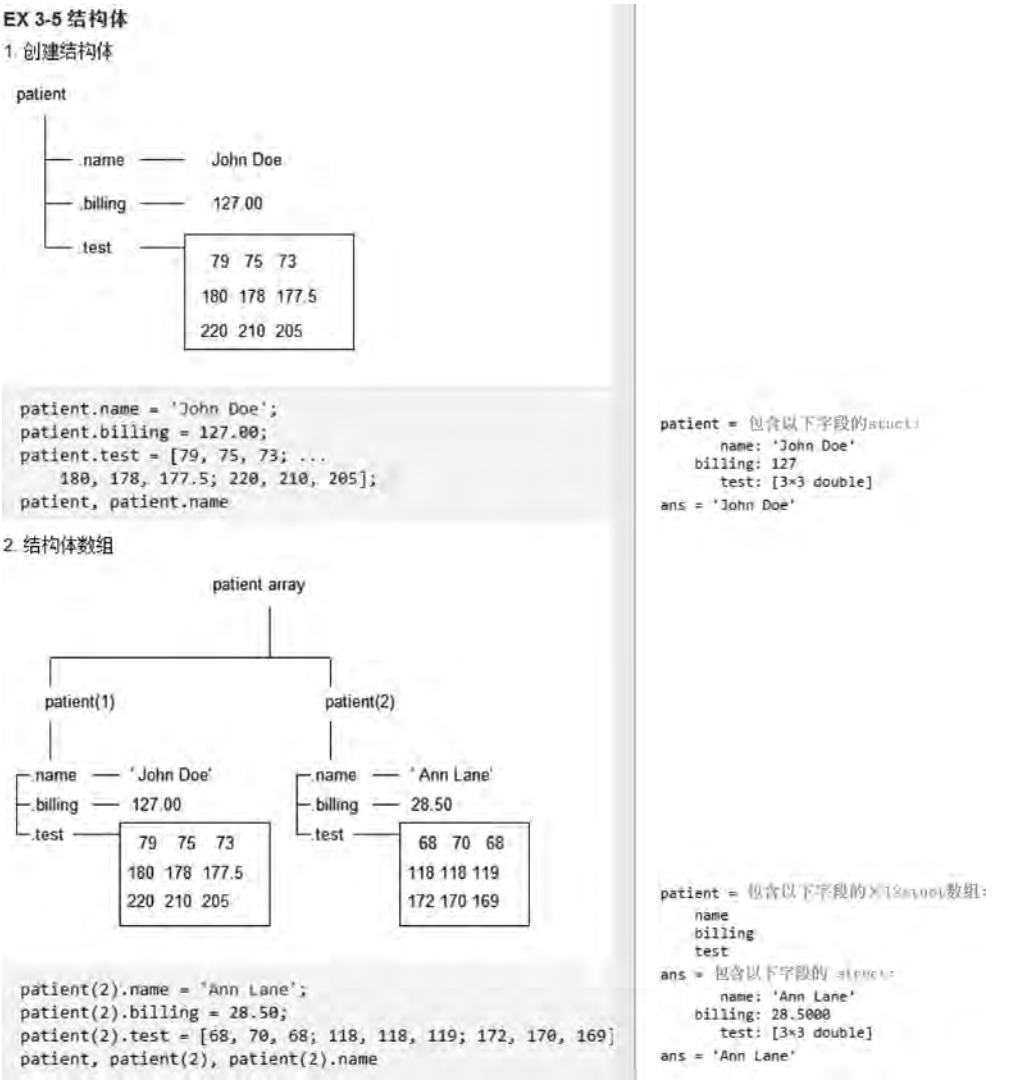

图 3-7　结构体例程

“面向对象编程”还有一定距离，但是往往可以大大简化程序结构、提高代码的清晰度。

（4）结构体中的显示顺序与创建顺序相同，并且修改其值后也不会改变显示顺序，为程序设计中创建数据和存储数据提供了方便。

3.2.3　表

表（Table）是一种以矩阵为基础的异化数据结构，正如表格一样，表中每个变量可以具有不同的数据类型和大小，但是每个变量的行数必须相同；表中的变量并不限于列向量，也

可以是包含多列的矩阵，只要行数相同即可。表作为一种程序设计中非常实用的数据结构，在 R2013b 版本首次引入，以取代统计工具箱中的 dataset 数据类型，从此大受欢迎，成为数据处理和程序设计的神器。

表适用于在数据分析和处理行为中，典型用途是存储试验数据，行表示不同的观测点，列表示不同的测量变量，还可以对于文本文件或电子表格中的数据进行存取。表可以实现名称索引，速度很快类似于哈希表功能，同样的功能在矩阵或者元胞数组中则需要全数据比对，当数据量较大时效率略低。MATLAB 中的表格式可与两类格式无缝对接：

（1）.txt、.dat 或 .csv（适用于带分隔符的文本文件）。

（2）.xls、.xlsm 或 .xlsx（适用于 Excel 电子表格文件）。

使用 writetable()函数可以将表保存为上述格式，使用 readtable()函数可以读取上述格式，如图 3-8 所示。

EX 3-6 表

1 创建表

```
% 清空工作区
clear
% 载入内存变量
load patients
% 取变量建立表
tablePatients = table(Gender, Age, Weight, Smoker)
```

