第 5 章

数 据 格 式

我们在使用 Octave 进行编程时，会不可避免地产生数据。这些数据被放在 Octave 内存中或者被输出到外部文件系统中。在 Octave 内存中的数据都拥有自己的格式。

5.1 变量属性

5.1.1 由 Octave 工作空间管理的属性

我们如果使用 Octave 的 GUI 模式进行编程，则可以在 Octave 的工作空间之内查看所有存放在内存中的变量。Octave 将变量分为名称、类、维度、值、属性。下面是手动获取变量的属性的方法：

(1) 调用 whos()函数可以获取变量的名称。

(2) 调用 class()函数可以获取变量的类型。

(3) 调用 size()函数可以获取变量的维度。

(4) 调用变量名可以获取变量的值。

(5) 通过查询方式可以筛选出变量的属性。

示例查询方式如下：

```
>> b = whos_line_format(" % a:4;");
>> whos
```

其中的%a 是对于属性值的格式化字符串标志。

5.1.2 数字类型数据的输入方式

Octave 支持多种数字类型数据的输入方式。除直接输入数字外，Octave 还允许我们进行其他风格的数字输入，代码如下：

```
>> a = 0x23
a =  35
```

这里的 0x23 是十六进制的输入风格。Octave 支持的输入风格如表 5-1 所示。

表 5-1　Octave 支持的输入风格

输入方式	示　例
十进制	1
十六进制	0x1a
	0x1_a
	0x1A
	0x1_A
	0X1a
	0X1_a
	0X1A
	0X1_A
二进制	0b11
	0b1_1
	0B11
	0B1_1
科学记数法	1.1e10
	1.1E10

5.2　数据精度

在 Octave 中，数据精度完全由数据类型决定。在内存中，数据的存储空间使用位计数，下面给出以位为单位的数据精度。Octave 的基本数据类型与数据精度的对应关系如表 5-2 所示。

表 5-2　Octave 的基本数据类型与数据精度的对应关系

基本数据类型	数 据 精 度
uint8	8 位无符号整型
int8	8 位有符号整型
uint16	16 位无符号整型
int16	16 位有符号整型
uint32	32 位无符号整型
int32	32 位有符号整型
uint64	64 位无符号整型
int64	64 位有符号整型
single	32 位浮点型
double	64 位浮点型
char	8 位字符型
complex	128 位复数型

5.2.1　预置的最大值和最小值

Octave 根据数据类型预置了最大值和最小值。这些数值的调用方法如下：

```
>> intmax
ans = 2147483647
>> intmin
ans = -2147483648
>> realmax
ans = 1.7977e+308
>> realmin
ans = 2.2251e-308
```

5.2.2　预置的无穷小量

在计算机的浮点运算当中，很容易在应该计算出 0 的场合出现极小偏差，代码如下：

```
>> a = 0.1 + 0.2;
>> sprintf('%.17f',(a-0.3))
ans = 0.00000000000000006
```

为避免因为浮点数运算的误差造成的计算结果错误，Octave 预置了无穷小量 eps，当一个运算结果小于 eps 时，Octave 将认为这个结果等于 0，然后将这个结果作为 0 输出。

Octave 为 eps 指定了两种数值：

- 在 double 精度时，eps 的值为 2.2204e－16。
- 在 single 精度时，eps 的值为 1.1921e－07。

这两个数值在 Octave 中的调用方式如下：

```
>> eps(double(1.0))
ans =  2.2204e-16
>> eps(single(1.0))
ans =  1.1921e-07
```

5.2.3　浮点型格式能够存储的最大整数值

flintmax()函数用于获得某个浮点型格式能够存储的最大整数值。使用这一指标也可以表示其数值精度。如果在调用 flintmax()函数时不提供参数，则下面的代码：

```
>> flintmax
```

等效于：

```
>> flintmax("double")
```

获得某个浮点型格式能够存储的最大整数值的代码如下：

```
>> flintmax("double")
ans = 9007199254740992
>> flintmax("single")
ans = 16777216
```

5.3 数据的存储空间

5.3.1 基本数据类型的存储空间

数据存储空间根据变量类型的不同而改变。我们可以使用 sizeof()函数来方便地查看一个变量占用的内存空间。在 Octave 中，内存空间使用字节作为内存空间的最小计数单位，并且，sizeof()函数的返回值也是一个正整数。事实上，sizeof()函数返回的数值就是对应变量所占据的内存空间的字节数量，代码如下：

```
>> a = char(1);
>> sizeof(a)
ans = 1
```

结果表明：一个 char 类型变量的存储空间为 1 字节。

5.3.2 基本变量类型的 0 值

在 Octave 中定义的 0 值指的是变量在内存中的全 0 状态，这和实际意义上的零值不是一个概念。对于基本变量类型而言，它们的 0 值分别是：

(1) char 类型的“\0”。

(2) int8 类型的 0。

(3) uint8 类型的 0。

(4) int16 类型的 0。

(5) uint16 类型的 0。

(6) int32 类型的 0。

(7) uint32 类型的 0。

(8) int64 类型的 0。

(9) uint64 类型的 0。

（10）single 类型的 0。

（11）double 类型的 0。

（12）complex 类型的 0＋0i。

5.3.3　单引号和双引号与字符串的关系

我们可以使用单引号或者双引号来创建字符串，而在 Octave 中，使用单引号或者双引号所创建的字符串的形式不同，代码如下：

```
>> a = 'a\nb'
a = a\nb
```

在以上程序当中，a 被赋值为 a\nb。

```
>> a = "a\nb"
a = a
b
```

在以上程序当中，a 被赋值为 a↵b（这里的↵代表换行），而其中的\n 字符没有显示。这是因为输出结果在\n 字符的位置上自动换行了。事实上，用双引号表示的\n 字符会被转义成换行符，在输出的结果中也确实换行显示了。

由以上两个示例可得出，用单引号表示的字符串不会被转义，用双引号表示的字符串却会被转义。对于某些难以打出的字符而言，只要这些字符支持转义，我们就可以通过转义字符的方式，方便地在键盘上进行输入。

5.3.4　转义字符

1. 反斜杠转义

反斜杠转义支持的转义字符的详细列表如表 5-3 所示。

表 5-3　反斜杠转义支持的转义字符

转义前字符	转义后字符
\\	\
\"	"
\'	'
\0	ASCII 0
\a	ASCII 7 蜂鸣器响
\b	ASCII 8 退格
\f	ASCII 12 换页
\n	ASCII 10 换行
\r	ASCII 13 回车

续表

转义前字符	转义后字符
\t	ASCII 9 制表符
\v	ASCII 11 垂直制表符
\三位八进制数字	对应的 ASCII 字符
\x 两位十六进制数字	对应的 ASCII 字符

2. 简便单引号转义

看到这里，我们不难想到：由于被单引号括起来的字符串会默认进行转义，那么能不能在需要单引号字符的位置上，无须转义步骤就可以正常赋值呢？答案是可以的。在一个用单引号括起来的字符串中，可以连写两个单引号来使 Octave 将两个单引号解释为一个单引号，并且将这一个单引号理解为字符串中的单引号，示例代码如下：

```
>> a = 'A Single Quote: '''
a = A Single Quote: '
```

上面的代码使用了单引号作为字符串的括号，并且使用了两个单引号连写来表示句中的单引号。

我们不难看出，一个字符串中的两个字符可以被识别为一个字符，所以有理由相信在 Octave 中的字符和字符串之间存在某种联系。下面继续探究在 Octave 内部的字符和字符串之间的关系。

5.4 字符串

5.4.1 字符和字符串的关系

首先看一下下面的示例程序：

```
>> a = 'A Single Quote: ''';
>> class(a)
ans = char
```

Octave 将一个字符串变量的类型显示为 char，也就是字符。

那么，既然字符和字符串的表示方法都是用引号表示，为什么不合并说明呢？这是因为在 Octave 内部字符串类型的变量是用字符类型的变量拼合而成的，为了方便编写程序，才统一使用引号进行表示。

5.4.2 字符串的索引和切片

在调用 Octave 内部的字符串类型的变量时，有两种调用形式：一种是直接使用变量名

调用整个字符串，另外一种是使用冒号切片从而获得字符串的子串，代码如下：

```
>> a = 'A Single Quote: ''';
>> a
a = A Single Quote: '
>> a(2:9)
ans = Single
```

5.4.3　字符串拼接

我们可以使用逗号或者空格分割，再加上方括号来方便地进行字符串拼接，代码如下：

```
>> a = "Octave";
>> b = " is ";
>> c = "good.";
>> d = [a,b,c]
d = Octave is good.
>> d = [a b c]
d = Octave is good.
```

此外，我们还可以调用字符串拼接函数完成字符串拼接的操作。

1. strvcat()函数

strvcat()函数用于从字符串、数组和/或元胞当中拼接并创建一个新的字符串数组。调用 strvcat()函数拼接而成的数组采用竖直方式排列，代码如下：

```
>> strvcat('12',[51 52 53],{54 55},[56;57])
ans =

12
345
6
7
8
9
```

2. strcat()函数

strcat()函数用于从字符串、数组和/或元胞当中拼接并创建一个新的字符串数组。调用 strcat()函数拼接而成的数组采用水平方式排列，代码如下：

```
>> strcat('1;1',['2;2'],{'3;3'},['4;4'])
ans =
{
  [1,1] = 1;12;23;34;4
}
```

此外，strcat()函数也支持隐式的 ASCII 码转换。如果传入了数字类型参数，则 strcat()函数会将对应的数字隐式转换为字符类型，代码如下：

```
>> strcat('1',[51],{54},[56])
warning: implicit conversion from numeric to char
warning: called from
    strcat at line 117 column 8
ans =
{
  [1,1] = 1368
}
```

3. cstrcat()函数

cstrcat()函数用于从字符串数组当中拼接并创建一个新的字符串数组。调用 cstrcat()函数拼接而成的数组采用水平方式排列，并且保留参数中的所有空格，代码如下：

```
>> cstrcat('1 2 ','3 4 5')
ans = 1 2 3 4 5
```

4. strjoin()函数

strjoin()函数用于从字符串元胞当中拼接并创建一个新的字符串数组。调用 strjoin()函数拼接时，将每个元胞分量之间加入一个分隔符，最终返回一个字符串，代码如下：

```
>> strjoin({'a';'b';'c';'d'},'&')
ans = a&b&c&d
>> strjoin({'a';'b';'c';'d'},'&#!')
ans = a&#!b&#!c&#!d
```

5.4.4 创建字符串数组

虽然 Octave 的字符串拼接很方便，但是如果想要新建一个字符串数组，就不能使用逗号或者空格分割的方式，而需要使用分号进行处理，代码如下：

```
>> a = "Octave";
>> b = " is ";
>> c = "good.";
>> d = [a;b;c]
d =

Octave
is
good.
```

