第 3 章

矩阵：MATLAB 的核心概念

MATLAB 即“矩阵实验室”，以其矩阵中心的编程哲学和数据结构而著称。在学习了 MATLAB 的基本操作和软件架构之后，本章将引导读者深入探索 MATLAB 语言的核心——矩阵。掌握矩阵的基本概念、操作技巧、运算规则以及独特的编程风格，读者将能够轻松步入高效编程的新纪元，这是 MATLAB 与众不同的关键所在。

本章将带领读者从编程的根基——“数据类型与结构”出发，深入理解矩阵与它们的紧密联系。我们将通过探讨矩阵的操作方法和运算技巧，一步步带领读者进入以矩阵为核心的编程世界。最终，通过精选的编程实例，展示矩阵编程技术的精华要义。这一章的内容既基础又重要，学习时不能只满足于记忆理论，关键在于通过大量实践将知识转换为自己信手拈来的技能。

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 中的三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。本书中为了呼应 MATLAB 的名字“矩阵实验室”，同时为了强化软件核心思想，使用“矩阵”这个词用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 $1\times n$ 的 1 维矩阵。

(4) 列向量：规模 $n\times1$ 的 1 维矩阵。

(5) 普通矩阵：规模 $n\times m$ 的 2 维矩阵。

(6) 多维数组：规模 $n\times m\times\cdots$ 的多维矩阵。

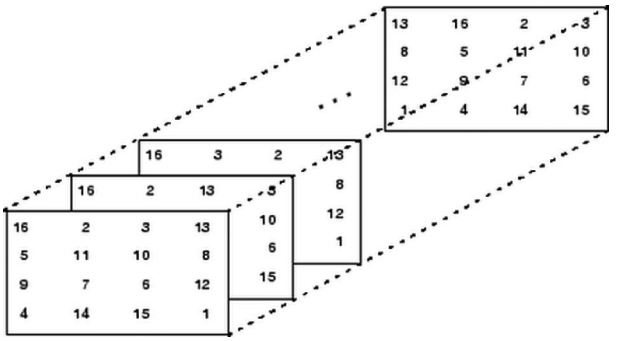

图 3-1　多维数组/矩阵示意

MATLAB 的所有数据类型均是以矩阵为核心及基础的，矩阵中的元素也可以是实数、复数、字符或符号变量。

3.1.1　数值矩阵："数"的结构

MATLAB 中的数值，拥有符合电气与电子工程师协会（Institute of Electrical and Electronics Engineers，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型（double）和单精度浮点型（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，基本可以涵盖所有应用场合；MATLAB 中次常用的是带符号 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
% 整型判断
e = b == 2*pi/3.14
f = b == int8(2*pi/3.14)
```

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]
```

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

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）不建议直接判断两个浮点数是否相等，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。

(2) 复数中的 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：“字”的结构

在 MATLAB 的编程语言中，文本数据主要以两种形式存储：“字符”和“字符串”。字符矩阵是由单个字符组成的数组，其中每个字符实际上是矩阵的一个元素。用户可以简单地通过单引号(' ')来创建一个字符行向量。字符串矩阵由一系列字符串构成，每个字符串元素可以是任意长度，不受限制。从 MATLAB 的 R2017a 版本开始，开发者可以通过使用双引号(" ")来直接创建字符串，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

字符与字符串的区分在实际编程中至关重要，因为字符矩阵和字符串矩阵在处理和操作文本数据时各有优势。字符矩阵便于进行传统的字符级操作，而字符串矩阵则提供了更高级的文本处理功能，如更便捷的字符串拼接、搜索和替换操作。透过这一细微差别，MATLAB 使得文本处理既灵活又强大，可以满足不同场景下的编程需求。

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来，但这并不表示它是一个字符串，它的类型仍是字符类型。字符矩阵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，这时可以使用 strlength()函数。

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

（4）在工作区中，字符变量的图标是“ch”，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”的结构

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 MATLAB 一直强力推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不需要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；另一种方式是使用 sym()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“数学：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，“矩阵”的概念可以说是整个现代科学的基础，其底层逻辑就在于，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，比如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、需要使用名称而不是数字来索引元素，以及处理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，还有三种核心数据结构：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 四种数据结构比对表

