Chapter 2:

Files and File Management

2.1 File and File Management

One can read and write numerous types of *files*, or collections of computer data while using Matlab. In this subdivision, the below topics can be studied.

• File management definitions and commands

• Saving and restoring information

• Script M-files

• Errors and debugging

• Search path, path management, and start up.

2.1.1 File Management Definitions and Commands

To portray file management, numerous conceptions and descriptions are established and subsequently, a certain Unix commands are described in brief, followed by a portrayal of Matlab commands in detail.

File: A compilation of computer data, either binary or ASCII text, which is stored on an external memory device, namely, a tape drive or disk. The order of the file, or the format, location, and size of the data items that are stored in the file, is illustrated by the program, which writes the file on the external memory device.

File System: The amalgamation of software and hardware, which permits files to be retrieved (read), stored (written), and managed.

Structure of file system: The file system in MS Windows and UNIX comprises of a tree structure, with the base termed as the root, partitioning sequentially into branches, known as folders or directories. Files could be considered of as leaves all along the branches.

File types: There are two kinds of files: **text** and **binary** (also called plain text or ASCII text) for our purposes. Binary files include both the data stored in machine-readable type and machine commands of executable programs. Text files enclose keyboard characters indicated by 8 bits (one byte) for each character. Text files can be generated and modified with text editors and printers. Binary files can be read and operated upon by programs modelled to deal with the internal formats of these files.

File management: This procedure is executed with a group of user commands, to deal with the files in a file system. This entails portraying the tree structure by generating or removing

directories and handling files by generating, moving, renaming, or deleting them and arranging their names and attributes. In a Matlab Command window, UNIX terminal window, or MS-DOS window, file management is by command-line driven (known as typed commands). In MS-Windows, file management is managed by a Graphical User Interface (GUI).

Directory: Branch of a file tree, including subdirectories and files is termed as a folder.

Root directory: Base of file system, indicated by "C:\" in MS-DOS and MS Windows and "/" in UNIX.

Present working directory: The branch of file system, where the user is located at present. It was also known as "current directory." The user can access the files directly by means of this directory. By means of "change directory" instructions, the user can shift to a different directory.

Home directory: When the user log into his/her UNIX account, the user's current working directory, is termed as "home directory."

User files: The user is offered with authorization to deal with the entire files in directories (branches) that is known to be lower than the home directory.

Path: Sequence of branches (directories) that leads to a particular file or directory.

Path separator: Character deployed to divide the folders (directories) in the path "\" in MS-DOS and MS Windows and – "/" in UNIX.

Absolute path: A path commencing at the file system root, like, "/home/ford/teach/e6/". It is also termed as the "full path".

Relative path: A path commencing at the current working directory. For instance, if the current working directory is "/home/ford/", then "teach/e6" will be the relevant path to the directory e6.

2.1.2 UNIX File Management

The following are the fundamental UNIX commands for file management. For additional comprehensive information, type **man *command***. The majority of these functions can be simulated from the Matlab, as a result it won't be essential to study them in great detail.

pwd:	Print working directory – displays the entire path of the current working directory.
cd *path*:	Change to directory specified by *path* that can be either an absolute path or a relative.

ls:	Display a record of the names available in files and directories in the current working directory.
ls -l:	Display a record of names and related access permissions, size, owner, and modification date of the files and directories in the current working directory.
mkdir *dir*:	Generate the directory, *dir* in the current working directory.
rm *file*:	Delete *file* from present directory.
more *file*:	Display the content of *file* (text file only), for a single screen at a time (press spacebar to view the subsequent screen, q to quit).
cp *file1* *file2*:	Create a copy of *file1* named *file2*.
mv *file1* *file2*:	Vary the name of *file1* to *file2*.
lp *file*:	Prints *file* (which should be a text file) on the user's default printer.

2.1.3 Matlab File Management

Matlab offers a set of commands to deal with user files, which are comparable to those of UNIX. For more data, type **help iofun**.

pwd:	Print working directory – displays the full path of the current working directory.
cd *path*:	Vary to directory (folder) specified by *path* that can be either an absolute or relative path.
dir or ls:	Display the names of the files and folders (directories) in the current working directory.
what:	Display the names of the MAT-files and M-files in the present directory.
delete *file*:	Delete *file* from present directory
type *file*:	Display contents of *file* (text file only, like an M-file).

