CHAPTER 4

Input and Output

4.1 Introduction

The fundamental data type for MATLAB is an array. Most of numerical computations for scientific and engineering purposes involve dealing with data in various file formats. Scientific devices and computer programs themselves generate data as files. These files are then read and converted into arrays (mostly). These arrays can be manipulated as per mathematical requirements by the files of matrix algebra. The results generate a new set of arrays. These arrays are further converted into files for visualization.

Using the information in Chapters 2 and 3 (arrays and plotting), you can now formulate physical problems in terms of numerical computations and solve them on a digital computer. This process has some requirements such as:

- The data should be in a digital form (a digital file).
- The computer program should be able to read the file and make arrays from it without errors. If errors occur, a mechanism to check those errors and warning the user should be in place. If possible, a mechanism for correcting them should also be in place.
- The data should be stored as an array in the proper data type and should be displayed on demand in the proper format.

© Sandeep Nagar 2017
S. Nagar, *Introduction to MATLAB for Engineers and Scientists*,
https://doi.org/10.1007/978-1-4842-3189-0_4

- Array operations on data will result in memory usage in terms of reading and writing data on disk. This should be facilitated by the system. Users should be able to check the status of memory as and when required.
- Post-processing tasks include displaying data in various formats—as a printout from a printer, on a terminal, as a graph on a terminal or printer/plotter, etc.
- If a report for a particular experiment has input parameters, processing the data and output as a file or graph will make the user's task easier.

MATLAB has some features for each of these steps. This chapter discusses them in brief.

4.2 Interactive Input from a Keyboard

A user interacts with MATLAB using a keyboard. Keyboards generate ASCII or Unicode strings for specific characters. These are fed into MATLAB, which then interprets them to perform a specific task. For an interactive session with MATLAB during the course of programming, MATLAB offers the functions discussed in the following sections.

4.2.1 input()

`input("Text")` displays the `Text` string at the MATLAB prompt (the default symbol is `>>>`) and waits for the user to input a value and press Enter. Users may enter any type of data. The input is treated as a MATLAB expression and it is *evaluated* in the current workspace. If the `input()` function is used for an assignment operation, then the data is assigned to a variable appropriately. If the user presses the Enter key without entering anything, then the input returns an empty matrix. When the user enters an invalid expression into the prompt, a relevant error message is displayed at the prompt. When a character or a string of character vectors needs to be fed into `input()`, the user must define them as a string. Otherwise, `'s'` is used as a second argument and then input is treated as a string. The usage is demonstrated in this code.

```
>> prompt = 'What is your name:';
>> name = input(prompt)
What is your name:Sandeep
Error using input
Undefined function or variable 'Sandeep'.

What is your name:'Sandeep'

name =

'Sandeep'

>> name = input(prompt,'s')
What is your name:Sandeep

name =

'Sandeep'

```

While dealing with numerical values, a single value or an array must be used, but with valid MATLAB syntax. For example, in the following code, the user can type a single value of `r` (storing the radius of a circle) or multiple values as arrays. This information can then be used to find the circumference (= $2\pi r$) and area (= $\pi r2$).

```
>> r = input('Enter value of radius:');
Enter value of radius:2
>> circumference = 2*pi*r

circumference =

12.5664

>> area = pi*r^2

area =

12.5664

>> r = input('Enter value of radius:');
Enter value of radius:[1 2 3]
>> r

r =

1     2     3

>> circumference = 2*pi*r

circumference =

6.2832   12.5664   18.8496

>> area = pi*r.^2

```

```
area =

3.1416   12.5664   28.2743

```

4.2.2 keyboard()

The `keyboard` keyword gives control to the user while running a program so that user can enter data or additional MATLAB commands, if required. This process can be effectively used by the user to check the program. It is called *debugging*.

When this is done, the MATLAB prompt changes from `>>>` to `k>`. The keyboard mode is terminated by executing the command `dbcont`. `dbquit` can also be used to exit keyboard mode, but in this case the invoking MATLAB code file is terminated. Control returns to the invoking MATLAB code file.