	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	否	是	是	否
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，如果仅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，可惜的是大多数教材与课堂并未给予其足够的重视。

3.2.1　元胞数组：多元数据的集成

在 MATLAB 中，元胞数组（cell array，或称为“元胞阵”）是一种特殊而强大的数据结构，它扩展了传统矩阵的概念。与普通矩阵不同，元胞数组中的每一个“元胞”都可以包含不同类型的数据，无论是数值、字符、字符串、符号表达式，还是另一个矩阵，甚至是另一个元胞数组，都可以轻松存储在这个灵活的容器中。

元胞数组的应用场景极其广泛，尤其是在处理不规则数据集时表现出其独到的优势。例如，当用户需要用数字索引来组织数据，但每个元素（比如子矩阵）的大小并不统一时，元胞数组成为最佳的选择，如图 3-5 所示。它的灵活性和实用性使其在 MATLAB 编程中不可或缺，无论是数据组织、信息存储还是高级编程技巧，元胞数组都能够发挥重要作用。

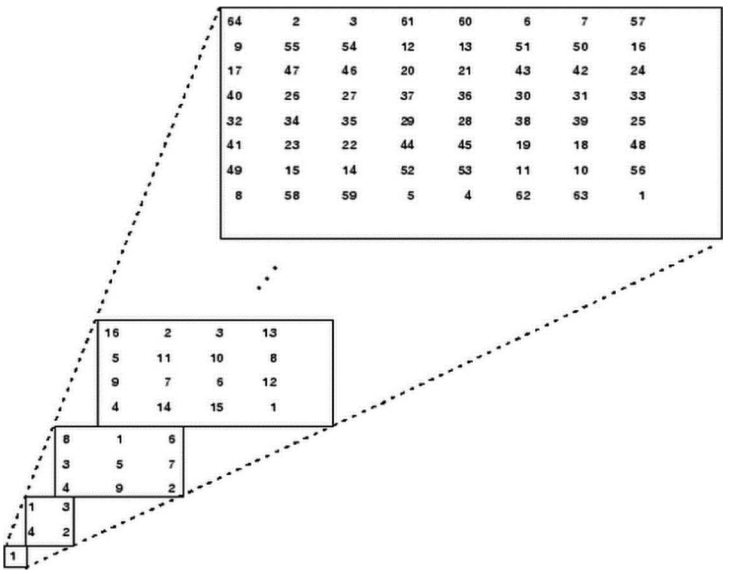

图 3-5　元胞数组的一种存储形式

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。

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

```
% 使用圆括号，访问的是元素，即元胞
a = cellA(1,1)
% 使用花括号，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

说明：

(1) 元胞数组采用花括号赋值，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，因此直接使用圆括号索引，得到的是元胞元素而不是其中的数据内容；其实，使用花括号索引方式，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()。

3.2.2　结构体：有序数据的框架

在众多编程语言中，结构体(Structure)扮演着至关重要的角色。它是一种特殊的数据结构，以一种类似于“树”的形式，通过名称(也称为字段)来索引和存储数据。MATLAB 在处理结构体时展现出了其独特的灵活性，使其成为数据组织和处理的强大工具。

