第 3 章

运算符与输入、输出

运算符分为两大类：以符号表示的运算符和以函数表示的运算符。根据惯例，将以符号表示的运算符称为“运算符号”，将以函数表示的运算符称为“运算函数”。

3.1 运算符号

3.1.1 代数运算符

对于 Octave 而言，运算符号分为以下几种：

1. 加号“+”

加号代表按矩阵相加。

2. 按元素加号“.+”

按元素加号和加号+等效。

3. 减号“-”

减号代表按矩阵相减。

4. 按元素减号“.-”

按元素减号和减号“—”等效。

5. 乘号“*”

乘号代表按矩阵相乘。使用乘号时，乘号前的矩阵的列数必须等于乘号后的矩阵的行数。

6. 按元素乘号“.*”

使用按元素乘号时，按元素乘号前的矩阵的列数必须等于按元素乘号后的矩阵的列数，而且按元素乘号前的矩阵的行数必须等于按元素乘号后的矩阵的行数。

7. 除号“/”

除号代表按矩阵右除。在使用除号时，如果参与除运算的两个矩阵当中至少有一个不是方阵，或者说是奇异矩阵，则计算结果将返回最小范数解。

8. 按元素除号“./”

按元素除号代表按元素右除。使用按元素除号时，按元素除号前的矩阵的列数必须等

于按元素除号后的矩阵的列数，而且按元素除号前的矩阵的行数必须等于按元素除号后的矩阵的行数。

9. 左除号“\”

左除号代表按矩阵左除。使用左除号时，如果参与左除运算的两个矩阵当中至少有一个不是方阵，或者说是奇异矩阵，则计算结果将返回最小范数解。

10. 按元素左除号“.\”

使用按元素左除号时，按元素左除号前的矩阵的列数必须等于按元素左除号后的矩阵的列数，而且按元素左除号前的矩阵的行数必须等于按元素左除号后的矩阵的行数。

11. 乘方符号“^”

乘方符号代表多种含义：

如果乘方符号前的数是底数，乘方符号后的数是指数，则乘方符号返回两数做乘方运算的值。如果乘方符号前的数是数字，乘方符号后的数是矩阵，则乘方符号返回一组展开值，其等于乘方符号后的数展开之后，得到的每个数和乘方符号前的数做乘方运算的值。如果乘方符号前的数是矩阵，乘方符号后的数是整型，则乘方符号返回一个矩阵，其等于乘方符号前的数中的每个数字与乘方符号后的数做乘方运算的值。如果乘方符号前的数是矩阵，乘方符号后的数是小数，则乘方符号返回一个矩阵，其等于乘方符号后的数展开之后，乘方符号前的数中的每个数字与乘方符号后的数做乘方运算的值。如果乘方符号前的数和乘方符号后的数都是矩阵，则乘方符号计算会出错。

12. 第二种乘方符号“”**

第二种乘方符号 ** 和乘方符号^等效。

13. 按元素乘方符号“.^”

如果按元素乘方符号前的数和按元素乘方符号后的数都是矩阵，则按元素乘方前的矩阵的列数必须等于按元素乘方后的矩阵的列数，而且按元素乘方前的矩阵的行数必须等于按元素乘方后的矩阵的行数。此外，如果按元素乘方得到多个复数域的解，则将返回最小非零解。

14. 第二种按元素乘方符号“”**

第二种按元素乘方符号.** 和按元素乘方符号.^等效。

15. 负号“-”

负号代表一元减号。负号的位置在数之前，且负号之前不得含有第二个数参与运算。

16. 正号“+”

正号代表一元减号。正号的位置在数之前，且正号之前不得含有第二个数参与运算。

17. 转置符号“.'”

转置符号又叫复共轭转置符号。转置符号代表多种含义：如果转置符号前的数是实数，则返回转置符号前的数的实转置运算结果。如果转置符号前的数是复数，则返回转置符号前的数的复共轭转置运算结果。

18. 实转置符号“'”

实转置符号代表多种含义：如果实转置符号前的数是实数，则返回实转置符号前的数的实转置运算结果。如果实转置符号前的数是复数，则运算可能会出错。

19. 赋值运算符“=”

将等号右侧的值赋给等号左侧的变量。因为赋值操作的特殊性，等号左侧必须是一个可变变量，例如下面的语句：

```
2 = 2
```

此语句是非法语句，因为等号左侧的 2 是一个不可变变量，而不可变变量无法被赋值。

20. 自增运算符“++”

自增运算符分为后缀自增运算符和前缀自增运算符。如果自增运算符和其他运算符共同参与运算，则后缀自增运算符相当于先将当前变量和其他运算符参与运算，再将当前变量自行加 1。前缀自增运算符相当于先将当前变量自行加 1，再将当前变量和其他运算符参与运算。

21. 自减运算符“--”

自减运算符分为后缀自减运算符和前缀自减运算符。如果自减运算符与其他运算符共同参与运算，则后缀自减运算符相当于先将当前变量和其他运算符参与运算，再将当前变量自行减 1。前缀自减运算符相当于先将当前变量自行减 1，再将当前变量和其他运算符参与运算。

3.1.2　逻辑运算符

1. 逻辑与“&&”