2. 表索引

```
% 提取表的一部分——数字索引
tablePatients(1:3,:)
% 制作命名索引
tablePatients.Properties.RowNames = LastName;
% 提取表的一部分——命名索引
tablePatients({'Smith','Johnson'},{'Gender','Smoker'})
```

3 表存储

```
% 存储为文本文档
writetable(tablePatients, 'tablePatients.txt')
% 存储为电子表格
writetable(tablePatients,'tablePatients.csv')
```

tablePatients = 100×4 table

	Gender	Age	Weight	Smoker
1	'Male'	38	176	1
2	'Male'	43	163	0
3	'Female'	38	131	0
4	'Female'	40	133	0
5	'Female'	49	119	0
6	'Female'	46	142	0
7	'Female'	33	142	1
8	'Male'	40	180	0
9	'Male'	28	183	0

ans = 3×4 table

	Gender	Age	Weight	Smoker
1	'Male'	38	176	1
2	'Male'	43	163	0
3	'Female'	38	131	0

ans = 2×2 table

	Gender	Smoker
1 Smith	'Male'	1
2 Johnson	'Male'	0

图 3-8　表例程

说明：

（1）patients 是 MATLAB 自带的工作空间数据，并保存为一个.mat 文件；同理，用户也可以将工作空间保存下来，命令为 save name.mat，其中 name 是用户自起的名字。

（2）table()函数用于创建表，输入的变量名同时将作为表头，输入变量可以是列向量、矩阵、元胞数组、字符矩阵、逻辑矩阵，只要它们拥有相同的行数即可，工作区中图标为一个“对号”的 Smoker 变量为逻辑矩阵，其中存储的只有 1（真）和 0（假）两个值。

（3）许多数据量较大的文件常以.csv 格式存在，MATLAB 甚至可以直接打开.csv 文件，并可以选择输入类型，常用的有表、列向量、数值矩阵、字符串数组、元胞数组，图 3-9 所示为数据文件导入窗口。

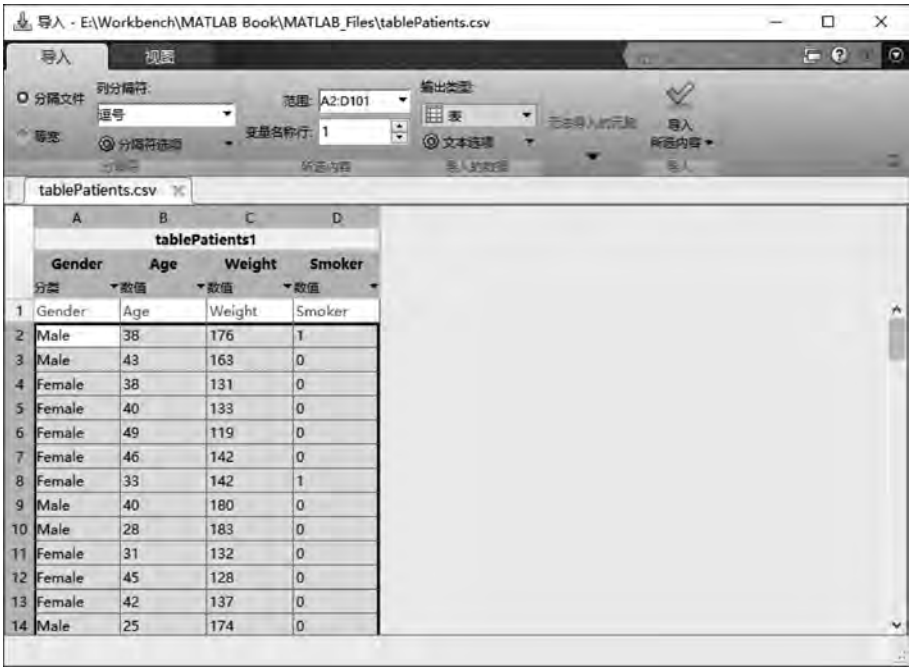

图 3-9　数据文件导入窗口

3.3　矩阵操作

在这个大数据与机器学习风靡全球的时代，矩阵成为各行各业炙手可热的词汇，原因在于矩阵作为一种数据结构，对其进行的操作相当于“批量化行为”，一方面计算效率高，另一方面将意义打包更利于理解。矩阵是 MATLAB 中的核心概念，灵活使用矩阵操作技能，可以让代码非常简洁高效，感受到 MATLAB 与其他语言相比的重大优势，因而矩阵操作是学习 MATLAB 必须掌握的核心技能，基本包括矩阵的索引操作、逻辑操作和函数操作。建议矩阵操作时养成如下 3 点习惯：