在 MATLAB 中，结构体能够存储的元素种类极为丰富，这一点与元胞数组相似。无论是数值、字符、字符串、符号表达式，还是一个完整的矩阵乃至另一个元胞数组，都可以成为结构体中的一个元素。更进一步，结构体的元素甚至可以是另一个结构体，允许我们通过多级结构直观地定义复杂的数据关系。此外，结构体还可以形成数组，这时可以局部使用数字索引来进行精准访问。

结构体的树状组织方式为程序设计带来了极大的便利，它使得将众多零散而复杂的数据有序地归纳和分类成为可能，如图 3-7 所示。在 MATLAB 编程实践中，灵活运用结构体不仅能够帮助用户更有效地管理和操作数据，还能让程序的设计更加清晰和高效。无疑，掌握结构体的使用，是每位 MATLAB 编程者提升技能的关键一步。

说明：

(1) 英文句点“.”可以用来定义结构体层级，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) 在程序设计中，常用结构体主名称作为一个“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“面向对象编程”还有一定距离，但是往往可以大幅简化程序结构、提高代码的清晰度。

(4) 结构体中内容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。

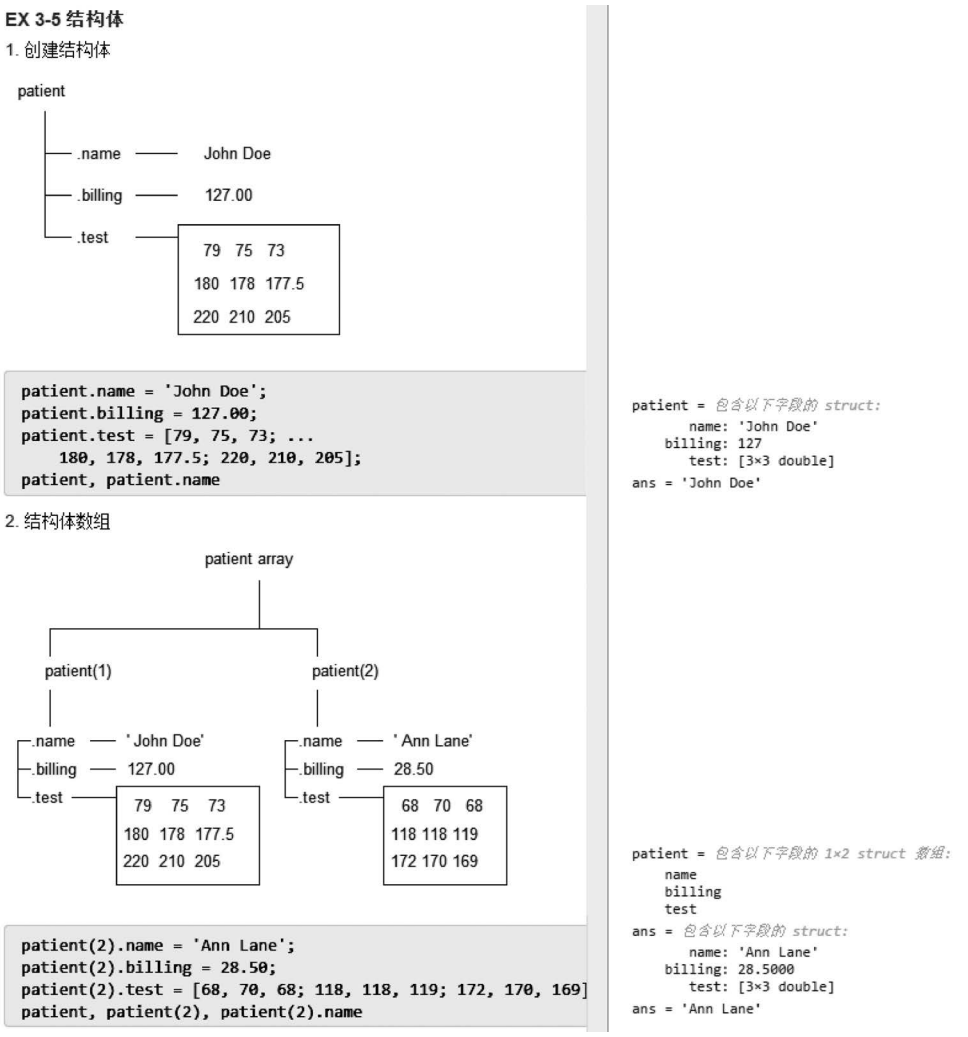

图 3-7　结构体例程

3.2.3　表：数据分析的利器

想象一下，如果有一种数据结构能够像电子表格那样直观，又具备 MATLAB 强大的数据处理能力，那会是怎样的便捷？这正是表(Table)所带来的革命性创新。表是一种特殊的异化数据结构，建立在矩阵之上，但提供了更多的灵活性和功能。

在表中，每个变量，就像电子表格中的一列，可以有不同的数据类型和大小。唯一的要求是，所有变量必须有相同的行数，即相同数量的观测记录。变量并不局限于单列数据，也可以是一个多列的矩阵，只要保持行数一致就可以。自从 R2013b 版本引入以来，表数据结构很快就取代了统计工具箱中的 dataset 数据类型，并迅速成为 MATLAB 用户在数据处理和程序设计中的宠儿。无论是存储实验数据、管理观测点(行)和测量变量(列)，还是从文本

文件和电子表格中提取数据，表都能够以其优雅的方式完美胜任。

