第八章 输入/输出函数

在第二章中，我们已经学到如何用 load 和 save 命令加载和保存 **MATLAB** 数据，以及如何使用 fprintf 函数格式化输出数据。在本章中，我们将学习更多的关于 **MATLAB** 输入和输出的功能。首先，我们将会学习函数 textread，在 maltab5。3 中它是一个非常有用的函数。然后，我们将花更多的时间学习 load 和 save 命令。最后，我们将查看其他的 **MATLAB**I/O 选择。

熟悉 C 语言的读者对这部分数据将会十分的熟悉。但是，在 **MATLAB** 函数和 c 函数之间有细微的不同。

8.1 函数 textread

命令 textread 最早出现于 **MATLAB**5.3 中。它可以按列读取 ascii 文件中的元素，每一列中可能含有不同的数据类型。这函数读取其他程序生成的数据表时非常地有用。

这个命令的形式如下

[a, b, c, ...] = textread(filename, format, n)

其中 filename 代表要打开的文件的名字，format 是用于每一行数据类型的字符串，n 代表要读取的行数(如果没有 n，则这个命令将读完这个文件)。格式化字符串与函数 fprintf 格式化描述的字符串相同。注意输出参数的个数必须与你读取的列数相区配。

例如，假设文件 test_input.dat 包含下列数据

James Jones O+ 3.51 22 Yes
Sally Smith A+ 3.28 23 NO

这些数据用下面的函数读取一系列的数组。

[first, last, blood, gpa, age, answer] = textread('test_input.dat','%s %s %s %f %d %s')

当这个函数被编译时产生如下结果

```
> [first, last, blood, gpa, age, answer] = textread('test_input.dat','%s %s %s %f %d %s')
first =
    'James'
    'Sally'
last =
    'Jones'
    'Smith'
blood =
    'O+'
    'A+'
gpa =
    3.5100
    3.2800
age =
    22
    23
answer =
    'Yes'
    'NO'
```

这个函数可以通过在格式描述符前面加一个星号的方式来跳过某些所选项。例如，下面的语句只从文件只读取 first，last 和 gpa。

```
>> [first, last, gpa] = textread('test_input.dat','%s %s %*s %f %*d %*s')
first =
    'James'
```

```
        'Sally'
    last =
        'Jones'
        'Smith'
    gpa =
        3.5100
        3.2800
```

函数 textread 要比 load 命令简单有效的多。load 命令假设输入文件中的所有数据都是同一类型——它不支持在不同的列上有不同的数据。此外，它把所有的数据都存储在一个数据中。相反地，函数 textread 允许每一列都有独立的变量，当和由不同类型的数据组成的列运算时，它更加的方便。

函数 textread 中有许许多多参数，它们增加了函数的灵活性。你可通过咨询 **MATLAB** 的在线文本得到这些参数的使用细节。

常见编程错误

应用函数 text 从 ascii 文件中按行格式读取数据，这个 ascii 文件可能是其他语言生成的，或是由其他的应用程序生成的，例如表格。

8.2 关于 load 和 save 命令的进一步说明

save 命令把 **MATLAB** 工作区数据存储到硬盘，load 命令把硬盘上的数据拷贝到工作区中。save 命令即可用特殊的二进制格式 mat- file 存储数据，也可用普通的 ascii 码格式存储数据。save 命令的形式为

save filename [list of variables] [options]

如果只有 save 命令，那么当前工作区内的所有数据存储在一个名为 matlab.mat 的文件中。如果后面有一个文件名，那么这些数据将会存储在“filename.mat”的文件。如果后面还包括一系列的变量，那么就只存储这些特殊的变量。

支持 save 命令的参数如表 8.1 所示。

表 8.1save 命令的参数

参数	描述
- mat	以 mat 文件格式存储数据（默认）
- ascii	用 ascii 格式保存数据
- append	给已存在 matf 文件增加变量
- v4	也存储为 mat 文件格式，但能被 **MATLAB**4.0 读取

load 命令可以加载 mat 文件或普通的 ascii 文件中的数据。load 命令的形式如下

load filename [option]

如果只有 load 命令，**MATLAB** 将加载 matlab.mat 文件中的所有数据。如果还跟着一个文件名，它 load 命令将会加载这个文件中的数据。

支持 load 命令的参数被列于表 8.1 中。

尽管它们的优点不是十分的明显，但是 save 和 load 命令是 **MATLAB** 中功能最强大，最有用的 I/O 命令。它的优点是

1. 这些命令易于使用
2. mat 文件的平台独立。在一个支持 **MATLAB** 的计算机上编写的文件，在另一种支持 **MATLAB** 的计算机上，可以被读取。这种格式可以在 PC，Mac，许多不同版本的 Unix 上互相转换。
3. mat 文件高效的硬盘空间使用者，它存储数据是高精度的，在 mat 文件和 ascii 文件转化过程中会出现精度下降的情况。
4. mat 文件存储了工作区内的每一个变量的所有信息，包括它的类属，名字和它是不是全局变量。在 I/O 其他类型数据存储格式中所有的这些信息都会丢失。例如，假设工作区

包含下面信息。

```
>> whos
   Name          Size                Bytes  Class
   a             10x10                 800  double array (global)
   ans           1x1                     8  double array
   b             10x10                 800  double array
   c             2x2                   332  cell array
   string        1x16                   32  char array
   student       1x3                  2152  struct array
Grand total is 372 elements using 4124 bytes
```

如果工作区用 save workspace.mat 命令存储，那么文件 workspace.mat 就会被自动创建。当这个文件被加载时，工作区中的所有信息都会被恢复，包括每一项的类型和一变量是否为全局变量。

这个命令的缺点是生成的 mat 文件只能由 **MATLAB** 调用，其他的程序不可能利用他共享数据。如要你想要与其他程序共享数据，可以应用-ascii 参数，但它有诸多的限制。

表 8.2 load 命令参数

参数	描述
-mat	把文件当作 mat 文件看待（如果扩展名是 mat，此为默认格式）
-ascii	把文件当作 ascii 格式文件来看待（如果扩展名不为 mat，此为默认格式）

好的编程习惯

除非我们必须与非 **MATLAB** 程序进行数据交换，存储和加载文件时，都应用 mat 文件格式。这种格式是高效的且移植性强，它保存了所有 **MATLAB** 数据类型的细节。

save -ascii 根本不能存储单元阵列和结构数据，在保存字符串之前，它要把字符串转化相应的数字形式。load -ascii 命令只能加载空间独立的数据，这些数据每一行的元素个数都相等，**MATLAB** 把所有的数据都存储于一个变量中，这个变量与输出文件同名。如果你要用更高的要求（例如，保存和加载字符串，单元阵列或结构数组并与其它程序进行交换），那么你需要本章后面介绍的 I/O 命令。

如果我们要加载的文件名或变量名是字符串，那么我们要用这些命令的函数形式。例如，下面的代码段要求用户提供一个文件名，并把当前工作区保存在那个文件中。

```
filename = input('Enter save file name: ','s');
save(filename);
```

8.3 MATLAB 文件过程简介

为了使用在 **MATLAB** 程序中的文件我们需要一些方法选出我们所要的文件，并从中读取或写入数据。在 **MATLAB** 中有一种非常灵活的读取/写入文件的方法，不管这个文件是在磁盘还是在磁带上或者是其他的存储介质。这种机制就叫做文件标识（file id）（有时可简写为 fid），当文件被打开，读取，写入或操作时，文件标识是赋值于一个文件的数。文件标识是一个正整数。两种文件标识是公开的——文件标识 1 是标准输出机制，文件标识 2 是标准错误机制（stderr）。其他的文件标识，在文件打开时创立，文件关闭时消逝。

许多的 **MATLAB** 语句可以控制磁盘文件的输入或输出。文件 I/O 函数被总结在表 8.3 中。

表 8.3 **MATLAB** 输入/输出语句

类别	函数	描述
加载/保存工作区	load	加载工作区
	save	保存工作区
文件打开/关闭	fopen	打开文件

	fclose	关闭文件
二进制 I/O	fread	从文件中读取二进制数据
	fwrite	把二进制数据写入文件
格式化 I/O	fscanf	从文件中读取格式化数据
	fprintf	把格式化数据写入文件
	fgetl	读取文件的一行，忽略换行符
	fgets	读取文件的一行，不忽略换行符
文件位置、状态	delete	删除文件
	exist	检查文件是否存在
	ferror	所需文件的 I/O 错误情况
	feof	检测文件的结尾
	fseek	设置文件的位置
	ftell	检查文件的位置
	frewind	回溯文件
临时情况	tempdir	得到临时目录名
	tempname	得到临时文件名

我们可以用 fopen 语句把文件标识传递给磁盘文件或设备，用 fclose 语句把他们从中分开。一旦一个文件用 fopen 语句得到一个文件标识，我们就可以利用 **MATLAB** 输入输出语句。当我们对这个文件操作完后，fclose 语句关闭并使文件标识无效。当文件打开时，函数 frewind 和 fseek 常用于改变当前文件读取和写入的位置。