运行这个例子，会生成一个含有 3 个字符串的数组。

注意：在创建字符串数组时，必须使用分号分割每两个字符串，否则，使用逗号分隔的两个字符串和使用空格分割的两个字符串会拼合成一个字符串。

5.4.5　字符串数组自动扩充

在 Octave 中，如果使用一个较短的字符串生成一个较长的字符数组，则 Octave 会自动进行优化，将字符串索引之外的字符数组空间之内统一放置空格字符，因此，使用一个较短的字符串生成一个较长的字符数组时，不必考虑二者的长度关系，而可以大胆地增加字符数组的长度，最后的数组内的内容不受影响，代码如下：

```
>> a = 'abc';
>> size(a)
ans =

   1   3

>> b = [a;'abcd'];
>> size(b)
ans =

   2   4
```

在上面的代码中，尽管其第一个元素的列数为 3，但是新生成的数组的列数自动被扩展为 4，多出的字符同时被填充为\0，这就体现了 Octave 的字符串数组自动扩充的特性。

Octave 在传递参数时也利用了这个特性。在调用函数时，如果要传入多个参数，则可以使用矩阵方式进行传参。如果多个参数的长度不同，则 Octave 统一将所有参数的长度设定为最长的那个参数的长度，再进行进一步处理。这就解决了多个参数之间即便长度不一致也能正常传参的问题。

此外，可以调用 string_fill_char()函数来手动指定自动填充的字符。调用 string_fill_char()函数时，如果不传入参数，则函数将返回当前的自动填充字符(该字符默认为空字符\0)。调用 string_fill_char()函数时，如果传入一个参数，则这个参数代表被替换的自动填充的字符，代码如下：

```
>> string_fill_char("_")
>> ["apple";"pear"]
ans =

apple
pear_
```

在指定了新的自动填充的字符之后，直到重启 Octave 或指定其他的自动填充的字符之前，自动填充的字符都会是本次指定的字符。

5.4.6 字符串截取

strtrunc()函数用于截取字符串。调用 strtrunc()函数时，我们需要传入两个参数，此时第一个参数代表被截取的字符串，第二个参数代表截取的长度，代码如下：

```
>> strtrunc('abcd',3)
ans = abc
```

此外，还可以传入字符串元胞，此时 strtrunc()函数将对元胞中的每个字符串分量进行截取，代码如下：

```
>> strtrunc({'abcd','efghij'},3)
ans =
{
  [1,1] = abc
  [1,2] = efg
}
```

5.4.7 字符串分割

1. strtok()函数

strtok()函数用于分割字符串。调用 strtok()函数时，至少需要传入两个参数，此时第 1 个参数代表被分割的字符串，第 2 个参数代表分隔符，代码如下：

```
>> strtok('abcdefg','abcd')
ans = efg
```

strtok()函数在分割字符串时，只在第 1 个分隔符处进行分隔。如果字符串中含有多个分隔符，则 strtok()函数也只进行一次分割操作，代码如下：

```
>> strtok('abababa','b')
ans = a
```

此外，可以指定两个返回参数，此时这两个参数代表分隔符前的字符串和分隔符后的字符串。在指定两个返回参数时，分隔符前的字符串不包含分隔符，分隔符后的字符串包含分割符，代码如下：

```
>> [a,b] = strtok('abababa','b')
a = a
b = bababa
```

还可以指定多个分隔符。如果第 2 个参数不能完全匹配被分割的字符串，则第 2 个参数代表多个分隔符，代码如下：

```
>> strtok('abcdefg','abcd')
ans = efg
>> strtok('abcdefg','abce')
ans = d
>> strtok('abcdefg','xbcd')
ans = a
```

2. strsplit()函数

strsplit()函数用于分割字符串。调用 strsplit()函数时，至少需要传入一个参数，此时这个参数代表被分割的字符串。strsplit()函数在分割字符串时使用空白符号进行字符串之间的识别，返回一个字符串元胞，代码如下：

```
>> strsplit('a b cd')
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = cd
}
```

此外，可以传入第 2 个参数，此时这个参数代表分隔符。如果指定了分隔符，则 strsplit()函数将不再使用默认的空白符号进行分割，代码如下：

```
>> strsplit('a b cd','c')
ans =
{
  [1,1] = a b
  [1,2] = d
}
```

还可以指定不同的分隔符。如果指定不同的分隔符，则需要传入字符串元胞，元胞中的每个元素都代表一个分隔符，代码如下：

```
>> strsplit('a b cd',{'c',' '})
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}
```

可以指定 collapsedelimiters 选项，以此来决定分隔符是否出现在返回结果当中：

- 如果 collapsedelimiters 选项为 true，则分隔符将不会出现在返回结果当中；
- 如果 collapsedelimiters 选项为 false，则分隔符将出现在返回结果当中。

代码如下：

```
>> strsplit('a b cd',{'c',' '},'collapsedelimiters',true)
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}

>> strsplit('a b cd',{'c',' '},'collapsedelimiters',false)
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] =
  [1,4] = d
}
```

可以指定 delimitertype 选项，以此来决定分隔符的模式：

- 如果 delimitertype 选项为 simple，则分隔符将“所见即所得”；
- 如果 delimitertype 选项为 regularexpression，则分隔符将视为正则表达式。

代码如下：

```
>> strsplit('a b cd',{'c',' '},'delimitertype','simple')
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}

>> strsplit('a b cd',{'[c]','[ ]'},'delimitertype','regularexpression')
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}

>> strsplit('a b cd','[c ]','delimitertype','regularexpression')
```

```
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}
```

3. ostrsplit()函数

ostrsplit()函数用于分割字符串。调用 ostrsplit()函数时,我们至少需要传入一个参数,此时这个参数代表被分割的字符串。ostrsplit()函数在分割字符串时使用空白符号进行字符串之间的识别,返回一个字符串元胞,代码如下:

```
>> ostrsplit('a b cd','c')
ans =
{
  [1,1] = a b
  [1,2] = d
}
```

此外,还可以指定不同的分隔符。如果指定不同的分隔符,则需要将所有分隔符组成一个字符串,代码如下:

```
>> ostrsplit('a b cd','c ')
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] =
  [1,4] = d
}
```

可以发现,ostrsplit()函数分割字符串的逻辑是将字符串先进行分割,然后将分割后的部分存放在元胞中,再将分隔符替换为空字符,最后返回结果。

可以指定多个字符串,此时需要传入一个字符串数组,代码如下:

```
>> ostrsplit(['a b cd';'e'],'c ba')
ans =
{
  [1,1] =
  [1,2] =
  [1,3] =
  [1,4] =
```

```
  [1,5] =
  [1,6] = d
  [1,7] = e
  [1,8] =
  [1,9] =
  [1,10] =
  [1,11] =
  [1,12] =
}
```

可以追加传入第 3 个参数，此时这个参数代表是否去除返回结果中的空字符分量：

- 如果第 3 个参数为 true，则返回结果中的空字符分量将被去除；
- 如果第 3 个参数为 false，则返回结果中的空字符分量将不会被去除。

追加传入第 3 个参数的代码如下：

```
>> ostrsplit('a b cd','c ',true)
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] = d
}

>> ostrsplit('a b cd','c ',false)
ans =
{
  [1,1] = a
  [1,2] = b
  [1,3] =
  [1,4] = d
}
```

5.4.8 字符串替换

strrep()函数用于替换字符串中的某一部分。调用 strrep()函数时，我们至少需要传入 3 个参数，此时第 1 个参数代表被替换的字符串，第 2 个参数代表匹配格式，第 3 个参数代表替换字符，然后 strrep()函数找到匹配格式中的字符并将其替换为替换字符，最后返回新的字符串，代码如下：

```
>> strrep('aabcd','ab','cd')
ans = acdcd
```

此外，可以传入字符串元胞，此时 strrep()函数将对元胞中的每个字符串分量进行替

换，代码如下：

```
>> strrep({'aabcd','babcd'},'ab','cd')
ans =
{
  [1,1] = acdcd
  [1,2] = bcdcd
}
```

5.4.9　字符串清除

erase()函数用于清除字符串中的某一部分。调用 erase()函数时，需要传入两个参数，此时第 1 个参数代表被清除的字符串，第 2 个参数代表匹配格式，然后 erase()函数找到匹配格式中的字符并将其清除，最后返回新的字符串，代码如下：

```
>> erase('aabcd','ab')
ans = acd
```

此外，可以传入字符串元胞，此时 erase()函数将对元胞中的每个字符串分量进行清除，代码如下：

```
>> erase({'aabcd','babcd'},'ab')
ans =
{
  [1,1] = acd
  [1,2] = bcd
}
```

5.5　数组

5.5.1　数组元素的索引

Octave 使用圆括号对数组进行索引。在索引时，可以手动指定元素在每个维度下的位置进行数组元素的索引。此时，不同维度之间的位置必须以逗号隔开，代码如下：

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

   1  2  3
   4  5  6
>> a(1,2)
ans = 2
```

此外，可以直接指定一个索引数字进行索引。此时，这个数字代表数组元素在数组中存储的顺序，代码如下：

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

   1  2  3
   4  5  6

>> a(1)
ans = 1
>> a(2)
ans = 4
>> a(3)
ans = 2
>> a(4)
ans = 5
>> a(5)
ans = 3
>> a(6)
ans = 6
```

此时可以发现，数组元素“1、2、3、4、5、6”在数组中存储的顺序并不是按照“1、2、3、4、5、6”的顺序，而是按照“1、3、5、2、4、6”的顺序。这是因为 Octave 中的数组元素在数组中存储的顺序是按照维度从低到高进行叠加计算的。以一个 2×2×2 的矩阵 a 为例：

(1) 数组中存储的第 1 个元素 a(1)是 a(1,1,1)。

(2) 数组中存储的第 2 个元素 a(2)是 a(2,1,1)。

(3) 数组中存储的第 3 个元素 a(3)是 a(1,2,1)。

(4) 数组中存储的第 4 个元素 a(4)是 a(2,2,1)。

(5) 数组中存储的第 5 个元素 a(5)是 a(1,1,2)。

(6) 数组中存储的第 6 个元素 a(6)是 a(2,1,2)。

(7) 数组中存储的第 7 个元素 a(7)是 a(1,2,2)。

(8) 数组中存储的第 8 个元素 a(8)是 a(2,2,2)。

然而，有一种方法可以使得数组元素在数组中存储的顺序和索引数字在数学意义上保持一致，即在定义数组后进行转置运算，代码如下：

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

   1  4
   2  5
   3  6
```

```
>> a(1)
ans = 1
>> a(2)
ans = 2
>> a(3)
ans = 3
>> a(4)
ans = 4
>> a(5)
ans = 5
>> a(6)
ans = 6
```

注意：由于高维矩阵不支持二维矩阵的转置运算，所以这种方法只能用于二维矩阵当中。如果想要使用转置运算解决涉及高维矩阵的问题，则需自行实现适用于高维矩阵的转置函数。

5.5.2　数组的切片

我们可以使用冒号进行切片操作。以一个数组为例：

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

   1   2
   3   4
```