A valid MATLAB `expression` must be entered here. This keyword can be used to change values of variables in the middle of programs very effectively. Its usage is shown in the sample code in Listing 4-1.

Listing 4-1. The keyboardCommand.m Program

```
%program to demonstrate
%usage of keyboard command

x = 10;
y = 12;
keyboard %change value of x here
answer = x^2
```

When this is executed, you'll see the following session:

```
>> keyboardCommand
K>> x=2.5

x =

2.5000

K>> dbcont

answer =

6.2500

>> x=2.5

x =

2.5000

>> x.^2

ans =

6.2500

```

When the `keyboard` keyword is encountered, the MATLAB session goes into debug mode and the user then alters the values of `x=2.5`. Typing `dbcont` continues the execution of the program. The answer is calculated as per the new assignment of value.

4.2.3 menu()

A graphical way of inputting values can be performed using the `menu()` command, where a title and a set of options are given as inputs (separated by commas). This works if the user has a computer terminal with graphics capabilities. Otherwise, a list of options is presented at the command prompt. The user is presented with a graphical window and can use a mouse or keyboard to select an option. The options return a scalar value, which can be stored in a variable. The options are numbered internally.

Let's look at the usage with an example. Create a menu with the title "Even or Odd Numbers" and two options—`Even` and `Odd`. Store this value in the variable `I`. When this command is executed, the graphical window shown in Figure 4-1 appears. The following outputs are possible:

- If the user closes the window without entering a value, the output is 0.
- If the user clicks on `Even`, the output is 1.
- If the user clicks on `Odd`, the output is 2.

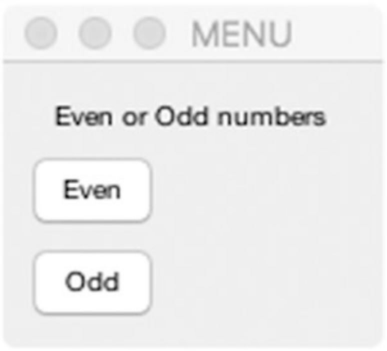


Figure 4-1. *The Menu window output*

This is shown in the following code:

```
>> I = menu('Even or Odd Numbers','Even','Odd')

I =

0

>> I = menu('Even or Odd numbers','Even','Odd')

I =

1

>> I = menu('Even or Odd numbers','Even','Odd')

I =

2

```

- `pause()`: If you want to temporarily halt the program, you can use the `pause()` command. Press any key to continue the program execution. To understand its usage, consider an example. Suppose you have an 3×3 matrix of random numbers. Then MATLAB program `pauseCommand.m` will show its rank, transpose, and size. Each item is shown if the user presses a key. See Listing 4-2.

Listing 4-2. The pauseCommand.m Program

```
%Program to show usage
%of pause command

x = rand(3,3);
r = rank(x);
t = x';
s = size(x);
disp('Given matrix is:')
disp(x)
disp('To continue checking its rank, press any key')
pause
disp('Rank of matrix is:')
disp(r)
pause
disp('To continue checking its transpose, press any key')
pause
disp('Transpose of matrix is:')
disp(t)
pause
disp('To continue checking its size, press any key')
pause
disp('Size of matrix is:')
disp(s)
pause
```

The output for running `pauseCommand.m` is shown here:

```
>> pauseCommand
Given matrix is:
0.8147    0.9134    0.2785
0.9058    0.6324    0.5469
0.1270    0.0975    0.9575
```

```

To continue checking its rank, press any key
Rank of matrix is:
3

To continue checking its transpose, press any key
Transpose of matrix is:
0.8147    0.9058    0.1270
0.9134    0.6324    0.0975
0.2785    0.5469    0.9575

To continue checking its size, press any key
Size of matrix is:
3     3

```

4.3 File Path

A MATLAB session starts from a default folder, which depends on the installation of a particular operating system. Usually a dedicated folder is created at the time of installation and it is generally named `MATLAB` by default. This folder's path can be viewed by typing `pwd` at the command prompt.