2.2 Saving and Restoring Matlab Information

It is an excellent engineering custom to maintain records of computations. These records can be deployed for numerous purposes, which includes,

• To modify the calculations in a while.

• To set up a report on the project.

Matlab offers numerous techniques for saving information from a workspace session. Saving the session output with the diary instruction and saving and loading the constraints with the save and load instruction are portrayed in this subdivision.

2.2.1 Diary Command

The diary commands permits the user to record the entire input and displayed output from a Matlab interactive workspace session. The instructions consist of the below mentioned parameters.

diary *file*:	Saves the entire text from the Matlab session, apart from the prompts (>>), as text in *file*, written to the current working directory. If *file* is not given, the information is written to the file known as diary.
diary off:	Suspends diary operation
diary on:	Turns diary operation back on.
diary:	Toggles diary state

Example 2.1 *Use of diary command*

Consider the example concerning roots of the quadratic equation.

Problem: Solve for s: $s2+5s+6=0$

$$s=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Matlab session:

```
>> diary roots
>> a=1;
>> b=5;
>> c=6;
>> x = -b/(2*a);
>> y = sqrt(b^2-
4*a*c)/(2*a);
>> s1 = x+y
s1 =
                    -2
>> s2 = x-y
```

s2 =

-3

The file roots are written in the user's current working directory. It can be displayed by the Matlab command type roots. It can also be displayed in a Unix terminal window by the command more roots or printed with the command lp roots.

```
a=1;
b=5;
c=6;
x = -b/(2*a);
y    =    sqrt(b^2-
4*a*c)/(2*a);
s1 = x+y
s1 =
                    -2
s2 = x-y
s2 =
                    -3
diary off
```

Observe that this is almost similar as the display in the command window, with the exemption that the Matlab prompt (>>) is not integrated.

2.2.2 Saving and Retrieving Matlab Variables

There may be instances when the user desires to save his/her Matlab constraints and as a result, that the user can shortly recover them to carry on his/her work. Under such circumstances, the user should save the information in the Matlab binary format, and hence, the complete exactness of the constraints is maintained. Files in this Matlab binary format are identified as MAT-files and they encompass an extension of mat.

Storing and Loading Workspace Values

save	Stores workspace values (parameter names, values, and sizes), in the binary file **matlab.mat** in the current working directory
save data	Stores the entire workspace values in the file **data.mat**
save data -	Stores only the parameters x and y in the file **data_1.mat**

1 x y	
load data-1	Loads the values of the workspace values earlier stored in the file **data_1.mat**

2.2.3 Exporting and Importing Data

There may be circumstances in which the user desires to export Matlab information to be functioned upon with erstwhile programs, or to introduce data produced by other programs. These have to be made with text files written with read or **save** with **load**.

For writing a text file **data1.dat** in the present working directory including values of Matlab parameters in **long e** format:

save data1.dat -ascii

Observe that the individual constraints will not be recognized with labels or partitioned in any case. Accordingly, if the user have defined parameters a and b in the Matlab workspace, the instruction above will initially output the values in a, then output the values in b, without partitioning them. Therefore, it is often enviable to write text files with the values of only a single parameter:

save data2.dat a -ascii

This command makes every row of the array a (further on arrays later) to be written to a detached line in the data file. By means of spaces, the array elements present on a line are separated. The separating quality can be done with a tab with the command:

save data2.dat a -ascii -tab

The **.mat** extension is not attached to an ASCII text file. On the other hand, it is suggested that ASCII text file names consist of the extension **.dat** and as a result, it is uncomplicated to differentiate them from M-files and MAT-files (to be portrayed in the subsequent section).

For an exemplar of importing text data, presume that a text file termed as, data3.dat includes a set of values, which symbolize related distance and the time of a runner from the initial line in a race. Every time and its corresponding distance value are on a detached line of the data file. The values on a line have to be detached by one or more spaces. Assume a data file known as data3.dat that includes the following:

0.0	0.0

0.1	3.5
0.2	6.8

The load command continued by the filename will read the data into an array with the similar name as the base name of the data file (extension deleted). For instance, the command

load data3.dat

reads the data in the file **data3.dat** and stores it in the Matlab array variable termed as data3, including three rows and two columns.