对上面的数组进行切片时，仍然使用圆括号调用的方式传入数组的索引数字，并且将切片的下界分量和切片的上界分量使用冒号隔开，代码如下：

```
>> a(2:4)
ans =

   3   2   4
```

上面的代码指定数组切片为 a 数组的第 2 个分量到第 4 个分量之间的范围。

但是，原则上切片的下界分量需要小于切片的上界分量。如果切片的下界分量大于切片的上界分量，则切片将返回空数组，代码如下：

```
>> a(4:2)
ans = [](1x0)
```

如果切片的下界分量等于切片的上界分量，则切片将返回一个单元素的数组，等效于直接按照索引数字进行索引，代码如下：

```
>> a(2:2)
ans = 3
```

此外，还可以将矩阵的维度大小指定为冒号来索引这一维度上的所有内容，并且指定需要切片的维度进行矩阵切片，代码如下：

```
>> a(1,:)
ans =

   1  2
```

上面的代码指定数组切片为 a 矩阵的第一维度是 1 的内容（第 1 行）和第二维度的所有内容（所有列）之间的范围。接下来，看一下下面的代码：

```
>> a(:,1)
ans =

   1
   3
```

上面的代码指定数组切片为 a 矩阵的第二维度是 1 的内容（第 1 列）和第一维度的所有内容（所有行）之间的范围。

如果将维度的值均指定为冒号，则切片将等于源数组，代码如下：

```
>> a(:,:)
ans =

   1  2
   3  4
```

5.5.3 创建高维数组

Octave 使用逗号或空格区分一行中的两个元素，也可以使用分号区分不同行间的两个元素，但是，使用分隔符只能至多创建二维的数组。如果要创建更高维度的数组，则可以采用高维度索引的方式，配合赋值运算进行高维数组创建，代码如下：

```
>> clear a
>> a(5,5,5,5,5) = 0;
```

上面的代码就可以创建一个大小为 5×5×5×5×5 的全 0 矩阵。

5.5.4　拼接二维数组

Octave 拼接二维数组也可以直接使用逗号、空格及分号作为分隔符，并且使用方括号进行包裹，代码如下：

```
>> [[1,2];[3] 4]
ans =

   1  2
   3  4
```

此外，也可以直接使用函数拼接二维数组。

1. horzcat()函数

horzcat()函数被用来以水平方向拼接二维数组。在使用 horzcat()函数进行数组拼接时，传入的每个参数都被视为进行拼接的源数组，而且每个源数组的尺寸必须相同，代码如下：

```
>> horzcat(1,2,3,4)
ans =

   1  2  3  4
```

2. vertcat()函数

vertcat()函数被用来以水平方向拼接二维数组。在使用 vertcat ()函数进行数组拼接时，传入的每个参数都被视为进行拼接的源数组，而且每个源数组的尺寸必须相同，代码如下：

```
>> vertcat(1,2,3,4)
ans =

   1
   2
   3
   4
```

5.5.5　拼接高维数组

我们可以调用 cat()函数将若干个相同尺寸的二维数组拼接为高维数组。

注意：cat()函数不能将三维数组或更高维度的数组直接进行拼接。如果想要拼接高维数组，则需要对高维数组的每两个相邻维度进行循环索引并拼接。

调用 cat()函数时需要至少传入 3 个参数，第 1 个参数为拼接后的矩阵维度，第 2 个和第 3 个参数为参与拼接的矩阵。在数组拼接完毕后，我们就得到了一个高维度的数组。

调用 cat()函数时，传入的维度不同，最终拼接成的矩阵也不同。拼接时的规则如下所示。

（1）当传入的维度为 2 时，将矩阵以水平方向进行拼接，代码如下：

```
>> a = cat(2,ones(2),eye(2),zeros(2))
a =

   1  1  1  0  0  0
   1  1  0  1  0  0
```

（2）当传入的维度为 3 时，将矩阵在第 3 个维度上进行拼接，代码如下：

```
>> a = cat(3,ones(2),eye(2),zeros(2))
a =

ans(:,:,1) =

   1  1
   1  1

ans(:,:,2) =

   1  0
   0  1

ans(:,:,3) =

   0  0
   0  0
```

（3）当传入的维度大于或等于 4 时，将矩阵的第 3 个维度之后的索引定义为 1，然后在最后一个维度上进行拼接，代码如下：

```
>> a = cat(4,ones(2),eye(2),zeros(2))
a =

ans(:,:,1,1) =

   1  1
   1  1

ans(:,:,1,2) =
```

```
    1  0
    0  1

ans(:,:,1,3) =

    0  0
    0  0

>> a = cat(5,ones(2),eye(2),zeros(2))
a =

ans(:,:,1,1,1) =

    1  1
    1  1

ans(:,:,1,1,2) =

    1  0
    0  1

ans(:,:,1,1,3) =

    0  0
    0  0
```

5.5.6　重新排列矩阵

1. permute()函数

permute()函数用于获取一个矩阵的广义转置。对于一个多维矩阵而言，广义转置运算需要指定一个置换向量，然后将每个维度的分量映射到置换向量相应的维度上，代码如下：

```
>> a = zeros(2,3,4,5);
>> b = permute(a,[3,4,2,1]);
>> size(b)
ans =

    4  5  3  2
```

在上面的代码中，广义转置后的矩阵将第一维度的分量映射到了第三维度，将第二维度的分量映射到了第四维度，将第三维度的分量映射到了第二维度，并且将第四维度的分量映射到了第一维度。

2. ipermute()函数

ipermute()函数用于获取一个矩阵的广义转置的逆矩阵,代码如下:

```
>> a = zeros(2,3,4,5);
>> b = ipermute(a,[3;4;2;1]);
>> size(b)
ans =

   5  4  2  3
```

5.5.7 循环更改矩阵

1. circshift()函数

circshift()函数用于循环更改矩阵中的行(和列)。调用 circshift()函数时需要传入源矩阵和需要更改的行数和列数,而且需要更改的行数或列数必须组成一个向量。除指定源矩阵外:

(1) 如果向量中只有一个元素,则这个数字代表矩阵需要更改的行数。

(2) 如果向量中只有两个元素,则这两个数字代表矩阵需要更改的行数和列数。

(3) 如果向量中有更多元素,则前两个数字代表矩阵需要更改的行数和列数,第 3 个数字代表需要更改的维度,代码如下:

```
>> circshift([1 2 3;4 5 6;7 8 9],1)
ans =

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

并且,行数和列数的正负也影响改变后的结果:

(1) 如果行数为正,则矩阵将循环向下更改相应的行数。

(2) 如果行数为负,则矩阵将循环向上更改相应的行数。

(3) 如果行数为正,则矩阵将循环向右更改相应的行数。

(4) 如果行数为负,则矩阵将循环向左更改相应的行数。

下面给出一个同时更改 1 行和-1 列的代码:

```
>> circshift([1 2 3;4 5 6;7 8 9],[1, -1])
ans =

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

此外，对于三维及三维以上的矩阵，可以增加元素的个数，来指定要更改的维度，代码如下：

```
> a = [1 2;3 4];
>> b = cat(4,a,a)
b =

ans(:,:,1,1) =

   1  2
   3  4

ans(:,:,1,2) =

   1  2
   3  4

>> circshift(b,[1, - 1,4])
ans =

ans(:,:,1,1) =

   4  3
   2  1

ans(:,:,1,2) =

   4  3
   2  1
```

在上面的代码中，矩阵 b 的第 4 个维度均被更改了 1 行和－1 列。

2. shift()函数

shift()函数也可以用于循环更改矩阵中的行(和列)。shift()函数是一个复用函数，其复用规则如下：

(1) 如果传入的第 1 个参数是一个向量，则按照第 2 个参数指定的行数或列数循环更改对应的行或列。

(2) 如果传入的第 1 个参数是一个矩阵，则按照第 2 个参数循环更改每一行。

(3) 如果追加了第 3 个参数，则第 3 个参数代表按维度更改矩阵。

shift()函数对于参数的规定如下：

(1) 如果第 2 个参数是负数，则 shift()函数将循环向左修改矩阵的元素。

(2) 如果第 2 个参数是零，则 shift()函数将输出一个相同的矩阵。

(3) 如果第 2 个参数是正数，则 shift()函数将循环向右修改矩阵的元素。

(4) 如果追加了第 3 个参数,而且第 3 个参数为 1,则将按行循环更改矩阵。

(5) 如果追加了第 3 个参数,而且第 3 个参数为 2,则将按列循环更改矩阵。

(6) 如果追加了第 3 个参数,而且第 3 个参数为 3、4、5、…,那么将按那个维度更改矩阵。

循环更改矩阵中的行(和列)的代码如下:

```
>> shift([1 2 3 4 5],2)
ans =

   4  5  1  2  3
>> shift([1 2 3 4 5]',2)
ans =

   4
   5
   1
   2
   3
>> shift([1 2 3 4 5;2 3 4 5 6],1)
ans =

   2  3  4  5  6
   1  2  3  4  5

>> shift([1 2 3 4 5;2 3 4 5 6],1,2)
ans =

   5  1  2  3  4
   6  2  3  4  5

>> shift([1 2 3 4 5;2 3 4 5 6],1,1)
ans =

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

5.5.8 改变矩阵维度

shiftdim()函数用于截取矩阵在某个维度之下的低维度矩阵,或者增加矩阵的维度并生成一个高维度矩阵。

调用 shiftdim()函数时至少要传入一个参数。如果传入了一个参数,则 shiftdim()函数会输出一个相同的矩阵,相当于什么也不做。如果传入了两个参数,则第 1 个参数代表要进行修改的源矩阵,第 2 个参数是要进行修改的维度。

shiftdim()函数对于第 2 个参数的规定如下所示。

(1) 如果第 2 个参数是负数,则 shiftdim()函数将增加对应数量的维度。

(2) 如果第 2 个参数是零,则 shiftdim()函数将输出一个相同的矩阵。

(3) 如果第 2 个参数是正数,则 shiftdim()函数将循环向左修改矩阵的元素,代码如下:

```
>> size(shiftdim(a, -1))
ans =

   1  2  3  4  5

>> size(shiftdim(a, -2))
ans =

   1  1  2  3  4  5

>> size(shiftdim(a, 0))
ans =

   2  3  4  5

>> size(shiftdim(a, 1))
ans =

   3  4  5  2

>> size(shiftdim(a, 2))
ans =

   4  5  2  3
```

5.5.9　矩阵排序

1. sortrows()函数

sortrows()函数用于将一个矩阵按行排序。排列的规则和某几个列有关。由于使用 sortrows()函数排序不会改变行内元素的组合顺序,所以这种方式非常适合对矩阵进行抽象排序。调用 sortrows()函数需要传入至少一个参数,此时这个参数代表需要排序的源矩阵,并且排序的结果参考第一列中的元素,对矩阵中的每行按照从上到下递增的顺序进行排列,代码如下:

```
>> a = magic(4)
a =

   16   2   3  13
```

```
    5  11  10   8
    9   7   6  12
    4  14  15   1

>> sortrows(a)
ans =

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

此外，还可以额外指定第 2 个参数，此时第 2 个参数代表排序参考的列数。sortrows()函数排序参考的列数的规则如下：

（1）第 2 个参数中的每个分量都代表一个排序时的参考量。

（2）如果第 2 个参数中的一个分量是正数，则将按照此列元素从上到下递增的顺序对所有的行进行排序。

（3）如果第 2 个参数中的一个分量是负数，则将按照此列元素从上到下递减的顺序对所有的行进行排序，代码如下：

```
>> a = magic(4)
a =

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

>> sortrows(a,[2, - 3,4])
ans =

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

2. sort()函数

sort()函数也可以用于对矩阵进行排序。调用 sort()函数时，至少要传入一个参数，这个参数代表要进行修改的源矩阵，代码如下：

```
>> sort([5 4 3 2 1])
ans =

   1  2  3  4  5
```

此外，还可以追加第 2 个参数，用于指定排序的维度，代码如下：

```
>> sort([5 4 3 2 1],1)
ans =

   5   4   3   2   1

>> sort([5 4 3 2 1],2)
ans =

   1   2   3   4   5
```

可以看出，一个二维矩阵按行排序和按列排序的结果是不同的。

sort()函数默认将矩阵以升序排列。如果指定了第 2 个参数，则第 2 个参数也可以被用来指定排序的规则。

- 传入 ascend 参数代表升序排列；
- 传入 descend 参数代表降序排列，代码如下：

```
>> sort([1 2 3 4 5;2 3 4 5 6],'ascend')
ans =

   1   2   3   4   5
   2   3   4   5   6

>> sort([1 2 3 4 5;2 3 4 5 6],'descend')
ans =

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

可以看出，一个二维矩阵在指定 ascend 参数和 descend 参数时，将得到完全相反的结果。

还可以追加第 3 个参数，同时指定排序的维度和排序的规则，代码如下：

```
>> sort([1 2 3 4 5;2 3 4 5 6],2,'ascend')
ans =

   1   2   3   4   5
   2   3   4   5   6

>> sort([1 2 3 4 5;2 3 4 5 6],2,'descend')
ans =

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

对于复数域而言，复数矩阵排序先按照极径排序。当极径相等时，再按照极角进行排序，代码如下：

```
>> sort([1 + i 1 - i 2 - i 1 - 2i])
ans =

   1 - 1i  1 + 1i  1 - 2i  2 - 1i
```

如果矩阵当中含有 NA 或者 NaN 元素，则这些数据不进行排序，而是按照矩阵的先后顺序直接放在排序后的数组的一端，代码如下：

```
>> sort([nan inf; - inf NA;1 2],'ascend')
ans =

   - Inf    2
     1  Inf
   NaN   NA

>> sort([nan inf; - inf NA;1 2],'descend')
ans =

   NaN   NA
     1  Inf
   - Inf    2
```

事实上这个结果在 Octave 看来是对的。根据“数组元素的索引”规则，由于 sort()函数对数组元素进行了遍历，遍历之后又改变了数组的形状，所以数组在变形之后，存放的结果就和数学意义上的结果不一致了，最终结果往往不尽人意，因此建议在使用 sort()函数对数组进行排序前，先将数组的形状加以改变，将数组变为一个向量，然后调用 sort()函数。这样做之后，sort()函数排序的结果也是一个向量，不涉及维度增加的问题，于是就避开了数组元素的索引规则。

5.5.10 改变矩阵形状

1. reshape()函数

reshape()函数可以将一个矩阵中的元素按照给定的维度重新排列组合为一个新的矩阵。调用 reshape()函数时，需要传入源矩阵和改变形状后的维度，代码如下：

```
>> reshape(zeros(1,8),[2,2,2])
ans =

ans(:,:,1) =

   0  0
```

```
    0  0

ans(:,:,2) =

    0  0
    0  0
```

上面的代码将一个 1×8 的全 0 矩阵重新排列为尺寸为 2×2×2 的全 0 矩阵。

其中，在传入改变形状后的维度时，也可以分成多个参数传入，代码如下：

```
>> reshape(zeros(1,8),2,2,2);
```

2. resize()函数

resize()函数也可以用于改变矩阵的形状。调用 resize()函数时，需要传入至少两个参数，此时第 1 个参数代表将要改变形状的源矩阵，而第 2 个参数代表输出矩阵的维度大小。如果第 2 个参数是一个数字，则最终将输出一个方阵。对于维度大小不匹配的场合，resize()函数的处理规则如下所示。

- ❑ 如果源矩阵的尺寸大于指定的尺寸，则 resize()函数将剪切多余的部分；
- ❑ 如果源矩阵的尺寸小于指定的尺寸，则 resize()函数将不足的部分填入 0 值，代码如下：

```
>> resize([1 2 3 4],2)
ans =

    1  2
    0  0

>> resize([1 2 3 4],3)
ans =

    1  2  3
    0  0  0
    0  0  0
```

此外，还可以追加更多的参数，此时除源矩阵参数之外的所有参数将代表对应矩阵维度的大小，代码如下：

```
>> resize([1 2 3 4],2,2)
ans =

    1  2
    0  0
```

```
>> resize([1 2 3 4],2,3)
ans =

   1  2  3
   0  0  0

>> resize([1 2 3 4],3,3)
ans =

   1  2  3
   0  0  0
   0  0  0

>> resize([1 2 3 4],3,3,3)
ans =

ans(:,:,1) =

   1  2  3
   0  0  0
   0  0  0

ans(:,:,2) =

   0  0  0
   0  0  0
   0  0  0

ans(:,:,3) =

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

还可以将所有维度值作为一个矩阵传入函数，将输入参数的数量统一为两个，而最终结果等效于将维度大小参数分别传入函数，代码如下：

```
>> resize([1 2 3 4],2,2)
ans =

   1  2
   0  0

>> resize([1 2 3 4],[2,2])
```

```
ans =

   1   2
   0   0
```

3. vec()函数

vec()函数用于将矩阵的某一维度的值作为一个向量输出。调用 vec()函数时至少需要输入一个参数，此时这个参数代表源矩阵，代码如下：

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

   1
   3
   2
   4
```

此外，可以在调用 vec()函数时额外指定第 2 个参数，此时这个参数被用来指定要生成的向量的维度。事实上，如果不指定维度参数，则生成的向量按照第一维度（也就是按列）展开，代码如下：

```
>> vec([1 2;3 4],1)
ans =

   1
   3
   2
   4

>> vec([1 2;3 4],2)
ans =

   1   3   2   4

>> vec([1 2;3 4],3)
ans =

ans(:,:,1) = 1
ans(:,:,2) = 3
ans(:,:,3) = 2
ans(:,:,4) = 4
```

4. vech()函数

vech()函数用于获得一个方阵的下三角矩阵，并且将这个矩阵以列向量的形式输出。

调用 vech()函数时需要传入一个方阵,代码如下:

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

   1
   3
   4

>> vech([1 2 3;4 5 6;7 8 9])
ans =

   1
   4
   7
   5
   8
   9
```

5.5.11 截取或补齐矩阵元素

在进行运算时,有时需要将一个矩阵补零(或者补某种数字)以便达到指定的尺寸才能进行运算。Octave 提供了两种函数进行此类操作。

1. prepad()函数

prepad()函数被用来在矩阵左侧截取或补齐一个数组以便达到某个长度。调用 prepad()函数时至少需要传入两个参数,第 1 个参数代表源矩阵,第 2 个参数代表截取或补齐的分量个数。如果参数代表的分量个数大于源矩阵的分量个数,则 prepad()函数默认补零,代码如下:

```
>> a = [1 2 3];
>> prepad(a,2)
ans =

   2  3

>> prepad(a,4)
ans =

   0  1  2  3
```

此外,可以指定第 3 个参数,这个参数代表补齐矩阵所使用的分量,代码如下:

```
>> prepad(a,4,100)
ans =

   100    1    2    3
```

2. postpad()函数

postpad()函数被用来在矩阵右侧截取或补齐一个数组以便达到某个长度。调用postpad()函数时至少需要传入两个参数,第1个参数代表源矩阵,第2个参数代表截取或补齐的分量个数。如果参数代表的分量个数大于源矩阵的分量个数,则postpad()函数默认补零,代码如下:

```
>> a = [1 2 3];
>> postpad(a,2)
ans =

   1  2

>> postpad(a,4)
ans =

   1  2  3  0
```

此外,可以指定第3个参数,这个参数代表补齐矩阵所使用的分量,代码如下:

```
>> postpad(a,4,100)
ans =

     1    2    3  100
```

5.6　元胞

5.6.1　元胞的索引

在上述例子中,分别使用了圆括号和花括号对字符串元胞进行了索引。事实上,通过这个例子,我们也知道了元胞的两种索引方式:圆括号索引方式和花括号索引方式。其中,圆括号索引方式需要同时传入所有的下标才能返回唯一的元胞,而使用花括号进行索引则是对于元胞的独有索引方式。使用花括号进行索引相当于将元胞内的所有元素按照定义的顺序(也就是每个下标从小到大,下标按照先后顺序)进行顺序排列,最终可以返回一个单一的元素。

元胞的索引规则和数组略有不同。元胞没有维度的概念,而只有索引数字的概念,因此,元胞的使用非常灵活,可以在每个索引数字对应的元素中放入任意类型的数据,数据占

据的空间大小也可以不同。

事实上，元胞的圆括号索引方式和花括号索引方式的根本区别在于：

- 使用圆括号对元胞索引将得到另一个元胞；
- 使用花括号对元胞索引将得到另一个元胞内部的元素。

5.6.2 元胞的串级索引

由于元胞的内部可以放置任意类型的数据，所以元胞内部也自然可以放置元胞，而元胞又可以被继续索引，因此我们就可以继续对这个元胞进行索引。这种索引方式被叫作串级索引。

元胞的优势在于可以使用花括号对元胞中的元素进行层级式排布，以便定义一个更复杂的元胞，来展示元胞的花括号索引方式的强大之处，代码如下：

```
>> a = {{{1,2},3},4}
a =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

在这个元胞中很难使用圆括号对其中的某个元素进行精确索引。那么，使用花括号索引方式，对此元胞的第一个元素进行索引，得到以下结果：