在文件中读取或写入数据的方法有两种方法：像二进制数据或像格式化字符数据。由实际位样式组成的二进制数据常用于存储于计算机内存中。读取和编写二进制数据是非常高效的，但是用户不能读取存在于文件中的数据。在格式化文件中的可以转化为符串的数据可以由用户直接读取。格式化 I/O 操作比二进制 I/O 操作要慢得多，效率要低得多。在本章中，我们将讨论两种类型的 I/O 的操作。

8.4 文件的打开与关闭

文件的打开与关闭函数，fopen 和 fclose 将在本节描述。

8.4.1 fopen 函数

fopen 函数打开一个文件并对返回这个文件的文件标识数。它的基本形式如下：

fid = fopen(filename, permission)

[fid, message] = fopen(filename, permission)

[fid, message] = fopen(filename,permission, format)

其中 filename 是要打开的文件的名字，premission 用于指定打开文件的模式，format 是一个参数字符串，用于指定文件中数据的数字格式。如果文件被成功打开，在这个语句执行之后，fid 将为一个正整数，message 将为一个空字符。如果文件打开失败，在这个语句执行之后，fid 将为-1，message 将为解释错误出现的字符串。如果 **MATLAB** 要打开一个不为当前目录的文件，那么 **MATLAB** 将按 **MATLAB** 搜索路径搜索。

permisson 的字符串被列在表 8.4 中。

对于一些如 PC 一样的平台，它更重要的是区分文本文件和二进制文件。如果文件以文本格式打开，那么一个“t”就应加入到 permission 字符串中（例如“rt”或“rt+”）。

表 8.4 fopen 文件 permissions

文件 permission	意义
r	以只读格式读取文件

r+	可对文件进行读写
w	删除一个已存在文件的内容（或创建一个新文件），并以只写格式打开
w+	删除一个已存在文件的内容（或创建一个新文件），并以读写格式打开
a	打开一个已存在的文件（或创建一个新文件），并以只写文件格式打开把写入的内容增加到文件的结尾
a+	打开一个已存在的文件（或创建一个新文件），并以只写文件格式打开把写入的内容增加到文件的结尾
W	不进行自动洗带的写入数据（针对于磁带机的特殊命令）
A	不进行自动洗带的添加数据（针对于磁带机的特殊命令）

如果是以二进制模式找开，那么“b”应加到 permission 字符串中（例如“rb”）。这实际上是不需要的，因为文件默认打开的方式是二进制模式。文本文件和二进制文件在 Unix 系统上是没有区别的，所以在这系统上，r 和 b 都不需要。

在 fopen 函数中的 format 字符串数据存储在文件中的格式。当在两计算机中传递互相矛盾的数据格式时，这个字符串才是必须的。一些可能的数字格式被总结在表 8.5 中。你可以从 **MATLAB** reference manual（参考手册）中得到所有可能的数字格式。

这个函数有两种提供信息的格式。函数

fids = fopen('all')

返回一个行向量，这个行向量由当打开的所有文件的文件标识组成（除了 stdcut 和 stderr）。在这个向量中的元素的个数与所要打开的文件的个数相等。函数

[filename, permission, format] = fopen(fid)

对于一指定文件标识的打开文件，返回它的名字，权限（permission）字符串和数字格式。

下面是一些正确应用 fopen 函数的例子。

8.4.1.1 情况 1：为输入而打开一二进制文件

下面的函数只为输入二进制数据而打开文件 example.dat。

fid = fopen('example.dat','r')

权限（permission）字符串是“r”，它指出这个文件的打开方式为只读。这个字符串也可以是“rb”，但这是没有必要的，因为 **MATLAB** 默认打开的是二进制文件。

8.4.1.2 情况 2：为文本输出打开一文件

下面的函数以文本输出打开文件 outdat。

fid = fopen('outdat','wr')

或

fid = fopen('outdat','at')

权限字符串“wt”指定这个文件为新建文本文件。如果这个文件已存在，那旧文件就会被删除，打开新建的文件等待写入数据。如果我们要替换先前已存在的数据，那么就可以采用这个形式。

权限运算符“at”指定一个我们想要增加数据的文本文件。如果这个文件已经存在了，那么它将会被打开，新的数据将会添加到已存在的数据中。如果我们不想替换已存在的数据，那么就可以采用这个方式。

8.4.1.3 以读写模式打开文件

下面的函数打开文件 junk，可以对它进行二进制输入和输出。
fid = fopen('junk', 'r+')
或
fid = fopen('junk','w+')
每一个语句与第二个语句的不同为第一句打开已存在文件，而第二个语句则会删除已存在的文件。

好的编程习惯
在使用 fopen 语句时，一定要注意指定合适的权限，这取决于你是要读取数据，还是要写入数据。好的编程习惯可以帮助你避免（类似于覆盖的）错误。

在试图打开一个文件之后，检查错误是非常重要的。如果 fid 的值为-1，那么说明文件打开失败。你将把这个问题报告给用户，允许他们选择其他的文件或跳出程序。

好的编程习惯
在文件打开操作后检查它的状态以确保它被成功打开。如果文件打开失败，提示用户解决方法。

8.4.2 fclose 函数

fclose 函数用于关闭一文件。它的形式为
status = fclose(fid)
status = fclose('all')
其中 fid 为文件标识，status 是操作结果，如果操作成功，status 为 0，如果操作失败，status 为-1。
函数 status = fclose('all')关闭了所有的文件，除了 stdout（fid = 1）和 stderr（fid = 0）。如果所有的文件关闭成功，status 将为 0，否则为-1。

8.5 二进制 I/O 函数

二进制 I/O 函数，fwrite 和 fread，将在本节讨论。

8.5.1 fwrite 函数

函数 fwrite 以自定义格式把二进制数据写入一文件。它的形式为
count = fwrite(fid, array, precision)
count = fwrite(fid, array, precision skip)
其中 fid 是用于 fopen 打开的一个文件的文件标识，array 是写出变量的数组，count 是写入文件变量的数目。
MATLAB 以列顺序输出数据，它的含义为第一列全部输出后，再输出第二列等等。例如，如果 array = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$，那么数据输出的顺序为 1，3，5，2，4，6。
参数 precision 字符串用于指定输出数据的格式。**MATLAB** 既支持平台独立的精度字符

串，在所有的有 **MATLAB** 运行的电脑，它是相同的，也支持平台不独立的精度字符串，它们在不同类型的电脑上精度也不同。你应当只用平台独立的字符串，在本书中出现的字符串均为这种形式。

表 8.6 MATLAB 精度字符串

精度字符串	C/Fortan 形式	意义
'char'	'char*1'	6 位字符
'schar'	'signed char'	8 位有符号字符
'uchar'	'unsigned char'	8 位无符号字符`
'int8'	'integer*1'	8 位整数
'int16'	'integer*2'	16 位整数
'int32'	'integer*4'	32 位整数
'int64'	'integer*8'	64 位整数
'uint8'	'integer*1'	8 位无符号整数
'uint16'	'integer*2'	16 位无符号整数
'uint32'	'integer*4'	32 位无符号整数
'uint64'	'integer*8'	64 位无符号整数
'float32'	'real*4'	32 位浮点数
'float64'	'real*8'	64 位浮点数
'bitN'		N 位带符号整数(1≤N≤64)
'ubitN'		N 位无符号整数(1≤N≤64)

平台独立的精度显示在表 8.6 中。所有的这些精度都以字节为单位，除了“bitN”和“ubitN”，它以位为单位。

选择性参数 skip 指定在每一次写入输出文件之前要跳过的字节数。在替换有固定长度的值的时侯，这个参数将非常的有用。注意如果 precision 是一个像“bitN”或“ubitN”的一位格式，skip 则用位当作单位。

8.5.2 fread 函数

函数 fread 读取用用户自定义格式从一文件中读取二进制数据。它的格式如下

[array, count] = fread(fid, size, precision)

[array, count] = fread(fid, size, precision, skip)

其中 fid 是用于 fopen 打开的一个文件的文件标识，array 是包含有数据的数组，count 是读取文件中变量的数目，size 是要读取文件中变量的数目。

参数 size 用于指定读取文件中变量的数目。这个参数有三种形式。

- n 准确地读取 n 个值。执行完相应的语句后，array 将是一个包含有 n 个值的列向量
- Inf 读取文件中所有值。执行完相应的语句后，array 将是一个列向量，包含有从文件所有值。
- [n，m]从文件中精确定地读取 n×m 个值。array 是一个 n×m 的数组。

如果 fread 到达文件的结尾，而输入流没有足够的位数写满指定精度的数组元素，fread 就会用最后一位的数填充，或用 0 填充，直到得到全部的值。如果发生了错误，读取将直接到达最后一位。