Matlab can moreover import data from and export data to two general formats, which are deployed in database program and spreadsheet formats. The more familiar of these file formats is the **.csv** format or comma separated values. This is an ASCII text file, with values on every line detached by commas, as the name entails. The subsequent format is the spreadsheet **.wk1** format, often deployed as an interchange format for diverse spreadsheet programs. The Matlab export and import commands for these formats are:

A = csvread('file')	Reads a comma detached value formatted file **file**. The result is returned in **A**. The file can only include numeric values.
csvwrite('file',A)	Writes matrix **A** into **file** as comma detached values
A = wk1read('file')	Reads the entire data from a WK1 spreadsheet **file** termed **file** into matrix **A**.
wk1write('file',A)	Writes matrix **A** into a WK1 spreadsheet file with the name file. '**.wk1**' is attached to the filename if no extension is specified.

Utilize the help service for information on options for these instructions.

2.3 Script M-Files

For uncomplicated issues, entering commands at the Matlab prompt in the Command window is efficient and simple. On the other hand, when the amount of commands raises, or when the user desires to vary the value of one or more constraints, re-examine the count of commands, typing at the Matlab turns out to be tedious. The user will discover that for the majority usages of Matlab, he/she have to set up a *script* that is a sequence of commands written to a file. Subsequently, by just typing the script file name at a Matlab, every command in the script file is implemented as if it were included at the prompt.

For additional information, type **help script.**

2.3.1 Script File

It is the collection of Matlab commands located in a text file with a text editor. Matlab can open and implement the commands precisely as if they were instructed at the Matlab prompt. The expression "script" points out that Matlab reads from the "script" established in the file. It is moreover termed as "M-files," asthe filenames have to finish with the extension '.m', e.g. **example1.m.**

M-files are text files and might be produced and modified with every text editors. The user must be acquainted with how to open, modify, and save a file with the text editor that he/she is using. On a Macintosh or PC, an **M-file editor** can be brought up by selecting **New** from the **File** menu in the Matlab Command window and choosing **M-file**.

The script M-file is implemented by selecting **Run Script...** from the **File** menu on a Macintosh or PC, or merely typing the name of the script file in the Matlab prompt command window.

Example 2.2 *Quadratic root finding script*

Create the file named ***qroots.m*** *in the user's current working directory by means of a text editor:*

```
% qroots: Quadratic root finding script
format compact;
a=1
b=5
c=6
x = -b/(2*a);
y = sqrt(b^2-4*a*c)/(2*a);
s1 = x+y
s2 = x-y
```

To implement the script M-file, just type the name of the script file qroots at the Matlab prompt:

```
>> qroots
a =
     1
b =
```

```
     5

c =

     6

s1 =

    -2

s2 =

    -3
```

Comments:

- % qroots: % permits a comment to be included
- format compact: Represses the additional lines in the output display
- a=1, b=5, c=6: Sets values of a, b, c, will exhibit result
- x & y: Semicolon at conclusion of command means these intermediary values won't be viewed.
- s1, s2: Calculate and view roots

Search rules for qroots command:

1. If defined, exhibit present Matlab variable qroots
2. Implement integrated Matlab command qroots if it subsists.
3. Implement qroots.m if it could be discovered in the Matlab search path

Commands contained by the M-file have access to the entire constraints in the Matlab workspace and the entire variables constraints by the M-file turn out to be component of the workspace.

Commands themselves are not usually displayed as they are estimated. Positioning the echo on command in the M-file will cause instructions to be exhibited. The command echo off (the default) turns off echo and command display by itself toggling the command echo state.

The user could continually edit **qroots.m**, vary the values of a, b, and c, save the file, and then it executes the revised script in Matlab to calculate a novel pair of roots.