逻辑与的判断逻辑如下：如果逻辑与两边的元素均为非零值，则表达式返回 1；如果逻辑与两边的元素至少一个为零，则表达式返回 0。

2. 逻辑或“||”

逻辑或的判断逻辑如下：如果逻辑或两边的元素均为零，则表达式返回 0；如果逻辑或两边的元素至少一个为非零，则表达式返回 1。

3. 按元素与“&”

对按元素与两边的每个分量进行如下判断：如果按元素与两边的对应分量均为非零，则返回分量 1；如果按元素与两边的对应分量至少一个为零，则返回分量 0。最后将所有的分量按照对应位置组合起来并返回。

4. 按元素或“|”

对按元素或两边的每个分量进行如下判断：如果按元素或两边的对应分量均为 0，则返回分量 0；如果按元素或两边的对应分量至少一个为非零，则返回分量 1。最后将所有的分量按照对应位置组合起来并返回。

3.1.3 逻辑运算的零值

逻辑运算的零“0”指的是：

(1) char 类型的“\0”。

(2) logical 类型的 false。

(3) int8 类型的 0。

(4) uint8 类型的 0。

(5) int16 类型的 0。

(6) uint16 类型的 0。

(7) int32 类型的 0。

(8) uint32 类型的 0。

(9) int64 类型的 0。

(10) uint64 类型的 0。

(11) single 类型的 0。

(12) double 类型的 0。

在进行逻辑运算时，只要参与运算的数符合以上的某一种零，那么这个值对应的含义就是逻辑意义上的假值，否则代表真值。

3.1.4 按元素逻辑运算和(狭义的)逻辑运算的区别

举一个例子：

令

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

那么，a&b 和 a&&b 的结果是不一样的，结果如下：

```
>> a&b
ans =

  1 1
  1 1

>> a&&b
ans = 1
```

其中，按元素逻辑运算将得到和参与运算的矩阵尺寸相同的矩阵，而(狭义的)逻辑运算将得到逻辑值 0 或 1。

3.1.5　赋值运算符

赋值运算使用等号表示，代表将等号右侧的值赋值给等号左侧的值，代码如下：

```
>> a = 1;
>> a
a = 1
```

3.1.6　复合运算符

为使迭代运算的表达式更加简洁，Octave 提供一种将赋值运算符和其他运算符合写的方式，包括：

(1) 加等于‘+=’。
(2) 减等于‘-=’。
(3) 乘等于‘*=’。
(4) 除等于‘/=’。
(5) 左除等于‘\=’。
(6) 乘方等于‘^=’。
(7) 按元素乘等于‘.*=’。
(8) 按元素左除等于‘./=’。
(9) 按元素除等于‘.\=’。
(10) 按元素乘方等于‘.^=’。
(11) 或等于‘|=’。
(12) 与等于‘&=’。

这种运算符一般称为“复合运算符”。使用复合运算符可以方便地表示运算逻辑。例如在下面的程序中：

```
>> a = 2;
>> a += 1
```

a+=1 代表的是 a=a+1，有效避免了在同一个语句中，相同的变量名 a 出现两次。由此可见，使用复合运算符也减少了由于变量输入错误导致的语句错误。

3.1.7　其他符号

1. 括号运算符

在 Octave 中，凡是改变运算顺序或者指定运算顺序所使用的括号统一使用圆括号，这一点和一般意义上的数学表达式不同(因为在数学表达式中，如果需要在圆括号之外再增加新的括号，则应该使用方括号、花括号等符号组合)。

在设计一个较长的表达式时，建议将括号的位置再三进行检查，以免造成不可预知的计算顺序错误。

2. 点号运算符

点号运算符可用于索引类中的方法、成员变量或成员常量，代码如下：

```
>> a.b
```

如果点号不能被解释为点号运算符，则 Octave 在处理点号时，实际上相当于将点号优先判断为点号是否可以被组合成复合运算符。如果点号不可以被组合成复合运算符，则点号将被视为小数点。

一个小数可以省略小数部分，直接在整型部分的最后加入一个小数点，那么这个小数的小数部分将被视为 0，代码如下：

```
>> 1.
ans = 1
>> 1.0
ans = 1
```

在 Octave 中，1. 和 1.0 是等效的，前者的小数部分被省略了。

3. 范围运算符

Octave 使用冒号“:”进行范围运算。

我们可以只写冒号，前后不加任何数字，此时表示的范围称为零元范围，等效于某一维度上的所有取值，代码如下：

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

   1  2  3
   4  5  6

>> a(:,1)
ans =

   1
   4

>> a(:,:)
ans =

   1  2  3
   4  5  6
```

此外，可以在一个冒号前后分别写入一个范围的下界和上界，此时表示的范围称为二元范围，等效于枚举在范围闭区间内间隔为 1 的所有元素，代码如下：

```
>> 1:7.4
ans =

   1   2   3   4   5   6   7

>> 0.8:7.4
ans =

    0.80000    1.80000    2.80000    3.80000    4.80000    5.80000    6.80000
```

如果在冒号前的实数小于在冒号后的实数，则范围运算将返回一个空矩阵，代码如下：

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

如果有复数参与范围运算，则复数的虚部将被当作 0 或－0 处理，代码如下：