表的一个关键优势在于其基于名称的索引功能，这使得数据检索速度极快，类似于哈希表的效率。与之相比，在矩阵或元胞数组中进行同样的操作，可能需要遍历全量数据，尤其在数据量庞大时，效率就显得不那么理想了。熟练运用表会极大提升数据分析的效率和准确性。因此，在 MATLAB 中，表不仅是数据分析师的好帮手，更是程序设计师的得力助手。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常是以.csv 格式存在的，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。

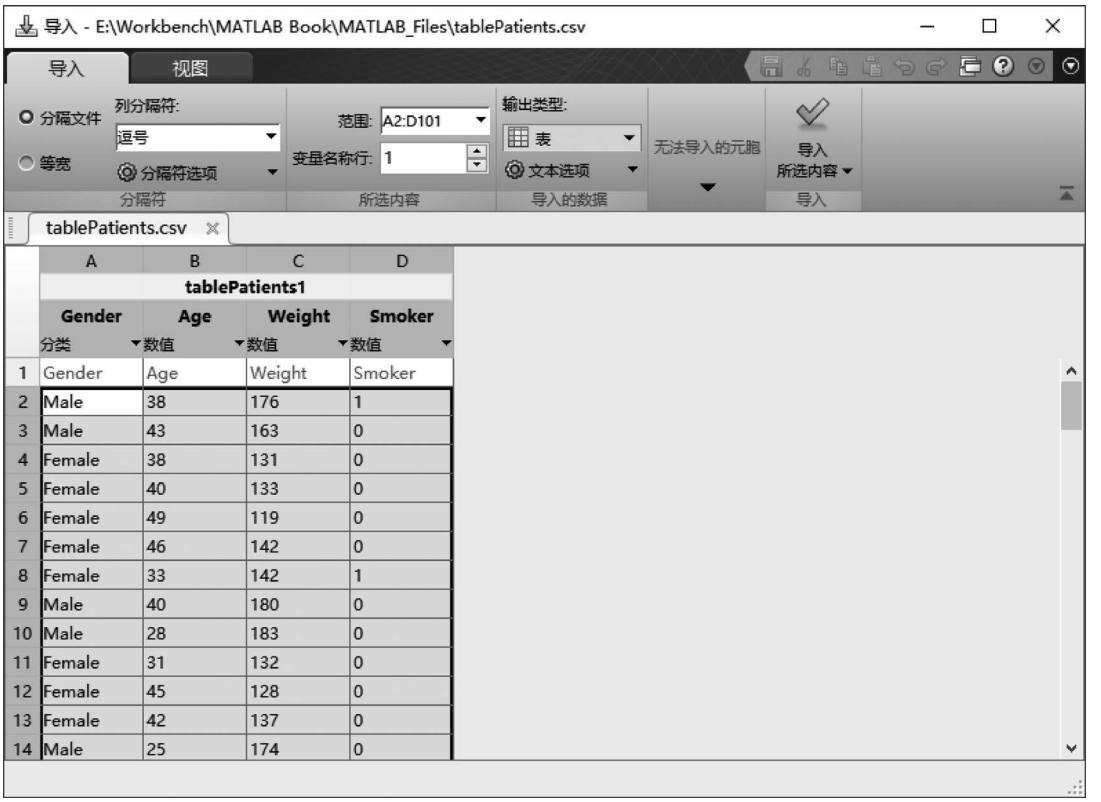

图 3-9　数据文件导入窗口

3.3　矩阵操作

在今天这个数据驱动的时代，大数据与机器学习已成为全球的热门话题。矩阵，作为这些领域不可或缺的数据结构，之所以颇受青睐，原因何在？答案在于矩阵的“批量操作”特性。通过矩阵，我们能够高效地进行批量计算，同时将数据打包，让其更易于理解和分析。

作为 MATLAB 的核心，矩阵的强大之处不仅在于其数据结构本身，还在于那些针对矩阵设计的操作技能。通过灵活运用这些操作，用户的代码可以达到令人惊叹的简洁与高效，体验到 MATLAB 相比其他编程语言的巨大优势。精通矩阵操作是学习 MATLAB 的关键之一，这包括熟悉矩阵的索引、逻辑以及函数操作。为了提升效率和保持代码的清晰，推荐在操作矩阵时养成以下三个良好习惯。

（1）整存整取：在可能的情况下，对整个矩阵进行操作，而非单独处理每个元素。这样可以最大限度地减少循环的使用，提升代码效率。

（2）清晰的维度意义：在初始化矩阵时，清楚地注释每个维度的含义，并在维度变换时及时更新注释，确保维度意义的明确性。

(3) 统一的维度数据地位：避免在同一维度中混合不同意义或不同层级的数据，这样做可以避免在编程时引入不必要的复杂性和混乱。

掌握 MATLAB 中的矩阵操作，将使用户在数据处理的道路上游刃有余，无论是进行简单的数据分析还是构建复杂的模型，都能够高效地处理和解读数据，让科学计算变得更加直观和有趣。

3.3.1　索引操作：矩阵的定位术

在 MATLAB 的世界里，掌握如何精确地访问矩阵中的元素是基础中的基础。这一切，都离不开一个关键技巧——索引。通过索引，我们可以轻松实现对矩阵中特定部分的提取和赋值，使数据处理变得既高效又直观。索引的魅力在于其两种形式：单索引(index)和角标索引(subscript)。单索引是将矩阵视为一个长向量，通过一个从 1 开始的数字序列来访问元素。在处理多维矩阵时，单索引按照维度的顺序(如行、列、页等)将矩阵展开成向量，从而用一个数字来定位任意元素。角标索引直接利用各维度的角标来定位元素，就像我们通常在数学中用行和列的坐标来找到矩阵中的一个特定元素一样简单。