Matlab operations constructive in M-files:

Command	Description
disp(ans)	Display outcomes devoid of identifying variable names
echo [on|off]	Control Command window echoing of script commands
input('prompt')	Prompt user with text in quotes, accept input till "**Enter**" is typed

keyboard	Give control to keyboard temporarily. Type **Return** to return control for implementing script M-file.
pause	Pause till the user presses any keyboard key
pause(n)	Pause for n seconds
wait for button press	Pause until user presses keyboard key or mouse button

The input command is utilized for user data input in a script and it is deployed in the form of an assignment statement. For instance:

a = input ('Enter quadratic coefficient a: ');

While this command is implemented, in the command window, a text string **Enter quadratic coefficient a:** is displayed based on the user prompt. Then, the user types, **in data value** that is allocated to the variable a. Since this input command terminates with a semicolon, the entered value of a is not exhibited while the command is ended.

Example 2.3

Revised quadratic roots script

Vary script for evaluating quadratic roots by:

1. Prompting for input of coefficients a, b, and c;

2. Format the display of the evaluated roots s1 and s2;

Script rqroots.m:

```
% rqroots: Revised quadratic root finding script
format compact;
% prompt for coefficient input
a = input('Enter quadratic coefficient a: ');
b = input('Enter quadratic coefficient b: ');
c = input('Enter quadratic coefficient c: ');
disp('')
% compute intermediate values x & y
x = -b/(2*a);
y = sqrt(b^2-4*a*c)/(2*a);
% compute and display roots
s1 = x+y;
disp('Value of first quadratic root: '),disp(s1);
```

s2 = x-y;
disp('Value of second quadratic root: '),disp(s2);

Two exemplars of the outcomes from this script:

>> rqroots
Enter quadratic coefficient a: 1
Enter quadratic coefficient b: 5
Enter quadratic coefficient c: 6
Value of first quadratic root: *-2*
Value of second quadratic root: *-3*
>> rqroots *Enter quadratic coefficient a: 1*
Enter quadratic coefficient b: 4
Enter quadratic coefficient c: 8
Value of first quadratic root:
-2.0000+ 2.0000i
Value of second quadratic root:
-2.0000- 2.0000i

2.3.2 M-file Commands

Command	Description
edit test	Open **test.m** for editing by means of the built-in Matlab text editor, similar as **Open...** in **File** menu

2.3.3Effective Use of Script Files

The following are various implications on the effectual utilization of Matlab scripts:

1. The name of a script file has to follow the Matlab standard for identifying the variables; i.e., the name should commence with a letter and might comprise underscore character and the digits.
2. Do not provide a script file the similar name as a parameter it evaluates, since Matlab will not be capable to implement that script file above once unless the parameter is

known. Recall that typing a variable name at the command prompt initiates Matlab to exhibit the value of that parameter. If there is no any parameter by that name, then Matlab looks for a script file containing that name. For exemplar, if the parameter **rqroot** was produced in a script file with the name **rqroot.m**, then subsequent to the execution of the script for the first time, the parameter **rqroot** subsists in the Matlab workspace. If the script file is adapted and an effort is made to run it a next time, Matlab will exhibit the value of rqroot and will not implement the script file.

3. Do not provide a script file the similar name as a Matlab function or command. The user can make sure to notice if a function exists already by employing the **which** command. For exemplar, to observe if **rqroot** exists already, type **which rqroot**. If it does not exists, Matlab will exhibit **rqroot** not found. If it does subsist, Matlab will exhibit the entire path to the function. For additional details concerning the subsistence of a function, script, or variable including the name **rqroot,** type exist (**'rqroot'**). This command returns one of the subsequent values:

0	if rqroot does not subsist
1	if rqroot is a variable in the workspace
2	if rqroot is an M-file or a file of unidentified type in the Matlab search path
3	if rqroot is a MEX-file in the Matlab search path
4	if rqroot is a MDL-file in the Matlab search path
5	if rqroot is a built-in Matlab function
6	if rqroot is a P-file in the Matlab search path
7	if rqroot is a directory

4. As in interactive mode, the entire parameters produced by a script file are portrayed as variables in the workspace. After script implementation, the user can type **who** or **whos** to demonstrate the information regarding the sizes, data types and names of these parameters.
5. The user can utilize the type command to exhibit an M-file without unlocking it with a text editor.