（1）整存整取：如果可以对矩阵整体进行操作，就一定整体操作，尽量减少使用循环或单独提取某一元素。

（2）每个维度要定义明确的意义：在初始化矩阵时，注释清楚每个维度所代表的涵义，遇到维度变换时要马上注释新维度的涵义。

（3）同一维度里的数据的地位一致：避免将不同意义或不同地位的数据放在一个维度中，否则将对代码编写带来许多不必要的麻烦。

3.3.1　索引操作

索引是探查矩阵中元素的主要方法，利用索引可以实现矩阵特定局部的提取与赋值；

索引分为“单索引”(Index)和“角标索引”(Subscript),角标索引即是直接用各维度上的角标做索引,单索引是一个从 1 开始的单个数字,对于任意维矩阵,按照维度的先后(行、列、页等)整理为向量,就可以用一个单索引数字来定位元素,图 3-10 所示为单索引与角标索引的对应关系。

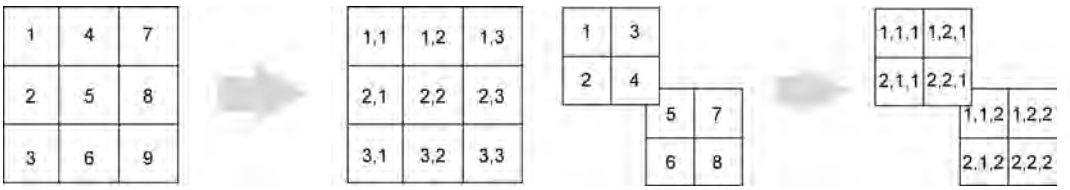

图 3-10　单索引与角标索引的对应关系

使用索引进行矩阵操作时,要灵活利用冒号“:”与 end 关键字,理解掌握单索引与角标索引的转换方法,如图 3-11 所示。

EX 3-7 矩阵索引操作

1. 冒号与 end 关键字

```
a = [1 2 3; 4 5 6]
% 索引方式取矩阵的一部分
b = a(:,1:2)
c = a(:,[1 3])
% 取矩阵最后一行最后一列的元素
d = a(end,end-1)
```

2. 单索引与角标索引

```
% 单索引整理
e = a(:)
reshape(e,size(a))
% 取 2 行 3 列位置的单索引
index = sub2ind(size(a),2,3)
% 验证两者是否相同
e(index) == a(2,3)
% 反算角标索引
[sub1, sub2] = ind2sub(size(a),index)
```

3. 局部赋值

```
f = zeros(2,4)
% 取矩阵的一部分赋值
f(1,2:4) = [1 2 3]
```

```
a = 2×3
     1     2     3
     4     5     6

b = 2×2
     1     2
     4     5

c = 2×2
     1     3
     4     6

d = 5
e = 6×1
     1
     4
     2
     5
     3
     6
ans = 2×3
     1     2     3
     4     5     6
index = 6
ans = Logical
   1
sub1 = 2
sub2 = 3
f = 2×4
     0     0     0     0
     0     0     0     0
f = 2×4
     0     1     2     3
     0     0     0     0
```

图 3-11　矩阵索引操作例程

说明:

(1) end 代表该维度最后一个角标的值,在程序设计中,对于要操作的矩阵,尽量使用 end 而不是实际的数字来取到最后一位,这样的程序鲁棒性强,不会受到矩阵规模变化的影响。

(2) a(:)这种形式非常重要和实用,它可以将任意维度矩阵整理为列向量,进而可以当

作向量来处理和计算，如需回归原形式，使用 reshape()函数即可。

（3）角标索引换算为单索引使用 sub2ind()函数，单索引换算为角标索引使用 ind2sub()函数，两者的第一输入变量均为矩阵的规模向量，可使用 size()函数得到。

（4）矩阵可以通过索引实现局部的赋值，注意等号两边的规模一定要相同，否则会报错。

3.3.2 逻辑操作

逻辑操作是极为实用的操作技术，一方面可以通过逻辑判断式来代替索引的位置，对矩阵中满足特定要求的元素进行批量操作；另一方面可以将逻辑矩阵整体作为程序流中的判断依据；另外，利用 find()函数可以极速实现矩阵索引的按逻辑提取。

在程序设计过程中，往往需要经常进行逻辑判断，根据判断结果决定程序流向；MATLAB 基于矩阵操作的整体判断方法在程序设计语言中独树一帜，简洁高效，节省用户大量的操作与构思时间，如图 3-12 所示。

EX 3-8 矩阵逻辑操作

1. 逻辑判断式做索引

```
a = [1 4 2; 4 2 6]
% 将大于 3 的位置标记为 1
b = a>3
% 提取 a 中大于 3 的元素 - 列向量形式
c = a(a>3)
% aa:将 a 中不大于 3 的位置赋值为 0
aa = a;
aa(~b) = 0
```

2. 矩阵整体逻辑判断

```
% 普通矩阵转化为逻辑矩阵
b = logical(a-2)
% 判断逻辑矩阵 b 是否全为真
allTrue = all(b(:))
% 判断逻辑矩阵 b 是否至少一项为真
anyTrue = any(b(:))
% 判断两个矩阵是否完全一样
allSame = isequal(a,aa)
allSame2 = all(a(:)==aa(:))
```