参数 precision 和 size 在函数 fread 和函数 fwrite 中有相同的意义。

例 8.1 读写二进制数据

在本例中显示的脚本文件创建了一个含有 10000 个随机数的数组，以只写方式打开一个

自定义文件，用 64 位浮点数格式把这个数据写入磁盘，并关闭文件。程序打开所要读取的文件，并读取数组，得到一个 100×100 的数组。它用来说明二进制 I/O 操作。

```
% Script file: binary_io.m
%
% Purpose:
% To illustrate the use of binary i/o functions.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== ====================
% 12/19/98 S. J. Chapman Original code
%
% Define variables:
% count                    -- Number of values read / written
% fid                      -- File id
% filename                 -- File name
% in_array                 -- Input array
% msg                      -- Open error message
% out_array                -- Output array
% status                   -- Operation status
% Prompt for file name
filename = input('Enter file name: ','s');
% Generate the data array
out_array = randn(1,10000);
% Open the output file for writing.
[fid,msg] = fopen(filename,'w');
% Was the open successful?
if fid > 0
    % Write the output data.
    count = fwrite(fid,out_array,'float64');
    % Tell user
    disp([int2str(count) ' values written...']);
    % Close the file
    status = fclose(fid);
else
    % Output file open failed. Display message.
    disp(msg);
end
% Now try to recover the data. Open the
% file for reading.
[fid,msg] = fopen(filename,'r');
% Was the open successful?
if fid > 0
    % Read the input data.
    [in_array, count] = fread(fid,[100 100],'float64');
    % Tell user
    disp([int2str(count) ' values read...']);
    % Close the file
    status = fclose(fid);
else
    % Input file open failed. Display message.
    disp(msg);
end
```

当这个程序运行时，结果如下

```
>> binary_io
Enter file name: testfile
10000 values written...
```

10000 values read...

在当前目录下，有一个 80000 字节的文件 testfile 被创建，这个文件之所以占 80000 个字节，是因为它含有 10000 个 64 位的值，每一个值占 8 个字节。

测试 8.1

本测试提供了一个快速的检查方式，看你是否掌握了 8.1 到 8.5 的基本内容。如果你对本测试有疑问，你可以重读 8.1 到 8.5，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

1. 为什么函数 textread 尤其用于读取由其他的语言编写的数据。
2. 用 mat 存储数据的优缺点？
3. 哪些 **MATLAB** 函数用于找开和关闭文件？打开一个二进制文件和打开一个文本文件有什么不同？
4. 编写 **MATLAB** 语言找开一个已存在的文件 myinput.dat，把新增加的内容添加到最后一行。
5. 编写 **MATLAB** 语句，需要以只读模式打开一个无格式的二进制输入文件 input.dat。检测文件是否存在，如果不存在，它就产生一个合适的错误信息。

看第 6 题和第 7 题，判断 **MATLAB** 语句是否正确。如果有错误，指出错在那里。

6.
```
fid = fopen('file1', 'rt');
array = fread(fid, Inf)
fclose(fid);
```
7.
```
fid = fopen('file1', 'w');
x = 1:10;
count = fwrite(fid, x);
fclose(fid);
fid = fopen('file1','r');
array = fread(fid, [2 Inf])
fclose(fid);
```

8.6 格式化 I/O 函数

在本节中，我们向大家介绍格式化 I/O 函数。

8.6.1 fprint 函数

函数 fprint 把以户自定义格式编写的格式化数据写入一个文件。它的形式为

count = fprintf(fid, format, val1, val2, ...)

fprintf(format, val1, val2, ...)

其中 fid 是我们要写入数据那个文件的文件标识，format 是控制数据显示的字符串。如果 fid 丢失，数据将写入到标准输出设备（命令窗口）。这些格式已经在第二章介绍过。

格式（format）字符串指定队列长度，小数精度，域宽和输出格式的其他方面。它包括文字数字字符（%）和字符序列（用于指定输出数据显示的精确格式）。一个典型的数据输出格式字符串图 8.1 所示。字符%总是标志着格式化字符串的开始，在字符%之后，这字符串应包括一个标识（flag），一个域宽，一个精度指定符和一个转换指定符。字符%，转换指定符一般会要求出在任何格式中，而标识，域宽，精度指定符是可选的。

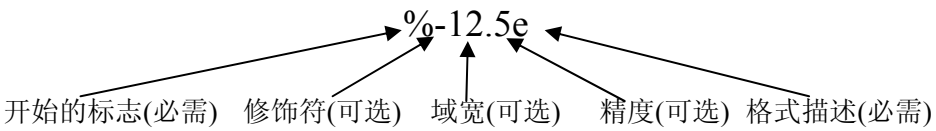


图 8.1 数据输出格式字符串

可能的转换指定符被列在表 8.7 中，可能的修改符（标识）被列在了表 8.8 中。如果我们用格式化字符串指定域宽和精度，那么小数点前的数就是域宽，域宽是所要显示的数所占的字符数。小数点后的数是精度，是指小数点后应保留的位数。

除了普通的字符和格式字符，还有转义字符常用在格式化字符串。这些特殊的字符被列在了表 8.9 中。

表 8.7 函数 fprintf 的格式转换指定符

指定符	描述
%c	单个字符
%d	十进制表示（有符号的）
%e	科学记数法（用到小写的 e，例 3.1416e+00）
%E	科学记数法（用到大写的 e，例 3.1416E+00）
%f	固定点显示
%g	%e 和%f 中的复杂形式，多余的零将会被舍去
%G	与%g 类似，只不过要用到大写的 E
%o	八进制表示（无符号的）
%s	字符串
%u	十进制（无符号的）
%h	用十六进制表示（用小写字母 a- f 表示）
%H	用十六进制表示（用大写字母 A- F 表示）

表 8.8 格式标识（修改符）

标识（修改符）	描述
负号(-)	数据在域中左对齐，如果没有这个符号默认为右对齐
+	输出时数据带有正负号
0	如果数据的位数不够，用零填充前面的数

表 8.9 格式字符串的转义字符

转义序列	描述
\n	换行
\t	水平制表
\b	退后一格
\r	回车符，使屏幕光标移到当前行开关，下移到下一行
\f	跳页符号
\\	打印一个普通反斜杠
\'or'	打印一个省略号或单一引证
%%	打印一个百分号（%）

8.6.2 格式转换指定符的理解

理解大量的格式指定符最好的方法是用例子，现在我们看一些例子以及它们的结果。

8.6.2.1 情况 1：显示十进制整数数据

显示十进制整数数据要用到%d 格式转换指定符。如果需要的话，d 可能出现在标识（flag），域宽和精度指定符之前。如果有用的话，精度指定符可以指定要显示的数据的最小数字个数，如果没有足够多的数字，那么 **MATLAB** 将在这个数之前添加 0。

函数	结果	评论
fprintf('%d\n',123)	----|----| 123	按需要字符的个数，显示这个数据。例如数 123，需要三个字符
fprintf('%6d\n',123)	----|----| 123	用 6 字符域宽显示数字。在这个域中的数是右对齐的。
fprintf('%6.4d\n',123)	----|----| 0123	用 6 字符域宽显示数字，最少也要用 4 字符域宽。在这个域中的数是右对齐的。
fprintf('%-6.4d\n',123)	----|----| 0123	用 6 字符域宽显示数字。最小也要用到 4 字符域宽，在这个域中的数是左对齐的。
fprintf('%+6.4d\n',123)	----|----| +0123	用 6 字符域宽显示数字，最少也要用到 4 字符域宽，加上一个正/负号。在这个域中的数是右对齐的。

如果用格式指定符%d 显示一个非十进制数，这个指定符将会被忽略，这个数将会以科学计算法格式显示。例如

fprintf('%6d\n',123.4)

将产生结果 1.234000e+002。

8.6.2.2 情况 2：显示浮点数数据

浮点数数据的显示要用到%e，%f。，%g 格式转换指符。如果需要的话，这些格式转换指符可能出现在标识（flag），域宽和精度指定符之前。如果指定的域宽太小了，不能显示这个数，则这个域宽是无效的。否则，则应用指定的域宽。

函数	结果	评论
fprintf('%f\n',123.4)	----|----| 123.400000	按需要字符的个数显示这个数据。%f 默认的格式是精确到小数点后 6 位
fprintf('%8.2f\n',123.4)	----|----| 123.40	用 8 字符域宽显示这个数，其中两域宽用于显示小数。
fprintf('%4.2f\n',123.4)	----|----| 123.40	用 6 字符域宽来显示这个数，指定的域宽因太小而被忽略。
fprintf('%10.2e\n',123.4)	----|----| 1. 23e+002	以科学记数法显示数据，域宽为 10,小数点占 2 位。默认这个数是右对齐的。
fprintf('%10.2e\n',123.4)	----|----| 1 .23E+002	与上面相同，只不过 E 为大写

8.6.2.3 情况 3：显示字符数据

字符数据的显示要用到%e，%c 格式转换指符。如果需要的话，这些格式转换指符可能出现在标识（flag），域宽和精度指定符之前。如果指定的域宽太小了，不能显示这个数，则这个域宽是无效的。否则，则应用指定的域宽。

函数	结果	评论
fprintf('%c\n','s')	----|----| s	显示单个字符
fprintf('%s\n','string')	----|----| string	显示一个字符串
fprintf('%8s\n','string')	----|----| string	用 8 字符串域宽显示字符串，默认是右对齐格式
fprintf('%-8s\n','string')	----|----| string	用 8 字符串域宽显示字符串，这个字符串是左对齐的