```
>> -5-1.2i:i
ans =

  -5   -4   -3   -2   -1    0

>> -i:2i
ans =  -0
```

此外，还可以追加第 2 个冒号，然后追加第 3 个数字，此时表示的范围称为三元范围，等效于枚举在范围闭区间内间隔为第 2 个数字的所有元素，代码如下：

```
>> -3:1.2:3
ans =

   -3.0000   -1.8000   -0.6000    0.6000    1.8000    3.0000
```

注意：Octave 不支持一元范围。

4. 定义符号

在 Octave 中定义矩阵需要用到“[ ]”符号，代码如下：

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

   1  2  3
```

在 Octave 中定义元胞需要用到“{ }”符号，代码如下：

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

5. 索引运算符

在 Octave 中索引矩阵需要用到“( )”符号，代码如下：

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

在 Octave 中索引元胞需要用到“{ }”符号或“( )”符号，代码如下：

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

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

6. 换行符

Octave 使用反斜杠“\”或 3 个点“...”标识换行。换行符前后的两行被视为在同一行中。换行符不中断前后两行的语义，代码如下：

```
>> a = \
warning: using continuation marker \ outside of double quoted strings is deprecated and will be
remov
ed from a future version of Octave, use ... instead
1
a = 1
```

```
>> a = ...
1
a = 1
```

7. 消除返回值符号

Octave 使用分号“;”消除返回值的输出。如果需要同时进行大量表达式的计算，则可以使用分号合理消除输出，代码如下：

```
>> a = [1,2,3];
>> a = a';
>> a = a + [2,3,4]
```

在上面的例子中，变量 a 的赋值步骤被消除了返回值的输出，变量 a 的转置步骤也被消除了返回值的输出。

8. 逗号分隔符

Octave 使用逗号“,”在水平方向上分隔多个变量，代码如下：

```
>> sum([1,2],2)
```

9. 空格分隔符

Octave 使用空格“ ”在水平方向上分隔多个变量，代码如下：

```
>> sum([1 2],2)
```

10. 分号分隔符

Octave 使用分号“;”在垂直方向上分隔多个变量，代码如下：

```
>> sum([1;2],2)
```

11. 表达式连接符

Octave 使用逗号“,”将多个表达式连接为一个表达式。如果需要同时进行大量表达式的计算，就可以使用逗号合理连接表达式，从而减少返回值的个数，提高运算速度，代码如下：

```
>> a = 1:2:100000;
>> a = log(a) + exp(a),a/ = - 3
```

在上面的例子中，变量 a 的计算步骤被连接为一个表达式，减少了上万个数字的一次返回过程，运算时间也减少了将近一半。

12. 水平连接符

Octave 使用方括号“[ ]”和逗号分隔符或空格分隔符组合完成变量在水平方向上的连

接,代码如下:

```
>> a = {1};
>> b = a
b =
{
  [1,1] = 1
}

>> a = {1};
>> b = a;
>> [a b]
ans =
{
  [1,1] = 1
  [1,2] = 1
}

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

13. 垂直连接符

Octave 使用方括号“[ ]”和分号分隔符组合完成变量在垂直方向上的连接,代码如下:

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

3.1.8 运算符的运算顺序

Octave 的运算符一般遵循从左往右的运算顺序,但只有赋值运算符和复合运算符遵循从右往左的运算顺序。

3.1.9 运算符的优先级

Octave 中的运算符的优先级完全兼容数学意义上的运算顺序。例如,乘和除优先于加和

减，乘和除优先级相同，圆括号可以改变表达式的运算顺序等。

下面给出所有 Octave 中的运算符优先级，如表 3-1 所示。

表 3-1　Octave 中的运算符优先级

运　算　符	含　　义	运算方向
'()' '[]' '{}' '.'	圆括号、方括号、花括号、点号	从左向右
'++' '--'	后缀自增、后缀自减	从左向右
''' '.'' '^' '**' '.^' '.**'	复共轭转置、实转置、乘方、按元素乘方	从左向右
'+' '-' '++' '--' '~' '!'	一元加、一元减、前缀自增、前缀自减、取非	从左向右
'*' '/' '\' '.\' '.*' './'	乘、除、左除、按元素乘、按元素除、按元素左除	从左向右
'+' '-'	加、减	从左向右
':'	范围	从左向右
'<' '<=' '==' '>=' '>' '!=' '~='	小于、小于或等于、等于、大于或等于、大于、不等于	从左向右
'&'	按元素与	从左向右
'|'	按元素或	从左向右
'&&'	逻辑与	从左向右
'||'	逻辑或	从左向右
'=' '+=' '-=' '*=' '/=' '\=' '^=' '.*=' './=' '.\=' '.^=' '|=' '&='	赋值、加等于、减等于、乘等于、除等于、左除等于、乘方等于、按元素乘等于、按元素除等于、按元素左除等于、按元素乘方等于、或等于、与等于	从右向左

(1) 在同一个方格中的所有运算符具有相同优先级。

(2) 在不同方格中的所有运算符，优先级从上至下依次降低，运算时先使用上面的运算符进行运算，后使用下面的运算符进行运算。

3.2　简单的运算函数

1. ctranspose(x)

返回复共轭转置矩阵的值，等效于转置符号。

2. transpose(x)

返回实转置矩阵的值，等效于实转置符号。

3. ldivide(x,y)

返回按元素左除的值，等效于按元素左除符号。

4. mldivide(x,y)

返回矩阵左除的结果，结果相当于左除号之前的矩阵除以左除号之后的矩阵。这个函

数和 x\y 是等效的。

5. mtimes(x,y)

返回多个矩阵的乘积。这个函数和 x * y 是等效的。运算顺序从左到右。

6. plus(x,y)

这个函数和 x+y 是等效的。

7. minus(x,y)

返回两个矩阵相减的值。这个函数和 x-y 是等效的。

8. power(x,y)

返回两个矩阵按元素乘方的值。这个函数和 x.^ y 是等效的。

如果计算结果当中含有多个复数结果,则函数将返回最小的非负(正数或者 0)角度。如果只需实数域的解,则需要调用 realpow()、realsqrt()、cbrt()或者 nthroot()函数。

9. mpower(x,y)

返回两个矩阵按照乘方计算得到的值。这个函数和乘方符号"^"是等效的。

10. rdivide(x,y)

返回按元素右除的值。这个函数和 x./y 是等效的。

11. mrdivide(x,y)

返回两个矩阵右除的值。这个函数和 x/y 是等效的。

12. times(x,y)

返回按元素乘方的值。这个函数和 x. * y 是等效的。如果对两个以上的数字或者矩阵进行连续的 times()函数调用,则乘方结果会按照由左向右的顺序依次进行数值传递直至得出最终的乘方结果。

13. uminus(x)

返回一个矩阵取负数运算的结果,等效于一元减号。

14. uplus(x)

返回一个矩阵取正数运算的结果,等效于一元加号。

3.3 运算符重载

通过对运算符对应的函数进行重载,即可起到重载运算符的效果。下面给出运算符重载的代码:

```
#!/usr/bin/octave
#第 3 章/plus.m
function plus(a,b)
  fprintf("Function overloaded.\n")
endfunction
```

然后，运行下面的代码：

```
>> plus(1,2)
```

可以看到 Octave 输出如下结果：

```
Function is overloaded.
```

需要注意的是，运算符重载不会改变运算符号的计算方式。例如，在重载了 plus()函数之后，如果进一步计算：

```
>> 1 + 2
```

其输出仍然是

```
ans = 3
```

可以进行重载的运算符如表 3-2 所示。

表 3-2　可以进行重载的运算符

运算符用法	运 算 函 数	含　　义
a＋b	plus(a,b)	加
a－b	minus(a,b)	减
＋a	uplus(a)	一元加
－a	uminus(a)	一元减
a. * b	times(a,b)	按元素乘
a * b	mtimes(a,b)	按矩阵乘
a./b	rdivide(a,b)	按元素右除
a/b	mrdivide(a,b)	按矩阵右除
a.\b	ldivide(a,b)	按元素左除
a\b	mldivide(a,b)	按矩阵左除
a.^b	power(a,b)	按元素乘方
a^b	mpower(a,b)	按矩阵乘方
a＜b	lt(a,b)	小于
a＜＝b	le(a,b)	小于或等于
a＞b	gt(a,b)	大于
a＞＝b	ge(a,b)	大于或等于
a＝＝b	eq(a,b)	等于
a!＝b	ne(a,b)	不等于
a&b	and(a,b)	逻辑与
a|b	or(a,b)	逻辑或

续表

运算符用法	运 算 函 数	含　　义
!a	not(a)	逻辑非
a'	ctranspose(a)	复共轭转置
a. '	transpose(a)	转置
a : b	colon(a,b)	二元范围
a : b : c	colon(a,b,c)	三元范围
[a,b]	horzcat(a,b)	水平连接
[a;b]	vertcat(a,b)	垂直连接
a(s_1,…,s_n)	subsref(a,s)	下标选择
a(s_1,…,s_n) = b	subsasgn(a,s,b)	下标赋值
b(a)	subsindex(a)	返回对象的索引
disp	disp(a)	输出对象

3.4 输入、输出函数

3.4.1 文件输入、输出函数

Octave 支持的文件输入、输出函数如表 3-3 所示。

表 3-3 Octave 支持的文件输入、输出函数

文件输入、输出函数	含　　义
save	将变量存入文件中
load	从文件中导入变量
fdisp	从文件中显示一个变量的值
dlmwrite()	将变量加入分隔符存入文件中
dlmread()	从文件中导入加入分隔符的变量
csvwrite()	将变量加入分隔符,以 csv 格式存入文件中
csvread()	从 csv 格式的文件中导入变量
textread()	从文件或字符串中导入变量
textscan()、importdata()	从文件中导入变量

1. save 函数

使用 save 函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = [1;2;3];
>> save output_csv.csv
```

此时，第 3 章文件夹下将增加一个 output_csv.csv 文件，其中的内容类似于：

```
# Created by Octave 5.2.0, Thu Dec 17 07:40:49 2020 GMT <unknown@DESKTOP-0000000>
# name: a
# type: matrix
# rows: 3
# columns: 1
1
2
3
```

此外，可以在调用 save 函数时指定额外选项。调用 save 函数时指定额外选项的示例代码如下：

```
>> save -append output_csv.csv
```