3. 索引的逻辑提取

```
% 取 a 中大于 3 的元素的单索引
index = find(a>3)
% 取 a 中第一个大于 3 的元素的单索引
firstIndex = find(a>3,1)
% 取 a 中大于 3 的元素的角标索引
[sub1, sub2] = find(a>3)
```

```
a = 2×3
     1     4     2
     4     2     6
b = 2×3 logical 数组
   0   1   0
   1   0   1
c = 3×1
     4
     4
     6
aa = 2×3
     0     4     0
     4     0     6
b = 2×3 logical 数组
   1   1   0
   1   0   1
allTrue = Logical
   0
anyTrue = Logical
   1
allSame = Logical
   0
allSame2 = Logical
   0
index = 3×1
     2
     3
     6

firstIndex = 2
sub1 = 3×1
     2
     1
     2
sub2 = 3×1
     1
     2
     3
```

图 3-12　矩阵逻辑操作例程

说明：

（1）“a > 3”这样的式子可以直接取得与 a 矩阵规模一致的逻辑矩阵。

(2) 逻辑矩阵做索引可以取得所有满足逻辑的元素,结果按列向量输出。

(3) logical()函数可以将输入变量换算成逻辑矩阵,其中的任意非零元素都将转换为逻辑值1。

(4) all()与any()函数的基本处理单元都是列向量,也就是说,输入一个 $m \times n$ 规模的矩阵,返回的是 $1 \times n$ 的行向量,其中每个元素对应的是每个列向量的逻辑值;如果想对矩阵中所有元素进行判断,先使用冒号索引把矩阵向量化即可。

(5) find()函数的默认输出是单索引,如需要输出角标索引,使用中括号承接输出变量即可。

3.3.3 函数操作

在MATLAB中为矩阵操作设置了大量的函数,可以说能想到的函数都已准备好,最基础和常用的是提取矩阵信息以及矩阵生成的函数(参见图3-13)。

EX 3-9 矩阵函数操作(1)

1. 基础信息

```
a = [1 4 2; 4 2 6]
% 求矩阵的维数
dimensionNumber = ndims(a)
% 求矩阵的规模
sizeMatrix = size(a)
% 求矩阵中元素总个数
elementNumber = numel(a)
% 判断矩阵是否为空阵
isEmptyMatrix = isempty(a)
```

2. 向量生成

```
% 冒号生成
g = 2:2:7
% 线性等距 从2~7
h = linspace(2,7,5)
% 对数等距 从10^2~10^7
l = logspace(2,7,5)
```

3. 矩阵生成

```
% 冒号生成
x = 1:3; y = 2:4;
% 二维和三维网格
[X, Y] = meshgrid(x, y)
% N 维空间中的矩形网格
z = 3:5;
[X, Y, Z] = ndgrid(x, y, z);
```

```
a = 2×3
     1     4     2
     4     2     6

dimensionNumber = 2

sizeMatrix = 1×2
     2     3

elementNumber = 6

isEmptyMatrix = logical
   0


g = 1×3
     2     4     6

h = 1×5
    2.0000    3.2500    4.5000    5.7500    7.0000

l = 1×5
10^7 ×
    0.0000    0.0002    0.0032    0.0562    1.0000


X = 3×3
     1     2     3
     1     2     3
     1     2     3

Y = 3×3
     2     2     2
     3     3     3
     4     4     4
```

图3-13 矩阵函数操作第一部分例程

说明:

(1) numel()函数意为元素个数(Element number),等于各个维度上规模的乘积。

(2) 使用冒号生成线性向量极为常用,其中间的数字为正数或负数均可,省略时表示为

1；最后一个数字表示生成范围，超过它时则结束数字生成算法。

（3）二维、三维、N 维网格的生成是 MATLAB 程序设计中常用的技巧，相当于准备了多维空间中的全遍历点，是矩阵化编程中必不可少的技术之一。

矩阵操作的函数远不止此，下面重点介绍矩阵操作的 8 大“神函数”，包括 sort()，permute()，squeeze()，fliplr()，reshape()，cat()，repmat()，kron()，掌握这 8 个核心函数将让 MATLAB 程序设计更加简洁，如图 3-14 所示。

EX 3-10 矩阵函数操作（2）

1. 对数组元素排序

```
a = [1 3 7 5]
sort(a)
% 矩阵所有元素沿第 2 维度（行）的排序元素
b = [1 2 3; 2 6 4; 4 2 1]
sort(b,2)
% 按第 2 列对行排序（行内不打散）
sortrows(b, 2)
```

2. 重新排列 N 维数组的维度

```
c = rand(3, 4, 5);
% 把第 1 维度与第 3 维度对换
d = permute(c,[3 2 1]);
sizeD = size(d)
```