8.6.3 如何使用格式字符串

函数 fprintf 包括一个格式字符串（在要打印出的 0 或更多的值之后）。当函数 fprintf 执行时，函数 fprintf 的输出参数列表将会按格式字符串的指示输出。这个函数从从变量的左端和格式字符的左端开始执行，并从左向右扫描，输出列表的第一个值与格式字符串中第一个格式输出符联合，等等。在输出参数列表中的值必须是相同的类型，格式必须与对应的格式描述符相对应，否则的话，意外的结果将会产生。例如，假设我们要用%c 或%d 描述符显示浮点数 123。4，这个描述符将会全部被忽略，这个数将会以科学记数的方式打印出来。

常见编程错误

保证 fprintf 函数中的数据类型与格式字符串中的格式转换指定符的类型要一一对应，否则将会产生意料之外的结果。

程序从左向右读取函数 fprint 中的变量列表，它也从左向右读取相应的格式字符串。按照下面的规则，程序扫描格式字符串

1. 按从左向右的顺序扫描格式字符串。

格式字符串中的第一个转换指定符与 fprint 函数输出参数列表中的第一个值相结合，依此类推。每一个格式转换指定符的类型与输出数据类型必须相同。在下面的例子中，指示符%d 与变量 a 联合，%f 与变量 b 结合，%s 与变量 c 相结合。注意指定符类型必须与数据类型相匹配。

```
a = 10; b = pi; c = 'Hello';
fprintf('Output: %d %f %s\n', a, b, c);
```

2. 在函数 fprintf 运行完所有的变量之前，如果扫描还未到达格式字符串的结尾，程序再次从头开始扫描格式字符串。例如，语句

```
a = [10 20 30 40];
fprintf('Output = %4d %4d\n',a);
```

将会产生输出

```
Output = 10 20
Output = 30 40
```

在打印完 a(2)后，函数到达格式字符串的结尾，它将会回字符串的开始打印 a(3)，a(4)

3. 如果函数 fprintf 在到达格式字符结束之前运行完所有的变量，格式字符串的应用停止在第一个格式指定符，而没有对应的变量，或者停止在格式字符串的末端。例如语句

```
a = 10; b = 15; c = 20;
fprintf('Output = %4d\nOutput = %4.1f\n', a, b, c);
```

将产生输出

```
Output =      10
Output = 15.0
Output =      20
Output = >>
```

格式字符串的应停止在%4.1f，这是它第一次与格式转换指示符不匹配。从另一方面来说，语句

```
voltage = 20;
fprintf('Voltage = %6.2f kv.\n', voltage);
```

将产生输出

```
Voltage =    20.00 kv.
```

因为它与格式转换字符串不匹配，所以格式的应用停止在格式字符串的结尾。

例 8.2 产生一个信息表

产生并打印一个数据表是说明函数 fprintf 函数就用的好方法。下面的脚本文件产生 1 到 10 中的所有整数的平方根，平方，立方，并在一个表中显示数据，并带有合适的表头。

```
% Script file: table.m
%
% Purpose:
% To create a table of square roots, squares, and
% cubes.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== ====================
% 12/20/98 S. J. Chapman Original code
%
% Define variables:
% cube                  -- Cubes
% ii                    -- Index variable
% square                -- Squares
% square_roots          -- Square roots
% out                   -- Output array
% Print the title of the table.
fprintf(' Table of Square Roots, Squares, and Cubes\n\n');
% Print column headings
fprintf(' Number Square Root Square Cube\n');
fprintf(' ====== =========== ====== ====\n');
% Generate the required data
ii = 1:10;
square_root = sqrt(ii);
square = ii.^2;
cube = ii.^3;
% Create the output array
out = [ii' square_root' square' cube'];
% Print the data
for ii = 1:10
    fprintf (' %2d %11.4f %6d %8d\n',out(ii,:));
end
```

程序运行后，产生的结果为

```
>> table
```

Table of Square Roots, Squares, and Cubes

Number	Square Root	Square	Cube
1	1.0000	1	1
2	1.4142	4	8
3	1.7321	9	27
4	2.0000	16	64
5	2.2361	25	125
6	2.4495	36	216
7	2.6458	49	343
8	2.8284	64	512
9	3.0000	81	729
10	3.1623	100	1000

8.6.4 fscanf 函数

函数 fscanf 可以从一个文件中按用户自定义格式读取格式化数据。形式如下：

array = fscanf(fid, format)

[array, count] = fscanf(fid, format, size)

其中 fid 是所要读取的文件的文件标识（fileid），format 是控制如何读取的格式字符串，array 是接受数据的数组，输出参数 count 返回从文件读取的变量的个数。参数 size 指定从文件读取数据的数目。这个函数有以下三个类型。

- n　准确地读取 n 个值。执行完相应的语句后，array 将是一个包含有 n 个值的列向量
- Inf　读取文件中所有值。执行完相应的语句后，array 将是一个列向量，包含有从文件所有值。
- [n,m]　从文件中精确定地读取 n×m 个值。Array 是一个 n×m 的数组。

格式字符串用于指定所要读取数据的格式。它由普通字符和格式转换指定符。函数 fscanf 把文件中的数据与文件字符串的格式转换指定符进行对比。只要两者区配，fscanf 把值进行转换并把它存储在输出数组中。这个过程直到文件结束或读取的文件当数目达到了 size 数组才会结束，无论那一种情况先出现。

如果文件中的数据与格式转换指定符不匹配，fscanf 的操作就会突然中止。

fscanf 格式转换指定符基本上与 fprintf 的格式转换指定符相同。最普通的指定符被总结在表 8.10 中。

为了说明函数 fscanf 的应用，我们将试着读取文件 x.dat，在两行中包含下面的值。

10.00 20.00

30.00 40.00

1. 如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%f');

z 值为$\begin{bmatrix}10\\20\\30\\40\end{bmatrix}$，cout 的值为 4。

2. 如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%f', [2 2]);

z 的值为$\begin{bmatrix}10 & 30\\20 & 40\end{bmatrix}$，cout 的值为 4。

3. 下一步，我们让我们从一文件中读取十进制小数数据。如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%d', Inf);

z 为 10，count 的值为 1。这种情况的发生是因为 10.00 的小数点与格式转义指定符不匹配，函数 fscanf 函数停止在第一次出现不匹配时。

4. 如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%d.%d',[1 Inf]);

z 为行向量[10 0 20 0 30 0 40 0]，count 的值为 8。这种情况的发生是因为小数点与格式转义指定符匹配，小数点前后的数可以看作独立的整数。

5. 现在让我们文件中读取一个单独的字符，如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%c');

变量 z 是一个包含文件中每一个字符的行向量，包括所有的空格和换行符!变量 count 等于文件中字符的个数。

6. 最后，让我们试着从文件中读取字符串，如果用下面的语句读取一文件

[z, count] = fscanf(fid, '%s');

z 是一个行向量，包括 20 个字符 10.0020.0030.0040.00，count 为 4。这种结果的产生是因为字符串指定符忽略空白字符，这个函数在这个文件中发现 4 个独立的字符串。

表 8.10 fscanf 的格式转化指定符

指定符	描述
%c	读取一单个字符。这个字符读取的是任意类型的字符，包括空格，换行符等
%Nc	读取 N 个字符
%d	读取一小数（忽略空格）
%e %f %g	读取一浮点数（忽略空格）
%i	读取一有符号数（忽略空格）
%a	读取一字符串。字符串可以被空格或其他类似于换行符的特殊符号隔开

8.6.5 fgetl 函数

函数 fgetl 从一文件中把下一行（最后一行除外）当作字符串来读取。它的形式为

line = fgetl(fid)

如果 fid 是我们所要读取的文件的标识（file id）。line 是接受数据的字符数组。如果函数 fgetl 遇到文件的结尾，line 的值为-1。

8.6.5 fgets 函数

函数 fgets 从一文件中把下一行（包括最后一行）当作字符串来读取。它的形式为

line = fgets(fid)

如果 fid 是我们所要读取的文件的标识（file id）。line 是接受数据的字符数组。如果函数 fgets 遇到文件的结尾，line 的值为-1。

8.7 格式化和二进制 I/O 函数的比较

格式化 I/O 数据产生格式化文件。格式化文件夹由可组织字符，数字等组成，并以 ASCII 文本格式。这类数据很容易辨认，因为当我们把在显示器上把他显示出来，或在打印机上打印出来。但是，为了应用格式化文件中的数据，**MATLAB** 程序必须把文件中的字符转化为计算机可以直接应用的中间数据格式。格式转换指定符为这次转换提供了指令。

格式化文件有以下优点：我们可以清楚地看到文件包括什么类型的数据。它还可以非常容易在不同类型的程序间进行转换。但是也有缺点程序必须作大量的工作，对文件中的字符串进行转换，转换成相应的计算机可以直接应用的中间数据格式。如果我们读取数据到其他的 **MATLAB** 程序，所有的这些工作都会造成效率浪费。而且一个数的计算机可以直接应用的中间数据格式要比格式化文件中的数据要大得多。例如，一个 64 位浮点数的中间数据格式需要 8 个字节的内存。而格式化文件中的字符串表达形为±d.ddddddddddddddEee，它将