想象一下，有一个三维矩阵，通过角标索引，我们可能需要提供三个坐标(i，j，k)来访问一个元素。而通过单索引，我们只需要一个数字。如图 3-10 所示为单索引与角标索引之间的对应关系，可以让读者能够根据需要灵活选择索引方式。

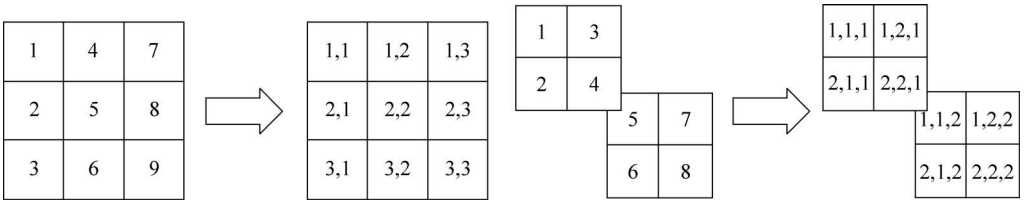

图 3-10　单索引与角标索引的对应关系

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。

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，这样的程序稳健性强，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。

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

3.3.2　逻辑操作：决策与筛选的智慧

当我们深入 MATLAB 的世界，逻辑操作便显现出其无可比拟的实用性。这种操作技术不仅是数据处理的利器，而且在程序设计中也扮演着至关重要的角色。

首先，逻辑操作允许我们通过简单的逻辑表达式来筛选矩阵中满足特定条件的元素，从而实施批量操作。这意味着，不需要烦琐的索引指定，仅凭一个逻辑判断就能高效地对数据集进行增、删、改、查操作。其次，逻辑矩阵本身可以作为控制流语句的条件，使得程序的分支决策更加直观和精确。MATLAB 这种基于矩阵的整体判断方法，在编程语言的世界里可谓独树一帜，它极大地简化了编程过程，节约了用户的时间和精力。此外，find()函数提供了一种快速的方式来从矩阵中提取满足逻辑条件的索引，它是逻辑操作的完美伴侣，使得从庞大数据集中定位特定元素变得轻而易举。

在 MATLAB 的程序设计中，逻辑操作的威力无处不在。它不仅能够帮助用户决定程序的走向，还能优化处理流程，使代码更加紧凑和高效。矩阵逻辑操作例程如图 3-12 所示，逻辑操作的实用性和强大功能为 MATLAB 用户提供了极大的方便，让复杂的决策变得简单且高效。

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。

EX 3-8 矩阵逻辑操作

1. 逻辑判断式作索引