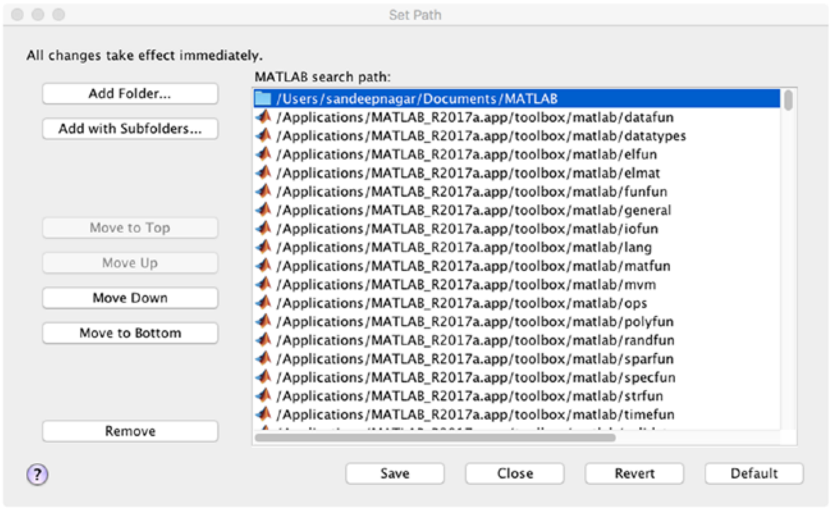
```
>> pwd

ans =

'/Users/.../MATLAB'
```

Note that the path may be different on your computer and the three dots are used to represent a generalized representation of the path.

When you want to run a MATLAB's `.m` file, the file is searched first in the default folder. If it is not there, you must change the working directory to the one where the file is stored. This can be initiated by typing `pathtool` at MATLAB's command prompt (see Figure 4-2).



All changes take effect immediately.

MATLAB search path:

```
/Users/sandeepnagar/Documents/MATLAB
/Applications/MATLAB_R2017a.app/toolbox/matlab/datafun
/Applications/MATLAB_R2017a.app/toolbox/matlab/datatypes
/Applications/MATLAB_R2017a.app/toolbox/matlab/elfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/elmat
/Applications/MATLAB_R2017a.app/toolbox/matlab/funfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/general
/Applications/MATLAB_R2017a.app/toolbox/matlab/iofun
/Applications/MATLAB_R2017a.app/toolbox/matlab/lang
/Applications/MATLAB_R2017a.app/toolbox/matlab/matfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/mvm
/Applications/MATLAB_R2017a.app/toolbox/matlab/ops
/Applications/MATLAB_R2017a.app/toolbox/matlab/polyfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/randfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/sparfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/specfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/strfun
/Applications/MATLAB_R2017a.app/toolbox/matlab/timefun
```

Figure 4-2. *Setting the path of working directory using the pathtool command*

You can simply add the directory to the list of directories for the session or save it. You can also choose to set the directory as the default for future MATLAB sessions. This is a good option if you will be working on a project for a long time and are sure that the directory will be used on a daily basis. It is strongly suggested that you keep all the files in either the default directory of the installed MATLAB program or make your working directory the default one.

4.4 File Operations

File operations constitute an important part of computation. It is important to note that the file system is OS (Operating System) dependent. Just like most scientific programs, MATLAB works with UNIX-like systems, so it works on Linux-based and MacOS X equally well with the same set of commands. On Windows, you use the same commands as the Linux version for dealing with files within the MATLAB environment. The code examples in this book were written and tested on Windows 8, MacOSX 10.10, and Ubuntu 14.04 systems.

4.4.1 Users

A computing system is accessed by different users. Each user defines a workspace to avoid damaging each other's work. After login, a user's workspace becomes active for that user. The workspace is made up of various files and folders. Some files are essential for the OS to define the workspace and its properties, hence they should not be altered. This is ensured by giving permissions to various users.

Reading and writing a file is restricted by permission. The administrator (fondly called the `admin`) is also called the superuser and has all privileges and permission to edit any file/folder. You must understand the defined user types for a computer system and then issue those commands accordingly. If you are not permitted to access certain folders and the input data you need is placed inside those files/folders, you will always get an error (unless the admin changes your permissions).