需要 21 个字节。所以用字符格式存储数据是低效的且浪费磁盘空间。

无格式化文件（二进制文件）克服上面的缺点，它其中的数据无需转化，就可以把内存中的数据写入磁盘。因为没有转化发生，计算机就没有时间浪费在格式化数据上。在 **MATLAB** 中，二进制 I/O 操作要比格式化 I/O 操作快得多，因为它中间没有转化。进一步说，数据占用的磁盘空间将更小。从另一方面来说，无格式的数据不能进行人工检查和人工翻译。还有，它不能移植到不同类型的计算机，因为不同类型的计算机有不同中间过程来表示整数或浮点数。

表 8.11 显示了格式化文件与无格式化文件的区别。在一般情况下，格式化文件，对于那些必须进行人工检查的数据，或对于那些必须在不同的计算机上运行的数据，是最好的选择。对于那些不需要进行人工检查的数据且在相同类型的计算机创建并运行的数据，存储最好用无格式文件。在这些环境下，无格式文件运算要快得多，占用的磁盘空间更小。

好的编程习惯

对于那些必须进行人工检查的数据，或对于那些必须在不同的计算机上运行的数据，用格式化文件创建数据。对于那些不需要进行人工检查的数据且在相同类型的计算机创建并运行的数据，用无格式文件创建数据，当 I/O 速度缓慢时，用格式化文件创建数组。

表 8.11 格式化文件和无格式化文件的比较

格式化文件	无格式化文件
能在输出设备显示数据	不能在输出设备显示数据
能在不同的计算机上很容易地进行移植	不能在不同的计算机上很容易地进行移植
相对地，需要大量的磁盘空间	相对地，需要较少的磁盘空间
慢：需要大量的计算时间	快：需要较少的计算时间
在进行格式化的过程中，产生截断误差或四舍五入错误	不会产生截断误差或四舍五入错误

例 8.3 格式化和二进制 I/O 文件的比较

在这个例子中的程序比较了用格式化和二进制 I/O 操作读写一个含 10000 个元素数组所花的时间。注意每一个操作运行 10 次求平均值。

```
% Script file: compare.m
%
% Purpose:
% To compare binary and formatted I/O operations.
% This program generates an array of 10,000 random
% values and writes it to disk both as a binary and
% as a formatted file.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== =====================
% 12/19/98 S. J. Chapman Original code
%
% Define variables:
% count          -- Number of values read / written
% fid            -- File id
% in_array       -- Input array
% msg            -- Open error message
% out_array      -- Output array
% status         -- Operation status
% time           -- Elapsed time in seconds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Generate the data array.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
out_array = randn(1,10000);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, time the binary output operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reset timer
tic;
% Loop for 10 times
for ii = 1:10
    % Open the binary output file for writing.
    [fid,msg] = fopen('unformatted.dat','w');
    % Write the data
    count = fwrite(fid,out_array,'float64');
    % Close the file
    status = fclose(fid);
end
% Get the average time
time = toc / 10;
fprintf ('Write time for unformatted file = %6.3f\n',time);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next, time the formatted output operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reset timer
tic;
% Loop for 10 times
for ii = 1:10
    % Open the formatted output file for writing.
    [fid,msg] = fopen('formatted.dat','wt');
    % Write the data
    count = fprintf(fid,'%23.15e\n',out_array);
    % Close the file
    status = fclose(fid);
end
% Get the average time
time = toc / 10;
fprintf ('Write time for formatted file = %6.3f\n',time);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Time the binary input operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reset timer
tic;
% Loop for 10 times
for ii = 1:10
    % Open the binary file for reading.
    [fid,msg] = fopen('unformatted.dat','r');
    % Read the data
    [in_array, count] = fread(fid,Inf,'float64');
    % Close the file
    status = fclose(fid);
end
% Get the average time
time = toc / 10;
fprintf ('Read time for unformatted file = %6.3f\n',time);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Time the formatted input operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Reset timer
tic;
% Loop for 10 times
for ii = 1:10
    % Open the formatted file for reading.
    [fid,msg] = fopen('formatted.dat','rt');
    % Read the data
    [in_array, count] = fscanf(fid,'%f',Inf);
    % Close the file
    status = fclose(fid);
end
% Get the average time
time = toc / 10;
fprintf ('Read time for formatted file = %6.3f\n',time)
```

当程序在奔腾 733MHz 机器上运行，操作系统为 windowsNT2000 专业版，得到的结果为

```
>> compare
Write time for unformatted file =    0.002
Write time for formatted file =    0.132
Read time for unformatted file =    0.002
Read time for formatted file =    0.157
```

写入磁盘的文件如下所示：

```
D:\MATLAB\work\chap8>dir
 驱动器 D 中的卷是 SINSTALL
 卷的序列号是 C33D-3233

 D:\MATLAB\work\chap8 的目录

2008-01-17   18:58      <DIR>              .
2008-01-17   18:58      <DIR>              ..
2008-01-17   18:51               250,000 formatted.dat
2008-01-17   18:51                80,000 unformatted.dat
               2 个文件            330,000 字节
               2 个目录   2,495,549,440 可用字节
```

注意写入格式化文件数据所需的时间是无格式文件的 60 倍，记取时间是无格式文件的 75 倍。还有格式化文件的大小是无格式文件的 3 倍。得到的结果是非常清楚的，除非你真得需要格式化数据，否则二进制 I/O 操作是 **MATLAB** 中存储数据的一个非常好的方法。

测试 8.2

本测试提供了一个快速的检查方式，看你是否掌握了 8.1 到 8.5 的基本内容。如果你对本测试有疑问，你可以重读 8.1 到 8.5，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

1. 格式化和二进制 I/O 操作的区别是什么？
2. 什么时侯我们应当用格式化 I/O 操作？什么时侯我们应当有二进制 I/O 操作？
3. 编写 **MATLAB** 语句创建一个表，由 x 的正弦值和余弦值（x = 0, 0.1pi, ..., pi），在表上有标题和标签。

看第 4 题和第 5 题，判断 **MATLAB** 语句是否正确。如果有错误，指出错在那里。

4.
```
a = 2*pi;
b = 6;
c = 'hello';
fprintf(fid, '%s %d %g\n',a, b, c);
```

```
5.  data1 = 1:20;
    data2 = 1:20;
    fid = fopen('xxx', 'w+');
    fwrite(fid, data1);
    fprintf(fid, '%g\n', data2);
```

8.8 文件位置和状态函数

正如我们前面所陈述的，**MATLAB** 文件是连续的——它们从第一条记录开始一直读到最后一条记录。但是，有时在一个程序中，我们需要多次调用一段数据或整个文件。在一个连续文件中，我们如何跳过无用的数据呢？

在打开文件之前，**MATLAB** 函数 exist 用于判断这个文件是否存在。一旦一个文件打开，我们就可以用函数 feof 和 ftell 判断当前数据在文件中的位置。还用两个函数帮助我们在文件中移动：frewind 和 fseek。

最后，当程序发生 I/O 错误时，**MATLAB** 函数 ferror 将会对这个错误进行详尽的描述。我们现在将向大家详细的介绍这 6 个函数，我们先看一下 ferror，因为它可以应用其他的函数中。

8.8.1 exist 函数

exist 函数用来检测工作区中的变量，内建函数或 **MATLAB** 搜索路径中的文件是否存在。它的形式如下

```
ident = exist('item');
ident = exist('item', 'kind');
```

如果“item”存在，函数就根据它的类型返回一个值。可能的结果被显示在表 8.12 中。

函数 exist 指定所要搜索的条目（item）的类型。它的合法类型为“var”，“file”，“builtin”和“dir”。

函数 exist 是非常重要的，因为我们可以利用它判断一个文件否存在。当文件被打开时，fopen 函数中权限运算符“w”和“w+”会删除文件已有的一个文件。在程序员允许 fopen 函数删除一个文件时，它必须征得用户的同意。

表 8.12 由函数 exist 的返回值

值	意义
0	没有发现条目
1	条目为当前工作区的一个变量
2	条目为 m 文件或未知类型的文件
3	条目是一个 MEX 文件
4	条目是一个 MDL 文件
5	条目是一个内建函数
6	条目是一个 p 代码文件
7	条目是一个目录

例 8.4 打开一个输出文件

这个程序从用户那里得到输出文件名，并检查它是否存在。如果存在，就询问用户是要把用新数据覆盖这个文件，还是要把新的数据添加到这个文件中。如果这个文件不存在，那么这个程序就会很容易地打开输出文件。

```
% Script file: output.m
```

```
%
% Purpose:
% To demonstrate opening an output file properly.
% This program checks for the existence of an output
% file. If it exists, the program checks to see if
% the old file should be deleted, or if the new data
% should be appended to the old file.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== ====================
% 11/29/98 S. J. Chapman Original code
%
% Define variables:
% fid -- File id
% out_filename -- Output file name
% yn -- Yes/No response
% Get the output file name.
out_filename = input('Enter output filename: ','s');
% Check to see if the file exists.
if exist(out_filename,'file')
      % The file exists
      disp('Output file already exists.');
      yn = input('Keep existing file? (y/n) ','s');
      if yn == 'n'
            fid = fopen(out_filename,'wt');
      else
            fid = fopen(out_filename,'at');
      end
else
      % File doesn't exist
      fid = fopen(out_filename,'wt');
end
% Output data
fprintf(fid,'%s\n',date);
% Close file
fclose(fid);
```