```
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

```
% 普通矩阵转换为逻辑矩阵
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

(4) all()函数与 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的元素进行判断，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，使用方括号承接输出变量即可。

3.3.3　函数操作：矩阵处理的魔法

在 MATLAB 中为矩阵操作设置了大量的函数，可以说用户能想到的函数都已经准备好了，最基础和常用的是提取矩阵信息以及矩阵生成的函数，如图 3-13 所示。

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；最后一个数字表示的是生成范围，生成的数字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 程序设计中非常常用的技巧，相当于生成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。

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)
% 判断矩阵是否为空矩阵
isEmptyMatrix = isempty(a)
```

2. 向量生成

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

矩阵操作的函数远不止此，下面隆重推出矩阵操作的八大核心函数，包括 sort()，permute()，squeeze()，fliplr()，reshape()，cat()，repmat()和 kron()函数，这八个核心函数的掌握将让 MATLAB 程序设计变得十分简洁，如图 3-14 所示。

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呈有序态；sort(a,'descend')可以实现从大到小排序；如果用户想进行类似于 Excel 表格中的排序操作，即按对行进行排序并且行内不打散对应关系，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。

（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。

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

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. 沿指定维度串联矩阵

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

2.按指定的行数/列数重复指定的矩阵

```
d = [1 2; 0 1]
%使矩阵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

说明：

（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，注意拼接之前需要确定输入矩阵规模是可以拼接的，否则会报错。

（2）repmat()函数用于按指定的行数/列数重复指定的矩阵，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个矩阵相乘，是一种高级的矩阵扩展方法，灵活使用此函数可以在一些场合四两拨千斤。

3.3.4 实用技巧：提升编程效率小妙招

矩阵操作还有一些常用的小技巧，比如关于“矩阵性质的判断”。

（1）如何判断矩阵 ***a*** 是否为空矩阵？

```
isempty(a)
```

（2）如何判断是否存在变量 *a*？

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

（3）如何判断矩阵 ***a*** 是否为行向量、列向量、标量？

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

再如关于“矩阵中指定元素的去除”。

（1）如何删除矩阵 ***a*** 中最后一个元素？

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

（2）如何删去向量 ***a*** 中偶数位置的元素？

```
a = a(1:2:end)
```

（3）如何删去向量 ***a*** 中重复的元素？

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

（4）如何删去矩阵 ***a*** 中重复的列向量？

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

（5）如何去除矩阵或向量中的 NaN、inf？

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

（6）如何删除矩阵 ***a*** 中全为 0 的列？

```
a = a(:,any(a))
```

类似的技巧还有很多，由于不同用户的习惯与熟练程度有所不同，每个人对技巧的定义也不同，建议用户在平日的编程积累中将看到想到的小技巧总结到电子文档中，日积月累逐

渐就能成长为 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，最后再考虑元素运算。

作为初学者，读者会发现 MATLAB 中有一整套预先打包好的函数可以使用。这些函数提供了编码的快捷方式，不仅加快了计算速度，而且提高了程序的稳定性。虽然有时候我们可能会出于学习的目的尝试手动编写这些函数，但在实际应用中利用这些现成的函数能让我们事半功倍，节约宝贵的时间。

3.4.1　算术运算：矩阵的计算法则

矩阵有一些基本的算术运算函数，可以批量对矩阵中的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) % 四舍五入后取最近整数
signA = sign(a) % 符号阵（1/0/-1）
```

2. 统计处理

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

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（欧氏距离/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，求四舍五入后小于或等于该元素的最接近整

数；ceil()天花板函数，求四舍五入后大于或等于该元素的最接近整数。

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

(4) mean()函数为平均值函数，求向量中元素的平均值，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()函数为点乘函数，求向量的内积，结果为一个数值；cross()函数为叉乘函数，求向量的外积，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：矩阵的真与假