上面的代码以追加写入方式将全部工作空间内的变量保存到文件 output_csv.csv 内。save 函数支持的额外选项如表 3-4 所示。

表 3-4　save 函数支持的额外选项

额 外 选 项	含　义
-	直接输出变量，而不保存到外部文件中
-append	将变量以追加写入的方式保存到外部文件中
-ascii	将变量以 ASCII 字符格式保存到外部文件中，且不保存文件头
-binary	将变量以 Octave 二进制格式保存到外部文件中
-float-binary	将变量以 Octave 二进制格式保存到外部文件中，且所有数据使用单精度浮点格式
-hdf5	将变量以 HDF5 格式保存到外部文件中
-float-hdf5	将变量以 HDF5 格式保存到外部文件中，且所有数据使用单精度浮点格式
-V7	将变量以 MATLAB 7 二进制格式保存到外部文件中
-v7	
-7	
-mat7-binary	
-V6	将变量以 MATLAB 6 二进制格式保存到外部文件中
-v6	
-6	
-mat-binary	

续表

额 外 选 项	含　　义
-V4	将变量以 MATLAB 4 二进制格式保存到外部文件中
-v4	
-4	
-mat4-binary	
-text	将变量以纯文本格式保存到外部文件中
-zip	对目标文件使用 gzip 格式进行压缩
-z	

此外，可以在调用 save 函数时指定通配符。示例代码如下：

```
>> save output_csv.csv var*
```

上面的代码将工作空间内名为 var 开头的变量保存到文件 output_csv.csv 内。save 函数支持的通配符如表 3-5 所示。

表 3-5　save 函数支持的通配符

通　配　符	含　　义
?	匹配任意一个字符
*	匹配零个以上字符
[ ]	匹配方括号内的字符
[!]	匹配除方括号内的字符
[^]	

2. dlmwrite()函数

使用 dlmwrite()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = [1;2;3];
>> dlmwrite('output_csv.dlm',a)
```

此时，第 3 章文件夹下将增加一个 output_csv.dlm 文件，其中的内容类似于：

```
1
2
3
```

此外，可以在调用 dlmwrite()函数时以键值对形式指定额外选项。示例代码如下：

```
>> dlmwrite('output_csv.dlm',a,'delimeter',',')
```

上面的代码将分隔符设置为逗号“,”。

dlmwrite()函数支持的键值对形式的额外选项如表 3-6 所示。

表 3-6　dlmwrite()函数支持的键值对形式的额外选项

键参数	值参数	含　义
append	on	开启追加写入
	off	关闭追加写入
delimeter	分隔符	使用这个分隔符代替默认的分隔符
newline	"\n"	使用 UNIX 风格换行
	"\r\n"	使用 Windows 风格换行
	"\r"	使用 Mac 风格换行
roffset	空行数字	在文件前增加若干空行
coffset	空列数字	在文件前增加若干空列
precision	精确度	指定保存数据的精确度

可以追加分隔符参数，指定不同的分隔符，代码如下：

```
>> dlmwrite('output_csv.dlm',a,',')
```

可以追加空行数字参数，指定文件前的空行数量，代码如下：

```
>> dlmwrite('output_csv.dlm',a,3)
```

可以追加空列数字参数，指定文件前的空列数量，代码如下：

```
>> dlmwrite('output_csv.dlm',a,3,4)
```

可以追加-append 参数开启追加写入，代码如下：

```
>> dlmwrite('output_csv.dlm',a,"-append")
```

3. csvwrite()函数

使用 csvwrite()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = [1;2;3];
>> csvwrite('output.csv',a)
```

此时，第 3 章文件夹下将增加一个 output.csv 文件，其中的内容类似于：

```
1
2
3
```

csvwrite()函数也支持 dlmwrite()函数的额外参数。

4. load 函数

使用 load 函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> load output_csv.csv
>> a
a =

   1
   2
   3
```

此外，可以在调用 load 函数时指定额外选项。示例代码如下：

```
>> save -binary output_csv.csv
```

上面的代码将以 Octave 二进制格式读取 output_csv.csv 内的变量。

load 函数支持的额外选项如表 3-7 所示。

表 3-7 load 函数支持的额外选项

额外选项	含　义
-force	如果在 Octave 的内存中存在同名变量，则导入后的变量将覆盖那些变量。此选项在新版本 Octave 中已经不起作用，因为 Octave 已经启用了此逻辑
-ascii	将变量以 ASCII 字符格式读取到内存空间中，且不读取文件头

续表

额外选项	含　义
-binary	将变量以 Octave 二进制格式读取到内存空间中
-hdf5	将变量以 HDF5 格式读取到内存空间中
-import	读取多维度的 HDF5 格式的文件。此选项在新版本 Octave 中已经不起作用，因为 Octave 已经启用了此逻辑
-mat	将变量以 MATLAB 6/7 二进制格式读取到内存空间中
-mat-binary	
-6	
-v6	
-7	
-v7	
-mat4-binary	将变量以 MATLAB 4 二进制格式读取到内存空间中
-4	
-v4	
-V4	
-text	将变量以纯文本格式读取到内存空间中

5. dlmread()函数

使用 dlmread()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> dlmread('output_csv.dlm')
ans =

   1
   2
   3
```