```
>> a{1}
ans =
{
  [1,1] =
  {
    [1,1] = 1
    [1,2] = 2
  }

  [1,2] = 3
}
```

于是，我们发现此元胞按照花括号方式索引第 1 个元素，得到的是一个小一些的元胞。使用串级索引方式继续缩小范围，代码如下：

```
>> a{1}{1}
ans =
{
  [1,1] = 1
  [1,2] = 2
}

>> a{1}{1}{1}
ans = 1
```

在使用花括号索引方式索引 3 次后，最终索引到单一的基本数据。

此外，元胞也支持使用圆括号串级索引。在下面的元胞当中：

```
>> a = {{{1,2},3},4}
a =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

元胞的第 1 个索引数字对应的元素是一个元胞，我们使用圆括号并使用索引数字 1 进行串级索引，即可得到以下结果：

```
>> a(1)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
```

```
      [1,2] = 2
    }

    [1,2] = 3
  }

}
```

继续使用索引数字 1 和另一个索引数字 1 进行串级索引，即可得到以下结果：

```
>> a(1)(1)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }
}
```

我们发现索引结果和使用索引数字 1 进行串级索引的结果没有区别。这是因为得到的元胞是单纯的一个元胞。我们可以再使用无数个索引数字 1 进行串级索引，得到的结果都会相同。换言之，使用圆括号进行串级索引可以对元胞进行无限索引，代码如下：

```
>> a(1)(1)(1)(1)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

}
```

那么，如果要取得元胞内部的元素并进行更深层次的索引，就必须使用花括号进行串级索引，代码如下：

```
>> a(1){1}
ans =
{
  [1,1] =
  {
    [1,1] = 1
    [1,2] = 2
  }

  [1,2] = 3
}
```

上面的代码表明，Octave 也支持同时使用圆括号和花括号进行混合串级索引，而且混合索引方式也具有等效形式，代码如下：

```
>> a(2){1}
ans = 4
>> a{2}
ans = 4
```

使用花括号索引某个元素等效于先使用圆括号索引这个元素，再使用花括号索引第一个元素。

5.6.3　元胞的切片

元胞可以使用冒号运算符进行切片。以一个元胞为例，代码如下：

```
>> a = {{{1,2},3},4}
a =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

对元胞进行切片时，仍然使用圆括号调用的方式传入元胞的索引数字，并且将切片的下界分量和切片的上界分量使用冒号隔开，代码如下：

```
>> a(1:2)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

上面的代码指定元胞切片为 a 数组的第 2 个～第 4 个分量的范围。

但是，切片的下界分量需要小于切片的上界分量。如果切片的下界分量大于切片的上界分量，则切片将返回空元胞，代码如下：

```
>> a(2:1)
ans = {}(1x0)
```

如果切片的下界分量等于切片的上界分量，则切片将返回一个元胞，等效于直接按照索引数字进行索引，代码如下：

```
>> a(2:2)
ans =
{
  [1,1] = 4
}
```

此外，可以将元胞的最后一个层级大小指定为冒号来索引前一层级的所有内容，并且指定需要切片的层级进行元胞切片，代码如下：

```
>> a(1,2,:)
ans =
{
  [1,1] = 4
}
```

上面的代码指定的元胞切片等效于不增加最后一个冒号的元胞。等效形式如下：

```
>> a(1,2)
ans =
{
  [1,1] = 4
}
```

还可以使用冒号，使用追加冒号的方式指定更多的层级。换言之，使用圆括号索引的同时使用冒号可以对元胞进行无限切片，代码如下：

```
>> a(1,2,:)
ans =
{
  [1,1] = 4
}

>> a(1,2,:,:)
ans =
{
  [1,1] = 4
}

>> a(1,2,:,:,:)
ans =
{
  [1,1] = 4
}
```

可以将元胞的层级大小指定为冒号来索引这一层级的所有内容，并且指定需要切片的层级进行元胞切片，代码如下：

```
>> a(:,1)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

}
```

上面的代码指定元胞切片为 a 矩阵的第 2 层级是 1 的内容和第 1 层级的所有内容之间的范围。

如果将层级的值均指定为冒号,则切片将等于源元胞,代码如下:

```
>> a(:,:)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

还可以同时使用冒号分隔切片的下界分量和切片的上界分量,而且将元胞的层级大小指定为冒号来索引这一层级的所有内容,进行元胞切片,代码如下:

```
>> a(:,1:2)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
    }

    [1,2] = 3
  }

  [1,2] = 4
}
```

5.6.4 元胞的串级切片

因为元胞支持串级索引,所以根据切片的定义,元胞也支持串级切片。以下面的元胞

为例：

```
a =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] = 1
      [1,2] = 2
      [1,3] = 3
      [1,4] = 4
    }

    [1,2] = 5
    [1,3] = 6
    [1,4] = 7
    [1,5] = 8
  }

  [1,2] = 9
  [1,3] = 10
  [1,4] = 11
  [1,5] = 12
}
```

可以使用冒号分隔索引数字的范围。其中的一个串级切片结果如下：

```
>> a{1}{1}{2:3}
ans = 2
ans = 3
```

还可以使用冒号指定层级。其中的一个串级切片结果如下：

```
>> a{1}{1}{:,2}
ans = 2
```

此外，还可以同时使用冒号分隔索引数字的范围，并且指定层级。其中的一个串级切片结果如下：

```
>> a{1}{1}{:,2:3}
ans = 2
ans = 3
```

Octave 还提供了一种更精准的切片函数，用于元胞的切片。

5.6.5 元胞的精确切片

cellslices()函数用于获取某个索引数字范围之内存储的内容，并返回这个元胞的精确切片。调用 cellslices()函数时，需要传入 4 个参数，其中第 1 个参数代表源元胞，第 2 个参数代表索引数字的下界，第 3 个数字代表索引数字的上界，第 4 个数字代表索引的深度，代码如下：

```
>> cellslices(a,1,1,2)
ans =
{
  [1,1] =
  {
    [1,1] =
    {
      [1,1] =
      {
        [1,1] = 1
        [1,2] = 2
        [1,3] = 3
        [1,4] = 4
      }

      [1,2] = 5
      [1,3] = 6
      [1,4] = 7
      [1,5] = 8
    }

  }

}
```

如果索引数字的下界参数大于索引数字的上界参数，则调用 cellslices()函数进行索引的结果为一个空元胞，代码如下：

```
>> cellslices(a,3,1,7)
ans =
{
  [1,1] = {1x5x1x1x1x1x0 Cell Array}
}
```

5.6.6 创建字符串元胞

一个字符串数组是一种所存放的元素均为字符串的数组。它和字符数组不同之处只是

字符数组中的所有元素都是 1×1 的字符串(或者称为字符)。如果想使用具体的字符串进行索引,就不能使用字符串数组了,而应该考虑使用元胞对这些字符串进行储存,代码如下:

```
>> a = {'apple','orange';'grape','peach'};
>> a(1)
ans =
{
  [1,1] = apple
}

>> a{1}
ans = apple
```

此外,字符串元胞中的每个字符串元素占用的空间可以不同。换言之,字符串元胞没有自动扩充的特性。我们在需要一个变量存储不同长度的字符串时,不能使用数组,而应该使用元胞。

5.7　数据格式转换

5.7.1　数字类型变量转换

1. num2cell()函数

num2cell()函数用于将数字转换为元胞。这个数字可以是单个数字,也可以是数组或元胞。

调用 num2cell()函数时,至少需要指定一个参数,这个参数代表需要转换的源数字,代码如下:

```
>> num2cell(1)
ans =
{
  [1,1] = 1
}
```

此外,可以追加第 2 个参数,用于指定转换的深度,代码如下:

```
>> a{2,3,4,5} = 0;
>> num2cell(a,4)
ans = {2x3x4 Cell Array}
>> num2cell(a,2)
ans = {2x1x4x5 Cell Array}
>> num2cell(a,3)
ans = {2x3x1x5 Cell Array}
```

```
>> num2cell(a,[2,3,4])
ans =
{
  [1,1] = {1x3x4x5 Cell Array}
  [2,1] = {1x3x4x5 Cell Array}
}
```

2. num2str()函数

num2str()函数用于将数字转换为字符串。这个数字可以是单个数字,也可以是数组或元胞。

调用 num2str()函数时,至少需要指定一个参数,这个参数代表需要转换的源数字,代码如下:

```
>> num2str(123.456)
ans = 123.456
>> num2str([1 2;3 4])
ans =

1  2
3  4
```

如果源数字以科学记数法的参数形式传入,则 num2str()函数将返回标准的科学记数法形式字符串,代码如下:

```
>> num2str(10e100)
ans = 1e+101
```

如果源数字的大小较大,则 num2str()函数将返回标准的科学记数法形式的字符串,代码如下:

```
>> num2str(11111111111111111111111111111111111111111111111111111111111111111111)
ans = 1.11111111111111e+67
```

如果源数字的大小超过 Octave 可以表示的范围,则 num2str()函数将返回字符串 Inf,代码如下:

```
>> num2str(1111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```

```
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111)
ans = Inf
```

还可以额外传入一个参数，这个参数可以代表精确度或者格式化字符串。

当额外传入一个整数时，最终结果将舍入到参数所指定的位数。如果传入的位数参数过大，则原来的数字将原样输出为一个字符串，代码如下：

```
>> num2str(123.456,4)
ans = 123.5
>> num2str(123.456,7)
ans = 123.456
```

如果传入的是格式化字符串，则输出的字符串将按照格式化字符串进行格式化，代码如下：

```
>> num2str(123.456,'%d')
ans = 123.456
>> num2str(123.456,'%f')
ans = 123.456000
>> num2str(123.456,'%8.8f')
ans = 123.45600000
```

注意：格式化后输出的字符串将脱掉字符串前方的空格缩进。

3. num2hex()函数

num2hex()函数用于将数字转换为用十六进制表示的字符串数组。这个数字可以是单个数字，也可以是数组或元胞。

我们在调用 num2hex()函数时，至少需要指定一个参数，这个参数代表需要转换的源数字。在进行十六进制数组转换时，num2hex()函数将数字中的每个数位上对应内存中的值进行十六进制转换，代码如下：

```
>> num2hex(123.456)
ans = 405edd2f1a9fbe77
>> num2hex([1 2;3 4])
ans =

3ff0000000000000
4008000000000000
4000000000000000
4010000000000000
```

由于 num2hex()函数转换的是数字在内存中的值,所以在进行转换时,根据传入的数字精度不同,返回的字符串长度也不同,代码如下:

```
>> num2hex(single([1 2;3 4]))
ans =

3f800000
40400000
40000000
40800000

>> num2hex(int8([1 2;3 4]))
ans =

01
03
02
04
```