3. 删除单一维度

```
e = rand(2, 1, 3);
f = squeeze(e)
```

4. 翻转向量

```
h = [1 2; 3 4]
h1 = fliplr(h) % 行向量翻转
```

5. 重构矩阵的形状

```
m = 1:28;
reshape(m, 4, 7)
```

```
a = 1×4
     1     3     7     5

ans = 1×4
     1     3     5     7

b = 3×3
     1     2     3
     2     6     4
     4     2     1

ans = 3×3
     1     2     3
     2     4     6
     1     2     4

ans = 3×3
     1     2     3
     4     2     1
     2     6     4

sizeD = 1×3
     5     4     3

f = 2×3
    0.0605    0.5269    0.6569
    0.3993    0.4168    0.6280

h = 2×2
     1     2
     3     4

h1 = 2×2
     2     1
     4     3

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

图 3-14　矩阵函数操作第二部分例程

说明：

（1）sort()函数用于对矩阵中的元素进行排序，对于向量实现的是从小到大的排序，对于矩阵可以指定排序所依据的维度，因为毕竟一个维度上的大小排序不会恰好使得其他维度也成有序态；sort(a,'descend')可以实现从大到小排序；如果想进行类似于 Excel 表格中的排序操作，即按行进行排序并且维持行内对应关系，则可使用 sortrows()函数，该函数对于 MATLAB 中的表结构也可以进行排序。

（2）permute()函数用于重新排列矩阵中的维度，permute 意为“置换”。

（3）squeeze()函数用于撤销长度为 1 的维度，使矩阵降维，该函数的灵活使用可以减少无用功，squeeze 意为“压榨”。

(4) fliplr()函数用于将矩阵中所有的行向量进行翻转,flipud()函数可以实现矩阵中所有列向量的翻转,而 wrev()函数用于实现所有向量翻转,也可以认为是接连进行了一次行向量翻转与列向量翻转,另外 rot90()函数用于实现把矩阵逆时针旋转 90°。

(5) reshape()函数,用于将一个矩阵在总元素不变的情况下,改变行列数与形状。

还有一些用于矩阵串联、重复、扩展的函数,如图 3-15 所示。

EX 3-11 矩阵函数操作 (3)

1. 沿指定维度串联数组

```
a = ones(2,3);
b = zeros(2,3);
c = cat(2, a, b)
```

2.按指定的行/列数重复指定的矩阵

```
d = [1 2; 0 1]
% 在两个维度上分别重复 2 次和 3 次
repmat(d, 2, 3)
```

3. 智能扩展矩阵

```
e = [1 2; 3 4];
f = [1 0 0; 0 0 0]
g = [1 -1; 0 1]
% Kronecker 张量积
h = kron(e, f)
m = kron(e, g)
```

```
c = 2×6
     1     1     1     0     0     0
     1     1     1     0     0     0
d = 2×2
     1     2
     0     1
ans = 4×6
     1     2     1     2     1     2
     0     1     0     1     0     1
     1     2     1     2     1     2
     0     1     0     1     0     1
f = 2×3
     1     0     0
     0     0     0
g = 2×2
     1    -1
     0     1
h = 4×6
     1     0     0     2     0     0
     0     0     0     0     0     0
     3     0     0     4     0     0
     0     0     0     0     0     0
m = 4×4
     1    -1     2    -2
     0     1     0     2
     3    -3     4    -4
     0     3     0     4
```

图 3-15 矩阵函数操作第三部分例程

(1) cat()函数可以把若干矩阵,沿“指定维”方向拼接为高维矩阵,函数的第一个输入变量就是指定的维度,在高维矩阵拼接操作时非常实用,注意拼接之前确定输入矩阵规模是可以拼接的,否则报错。

(2) repmat()函数用于按指定的行/列数重复(阵列)指定的矩阵,可以将一个小矩阵按规律快速扩展成一个周期性矩阵。

(3) kron()函数返回两矩阵的 Kronecker 张量积,简单地说,就是将第一个矩阵的各元素分散开,再将每个元素与第二矩阵相乘,是一种高级的矩阵扩展方法,灵活使用可以在一些场合四两拨千斤。

3.3.4 实用技巧

矩阵操作还有一些常用的小技巧,比如下面关于矩阵性质判断的技巧:

(1) 如何判断矩阵 ***a*** 是否为空矩阵。

```
isempty(a)
```

(2) 如何判断是否存在 a 变量。

```
exist('a')
```

(3) 如何判断矩阵 $\boldsymbol{a}$ 是否为行向量、列向量、标量。

```
isrow(a)
iscolumn(a)
isscalar(a)
```

再比如关于矩阵中指定元素的删除的技巧：

(1) 如何删除 a 中最后一个元素。