逻辑值只有两种，要么为真(true，1)，要么为假(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)
% 将矩阵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()函数为逻辑化函数，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，代表着逻辑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。短路逻辑运算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：比较与排序的逻辑

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间的距离关系，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：等于（==）、不等于（~=）、大于（>）、大于或等于（>=）、小于（<）、小于或等于（<=）。

（2）isequal()函数可以确定两个矩阵完全相等，如果矩阵中有 NaN 时，可使用 isequaln()函数，其将 NaN 值视为相等。

（3）diff()函数为差分函数，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，这里的距离是欧氏距离，即两坐标点连线的长度，取得的是向量形式，也可以使用 squareform()函数将向量形式转换为矩阵形式，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，也叫曼哈顿距离，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，或者说向量化编程，是 MATLAB 的一大亮点，提供了一种与众不同的编码风格，尤其与传统的 C 语言等编程风格相比，它有以下几个显著的优势。

（1）自然直观：它贴近我们的数学直觉，使得代码不仅易于编写，而且更加易于理解。

（2）简化代码：通过减少循环和迭代，代码变得更加简洁，同时降低了出错的可能性。

（3）计算提速：凭借 MATLAB 为矩阵运算量身定做的计算引擎，其执行速度远超传统的编程方法。

矩阵编程围绕矩阵展开，将矩阵置于编程的核心位置。通过前几节关于矩阵与数据类型、结构以及矩阵操作和运算的学习，读者已经掌握了矩阵编程的精髓。一旦读者能熟练并灵活地运用这些技巧，就已经掌握了向量化编程的精华。更深层次的探索和应用将在第 7 章的软件设计部分展开。简而言之，矩阵编程不仅提升了编码效率，还让我们的程序更加紧凑、高效。它是 MATLAB 编程的核心，为解决复杂的数学和工程问题提供了一个强大的工具。随着读者深入学习，将更加领略到它的威力。

3.5.1 矩阵编程举例：理论与实践的结合

在这一节中，我们精选了 5 组对比示例，展现了传统的 C 语言元素级别编程风格与 MATLAB 的矩阵化编程风格。虽然 C 语言风格的代码在 MATLAB 中同样可执行，但它往往意味着更低的效率、更多的代码行数以及较差的可读性。通过下面的实例，我们希望读者能够直观地感受到矩阵编程的概念和它带来的显著优势。

（1）如何计算 1001 个从 0 到 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的，这使得 MATLAB 的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;
```

利用点运算符进行元素级操作。例如，使用.*来替代逐元素的乘法循环，使用./进行逐元素的除法操作。这样的操作保证了代码的简洁性和执行速度。

（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()函数为累积和函数，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。MATLAB拥有丰富的库函数，比如sum()、mean()、max()函数等，它们对向量和矩阵的运算经过优化，比手写循环更快更可靠。

（4）假设矩阵 $\boldsymbol{A}$ 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 $\boldsymbol{A}$ 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，并且该运算将正常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));
```

首先，逻辑索引允许我们用条件表达式直接索引数组，这比传统的 for 循环寻找特定元素要高效得多。另外，“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()函数或 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：编程效率的秘诀

在掌握了前面章节所介绍的矩阵编程的概念、方法和技巧后，读者应该进一步理解以下几个关键的编程实践。

（1）内存预分配的智慧：在开始编码之前，务必使用 zeros()或 ones()等函数预分配矩阵内存。这样做能避免程序在运行过程中频繁改变矩阵大小，从而节省大量计算资源和时间。

（2）矩阵的单一性和明确性：确保每个矩阵都只承载一个清晰的含义，每个维度都有一个明确的意图。不要让一个矩阵承载过多的信息。通过恰当的命名和详细的注释来保持矩阵含义的清晰。

（3）矩阵规模的持续校验：在编码过程中不断检查矩阵的大小，确保在程序的每个关键点，矩阵的规模都是已知且被正确处理的，使用注释来记录这些信息，避免因规模不匹配而导致的错误。

（4）逻辑索引的巧妙运用：采用逻辑索引可以极大地简化代码。在处理矩阵时，一条逻辑索引语句经常可以轻松解决复杂的问题，这是 MATLAB 代码简洁和高效的关键。

（5）谨慎使用循环：大多数循环都可以通过矩阵运算来优化。对于刚开始学习的读者，可能会倾向于使用熟悉的循环，但应该培养在每次准备使用循环时先考虑是否能用矩阵运算来代替的习惯。