当这个程序执行后，产生的结果为

```
>> output
Enter output filename: xxx
>> type xxx
17-Jan-2008
>> output
Enter output filename: xxx
Output file already exists.
Keep existing file? (y/n) y
>> type xxx
17-Jan-2008
17-Jan-2008
>> output
Enter output filename: xxx
Output file already exists.
Keep existing file? (y/n) n
>> type xxx

17-Jan-2008
```

三种不同的情况均产生了正确的结果。

好的编程习惯
未经用户同意，不要用新数据覆盖原用的文件。

8.8.2 函数 ferror

在 **MATLAB** 的 I/O 系统中有许多的中间数据变量，包括一些专门提示与每一个打开文件相关的错误的变量。每进行一次 I/O 操作，这些错误提示就会被更新一次。函数 ferror 得到这些错误提示变量，并把它转化为易于理解的字符信息。

```
message = ferror(fid)
message = ferror(fid, 'clear')
[message, errnum] = ferror(fid)
```

这个函数会返回与 fid 相对应文件的大部分错误信息。它能在 I/O 操作进行后，随时被调用，用来得到错误的详细描述。如果这个文件被成功调用，产生的信息为“…”，错误数为 0。

对于特殊的文件标识，参数“clear”用于清除错误提示。

8.8.3 函数 foef

函数 feof 用于检测当前文件的位置是否是文件的结尾。它的形式如下

```
eofstat = feof(fid)
```

如果是文件的结尾，那么函数返回 1，否则返回 0。

8.8.4 函数 ftell

函数 ftell 返回 fid 对应的文件指针读/写的位置。这个位置是一个非负整数，以 byte 为单位，从文件的开头开始计数。返回值-1 代表位置询问不成功。如果这种情况发生了，我们利用 ferror 得知为什么询问不成功。函数的形式如下：

```
position = ftell(fid)
```

8.8.5 函数 frewind

函数 frewind 允许程序把文件指针复位到文件的开头，形式如下

```
frewind(fid)
```

这个函数不返回任何状态信息。

8.8.6 函数 fseek

函数 fseek 允许程序把文件指针指向文件中任意的一个位置。函数形式如下

```
status = fseek(fid, offset, origin)
```

函数用 offsett 和 origin 来重设 fid 对应文件的文件指针。offset 以字节为单位，带有一个正数，用于指向文件的结尾，带有一个负数，用于指向文件的开头。origin 是一个字符串，取值为下面三个中的一个。

- “bof” 文件的开始位置
- “cof” 指针中的当前位置
- “eof” 文件的结束位置

如果这个操作成功，status 的值为 0，如果操作失败 status 为-1。如果 status 为-1，用函数 ferror 判断错误出现在那里。

举一个例子，用于说明 fseek 和 ferror 的联合应用，考虑下面的语句。

```
[fid, msg] = fopen('x', 'r');
status = fseek(fid, -10, 'bof');
if status ~= 0
    msg = ferror(fid);
    disp(msg);
end
```

这些命令打开了一个文件，并把文件指针设置在文件开始之前的 10 个字节上面。这是不可能，所以 fseek 将会返回一个-1，用 ferror 得到对应的错误信息。当这些语句被编译时，产生下面的错误信息。

Offset is bad - before beginning-of-file.

例 8.5

用一系列带有噪声的值拟合一条直线

在例 4.7 中，我们学习了用最小二乘法拟合直线

$$y = mx + b \qquad (8.1)$$

确定待定系数 m 和 b 的标准方法为最小二乘法。之所以称为最小二乘法，是因为根据偏差的平方和为最小的条件来选择常数 m 和 b 的。公式如下:

$$m = \frac{(\Sigma xy)-(\Sigma x)\bar{y}}{(\Sigma x^2)-(\Sigma x)\bar{x}} \qquad (8.2)$$

$$b = \bar{y}-m\bar{x} \qquad (8.3)$$

其中，$\sum x$代表所有测量值x之和，$\sum y$代表所有测量值y之和，$\sum xy$代表所有对应的x与y的乘积之和，$\bar{x}$ 代表测值量x的数学期望，$\bar{y}$ 代表测值量y的数学期望。

编写一个程序，用最小二乘法计算出 m 和 b。其中有一系列含有噪声的数据(x,y)存储在输入数据文件中。

答案

1. 陈述问题

用最小二乘法计算直线的截距 b 和斜率 m，输入值任意数目的（x，y）坐标对。输入值（x，y）被存储在用户自定义输入文件中。

2. 定义输入输出值

程序所需的输入值是（x，y）坐标对，x，y 均为实数，每一个点都存储在磁盘文件独立的行中，输出是用最小二乘法计算出的直线的截距 b 和斜率 m。

3. 算法描述

本程序可以分为以下四大步骤

Get the name of the input file and open it
Accumulate the input statistics
Calculate the slope and intercept
Write out the slope and intercept

这个程序的第一大步得到输入文件的名字并打开这个文件。为了做到这一点，我们必须提示用户键入输入文件名。在文件打开之后，我们必须检查文件是否被成功打开。下一步，我们必须从文件中读取数据，并记录下这些数据，并计算$\sum x$，$\sum y$，$\sum x^2$和$\sum xy$。这些步骤的伪代码如下：

```
Initialize n, sum_x, sum_x2, sum_y, and sum_xy to 0
Prompt user for input file name
Open file 'filename'
Check for error on open
if no error
    READ x, y from file 'filename'
    while not at end-of-file
        n ← n + 1
        sum_x ← sum_x + x
        sum_y ← sum_y + y
        sum_x2 ← sum_x2 + x^2
        sum_xy ← sum_xy + x*y
        READ x, y from file 'filename'
```

```
    end
    (further processing)
end
```

下一步我们要用最小二乘法计算直线的截距 *b* 和斜率 *m*。我们可以根据公式（8.2）（8.3）得到。

```
    x_bar ← sum_x / n
    y_bar ← sum_y / n
    slop ← (sum_xy - sum_x*y_bar) / (sum_x2 - sum_x*x_bar)
    y_int ← y_bar - slop * x_bar
```

最后，我们要写出结果。

```
    Write out slope 'slope' and intercept 'y_int'.
```

4. 转化为 **MATLAB** 语言

```
    % Script file: lsqfit.m
    %
    % Purpose:
    % To perform a least-squares fit of an input data set
    % to a straight line, and print out the resulting slope
    % and intercept values. The input data for this fit
    % comes from a user-specified input data file.
    %
    % Record of revisions:
    % Date Programmer Description of change
    % ==== ========== ====================
    % 12/20/98 S. J. Chapman Original code
    %
    % Define variables:
    % count -- number of values read
    % filename -- Input file name
    % fid -- File id
    % msg -- Open error message
    % n -- Number of input data pairs (x,y)
    % slope -- Slope of the line
    % sum_x -- Sum of all input X values
    % sum_x2 -- Sum of all input X values squared
    % sum_xy -- Sum of all input X*Y values
    % sum_y -- Sum of all input Y values
    % x -- An input X value
    % x_bar -- Average X value
    % y -- An input Y value
    % y_bar -- Average Y value
    % y_int -- Y-axis intercept of the line
    % Initialize sums
    n = 0; sum_x = 0; sum_y = 0; sum_x2 = 0; sum_xy = 0;
    % Prompt user and get the name of the input file.
    disp('This program performs a least-squares fit of an');
    disp('input data set to a straight line. Enter the name');
    disp('of the file containing the input (x,y) pairs: ' );
    filename = input(' ','s');
    % Open the input file
    [fid,msg] = fopen(filename,'rt');
    % Check to see if the open failed.
    if fid < 0
        % There was an error--tell user.
        disp(msg);
    else
```

```
        % File opened successfully. Read the (x,y) pairs from
        % the input file. Get first (x,y) pair before the
        % loop starts.
        [in,count] = fscanf(fid,'%g %g',2);
        while ~feof(fid)
            x = in(1);
            y = in(2);
            n = n + 1; %
            sum_x = sum_x + x; % Calculate
            sum_y = sum_y + y; % statistics
            sum_x2 = sum_x2 + x.^2; %
            sum_xy = sum_xy + x * y; %
            % Get next (x,y) pair
            [in,count] = fscanf(fid,'%f',[1 2]);
        end
        % Now calculate the slope and intercept.
        x_bar = sum_x / n;
        y_bar = sum_y / n;
        slope = (sum_xy - sum_x*y_bar) / (sum_x2 - sum_x*x_bar);
        y_int = y_bar - slope * x_bar;
        % Tell user.
        fprintf('Regression coefficients for the least-squares line:\n');
        fprintf(' Slope (m) = %12.3f\n',slope);
        fprintf(' Intercept (b) = %12.3f\n',y_int);
        fprintf(' No of points = %12d\n',n);
    end
```