4.4.2 File Path

Directories/folders can contain sub-directories/sub-folders and files again. This can go to any level if this process if not restricted by the administrator.

The pwd command stands for *print working directory*. On the MATLAB terminal, typing pwd displays the path of the present working directory, as shown in this example:

```
>> pwd
ans = /home/sandeep
```

The user's /home directory contains another directory named /sandeep. This is the present working space. When pwd is typed into the terminal, a variable name named ans stores this data (file path). A variable name of your choice can be assigned to store the filename as a string.

A file/folder is accessed by typing the file path into the terminal. Consider this small exercise to understand this process. To create a new directory, you use mkdir *name* as follows:

```
>> mkdir matlab—practice
ans = 1
>> ls
Downloads                          Music
R
Templates
matplab—practice
Videos
Desktop                            software
Work
Documents                          Library
Pictures
>> cd matplab—practice
>>
```

At line 1, `mkdir matplab-practice` creates a directory named `matplab-practice`. To see the contents of the present directory, you can use the `ls` command, as is done at line 3, which stands for `list`. To change the directory, you can use the `cd` *`file path`* command, as shown in line 13. I suggest that you work in this directory for rest of the book.

4.4.3 Creating and Saving Files

The `save` and `load` commands allow you to write and read data to memory.

```
 1  >> matrix = rand(3,3);
 2  >> save MyFirstFile.mat matrix
 3  >> ls
 4  MyFirstFile.mat
 5  >> load MyFirstFile.mat
 6  >> matrix
 7  matrix =
 8
 9  0.467414     0.610273     0.429941
10  0.568490     0.037898     0.734682
11  0.547370     0.275421     0.539650
12
13  >>
```

At line 1, A variable named `matrix` is created first, which stores a random-value 3×3 matrix. At line 2, this data is stored as a `.mat` file named `MyFirstFile.mat`, which is passed the variable name as the argument. When required, this file can be loaded in the workspace using the `load MyFirstFile.mat` command and then by calling the variable named `matrix`. The random numbers recorded when the file was saved are loaded as the data for the 3×3 matrix. Note that this data does need not be numbers. It can be anything that a digital computer can handle, including pictures, videos, strings, and characters, just to name a few.

Multiple variables can be stored in the same file by passing the name of the variables at the time of saving.

```
>> matrix1 = rand(4,4);
>> matrix2 = rand(2,3);
>> matrix3 = rand(2,2);
>> save("SavingMultipleVariables.mat","matrix1","matrix2",
       "matrix3")
>> load SavingMultipleVariables.mat
>> matrix1
matrix1 =

   0.8598130   0.0118250   0.9803720   0.3044413
   0.6676748   0.0056845   0.1101545   0.2183920
   0.2547204   0.8192626   0.8056112   0.6961116
   0.7924558   0.9130480   0.1976146   0.4635055

>> matrix2
matrix2 =

   0.35215   0.55770   0.66650
   0.98515   0.98677   0.45513

>> matrix3
matrix3 =

   0.097693   0.540354
   0.923853   0.329501

>>>> save –binary SavedAsBinary m*
>> ls
MyFirstFile.mat  SavedAsBinary  SavingMultipleVariables.mat
```

The `help save` and `help load` commands provide very useful instructions about using `save` and `load`. Using the options, you can save the file in a specific format. For example, on line 26, all variables names starting with `m` are saved as binary data inside a binary file named `SavedAsBinary`. This is particularly important when data generated from MATLAB-based numerical computations is used in other software programs. You can also specify the precision of saved data using options. You can also compresses a big file using the `-zip` command. This is very useful when the data generated by MATLAB is large in size and needs to be transmitted.

The `load` function follows the same logic as the `save` function. Data can be unzipped and loaded from a particular formatted file as an array. The array, thus populated, can be used for computation and the resultant files can be made by using the `save` function again (if required). Elaborate computations require this procedure to be repeated successively many times, thus the functions have been optimized to locate and load the required data in a short time.