可以追加分隔符参数，指定不同的分隔符，代码如下：

```
>> dlmread('output_csv.dlm',a,',')
```

可以追加空行数字参数，指定跳过文件前的行数，代码如下：

```
>> dlmread('output_csv.dlm',a,3)
```

可以追加空列数字参数，指定跳过文件前的列数，代码如下：

```
>> dlmread('output_csv.dlm',a,3,4)
```

此外，还可以追加范围参数，指定读取文件的具体行数范围和具体列数范围。传入的范围参数为一个数组，其中：

(1) 数组中的第 1 个分量为读取文件的起始行数。

(2) 数组中的第 2 个分量为读取文件的终止行数。

(3) 数组中的第 3 个分量为读取文件的起始列数。

(4) 数组中的第 4 个分量为读取文件的终止列数。

指定读取文件的具体行数范围和具体列数范围的代码如下：

```
>> dlmread('output_csv.dlm',a,[1,2,3,4])
```

此外，在指定读取文件的具体行数范围和具体列数范围时，还可以使用工作表风格输入。上面的代码使用工作表风格输入的等效形式如下：

```
>> dlmread('output_csv.dlm',a,"A2:C4")
```

可以追加 emptyvalue 参数，指定空值所代表的数据。空值所代表的数据在读取到内存空间之后为空。指定空值所代表的数据为星号“*”的代码如下：

```
>> dlmread('output_csv.dlm',a,"emptyvalue","*")
```

6. csvread()函数

使用 csvread()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> csvread('output.csv')
ans =

   1
   2
   3
```

此外，csvread()函数也支持 dlmread()函数支持的额外参数。

7. textread()函数

使用 textread()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = textread('output.csv',"%d")
a =

  1
  2
  3
```

8. textscan()函数

使用 textscan()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = textscan("1\n2\n3\n","%d")
a =
{
  [1,1] =

    1
    2
    3

}
>> b = fopen('output.csv');
>> a = textscan(b,"%d")
a =
{
  [1,1] =

    1
    2
    3

}

>> fclose(b);
```

此外，还可以指定格式化字符串。textscan()函数支持的格式化字符串，如表 3-8 所示。

表 3-8　textscan()函数支持的格式化字符串

格式化字符串	含　义
%f	将文字解析为 double 类型数字
%f64	
%n	
%f32	将文字解析为 single 类型数字
%d	将文字解析为 int8 类型数字
%d8	将文字解析为 int8 类型数字
%d16	将文字解析为 int16 类型数字
%d32	将文字解析为 int32 类型数字
%d64	将文字解析为 int64 类型数字
%u	将文字解析为 uint8 类型数字
%u8	将文字解析为 uint8 类型数字
%u16	将文字解析为 uint16 类型数字
%u32	将文字解析为 uint32 类型数字
%u64	将文字解析为 uint64 类型数字
%s	将文字解析为字符串
%q	将文字解析为转义字符串
%c	读取下一个字符，包含分隔符、空白字符及行尾符
%[ ]	匹配括号内的字符
%[^]	匹配除括号内的字符
%N	配合 s、c、d、f、n、u 使用，将 N 替换为一个数字，指定字符的个数
% *	配合其他符号使用，跳过其他符号生效的字符
其他字符	跳过这些字符

可以在调用 textscan()函数时以键值对形式指定额外选项。示例代码如下：

```
>> a = textscan(b,"%d",'BufSize','10000')
```

上面的代码将缓冲区设为 10000。

textscan()函数支持的键值对形式的额外选项如表 3-9 所示。

表 3-9　textscan()函数支持的键值对形式的额外选项

键参数	值参数	含　义
BufSize	数字	指定这个数字为缓冲区大小
CollectOutput	1/0/true/false	如果值为 1/true，则读取的所有同类数值被存放在一个元胞中；如果值为 0/false，则读取的所有同类数值被存放在不同的列中

续表

键参数	值参数	含　义
CommentStyle	1×1 的元胞	跳过注释右面的内容
	含有两个分量的元胞	跳过左侧的内容和右侧的内容
Delimiter	字符串	指定这个字符串为分隔符
EmptyValue	字符串	指定这个字符串为空变量
EndOfLine	\n/\r\n/\r	指定这个字符串为换行符
HeaderLines	数字	指定这个数字为文件头的行数，用于跳过这些行数
MultipleDelimsAsOne	数字	如果这个数字为非 0，则多个连续的分割符将被视为一个分隔符；如果这个数字为 0，则多个连续的分割符将被视为多个分隔符
TreatAsEmpty	字符串	将单独的这个字符串视为丢失的变量
ReturnOnError	1/0	如果值为 1，则在读取出错时正常返回；如果值为 0，则在读取出错时返回 error，并丢弃读取的数据
WhiteSpace	字符串	将这个字符串视为空格，并且去除这些字符串

9. importdata()函数

使用 importdata()函数输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = importdata('output.csv',",")
a =

   1
   2
   3
```

此外，可以在调用 importdata()函数时指定分隔符。示例代码如下：

```
>> a = importdata('output.csv',",")
```

可以在调用 importdata()函数时指定文件头的行数。示例代码如下：

```
>> a = importdata('output.csv',",",0)
```

importdata()函数支持导入多种文件格式。支持的格式包括：

(1) ASCII 字符列表格式。
(2) 图像文件格式。
(3) MATLAB 文件格式。
(4) WAV 声频格式。

3.4.2 简单输入函数

Octave 支持的简单输入函数如表 3-10 所示。

表 3-10 Octave 支持的简单输入函数

简单输入函数	含 义
puts()	将字符串内容输入 stdout 中
fputs()	将字符串内容输入文件中

1. puts()函数

使用 puts()函数向 stdout 内输入以下数字：

```
1
2
3
```

代码如下：

```
#!/usr/bin/octave
#第 3 章/puts_123.m
puts("1\n2\n3\n");

>> puts_123
1
2
3
```