5. 检测程序

为了检测这个程序，我们可以用一些简单的数据集合进行检测。例如，如果输入数据所对应的点都在同一条直线，那么产生的斜率和截距必定是那条直线的斜率和截距。这组数据为

```
1.1   1.1
2.2   2.2
3.3   3.3
4.4   4.4
5.5   5.5
6.6   6.6
7.7   7.7
```

它的斜率和截距分别为 1.0 和 0.0。我们把这些值写入 input1 文件，运行这个程序，结果如下:

```
>> lsqfit
This program performs a least-squares fit of an
input data set to a straight line. Enter the name
of the file containing the input (x,y) pairs:
 input1.txt
Regression coefficients for the least-squares line:
 Slope (m) =              1.000
 Intercept (b) =          0.000
 No of points =               7
```

我们在这些测量值上加入一些噪声，数据变为

```
1.1   1.01
2.2   2.30
3.3   3.05
4.4   4.28
5.5   5.75
6.6   6.48
7.7   7.84
```

如果把这些值写入 input2 文件，运行程序，结果如下

```
>> lsqfit
This program performs a least-squares fit of an
input data set to a straight line. Enter the name
of the file containing the input (x,y) pairs:
  input2.txt
Regression coefficients for the least-squares line:
  Slope (m) =              1.024
  Intercept (b) =          -0.120
  No of points =                7
```

我们用手动计算很容易就能得到上面两个程序的正确结果。第二个程序图象如图 8.2 所示。

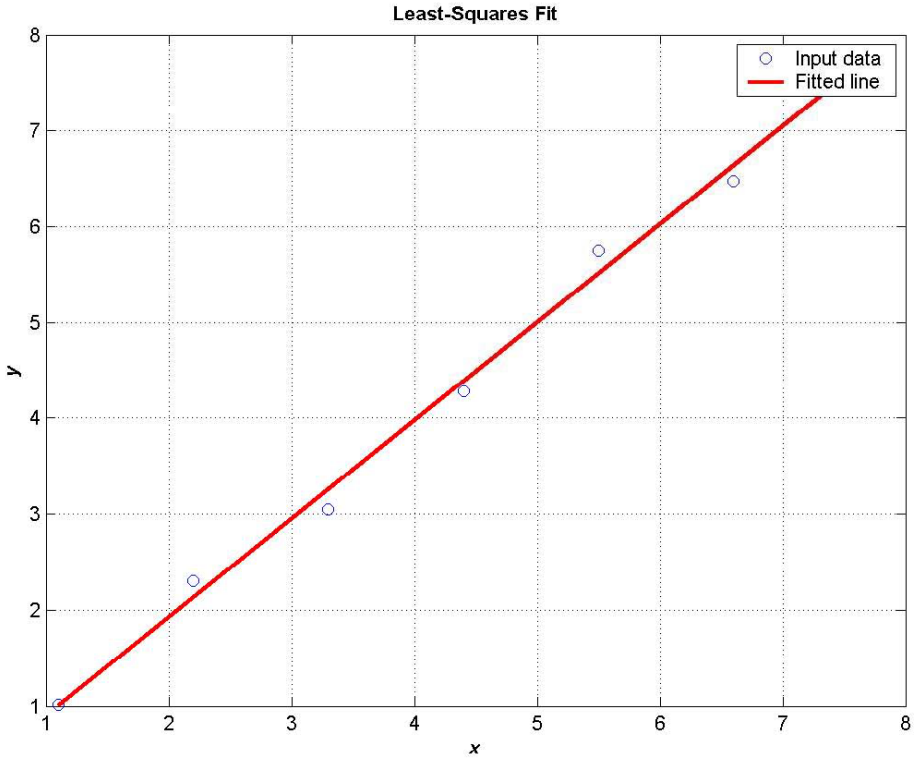


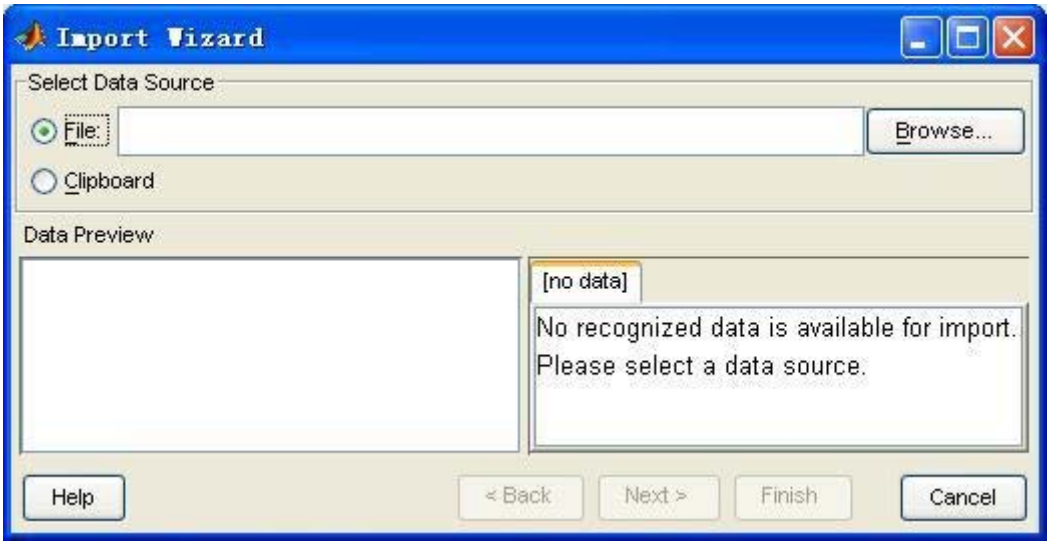
图 8.2

8.9 函数 uiimport

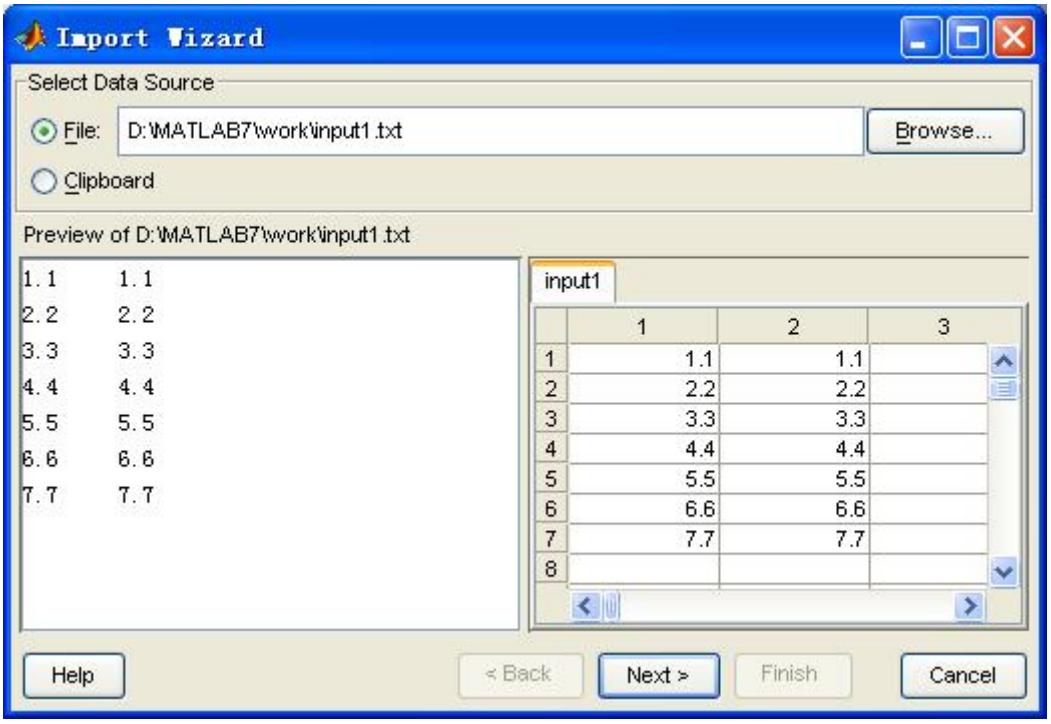
函数是一种基于 GUI 的方法从一个文件或从剪贴板中获取数据。这个命令的形式如下

```
uiimport
structure = uiimport;
```