Delimited numeric data files (numerical data values separated by a delimiter) can be read and written using `dlmread()` and `dlmwrite()`. The functions produce ASCII-delimited files. To illustrate this, the following MATLAB code performs the following task:

- Stores a 3×3 matrix in variable `A`.
- Using the `dlmwrite()` function, a file named `randomNumbers.txt` is written, which takes its inputs from the matrix stored in `A`.
  - The delimiter is defined to `;`
  - You can check the file in the working directory and open it with an appropriate text editor or spreadsheet software.
- A new variable named `B` is initialized to be an empty matrix.

- Using the function `dlmread()`, this file is read. It is important to define the delimiter used during the creation of the file. The results are stored in `B` and found to be exactly same as that of `A`.

```
>> A = randn(3,3)

A =

 0.3252   −1.7115      0.3192
−0.7549    −0.1022      0.3129
 1.3703   −0.2414     −0.8649

>> dlmwrite('randomNumbers.txt',A,';')
>> B = []
>> B = dlmread('randomNumbers.txt',';')

B =

 0.3252    −1.7115     0.3192
−0.7549     −0.1022     0.3129
 1.3703    −0.2414    −0.8649
```

4.4.4 Using the Diary and History Commands

A MATLAB session can be recorded in a file by using the command `diary`. Type `help diary` to see information about its use. Writing `help` *`filename`* allows recording the session in a file with given filename. The commands and their outputs are continuously updated using this function.

You can use the `history` command to display a list of executed commands. Various options are available to see this history in particular formats.

4.4.5 Opening and Closing Files

To read and write data files, they must be opened and defined as readable and/or writable. The `fopen` function returns a pointer to an open file that is ready to be read or written to. This is defined by the following options: `r` as readable, `w` as writable, `r+` as readable and writable, `a` for appending (i.e., writing new content at the end of the file), and `a+` for reading, writing, and appending. The opening mode can be set to `t` for text mode or `b` for binary mode. `z` enables opening a gzipped file for reading and writing.

Once all the data has been read from or written to, the opened file should be closed. The `fclose` function does this.

```
MyFile = fopen("a.dat","r");
```

A variable `MyFile` is created which is used to store the contents of the file `a.dat`. This file is opened in reading mode only in the sense that it cannot be edited. This is important if the author of the file wants the information to remain unchanged while sharing it. This might be necessary for files containing constants or important pieces of code that should not be changed.

The `freport()` command prints a list of opened files and whether they are opened for reading, writing, or both. For example:

```
>> freport

number    mode    arch        name
------    ----    ----        ----
0         r       ieee-le    stdin
1         w       ieee-le    stdout
2         w       ieee-le    stderr

>>
```

4.4.6 Reading and Writing Binary Files

A binary file is computer readable file. They are simply sequence of bytes. They are the same as the C functions `fread` and `fwrite`, which can read and write binary data from a file.

4.4.6.1 The csvread and csvwrite Functions

The `csvread` and `csvwrite` functions are used to read data from `.csv` files, which stands for comma separated values. Suppose the following data needs to be stored as a `.csv` file.

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

The following code creates an array using `csvwrite` to create a file named `csvTestData.dat` containing the matrix values. You can check this by simply opening this newly created file in a text editor. At line 3, a new file named `csvTestData1.dat` is created with an offset defined at row 1 and column 2.

```
 1  >> a = [1,2,3,4;5,6,7,8;8,7,6,5;4,3,2,1];
 2  >> a
 3  a =
 4
 5  1   2   3   4
 6  5   6   7   8
 7  8   7   6   5
 8  4   3   2   1
 9  >> csvwrite('csvTestData.dat',a)
10  >> csvwrite('csvTestData1.dat',a,1,2)
11  >> a1 = csvread('csvTestData.dat')
```

```
a 1 =

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

>> a1 = csvread('csvTestData.dat',1,2)
a1 =

7   8
6   5
2   1

>>
```

Now the `csvread` function can be used to create matrices with desired offsets just as the `csvwrite` function.

Note A number of other functions to read and write files exist, but the present section focuses on some of the most commonly used ones. You can access the documentation to learn about using these specialized functions, if required.

4.4.7 Working with Excel Files

A lot of data is presented on the Internet in the form of Excel files. Note that one must be connected to the Internet in this case.