```
a(end) = []
```

(2) 如何删除向量 $\boldsymbol{a}$ 中偶数位置的元素。

```
a(rem(a, 2) == 0) = []
```

(3) 如何删除向量 $\boldsymbol{a}$ 中重复的元素。

```
a = unique(a)
```

(4) 如何删除矩阵 $\boldsymbol{a}$ 中重复的列向量。

```
a = unique(a, 'row')
```

(5) 如何删除矩阵或向量中的 NaN、inf。

```
a(isnan(a)) = []
a(isinf(a)) = []
```

(6) 如何删除 a 中全为 0 的列。

```
a(:, sum(abs(a)) == 0) = []
```

类似的技巧还有很多，由于不同的用户习惯与熟练程度有所不同，每个人对技巧的定义也不同，建议在平日的编程积累中将看到或想到的小技巧总结成电子文档，日积月累逐渐成长为 MATLAB 的应用高手。

3.4 矩阵运算

向量是最基础的矩阵，向量运算往往可以实现“批量化的元素运算”；矩阵是由向量组成的，矩阵是“向量的向量”，所以矩阵的运算往往可以实现“批量化的向量运算”；因此，在 MATLAB 中优先考虑矩阵运算，其次为向量运算，最后再考虑用元素运算实现。学习使用 MATLAB 时，要注意 MATLAB 打包了相当多的函数，虽然这些函数也可以自己编程实现，但是寻找使用打包好的函数则是事半功倍，拥有更快的计算速度和稳定性，节省大量编程时间。

3.4.1 算术运算

矩阵/向量有一些基本的算术运算函数，可以批量对矩阵中元素进行算术处理或得到统计信息，如图 3-16 所示。

EX 3-12 矩阵算术运算

1. 算术处理

```
a = [-2 0 1; 2.1 3 1.5]
absA = abs(a) % 绝对值
roundA = round(a) % 四舍五入取整
signA = sign(a) % 符号阵（1/0/-1）
```

2. 统计处理

```
minA = min(a) % 列向量元素的最小值
maxA = max(a) % 列向量元素的最大值
meanA = mean(a) % 列向量元素的平均值
sumA = sum(a) % 列向量元素总和
```

3. 向量运算

```
b = [1 2 0]; c = [0 3 1];
% 向量 b 和 c 的内积
dotBC = dot(b, c)
% 向量 b 和 c 的外积
crossBC = cross(b, c)
% 求 b 向量的范数（欧氏长度/模）
normb = norm(b)
```

```
a = 2×3
   -2.0000         0    1.0000
    2.1000    3.0000    1.5000
absA = 2×3
    2.0000         0    1.0000
    2.1000    3.0000    1.5000
roundA = 2×3
    -2     0     1
     2     3     2
signA = 2×3
    -1     0     1
     1     1     1
minA = 1×3
    -2     0     1
maxA = 1×3
    2.1000    3.0000    1.5000
meanA = 1×3
    0.0500    1.5000    1.2500
sumA = 1×3
    0.1000    3.0000    2.5000
dotBC = 6
crossBC = 1×3
     2    -1     3
normb = 2.2361
```

图 3-16　矩阵算术运算例程

说明：

(1) abs()函数求取每个元素的绝对值，对于复数来说是求复数的模。

(2) round()函数求四舍五入后的最近整数，与其类似的函数还有：fix()函数无论正负，舍去小数至最近整数，floor()地板函数，求四舍五入到小于或等于该元素的最接近整数，ceil()天花板函数，求四舍五入到大于或等于该元素的最接近整数。

(3) sign()是很常用的符号函数，对每个元素计算返回值：1(元素大于 0)，0(元素等于 0)，−1(元素小于 0)。

(4) mean()为平均值函数，求向量中的元素平均值，类似的还有：median()中位数函数，求向量中元素的中位数，std()标准差函数，求向量中元素的标准差。

(5) dot()为点乘函数，求得向量的内积，结果为一个数值；cross()为叉乘函数，求得向量的外积，要求输入的向量长度必须为 3(或者是以长度为 3 的列向量组成的矩阵)，结果也是一个长度为 3 的向量(或矩阵)。

3.4.2 逻辑运算

逻辑值只有两种：真(true,1)或假(false,0)，在程序设计中为判断起到非常重要的作用。MATLAB可对矩阵中的元素进行批量的逻辑运算处理，灵活使用可使程序极致简洁，如图3-17所示。

EX 3-13 矩阵逻辑运算

1. 逻辑矩阵

```
a = [-2 0 3];
% 判断矩阵 a 是否为逻辑矩阵
islogical(a)
% 将 a 转换为逻辑矩阵
b = logical(a)
% 使用 true/false 赋值逻辑矩阵
c = [true true false]
```

2. 逻辑运算

```
andBC = b & c % 与
orBC = b | c % 或
notB = ~b % 非
xorBC = xor(b, c) % 异或
```

3. 逻辑判断

```
allB = all(b) % 所有为真
anyB = any(b) % 任意为真
find(b) % 寻找真
```

4. 短路逻辑

```
shortAnd = 0 && 1 % 与
shortOr = 1 || 0 % 或
```

```
ans = logical
   0
b = 1×3 logical 数组
     1   0   1

c = 1×3 logical 数组
     1   1   0

andBC = 1×3 logical 数组
     1   0   0

orBC = 1×3 logical 数组
     1   1   1

notB = 1×3 logical 数组
     0   1   0

xorBC = 1×3 logical 数组
     0   1   1

allB = logical
   0
anyB = logical
   1
ans = 1×2
     1     3

shortAnd = logical
   0
shortOr = logical
   1
```

图3-17 矩阵逻辑运算例程

说明：

(1) logical()为逻辑化函数，将矩阵中所有元素变成逻辑值，原则是“遇0为0，非0即1”。

(2) true和false在MATLAB中是关键字，代表逻辑1与逻辑0，通常用1和0代替更为简洁高效。

(3) 短路逻辑运算与普通逻辑运算有两处不同：一是短路逻辑运算要求符号两边必须为逻辑元素而不能是矩阵；二是如果计算第一个逻辑元素就得到了整体表达式的必然值，则不必再计算后面的式子；短路逻辑常用于程序流中的判断，如if或while后面的表达式。

3.4.3 关系运算

同规模矩阵可以进行关系运算，可以比较两矩阵对应元素的大小关系；在同一矩阵中相邻元素之间可以用导数函数求取大小关系；如果矩阵中的信息是点坐标，还可以求取点与点之前的距离关系，如图 3-18 所示。

EX 3-14 矩阵关系运算

1. 关系运算