（6）多向量计算的网格化：当涉及多个向量的运算时，不要误以为只有循环才能处理。网格化技术常常能简化这类问题的代码，并减少出错的可能性。

（7）元胞数组和结构体的适时使用：它们在存储和索引上提供了便利，解决了矩阵的局限性。但它们不适合高效的批量运算，因此只在特定场合使用。在大多数情况下，矩阵仍然是首选的数据结构。

（8）稀疏矩阵的高效利用：对于包含大量零元素的矩阵，优先考虑使用 MATLAB 的稀疏矩阵。这将大幅提升这类矩阵的运算速度，并节约存储空间。

这些编程要点的掌握，将有助于读者更加精通 MATLAB 的矩阵编程，提升代码的性能和可读性，让读者的 MATLAB 之旅更加顺畅。

常见问题解答

（1）为什么本书将各种数据类型统一称作“矩阵”？

在 MATLAB 的世界里，我们选择将数据类型统一称作“矩阵”，这不仅反映了

MATLAB的核心思想，而且也简化了学习过程。统一的名称意味着统一的理解和处理方式。这个统一的称谓有助于快速深入MATLAB的核心，让读者能够更加专注于如何有效地使用这个强大的工具，而不是在各种术语之间迷失方向。记住，“名字”只是标签，“名者，实之宾也”，真正的价值在于这些数据结构的特性及其应用。

(2) 为什么本书特别强调元胞数组、结构体、表等数据结构？

虽然很多MATLAB的图书着重介绍了矩阵，但忽略了元胞数组、结构体和表这些同样重要的数据结构。实际上，这些数据结构在编程中极为有用，能够极大地简化代码并减少错误。了解和掌握这些数据结构，将使读者能够更加灵活地处理各种编程挑战，从而成为更加全面的程序员。

(3) 为什么重点强调矩阵的操作与运算？

矩阵不仅是MATLAB的核心，也是其语言的灵魂。精通矩阵的操作和运算是掌握MATLAB编程的关键，它覆盖了MATLAB中大部分的功能和应用。通过重点强调这一部分，我们旨在增强读者对矩阵结构的敏感度，帮助读者在面对编程问题时能够快速、准确地构建解决方案。

(4) 矩阵编程(向量化编程)的重要性？

绝对重要。矩阵编程或向量化编程是学习MATLAB的精华所在，它不仅展示了MATLAB的独特性，也是其高效编程能力的核心。掌握了矩阵编程，读者就能体验到MATLAB编程的极致简洁和高效，迅速实现目标，验证创意。MATLAB的一大卖点就是其能够让编程工作变得更快、更简单。

本章精华总结

在本章中，我们深入探讨了MATLAB的核心理念——矩阵的世界。我们从五个关键方面——数据类型、数据结构、矩阵操作、矩阵运算和矩阵编程，全面阐述了MATLAB独特的特性和应用。这一章不仅让读者更加明确MATLAB“以矩阵为中心”的思想，而且为大家未来学习和使用MATLAB奠定了坚实的基础。

在MATLAB中，无论是数字、文字，还是符号，都可以被统一地构造成矩阵结构，它们遵循相同的属性和操作方法。除了矩阵，元胞数组、结构体和表这三种数据结构也扮演着举足轻重的角色，为编程带来了更多灵活和优雅的解决方案。我们还详细梳理了矩阵的各类操作——从索引、逻辑处理到函数应用，甚至是那些提高效率的小技巧。同时，我们概括了矩阵的各种运算，包括算术、逻辑和关系运算。为了让这些操作和运算更易于掌握，我们提供了极致简化的应用示例，涉及广泛的应用领域，同时确保学习成本低、易于快速上手。最后，但同样重要的，我们重申了矩阵编程的重要性。作为MATLAB语言的显著特征，矩阵编程不仅是MATLAB强大功能的体现，也是其编程效率高的标志。掌握矩阵编程意味着读者真正地理解了MATLAB的精髓，为读者未来的数学和工程挑战提供了有力的工具。