The `xlsopen`, `xlswrite`, `xlsclose`, `odsopen`, `odswrite`, and `odsclose` commands open, write, and close `.xls` and `.ods` files, respectively.

While `.xls` files are generated using Microsoft Excel, `.ods` files are generated using Open/Libre Office software, which is the open source equivalent of Microsoft Excel. The process of opening, reading, and writing data is as follows:

- `xlsopen('Filename.xls')`
- `a = xlsread ('Filename.xls', '3rd_sheet', 'B3:AA10');`

  Numeric data from the `Filename.xls` worksheet named `3rd sheet` will be read from cell B3 to AA10. This data is stored as an array named `a`.

- `[Array, Text, Raw, limits] = xlsread ('a.xls', 'hello');`

  The file `a.xls` is read from the worksheet named `hello`, and the whole numeric data is fed into an array named `Array`. The text data is fed into array named `Text`, the raw cell data into cell array named `Raw`, and the ranges where the actual data came in is saved in `limits`.

- `xlswrite('new.xls',a)` writes the data in an array named `a` into an `.xls` formatted Excel sheet named `new.xls`.
- `xlsclose`

```
>> a = rand(10,10);
>> odswrite('a.ods',a)
ans = 1
>> ls
a.ods
```

4.5 Reading Data from the Internet

Most often, large data sets that you need to access are kept on some remote server. Using `urlread()`, you can read a remote file. To save data to the local disk, you use the `urlwrite()` functions.

```
>> a = urlread('http://www.fs.fed.us/land/wfas/fdr_obs.
dat');
>> who
Variables in the current scope:

a    ans

>> whos
Variables in the current scope:

Attr Name        Size             Bytes    Class
==== ====        ====             =====    =====
a                1x147589         147589   char
ans              1x1              8        double

Total is 147590 elements using 147597 bytes

>> urlwrite('http://www.fs.fed.us/land/wfas/fdr_obs.
dat','fire.dat')
>> ls
fire.dat
>>
```

Here, a variable named a stores the data from the data file stored at http://www.fs.fed.us/land/wfas/fdr_obs.dat. Alternatively, the whole data is stored as a file named `a.dat` using the function `urlwrite(URL)`.

4.6 Printing and Saving Plots

Some commands, like `print` and `saveas`, exist to save graphs/figures generated by MATLAB programs, to be saved in desired formats. They are discussed in the following sections.

4.6.1 The print Command

The `print` command prints jobs, including printing using a printer and/or plotter, printing to a file, etc. This command is very useful if you need to save a figure automatically by a desired filename in a specified format.

```
%Saving in svg format
figure(1);
clf();
peaks();
print -dsvg figure1.svg

%Saving in png format
figure(1);
clf();
sombrero();
print -dpng figure2.png

%Printing to a HP DeskJet 550C
clf();
sombrero();
print -dcdj550
```

The `clf` function clears the current graphic window. A lot of other options for saving in different formats exist for the `print` command. To learn more, type `help print` into the MATLAB terminal.

4.6.2 The saveas Function

The saveas function saves a graphic object in a desired format, as follows:

```
clf();
a = sombrero();
saveas (a,"figure3.png");
```

The orient(a,orientation) function defines the orientation of an graphical object a. The valid values for the orientation parameters are portrait, landscape, and tall. The landscape option changes the orientation so the plot width is larger than the plot height. The tall option sets the orientation to portrait and fills the page with the plot, while leaving a 0.25 inch border. The portrait option (default) changes the orientation so the plot height is larger than the plot width.

4.7 Summary

This chapter explained various functions enabling reading and writing permission as well as taking data to and from a file. This becomes an essential part of a numerical computation exercise. The data can be generated in the form of files using software or hardware (an instrument). MATLAB does not care about its origin. It treats data by its type and by file type. Determining the appropriate function when using files has to be done by the user as per the situation.

File operations do provide faculties to trim the data so that only the useful part is used as an array. Further trimming can be performed by slicing operations. With the art of handling files under your belt, you can confidently proceed toward handling sophisticated numerical computations.