2. fputs()函数

使用 fputs()函数向 stdout 内输入以下数字：

```
1
2
3
```

代码如下：

```
#!/usr/bin/octave
#第 3 章/fputs_123.m
fputs(stdout,"1\n2\n3\n");
```

```
>> fputs_123
1
2
3
```

3.4.3　行输出函数

Octave 支持的行输出函数如表 3-11 所示。

表 3-11　Octave 支持的行输出函数

行输出函数	含　义
fgetl()	从文件中读取含有非换行符的一行并返回
fgets()	从文件中读取一行并返回
fskipl()	跳过文件中的若干行，返回文件中已经被跳过的行数
fclear()	将文件中已经被跳过的行数置 0

1. fgets()函数

使用 fgets()函数输出以下数字：

```
1
2
3
```

其中，第 3 章/number123.m 的内容如下：

```
1
2
3
```

代码如下：

```
#!/usr/bin/octave
#第 3 章/fgets_123.m
a = fopen('number123.m');
b = zeros(3,3);
b = [fgets(a) fgets(a) fgets(a)];
fclose(a);
[b(1) b(4) b(7)]'

>> fgets_123
ans =
```

```
1
2
3
```

2. fgetl()函数

使用 fgetl()函数输出以下数字：

```
1
2
3
```

代码如下：

```
#!/usr/bin/octave
#第 3 章/fgetl_123.m
a = fopen('number123.m');
b = zeros(3,3);
b = [fgetl(a) fgetl(a) fgetl(a)];
fclose(a);
b = char(b(1,:));
b'

>> fgetl_123
ans =

1
2
3
```

3.4.4 格式化输入、输出函数

Octave 支持的格式化输入、输出函数如表 3-12 所示。

表 3-12 Octave 支持的格式化输入、输出函数

格式化输入、输出函数	含 义
printf()	将格式化后的字符串输出到 stdout 中
sprintf()	将格式化后的字符串输出到一个字符串变量中
fprintf()	将格式化后的字符串输出到文件中
	将格式化后的字符串输出到 stdout 中
scanf()	从输入流中将格式化后的字符串输入
sscanf()	从一个字符串变量中将格式化后的字符串输入
fscanf()	从输入流中将格式化后的字符串输入
	从文件中将格式化后的字符串输入

此外，格式化输入、输出函数支持一种特定的格式化字符串，如表 3-13 所示。

表 3-13　特定的格式化字符串

格式化字符串	含　义
%d	格式化为整型
%s	格式化为字符串型
%%	百分号“%”
%o	格式化为八进制数字
%u	格式化为十进制数字
%x	格式化为十六进制数字
%c	格式化为单个字符
%f	格式化为浮点型
%e	格式化为科学记数法
%g	格式化为浮点型或科学记数法
%N	配合 d、s、o、u、x、c、f、e、g 使用，将 N 替换为一个数字，匹配相应的字符个数

1. printf()函数

使用 printf()函数输出以下数字：

```
1
2
3
```

代码如下：

```
>> printf('1\n2\n3\n')
1
2
3
```

2. sprintf()函数

使用 sprintf()函数输出以下数字：

```
1
2
3
```

代码如下：

```
>> sprintf('\n1\n2\n3\n')
ans =
1
2
3
```

3. fprintf()函数

使用 fprintf()函数输出以下数字：

```
1
2
3
```

代码如下：

```
>> fprintf('1\n2\n3\n')
1
2
3
```

4. scanf()函数

使用 scanf()函数在交互模式下输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = "1\n2\n3";
>> scanf('%d%d%d',3)
error: scanf: unable to read from stdin while running interactively
```

使用 scanf()函数在脚本模式下输入以下数字：

```
1
2
3
```

在 Shell 下执行代码如下：

```
#!/usr/bin/octave
#第 3 章/scanf_script.m
scanf('%d%d%d',3);

$ cd 第 3 章
$ ./scanf_script.m
1
2
3
```

在 PowerShell 下执行代码如下：

```
PS > C:\\WINDOWS\system32\wscript.exe "C:\Octave\Octave - 5.2.0\octave.vbs" -- no - gui .\
scanf_script.m
1
2
3
```

注意：scanf()函数在交互模式下无法正确读取输入的内容。

5. sscanf()函数

使用 sscanf()函数在交互模式下输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = "1\n2\n3";
>> sscanf(a,'%d%d%d',3)
ans =

   1
   2
   3
```

6. fscanf()函数

使用 fscanf()函数在交互模式下向文件内输入以下数字：

```
1
2
3
```

代码如下：

```
>> a = fopen('number123.m');
>> fscanf(a,'%d%d%d',a)
ans =

   1
   2
   3
>> fclose('number123.m');
```