可以追加传入一个参数 cell,在追加这个参数之后,数字将被转换为用十六进制表示的字符串元胞,代码如下:

```
>> num2hex([1 2;3 4],'cell')
ans =
{
  [1,1] = 3ff0000000000000
  [2,1] = 4008000000000000
  [3,1] = 4000000000000000
  [4,1] = 4010000000000000
}
```

5.7.2 整数类型变量转换

int2str()函数用于将整数转换为字符串。

调用 int2str()函数时,至少需要指定一个参数,这个参数代表需要转换的源数字,代码如下:

```
>> int2str(123)
ans = 123
```

可以对一个小数调用 int2str()函数进行转换。转换后的结果相当于先对小数进行四舍五入再转换为字符串,代码如下:

```
>> int2str(123.499)
ans = 123
>> int2str(123.501)
ans = 124
```

还可以对一个复数调用 int2str()函数进行转换。此时只有实部会被转换为字符串，代码如下：

```
>> int2str(123 + i)
ans = 123
```

5.7.3　元胞类型变量转换

1. cell2mat()函数

cell2mat()函数用于将元胞转换为矩阵。

调用 cell2mat()函数时需要指定一个参数，这个参数代表需要转换的源元胞，代码如下：

```
>> cell2mat({1 2 3})
ans =

   1   2   3
```

注意：cell2mat()函数中传入的参数不得是元胞、结构体或数组的混合元胞，否则 cell2mat()函数将报错：

```
error: cell2mat: wrong type elements or mixed cells, structs, and matrices
error: called from
    cell2mat at line 53 column 11
```

2. cell2struct()函数

cell2struct()函数用于将元胞转换为结构体。

调用 cell2struct()函数时，至少需要指定两个参数，第 1 个参数代表需要转换的源元胞，第 2 个参数代表附加的字段名，代码如下：

```
>> a = cell2struct({1 2 3;4 5 6},{'1st','2nd'});
>> a(1)
ans =

  scalar structure containing the fields:
```

```
    1st = 1
    2nd = 4

>> a(2)
ans =

  scalar structure containing the fields:

    1st = 2
    2nd = 5
```

还可以指定第 3 个参数，这个参数代表转换源元胞所进行的维度，代码如下：

```
>> a = cell2struct({1 {2 3};4 {5 6}},{'1st','2nd'},2);
>> a(1)
ans =

  scalar structure containing the fields:

    1st = 1
    2nd =
    {
      [1,1] = 2
      [1,2] = 3
    }


>> a(2)
ans =

  scalar structure containing the fields:

    1st = 4
    2nd =
    {
      [1,1] = 5
      [1,2] = 6
    }
```

5.7.4 二进制类型变量转换

bin2dec()函数用于将二进制数字表示的字符串转换为十进制的数字。

调用 bin2dec ()函数时需要指定一个参数，这个参数代表需要转换的源字符串，代码

如下：

```
>> bin2dec('111')
ans = 7
```

bin2dec()函数也支持对数组中的每个字符串元素进行转换，代码如下：

```
>> bin2dec(['111';'1110'])
ans =

     7
    14
```

如果转换的字符串含有 0、1 和空格之外的字符，则 bin2dec()函数将返回 NaN，代码如下：

```
>> bin2dec('2')
ans = NaN
```

5.7.5　十进制类型变量转换

1. dec2base()函数

dec2base()函数用于将十进制数字转换为以一个非零数字为基底的数字。事实上，dec2base()函数的作用是进行任意进制的数字转换。

在调用 dec2base()函数时，至少需要指定两个参数，第 1 个参数代表需要转换的源数字，第 2 个参数代表基底，代码如下：

```
>> dec2base(1234,3)
ans = 1200201
```

上面的代码中，小于基底的数位保持原样，大于基底的数位则进行进位，最终的结果是计算十进制数字 1234 的三进制并表示为 1200201。

dec2base()函数也可以进行矩阵类型数据的转换。转换矩阵时，矩阵内的每个元素都会按照转换规则分别进行转换，然后长度不足的元素使用前导 0 补齐转换后的数位，代码如下：

```
>> dec2base([1 2;3 4],3)
ans =

01
10
02
11
```

dec2base()函数还支持抽象的数字表示方法。将基底以一个字符串表示，那么输出结果将以字符串中的字符表示，此时的字符被抽象为某个数字，然后配合数字进位等运算，最终输出字符串结果，代码如下：

```
>> dec2base(0,'asd')
ans = a
>> dec2base(1,'asd')
ans = s
>> dec2base(2,'asd')
ans = d
>> dec2base(100,'asd')
ans = sadas
>> dec2base(1000,'asd')
ans = ssasaas
```

注意：dec2base()函数中传入的基底不得含有空格，否则 dec2base()函数将报错：

```
error: dec2base: whitespace characters are not valid symbols
error: called from
    dec2base at line 83 column 7
```

上面的代码中，0 被抽象为 a，1 被抽象为 b，2 被抽象为 c，以此类推进行数字表示。

2. dec2bin()函数

dec2bin()函数用于将十进制数字转换为二进制数字表示的字符串。

调用 dec2bin()函数时，至少需要指定一个参数，这个参数代表需要转换的源数字，代码如下：

```
>> dec2bin(1234)
ans = 10011010010
```

dec2bin ()函数也可以用于转换一个数组，代码如下：

```
>> dec2bin([1 2;3 4])
ans =

001
011
010
100
```

3. dec2hex()函数

dec2hex()函数用于将十进制的数字转换为十六进制数字表示的字符串。

调用 dec2hex()函数时，至少需要指定一个参数，这个参数代表需要转换的源数字，代码如下：

```
>> dec2hex(1234)
ans = 4D2
```

dec2hex()函数也可以用于转换一个数组，代码如下：

```
>> dec2hex([12 34;1234 0])
ans =

00C
4D2
022
000
```

还可以额外指定另一个参数，用于指定转换后的数字的位数。如果数字的位数大于应有的位数，则 dec2hex()函数将在数字的前面使用前导零补满返回的数字结果，代码如下：

```
>> dec2hex(1234,6)
ans = 0004D2
>> dec2hex(1234,2)
ans = 4D2
```

5.7.6　十六进制类型变量转换

1. hex2dec()函数

hex2dec()函数用于将十六进制数字表示的字符串转换为十进制的数字。

调用 hex2dec()函数时需要指定一个参数，这个参数代表需要转换的源字符串，代码如下：

```
>> hex2dec('aaa')
ans = 2730
```

hex2dec()函数也支持对数组中的每个字符串元素进行转换，代码如下：

```
>> hex2dec(['a';'b';'c';'d'])
ans =

   10
   11
   12
   13
```

如果需要转换的字符串含有 0～9、a～f、A～F 和空格之外的字符，则 hex2dec()函数将返回 NaN，代码如下：

```
>> hex2dec(['h'])
ans = NaN
>> hex2dec(["1\t"])
ans = NaN
```

2. hex2num()函数

hex2num()函数用于将十六进制数字表示的字符串数组转换为十进制的数组。

调用 hex2num()函数时，至少需要指定一个参数，这个参数代表需要转换的源数字，代码如下：

```
>> hex2num('123a')
ans =   7.1928e-221
```

hex2num()函数也支持对数组中的每个字符串元素进行转换，代码如下：

```
>> hex2num(['1000000000';'700000000a'])
ans =

   1.2882e-231
   3.1050e+231
```

还可以额外传入一个参数，来指定字符串元素的类型。hex2num()函数支持的字符串元素类型如表 5-4 所示。

表 5-4　hex2num()函数支持的字符串元素类型

类　　型	含　　义
int8	8 位有符号整型
uint8	8 位无符号整型
int16	16 位有符号整型
uint16	16 位无符号整型
int32	32 位有符号整型
uint32	32 位无符号整型
int64	64 位有符号整型
uint64	64 位无符号整型
char	8 位字符型
single	32 位浮点型
double	64 位浮点型

根据指定的字符串类型的不同，获得的转换结果也不同。下面的例子展示了同一个字

符串数字在指定不同的两种类型参数时，会得到不同的转换结果：

```
>> hex2num(['1';'a'],'int8')
ans =

   16
  -96

>> hex2num(['1';'a'],'int32')
ans =

    268435456
  -1610612736
```

5.7.7　任意进制类型变量转换

base2dec()函数用于将任意进制数字表示的字符串转换为十进制的数字。

调用 base2dec()函数时，至少需要指定两个参数，其中的第 1 个参数代表需要转换的源数字，第 2 个参数代表源字符串的基底。在进行转换后，base2dec()函数输出十进制的数字，代码如下：

```
>> base2dec('1234',9)
ans = 922
>> base2dec('1234',8)
ans = 668
```

如果转换的字符串含有不合理的字符，则 base2dec()函数将返回 NaN，代码如下：

```
>> base2dec('9',8)
ans = NaN
>> base2dec('8',7)
ans = NaN
```

base2dec()函数也可以进行矩阵类型数据的转换，代码如下：

```
>> base2dec(['12';'34'],5)
ans =

    7
   19
```

base2dec()函数还支持抽象的数字表示方法。将数字和基底以字符串表示，那么源数字在转换时将以字符串中的字符表示，此时的字符被抽象为某个数字，然后配合数字进位等

运算，最终输出数字结果，代码如下：

```
>> base2dec('a','abc')
ans = 0
>> base2dec('b','abc')
ans = 1
>> base2dec('c','abc')
ans = 2
>> base2dec('abcabc','abc')
ans = 140
```

5.7.8 字符串转换

1. strjust()函数

strjust()函数用于将字符串数组的对齐方式进行调整。

调用 strjust()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，代码如下：

```
>> a = ['1';'12';'123'];
>> strjust(a)
ans =

  1
12
123
```

在 strjust()函数的作用下，字符串数组被改为右对齐的方式。

此外，可以追加传入一个参数，这个参数代表设定字符串数组的对齐方式。strjust()函数支持的对齐方式如表 5-5 所示。

表 5-5 strjust()函数支持的对齐方式

对齐方式	含　义
left	左对齐
center	居中
right	右对齐

下面的例子展示了将字符串数组设定为居中的用法：

```
>> a = ['1';'12';'123'];
>> strjust(a,'center')
ans =

1
```

```
12
123
```

2. str2double()函数

str2double()函数用于将字符串转换为 double 类型的数字变量。

调用 str2double()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，代码如下：

```
>> str2double('12.34')
ans = 12.340
```

str2double()函数支持虚数字符串的转换。str2double()函数支持的虚数字符串的格式如表 5-6 所示。

表 5-6　str2double()函数支持的虚数字符串的格式

虚数字符串的格式	虚数字符串的格式	虚数字符串的格式
a+bi	a+i*b	b*i+a
a+b*i	bi+a	i*b+a

下面的例子展示了将虚数字符串转换为虚数的用法：

```
>> str2double('12.34i+34')
ans = 34.000 + 12.340i
```

str2double()函数也可以进行矩阵类型数据的转换，代码如下：

```
>> str2double(['1';'2';'3';'4'])
ans =

   1
   2
   3
   4
```