```
a = [1 2 3]; b = [3 2 1];
a == b
a >= b
a ~= b
% 确定两个矩阵完全相等
isEqual = isequal(a, b)
```

2. 相邻关系 - 差分

```
c = [1 4 7 0 5 5 2 1]
% 相邻元素的差：后一元素减前一元素
diffC = diff(c)
% 符号函数，明确大小关系
signDiffC = sign(diffC)
```

3. 点关系 - 距离

```
d = [1 2; 2 3; -1 0; 0 1]
% 各点之间的欧氏距离
distVector = pdist(d)
% 将距离向量化为更形象的矩阵
disMatrix = squareform(distVector)
```

```
ans = 1×3 logical 数组
   0   1   0

ans = 1×3 logical 数组
   0   1   1

ans = 1×3 logical 数组
   1   0   1

isEqual = logical
   0

c = 1×8
     1     4     7     0     5     5     2     1

diffC = 1×7
     3     3    -7     5     0    -3    -1

signDiffC = 1×7
     1     1    -1     1     0    -1    -1

d = 4×2
     1     2
     2     3
    -1     0
     0     1

distVector = 1×6
    1.4142    2.8284    1.4142    4.2426    2.8284    1.4142

disMatrix = 4×4
         0    1.4142    2.8284    1.4142
    1.4142         0    4.2426    2.8284
    2.8284    4.2426         0    1.4142
    1.4142    2.8284    1.4142         0
```

图 3-18 矩阵关系运算例程

说明：

(1) 关系运算符包括：确定等于(==)、确定不等于(~=)、确定大于(>)、确定大于等于(>=)、确定小于(<)、确定小于等于(<=)。

(2) isequal()函数可以确定两矩阵完全相等，如果矩阵中有 NaN 时，可使用 isequaln()函数，其将 NaN 值视为相等。

(3) diff()为差分函数，返回比原向量少一个元素的差分向量，比如差分向量位置为 n 的元素，即等于原向量位置 $n+1$ 的元素减去位置 n 的元素，配合符号函数即可得到两元素之间的关系是大于、小于还是等于。

(4) 当矩阵中存储的是点坐标信息时，可以使用 pdist()函数求取各点间的距离，这里的距离是欧几里得距离，也即两坐标点连线的长度，取得的是向量形式，也可以使用 squareform()函数将向量形式转化为矩阵形式，这样更为直观形象，也更容易通过角标直接取得两点之间的距离；pdist()函数还有一个很常用的选项，即 pdist(X,'cityblock')，从字面翻译为街区距离，也称曼哈顿距离，最早命名是用于在曼哈顿街区表征交通，因此它不是直

线距离，而是两点之间只走横线和竖线的路程距离。

3.5 矩阵编程

矩阵编程（也译为“向量化编程”）是MATLAB中的核心特色，相比于常规的C语言代码风格，矩阵编程是一种非常可取的编程风格：

（1）更接近自然数学语言，代码涵义更容易阅读。

（2）取代循环结构，代码大大简洁，减少出错概率。

（3）计算速度大大提高，使用MATLAB独有的针对矩阵的计算引擎，效率远远超出常规编程。

矩阵编程以矩阵为核心；本章前面小节中关于矩阵与数据类型、矩阵与数据结构、矩阵操作与运算的学习已经包含了矩阵编程的大部分内容，熟练掌握并灵活应用这些技术即可认为基本上实现了矩阵风格的编程，更深入的内容将在第6章学习。

3.5.1 矩阵编程举例

本节列举5组具体的代码示例，分别为以C语言为典型代表的元素风格，以及MATLAB的矩阵化编程风格，其中C语言风格也是完全可行的M代码，只不过执行的效率略低、代码略多、理解性较差。

（1）如何计算1001个从0～10之内的值的正弦值？

C语言风格：

```
i = 0;
for t = 0:0.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

MATLAB矩阵风格：

```
t = 0:0.01:10;
y = sin(t);
```

MATLAB中大量的计算函数都是既可以对元素计算，也可以对向量及矩阵进行批量计算，这使得代码异常简洁与高效。

（2）已知10000个圆锥体的直径D和高度H，如何求它们的体积？

C语言风格：

```
for n = 1:10000
   V(n) = 1/12 * pi * (D(n)^2) * H(n);
end
```

MATLAB矩阵风格：

```
V = 1/12 * pi * (D.^2). * H;
```

(3) 如何计算某向量每 5 个元素的累加和?