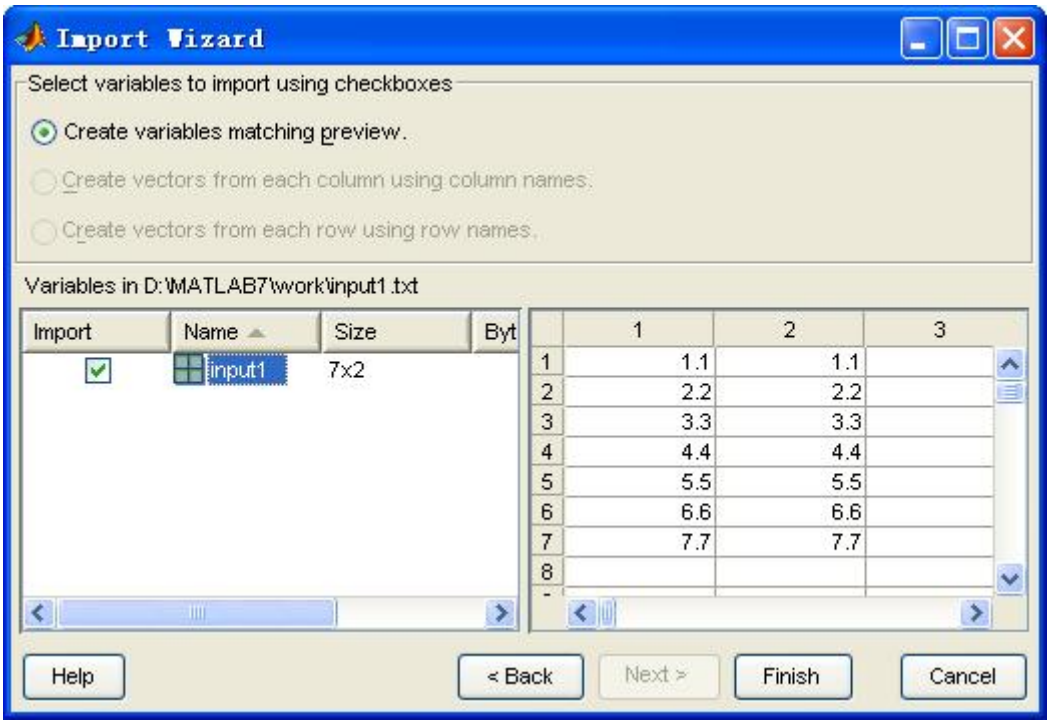
在第一种情况下，获取的数据可以直接插入当前的工作区。在第二种情况下，数据可以转化为一个结构数组。并保存在变量 structure 中。


Import Wizard
Select Data Source
File:
Browse...
Clipboard
Data Preview
[no data]
No recognized data is available for import.
Please select a data source.
Help
< Back
Next >
Finish
Cancel

(a)


Import Wizard
Select Data Source
File:
D:\MATLAB7\work\input1.txt
Browse...
Clipboard
Preview of D:\MATLAB7\work\input1.txt
1.1 1.1
2.2 2.2
3.3 3.3
4.4 4.4
5.5 5.5
6.6 6.6
7.7 7.7
input1
1 2 3
1 1.1 1.1
2 2.2 2.2
3 3.3 3.3
4 4.4 4.4
5 5.5 5.5
6 6.6 6.6
7 7.7 7.7
8
Help
< Back
Next >
Finish
Cancel

(b)



(c)

图 8.5 uiimport 的应用（a）导入向导 (b)选择一个数据文件后，创建一个或多个数组，它的内容可以被检测。（c）一旦一个文件被加载后，用户可以选择哪些数组要导入到 **MATLAB** 中。

当我们在命令窗口中键入命令 uiimport，importwizard 将会以窗口的形式显示出来。用户可以选择获取数据的文件或剪贴板。它支持许多的不同格式，它可以读取任意应用程序保存在剪贴板是的数据。当你想要把数据导入 **MATLAB** 并对其进行分析时，它的灵活性将会非常地有用。

8.10 总结

8.10.1 好的编程习惯总结

1. 除非我们必须与非 **MATLAB** 程序进行数据交换，存储和加载文件时，都应用 mat 文件格式。这种格式是高效的且移植性强，它保存了所有 **MATLAB** 数据类型的细节。

2. 在使用 fopen 语句时，一定要注意指定合适的权限，这取决于你是要读取数据，还是要写入数据。好的编程习惯可以帮助你避免（类似于覆盖的）错误。

3. 在文件打开操作后检查它的状态以确保它被成功打开。如果文件打开失败，提示用户解决方法。

4. 对于那些必须进行人工检查的数据，或对于那些必须在不同的计算机上运行的数据，用格式化文件创建数据。对于那些不需要进行人工检查的数据且在相同类型的计算机创建并运行的数据，用无格式文件创建数据，当 I/O 速度缓慢时，用格式化文件创建数组。

5. 未经用户同意，不要用新数据覆盖原用的文件。

8.10.2 MATLAB 总结

函数与命令

MATLAB 输入/输出语句

类别	函数	描述
加载/保存工作区	load	加载工作区
	save	保存工作区
文件打开/关闭	fopen	打开文件
	fclose	关闭文件
二进制 I/O	fread	从文件中读取二进制数据
	fwrite	把二进制数据写入文件
格式化 I/O	fscanf	从文件中读取格式化数据
	fprintf	把格式化数据写入文件
	fgetl	读取文件的一行，忽略换行符
	fgets	读取文件的一行，不忽略换行符
文件位置、状态	delete	删除文件
	exist	检查文件是否存在
	ferror	所需文件的 I/O 错误情况
	feof	检测文件的结尾
	fseek	设置文件的位置
	ftell	检查文件的位置
	frewind	回溯文件
临时情况	tempdir	得到临时目录名
	tempname	得到临时文件名

8.11 练习

8.1

二进制 I/O 与格式化 I/O 有什么区别？哪一个函数用于显示 I/O 操作的类型？

8.2

对数表。编写一个程序，能够产生 x 以十为底的对数，x 的取值范围为 x1:0.1:10，表应该在新的一页开始，表应该有标题以及用来说明行与列的标签。这个表如下所示

	x.0	x.1	x.2	x.3	x.4	x.5	x.6	x.7	x.8	x.9
1.0	0.000	0.041	0.079	0.114	...					
2.0	0.301	0.322	0.342	0.362	...					
3.0	...									
4.0	...									
5.0	...									
6.0	...									
7.0	...									
8.0	...									

```
 9.0      ...
10.0      ...
```

8.3

编写一个 **MATLAB** 程序，从一天的开始，每一秒读取一次时间（这个值应在 0.0 到 86400.0 之间）并打印出相应的时间 HH:MM:SS(以 24 时进制计时)用合适的格式转换符，把前面的零存储在 MM 和 SS 域。确保输入的秒数是合法的，如果输入了不合法的数据则提供一个合适的错误信息。

8.4

重力加速度。重力加速度 g 可由下面的公式计算得出

$$g = -G\frac{M}{(R+h)^2} \qquad (8.4)$$

编写一个程序，h 的取值为 0:500:40000（单位千米），计算出相应的重力加速度，打印出一个高度- 加速度表，并包括所有的输出变量。画出这些数据的图象。

8.5

在例 8.5 中的程序说明了格式化 I/O 命令从磁盘中读取（x，y）数据的应用。我们还可以用函数 load -sacii。重新编写程序应用函数 load -sacii。检测你重新编写的程序，确认它产生一个与例 8.5 相同的结果。

8.6

用函数 textread 重新编写例 8.5 中的程序。比较 8.4，8.5 和 8.6，看那一个函数更容易使用。

8.7

编写程序，从用户自定义的输入文件中读取任意数目的实数，对这些数进行四舍五入，然后把得到的整数值写到用户自定义输出文件中。确保输入文件存在，如果它不存在，通知用户并询问是否有其他的输入文件。如果输出文件存在，询问是否删除它。如果不删除，那么提示用另一个输出文件名。

8.8

正弦表和余弦表。编写一程序，产生一个 θ 正弦|余弦表 θ=0:1:90（单位为度）。程序应当合适的行标和列标。

8.9

本利计算。假设你在银行总共的存款为 P（p 代表本金）。如果银行的年利率为 i%（月利率则是年利率的 12 分之 1），那么存入银行 n 个月后，得到的本金 F 为

$$F = P\,(1 + \frac{i}{1200})^n \qquad (8.5)$$

编写一个程序，读取本金 P 和年利率 i，并计算出五年内每一个月的本金 F，并列出一个表。这个表被写入输出文件“interest”。确保表中的行有合适的标签。

8.10

编写一个程序从输入数据文件中读取一系列的整数，并找出这个文件中的最大值与最小值。并打印出来这两个值，和它所在的行。假设你不知道输入文件中数据的个数。

8.11

平均数。修改练习 4.27 中程序。从一个输入数据文件中读取任意数目的值，并计算出相应的算术平均数，几何平均数，均方根平均数，调和均数。为检测这个程序。输入数据文件中包含下面的数据

1.0, 2.0, 5.0, 4.0, 3.0, 2.1, 4.7, 3.0

8.12

把角度的弧度格式转换为相应的度/分/秒。

编写一程序，从磁盘文件中读取弧度格式的角度，并把它转化为相应的度/分/秒格式。为检测这个程序，磁盘文件中应包含下面的数据

0.0, 1.0, 3.141593, 6.0

8.13

在你的计算机中用其他程序创建一系列数据，例如，word，excel 或记事本。把这些数据复制到 window 的剪贴板上，然后用函数 uiimport 加载这些数据到 **MATLAB** 中。