在转换矩阵类型数据时，如果矩阵中含有空行，则 str2double()函数将略过空行，将剩余的矩阵分量正常进行转换，代码如下：

```
>> str2double(['1';'2';'';'4'])
ans =

   1
   2
   4
```

如果转换的字符串含有不合理的字符，则 str2double()函数将返回 NaN，代码如下：

```
>> str2double(['1';'2';' ';'4'])
ans =

     1
     2
   NaN
     4
```

3. str2num()函数

str2num()函数用于将字符串转换为数字。

调用 str2num()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，代码如下：

```
>> str2num('12.34')
ans = 12.340
```

此外，str2num()函数也可以进行矩阵类型数据的转换，代码如下：

```
>> str2num(['12';'34'])
ans =

   12
   34
```

此外，str2num()函数还提供了转换的状态作为可选返回参数。此时 str2num()函数的行为遵循以下规则：

- ❑ 如果 str2num()函数转换成功，则可选参数被赋值为逻辑变量 1；
- ❑ 如果 str2num()函数转换失败，则可选参数被赋值为逻辑变量 0。

```
>> [a,optional] = str2num('1234')
a = 1234
optional = 1
>> [a,optional] = str2num('aaaa')
a = [](0x0)
optional = 0
```

str2num()函数使用了 eval()函数实现数字的转换，因此，我们可以利用 str2num()函数执行 Octave 语句，然后 str2num()函数仍然可以继续其转换过程，代码如下：

```
>> [a,optional] = str2num("b = 2")
a = 2
optional = 1
```

5.7.9　函数句柄转换

1. str2func()函数

str2func()函数用于将字符串转换为函数的句柄。

调用 str2func()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，然后 str2func()函数将字符串解析为一个函数名称，最后返回这个函数的句柄，代码如下：

```
>> str2func('sin')
ans = @sin
```

如果函数名称对应的函数不存在，则 str2func()函数将报错，提示函数不存在，代码如下：

```
>> str2func('wrong_name')
error: @wrong_name: no function and no method found
```

注意：早期的 Octave 版本支持可选参数 global。在追加传入 global 参数时，str2func()函数在进行转换时将忽略局部函数。该参数现在已经不再被支持。

2. func2str()函数

func2str()函数用于将字符串转换为函数的句柄。

调用 func2str()函数时，至少需要指定一个参数，这个参数代表需要转换的源句柄，然后 func2str()函数将字符串解析为函数名，最后返回对应的字符串，代码如下：

```
>> func2str(@sin)
ans = sin
```

如果句柄对应的函数不存在，则 func2str()函数将报错，提示函数不存在，代码如下：

```
>> func2str(@wrong_name)
error: @wrong_name: no function and no method found
```

5.7.10　矩阵转换

1. mat2cell()函数

mat2cell()函数用于将矩阵转换为元胞。

调用 mat2cell()函数时，至少需要指定两个参数，其中，第 1 个参数代表需要转换的源矩阵，第 2 个参数代表转换矩阵的行数，代码如下：

```
>> a = [1 2 3;4 5 6;7 8 9];
>> mat2cell(a,[1 2])
ans =
{
  [1,1] =

     1  2  3

  [2,1] =

     4  5  6
     7  8  9

}
```

在上面的代码中，矩阵被转换为元胞，元胞包含两部分，第 1 部分含有源矩阵的第 1 行分量，第 2 部分含有源矩阵的第 2～3 行分量。

还可以追加传入第 3 个参数，这个参数代表转换矩阵的列数，代码如下：

```
>> mat2cell(a,[1 2],[2,1])
ans =
{
  [1,1] =

     1  2

  [2,1] =

     4  5
     7  8

  [1,2] = 3
  [2,2] =

     6
     9

}
```

在上面的代码中，矩阵被转换为元胞，元胞包含 4 部分，第 1 部分含有源矩阵的第 1 行和第 1～2 列围成的分量，第 2 部分含有源矩阵的第 2～3 行和第 1～2 列围成的分量，第 3 部分含有源矩阵的第 1 行和第 3 列围成的分量，第 4 部分含有源矩阵的第 2～3 行和第 3 列围成的分量。

以此类推，还可以继续追加传入的参数，这个参数代表转换矩阵的其他维度，代码如下：

```
>> a = [1:9 1:9 1:9];
>> a = reshape(a,[3 3 3])
a =

ans(:,:,1) =

   1  4  7
   2  5  8
   3  6  9

ans(:,:,2) =

   1  4  7
   2  5  8
   3  6  9

ans(:,:,3) =

   1  4  7
   2  5  8
   3  6  9

>> mat2cell(a,[1 2],[2 1],[1 2])
ans = {2x2x2 Cell Array}
```

2. mat2str()函数

mat2str()函数用于将元胞转换为结构体。

调用 mat2str()函数时，至少需要指定一个参数，这个参数代表需要转换的源矩阵，代码如下：

```
>> mat2str([1 2 3])
ans = [1 2 3]
```

mat2str()函数还可以转换复数矩阵，代码如下：

```
>> mat2str([1 + i 2 + j])
ans = [1 + 1i 2 + 1i]
```

还可以额外传入一个参数，这个参数代表转换后的字符串的精度，代码如下：

```
>> mat2str([123.456 2],1)
ans = [1e + 02 2]
>> mat2str([123.456 2],3)
ans = [123 2]
```

继续追加一个参数 class，此时 mat2str()函数将返回转换之后的矩阵的构造方法，然后调用 eval()函数可以调用这种方法，最后复原矩阵，代码如下：

```
>> a = mat2str([123.456 2],3,"class")
a = double([123 2])
>> eval(a)
ans =

   123     2
```

5.7.11 编码格式转换

1. unicode2native()函数

unicode2native()函数用于将 UTF-8 编码的字符串转换为本地格式的字符串。

调用 unicode2native()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，然后 unicode2native()函数将返回本地格式的字符串。此外，我们还可以追加第 2 个参数，这个参数代表转换之后的字符串编码，例如 UTF-16 编码，转换后的字符串在 Octave 的内存中仍然以本地格式进行存储。

2. native2unicode

native2unicode()函数用于将本地格式的字符串转换为 UTF-8 编码的字符串。

调用 native2unicode()函数时，至少需要指定一个参数，这个参数代表需要转换的源字符串，然后 native2unicode()函数将返回 UTF-8 编码的字符串。此外，还可以追加第 2 个参数，这个参数代表转换之后的字符串编码，例如 UTF-16 编码，转换之后的字符串在 Octave 的内存中仍然以 UTF-8 编码进行存储。

5.7.12 转义与反转义

1. do_string_escapes()函数

do_string_escapes()函数用于字符串转义。

调用 do_string_escapes()函数时需要指定一个参数，这个参数代表需要转换的源字符串，然后 do_string_escapes()函数将把反斜杠后面的第 1 个～第 3 个字符视为一个逃逸字符进行转义，最后返回新的字符串。逃逸字符如表 5-7 所示。

在逃逸字符的转换过程中，如果遇到不能显示的字符，例如字符\1，则转换后的字符也不能显示，代码如下：

```
>> do_string_escapes('0\1\12\123\1234\12345')
ans = 0
SS4S45
```

表 5-7　逃逸字符

字符	十进制	八进制	十六进制	字符	十进制	八进制	十六进制	字符	十进制	八进制	十六进制
(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
'	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
*	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
?	63	0077	0x3f	_	95	0137	0x5f				

2. undo_string_escapes()函数

undo_string_escapes()函数用于字符串反转义。

调用 undo_string_escapes()函数时需要指定一个参数，这个参数代表需要转换的源字符串，然后 undo_string_escapes()函数将对字符进行反转义，最后返回新的字符串，代码如下：

```
>> undo_string_escapes('''123\\\12''')
ans = '123\\\\\\12'
```

5.7.13 图形句柄转换

1. hdl2struct()函数

hdl2struct()函数用于将图形句柄转换为结构体。

调用 hdl2struct()函数时需要指定一个参数,这个参数代表需要转换的源句柄,代码如下:

```
>> hdl2struct(gca)
ans =

  scalar structure containing the fields:
#省略若干输出
    special =

       4  3  2  1
```

2. struct2hdl()函数

struct2hdl()函数用于将结构体转换为图形句柄。

调用 struct2hdl()函数时需要指定一个参数,这个参数代表需要转换的源结构体,代码如下:

```
>> struct2hdl('a')
```

5.8 数据查询

5.8.1 对比数组分量

diff()函数用于对比不同的数组分量,并且返回相邻的每两个分量之间存在的差别。调用 diff()函数时,我们至少需要传入一个参数,此时这个参数代表进行对比的数组。diff()函数进行对比时,将进行查询的数组内的前一个分量作为基准分量,然后将后一个分量分别与基准分量进行差分运算,且 diff()函数会在对应的下标位置输出两个分量的差值,代码如下:

```
>> a = [1 2 3 4 5];
>> diff(a)
ans =
```

```
    1   1   1   1

>> a = [1 2 3 2 1];
>> diff(a)
ans =

    1   1   -1   -1
```

此外，我们还可以额外指定第二个参数，此时这个参数代表差分运算的阶数。二阶差分运算的含义就是在进行一次差分运算的基础上，对返回结果再进行一次差分运算，然后返回新的结果。将同一个数组分别进行一次差分和二次差分，对比的代码如下：

```
>> a = [1 2 3 2 1];
>> diff(a,1)
ans =

    1   1   -1   -1

>> diff(a,2)
ans =

    0   -2   0
```

此外，可以额外指定第 3 个参数，此时这个参数代表进行差分运算的矩阵维度方向。

5.8.2　查询数组分量

find()函数用于查询非零数组分量，并且返回这些数组分量的下标。调用 find()函数时，我们至少需要传入一个参数，此时这个参数代表进行查询的数组，代码如下：

```
>> find([0 0 0 2 3])
ans =

    4   5
```

在上面的代码中，由于数组元素中的第 4 个分量和第 5 个分量为非零分量，所以 find()函数返回[4 5]。

如果查询无结果，则返回空数组，代码如下：

```
>> find(0)
ans = [](0x0)
```

可以额外指定第 3 个参数，此时这个参数代表返回数组的长度限制，代码如下：

```
>> a = [0 0 1 2];
>> find(a,4)
ans =

   3   4

>> find(a,2)
ans =

   3   4

>> find(a,1)
ans = 3
```

在上面的代码中，当第 2 个参数大于或等于非零分量的个数时，find()函数返回全部非零分量的下标。当第 2 个参数小于非零分量的个数时，find()函数只返回前一部分非零分量的下标，而且返回下标的个数等于第 2 个参数。