其中，第 3 章/number123.m 的内容如下所示：

```
1
2
3
```

3.4.5 终端输入、输出函数

Octave 支持的终端输入、输出函数如表 3-14 所示。

表 3-14 Octave 支持的终端输入、输出函数

终端输入、输出函数	含　义
disp	将格式化后的字符串显示到命令行窗口中
	原样显示一行输入的内容
	显示一行输入的格式化后的内容
input	先输出提示字符串，再从输入流读取输入。输入的内容被视为表达式
	先输出提示字符串，再从输入流读取输入。输入的内容被视为字符串

注意：stdout 本身不属于终端，而属于文件，即便输出到 stdout 中的内容也可以在终端中显示，所以 sprintf()等函数不属于终端输入、输出函数。

1. disp()函数

使用 disp()函数输出以下数字：

```
1
2
3
```

代码如下：

```
>> disp(1);disp(2);disp(3);
1
2
3
```

2. input()函数

使用 input()函数输入以下数字：

```
1
2
3
```

代码如下：

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

   1
   2
   3
```

3.4.6　二进制输入、输出函数

Octave 支持的二进制输入、输出函数如表 3-15 所示。

表 3-15　Octave 支持的二进制输入、输出函数

二进制输入、输出函数	含　义
fread()	以二进制形式将文件内容读取到变量中
fwrite()	以二进制形式将变量内容写入文件中

fread()函数和 fwrite()函数接受如下读取大小或写入大小选项，指定读取或写入变量的大小选项，如表 3-16 所示。

表 3-16　fread()函数和 fwrite()函数的读取大小或写入大小选项

读取大小或写入大小选项	含　义
Inf	尽可能多地读取或写入
nr	至多读取或写入 nr 个分量
[nr,Inf]	至多读取或写入 nr 的倍数个分量。在返回的结果中，每列存储 nr 个分量，最后一列不足则补零
[nr,nc]	至多读取或写入 nr 乘 nc 个分量。在返回的结果中，每列存储 nr 个分量，不足则补零

注意：这里的零指的是基本变量类型的“0”值，在 5.3.2 节中将进行详细介绍。

fread()函数和 fwrite()函数接受如下精确度选项，指定读取或写入变量的精确度，如表 3-17 所示。

表 3-17　fread()函数和 fwrite()函数的精确度选项

精确度选项	含　义
uint8	默认选项，8 位无符号整型
int8	8 位有符号整型
integer * 1	

续表

精确度选项	含　义
uint16	16 位无符号整型
ushort	
unsigned short	
int16	16 位有符号整型
integer * 2	
short	
uint	32 位无符号整型
uint32	
unsigned int	
ulong	
unsigned long	
int	32 位有符号整型
int32	
integer * 4	
long	
uint64	64 位无符号整型
int64	64 位有符号整型
integer * 8	
single	32 位浮点型
float	
float32	
real * 4	
double	64 位浮点型
float64	
real * 8	
char	8 位单纯字符型
char * 1	
uchar	8 位无符号字符型
insigned char	
schar	8 位有符号字符型
signed char	

精确度 integer、real 和 char 可以加上“*”符号和数字进行组合，实现文件分块读取或写入。例如：调用 fread()函数时附加 real * 32 选项代表读入文件时按照每 32 个 real 空间的大小读取为一块分量的模式。

这些精确度选项可以加上“=>”符号进行排列组合。例如：调用 fread()函数时附加 int16=> int32 选项代表读入文件时按照 int16 模式，但返回变量时按照 int32 模式进行每个分量的写入。

此外，可以额外传入一个参数，代表读取每个（或每一块）分量之后跳过多少个（或多少块）元素。

fread()函数和 fwrite()函数接受如下架构模式选项，指定读取或写入文件的方式，如表 3-18 所示。

表 3-18　fread()函数和 fwrite()函数的架构模式选项

架构模式选项	含　义
native	使用当前机器的架构
ieee-be	使用大端架构
n	
ieee-le	使用小端架构
l	

以常用的 AMD64 机器为例，由于其通常使用小端架构进行文件存储，所以 fread()函数和 fwrite()函数在 AMD64 机器上通常也默认使用小端架构读取或写入文件。

1. fread()函数

使用 fread()函数输出以下数字：

```
1
2
3
```

代码如下：

```
#!/usr/bin/octave
#第 3 章/fread_123.m
a = fopen('number123.m');
b = fread(a,[3,Inf]);
fclose(a);
b = char(b);
(b(1,:))'

>> fread_123
ans =

1
2
3
```

2. fwrite()函数

使用 fwrite()函数输入以下数字：

```
1
2
3
```

注意：根据操作系统的不同，fwrite()函数的实现方式也不同。

第 1 种使用 fwrite()函数的实现方式，代码如下：

```
#!/usr/bin/octave
#第 3 章/fwrite_123.m
a = fopen('number123.m');
fwrite(a,"1\n2\n3\n");
fclose(a);

>> fwrite_123
```

第 2 种使用 fwrite()函数的实现方式，代码如下：

```
#!/usr/bin/octave
#第 3 章/fwrite_123_2.m
a = fopen('number123.m');
fwrite(a,"1\r\n2\r\n3\r\n");
fclose(a);

>> fwrite_123_2
```

第 3 种使用 fwrite()函数的实现方式，代码如下：

```
#!/usr/bin/octave
#第 3 章/fwrite_123_3.m
a = fopen('number123.m');
fwrite(a,"1\r2\r3\r");
fclose(a);

>> fwrite_123_3
```