C 语言风格:

```
x = 1:10000;
ylength = ...
(length(x) - mod(length(x),5))/5;
y(1:ylength) = 0;
for n = 5:5:length(x)
    y(n/5) = sum(x(1:n));
end
```

MATLAB 矩阵风格:

```
x = 1:10000;
xsums = cumsum(x);
y = xsums(5:5:length(x));
```

cumsum()为累积和函数,求取的是向量或矩阵中列向量从第 1 个元素开始的累积和,是非常常用的函数。

(4) 假设矩阵 **A** 代表考试分数,行表示不同的班级,如何计算每个班级的平均分数与各分数的差?

C 语言风格:

```
A = [97 89 84; 95 82 92; 64 80 99;76 77 67; 88 59 74; 78 66 87; 55 93 85];
mA = mean(A);
B = zeros(size(A));
for n = 1:size(A,2)
    B(:,n) = A(:,n) - mA(n);
end
```

MATLAB 矩阵风格:

```
A = [97 89 84; 95 82 92; 64 80 99;76 77 67; 88 59 74; 78 66 87; 55 93 85];
devA = A - mean(A)
```

即使 A 是一个 7×3 矩阵,mean(A)是一个 1×3 向量,MATLAB 也会隐式扩展该向量,就好像其大小与矩阵相同一样,并且该运算将作为正常的按元素减法运算来执行。

(5) 如何计算两个向量形成的所有组合乘积的正弦值?

C 语言风格:

```
x = -5:0.1:5;
y = (-2.5:0.1:2.5)';
N = length(x);
M = length(y);
for ii = 1:M
    for jj = 1:N
        XO(ii,jj) = x(jj);
        YO(ii,jj) = y(ii);
        ZO(ii,jj) = …
sin(abs(x(jj) * y(ii)));
    end
end
```

MATLAB 矩阵风格:

```
[X,Y] = meshgrid(-5:0.1:5, -2.5:0.1:2.5);
Z = sin(abs(X. * Y));
```

“网格”是一个重要的概念，当目标是计算多个向量中的每个点的所有组合时，多个向量将形成一个网格，把向量扩展成矩阵以形成网格的方式称为“显式网格”，可以使用meshgrid()或 ndgrid()函数来进行创建。遇到此类问题，往往第一反应是建立循环，然而凡是循环建立之时，都应该思考是否可以使用矩阵化编程风格。

3.5.2 矩阵编程要点

关于矩阵编程，除了掌握本章前述的概念、方法与技巧之外，还要理解以下编程要点。

(1) 编程前要对矩阵进行内存预分配。

编程前，首先使用如 zeros()或 ones()函数，将要处理的矩阵的内存空间开辟出来，以避免在程序运行的中途反复更改矩阵的规模，这会消耗大量的时间与算力。

(2) 矩阵意义要单一且明确。

每个矩阵只有一个涵义，矩阵中的每个维度也只有单一意义，不要在一个矩阵中存储多种类型信息；要借助良好的命名以及清晰的注释保证每个矩阵的意义明确。

(3) 编程过程中检查和确认矩阵的规模。

通过时刻检查和确认，保证在程序流中每一阶段，矩阵的规模都清晰可知，并通过注释进行标记，以避免出现由于规模异常导致的程序报错。

(4) 优先使用逻辑索引操作。

逻辑索引的使用将大大简化代码实现，也是 M 代码简洁高效的重要基石，在对矩阵进行操作时，往往一句逻辑索引即可解决一些比较复杂的问题。

(5) 慎用循环语句。

绝大多数的循环语句都可以使用矩阵编程来替代，刚入门的初学者可能由于不擅长矩阵编程而弃之不用，这是不明智的，反而应该是每当准备使用循环时，都要考虑是否有升级为矩阵编程的可能性。

(6) 多向量计算时的遍历网格化。

多向量计算时最容易形成无法矩阵化编程的错觉，以为循环是唯一的解决方式，然而网格生成技术常可以将代码大大简化，同时也降低了代码中出错的可能性。

(7) 元胞数组和结构体的使用。

元胞数组和结构体很好地解决了矩阵的存储和索引局限，然而它们无法像矩阵一样进行高效的批量计算，因此只是在特殊场合下使用，除此之外矩阵仍是最优先使用的数据结构。

(8) 对于以 0 元素为主的矩阵，尽量使用稀疏矩阵。

顾名思义，稀疏矩阵就是指矩阵中的非 0 元素较少，因此使用常规矩阵的存储与计算就比较低效，MATLAB 提供了针对于此的“稀疏矩阵模式”，可以大大提高此类矩阵的运算速度，同时减少存储空间。

本章小结

本章集中研究了 MATLAB 的核心概念与思想——矩阵，从数据类型、数据结构、矩阵操作、矩阵运算、矩阵编程 5 个方面全方位介绍了 MATLAB 的特点与应用，帮助读者加深 M 语言是以矩阵为核心的印象，为今后对于 MATLAB 的学习打下坚实的基础。