此外，我们还可以额外指定第 3 个参数，此时这个参数代表返回数组的方向。方向的取值为 first 或 last，代码如下：

```
>> a = [0 0 1 2 0 0 3 4];
>> find(a,2,'first')
ans =

   3   4

>> find(a,3,'last')
ans =

   4   7   8
```

在上面的代码中，

- 当第 3 个参数为 first 时，find()函数返回前一部分非零分量的下标，而且返回下标的个数等于第 2 个参数；
- 当第 3 个参数为 last 时，find()函数返回后一部分非零分量的下标，而且返回下标的个数等于第 2 个参数。

一种常用的用法是：将 find()函数和其他逻辑表达式配合使用，以查询满足特定条件的数组分量。下面的代码用于查询数组中所有等于 2 的分量，并返回这些分量的下标：

```
>> a = [1 2 3 4 0 0 2];
>> b = a == 2
b =
```

```
  0  1  0  0  0  0  1

>> find(b)
ans =

   2  7
```

5.8.3　查询图形对象

1. findobj()函数

可以调用 findobj()函数查询图形对象。findobj()函数可以不带参数而直接调用，此时 findobj()函数将返回所有图形对象，代码如下：

```
>> findobj
```

还可以使用键值对方式传入筛选参数，此时 findobj()函数将只返回符合要求的图形对象。下面的代码将返回键参数为 a，且值参数为 b 的全部图形对象：

```
>> findobj a b
```

此外，还可以同时指定多个键值对，此时 findobj()函数将只返回符合全部要求的图形对象。指定了两个键值对{“a”:”b”}和{“c”:”d”}的代码如下：

```
>> findobj a b c d
```

可以在多个键值对之间加入逻辑参数-and、-or、-xor 或者-not，指定多个键值对参数之间的逻辑关系。指定两个键值对{“a”:”b”}和{“c”:”d”}，且逻辑关系为或关系的代码如下：

```
>> findobj a b - or c d
```

还可以指定-property 参数，此时 findobj()函数将只返回含有此键参数的图形对象。下面的代码将返回含有键参数为 a 字符的全部图形对象：

```
>> findobj - property a
```

指定-regexp 参数，配合正则表达式返回匹配的图形对象。下面代码将返回键参数为 a 字符，且值参数内含有 b 字符的全部图形对象：

```
>> findobj - regexp a [b]
```

指定 findobj()函数的查询范围，只需额外添加图形句柄数组参数，并且将这个参数放

在第一个参数的位置上，代码如下：

```
>> a = figure
a = 3
>> b = figure
b = 4
>> findobj([a b],'-regexp','a','[b]')
```

还可以指定-depth 参数，然后追加一个查询层数，限制返回的查询结果在这个查询层数之内，代码如下：

```
>> a = figure
a = 3
>> b = figure
b = 4
>> findobj([a b],'-depth',3)
ans =

   3
   4
```

其中，追加的查询层数最小为 0，代表对象列表中的对象本身。层数每增加 1，则代表每含有一层继承关系。

2. findall()函数

可以调用 findall()函数查询图形对象。findall()函数和 findobj()函数的用法大体相同，唯一的区别是 findall()函数可以额外查询隐藏的对象。

5.8.4 查询图像对象

findfigs 函数用于查询图像对象。

(1) findfigs 函数只能查询可见的图像对象。

(2) 在 findfigs 函数成功返回图像对象后，如果那些对象原先不在屏幕上显示，则它们此时将被显示在屏幕上。

(3) 如果 findfigs 函数查询失败，则不返回任何值。

调用 findfigs 函数查询图像对象的代码如下：

```
>> findfigs
```

5.8.5 查询字符串分量

1. findstr()函数

findstr()函数用于查询字符串分量。调用 findstr()函数时，至少需要传入两个参数，此

时第 1 个参数代表被查询的字符串，第 2 个参数代表匹配格式，然后 findstr()函数返回匹配格式在被查询的字符串中的所有下标，代码如下：

```
>> findstr('aabbcc','a')
warning: findstr is obsolete; use strfind instead
ans =

   1   2
```

如果查询无结果，则返回空数组，代码如下：

```
>> findstr('b','a')
ans = [](0x0)
```

此外，我们还可以额外指定第 3 个参数，此时这个参数代表递归查询的开关：

- 当第 3 个参数为 0 时，findstr()函数将不会查询已经查询过的字符串部分；
- 当第 3 个参数为非 0 时，findstr()函数将查询所有字符串部分。

对于匹配格式中含有多个字符的场合，递归查询特性将非常好用。开启递归查询和关闭递归查询的代码如下：

```
>> findstr("abababa","aba",1)
ans =

   1   3   5

>> findstr("abababa","aba",0)
ans =

   1   5
```

在上面的代码中，调用 findstr()函数在开启递归查询和关闭递归查询时的返回结果不同。

2. strfind()函数

strfind()函数用于查询字符串分量。调用 strfind()函数时，至少需要传入两个参数，此时第 1 个参数代表被查询的字符串，第 2 个参数代表匹配格式，然后 strfind()函数返回匹配格式在被查询的字符串中的所有下标，代码如下：

```
>> strfind('aabbcc','a')
ans =

   1   2
```

如果查询无结果，则返回空数组，代码如下：

```
>> strfind('b','a')
ans = [](0x0)
```

还可以额外指定 overlaps 参数，此时接下来的参数代表递归查询的开关：

- 当接下来的参数为 false 时，strfind()函数将不会查询已经查询过的字符串部分；
- 当接下来的参数为 true 时，strfind()函数将查询所有字符串部分。

对于匹配格式中含有多个字符的场合，递归查询特性将非常好用。开启递归查询和关闭递归查询的代码如下：

```
>> strfind("abababa","aba","overlaps",true)
ans =

   1  3  5

>> strfind("abababa","aba","overlaps",false)
ans =

   1  5
```

在上面的代码中，调用 strfind()函数在开启递归查询和关闭递归查询时的返回结果不同。

此外，strfind()函数还支持多字符串匹配。只需将多个字符串写成一个字符串元胞，以第 1 个参数传入，代码如下：

```
>> strfind({'aa','ab','bb'},'a')
ans =
{
  [1,1] =

     1  2

  [1,2] = 1
  [1,3] = [](0x0)
}
```

3. strmatch()函数

strmatch()函数用于查询字符串分量。调用 strmatch()函数时，至少需要传入两个参数，此时第 1 个参数代表匹配格式，第 2 个参数代表被查询的字符串数组或字符串元胞，然后 strmatch()函数返回成功匹配的字符串分量索引。strmatch()函数在查询时只匹配和匹配长度相同的字符串长度，只要这个长度之内的字符相同，就视为匹配成功，代码如下：

```
>> strmatch('a',{'aabbcc','a','a '})
ans =

   1  2

>> strmatch('a',['aabbcc';'a';'a '])
ans =

   1
   2
```

如果查询无结果，则返回空数组，代码如下：

```
>> strmatch('b','a')
ans = [](0x0)
```

还可以额外指定 exact 参数，此时 strmatch()函数将进行全字匹配，代码如下：

```
>> strmatch('a',['aabbcc';'a';'a '])
ans =

   1
   2

>> strmatch('a',['aabbcc';'a';'a '],"exact")
ans = 2
```

4. strcmp()函数

strcmp()函数用于对比字符串分量。调用 strcmp()函数时，我们需要传入两个字符串参数，如果两个字符串相同，则返回 1；如果两个字符串不相同，则返回 0，代码如下：

```
>> strcmp('1','2')
ans = 0
>> strcmp('1','1')
ans = 1
```

5. strncmp()函数

strncmp()函数用于对比字符串分量。调用 strncmp()函数时，需要传入两个字符串参数，此时前两个参数为进行对比的字符串参数，第 3 个参数为对比的字符个数。如果两个字符串在前几个字符个数内相同，则返回 1；如果两个字符串在前几个字符个数内不相同，则返回 0，代码如下：

```
>> strncmp('1','123',2)
ans = 0
```

```
>> strncmp('1','123',1)
ans = 1
```

6. strcmpi()函数

strcmpi()函数用于对比字符串分量。调用 strcmpi()函数时,需要传入两个字符串参数,在认为大小写字母相同的前提下,如果两个字符串相同,则返回 1;如果两个字符串不相同,则返回 0,代码如下:

```
>> strcmpi('a','A')
ans = 1
>> strcmpi('ab','A')
ans = 0
```

7. strncmpi()函数

strncmpi()函数用于对比字符串分量。调用 strncmpi()函数时,需要传入两个字符串参数,此时前两个参数为进行对比的字符串参数,第 3 个参数为对比的字符个数。在认为大小写字母相同的前提下,如果两个字符串在前几个字符个数内相同,则返回 1;如果两个字符串在前几个字符个数内不相同,则返回 0,代码如下:

```
>> strncmpi('ab','A',1)
ans = 1
>> strncmpi('ab','A',2)
ans = 0
```

5.8.6 查询字符索引

1. index()函数

index()函数用于查询某个字符在字符串中的索引位置。调用 index()函数时,至少需要传入两个参数,此时第 1 个参数代表被查询的字符串,第 2 个参数代表匹配字符,然后 index()函数返回匹配格式在被查询的字符串中的第 1 个下标,代码如下:

```
>> index('aabbcc','a')
ans = 1
```

如果查询无结果,则返回 0,代码如下:

```
>> index('b','a')
ans = 0
```

2. rindex()函数

rindex()函数用于查询某个字符在字符串中的索引位置。调用 rindex()函数时,至少需要传入两个参数,此时第 1 个参数代表被查询的字符串,第 2 个参数代表匹配字符,然后

rindex()函数返回匹配格式在被查询的字符串中的最后一个下标，代码如下：

```
>> rindex('aabbcc','a')
ans = 2
```

如果查询无结果，则返回 0，代码如下：

```
>> rindex('b','a')
ans = 0
```

3. strchr()函数

strchr()函数用于查询某个字符在字符串中的索引位置。调用 strchr()函数时，至少需要传入两个参数，此时第 1 个参数代表被查询的字符串，第 2 个参数代表匹配字符，然后 strchr()函数返回匹配格式在被查询的字符串中的所有下标，代码如下：

```
>> strchr('aabbcc','a')
ans =

   1  2
```

如果查询无结果，则返回空数组，代码如下：

```
>> strchr('b','a')
ans = [](0x0)
```

可以追加第 3 个参数，此时这个参数代表查询的字符个数。当查询操作到达指定的字符个数时便不再继续查询，代码如下：

```
>> strchr('aabbcc','a')
ans =

   1  2

>> strchr('aabbcc','a',1)
ans = 1
```

还可以指定第 4 个参数，这个参数代表查询的方向。方向参数可以被指定为 first 或者 last，代码如下：

```
>> strchr('aabbcc','a',1,'first')
ans = 1
>> strchr('aabbcc','a',1,'last')
ans = 2
```