For exemplar, to outlook the file **rqroot.m**, the command is **type rqroot**.

2.4 Errors and Debugging

The user will discover that he/she will seldom acquire their scripts to run accurately for the initial time. On the other hand, the user shouldn't anguish, since this is also the case for practiced programmers. The user will require to be trained to recognize and correct the errors, identified in computer jargon as *bugs*.

2.4.1 Syntax Errors

The most widespread kind of error is the syntax error, a typing error in a Matlab command (for instance, **srqt** in place of **sqrt**. These errors are known to be *fatal*, since they cause Matlab to end up the execution and exhibit an error message. Regrettably, as the Matlab command interpreter can simply be mystified by these errors, the displayed error message might not be very useful. It is the user's job as a programmer to rectify the message and modify his/her script, which is identified as *debugging*.

Certain exemplars of syntax errors and related messages consist of:

- Missing parenthesis

```
>> 4*(2+5
??? 4*(2+5
           |
A closing right parenthesis is missing.
Check for a missing ")" or a missing operator.
```

This message is supportive, since it indicates the error location.

- Missing operator

```
>> 4*(2+5
??? 4(
        |
Missing operator, comma, or semi-colon.
```

An operator * was left out after 4 in this error.

- Misspelled variable name

```
>> 2x = 4*(2+5
??? 2
        |
Missing operator, comma, or semi-colon.
```

Accordingly, if the user intended the parameter name to be $x2$ instead of $2x$, the error message is deceptive. The user is at least revealed about the error location, thus permitting him/her to recognize the error.

There are huge counts of feasible syntax errors, a lot of which the user will find out in writing scripts for this process. With practice, the user will steadily turn out to be more proficient at identifying and rectifying her/his mistakes.

2.4.2 Run-time and Logic Errors

After rectifying syntax errors, the script of the user will implement, however it still may not generate the outcomes he/she desire.

Run-time Errors: These types of errors take place when the user's script is run on a specific set of data. They characteristically take place when the outcome of certain function show the way to an empty array (to be enclosed shortly in the course), **Inf** (infinity), or **NaN** (not a number).

Logic Errors: These are errors in the user's programming logic, when the user's script implements correctly, however it does not generate the expected result. When this takes place, the user requires re-thinking the improvement and execution of his/her problem-solving approach.

The following are implications to assist with the debugging of Matlab M-files or scripts:

- Implement the script on a simple version of the problem, by means of test data for which the outcomes are recognized, to find out which exhibited outcomes are erroneous.
- Delete semicolons from chosen commands in the script such that intermediary outcomes are exhibited.
- Add assertions that exhibit parameters of interest in the script.
- Locate the keyboard command at chosen places in the script to offer impermanent control to the keyboard, so that the workspace can be cross-examined and values can be transformed as required. Recommence script implementation by providing a **return** command at the keyboard prompt.

The Matlab debugging operations will be established in the course later to offer additional tools for rectifying these errors.

2.4.3 Programming Style

An excellent programming style necessitates that comments be incorporated generously in a script file as explained previously. In addition, the initial comment line (recognized as the **H1** line) previous to any executable statement is the line, which is seeked out by the **lookfor** command. This command assists the user in discovering an appropriate Matlab command:

lookfor xyz	Searches for the string *x y z* in the initial comment line (the **H1** line) in the entire M-files established on matlab path. On considering the entire files in which a match takes place, **lookfor** displays the **H1** line.
lookfor xyz -all	Searches all the initial comment block of every M-file.

For instance:

>> look for roots	
rqroots. m: % rq roots:	Revised quadratic root finding script
POLY	Convert roots to polynomial
ROOTS	Find polynomial roots

Therefore, the initial lines of a Matlab script have to be a comment that contains the key words and script name portraying its function. The subsequent numerous lines (to be looked for with the **-all** option of **lookfor**) have to include comments offering the author's name, the date the script was produced, and a comprehensive explanation of the script was also generated.

2.5 Matlab Search Path, Path Management, and Startup

The Matlab Search Path, Path Management, and Startup is described briefly in this subdivision.

Matlab search path:	Ordered list of directories that Matlab searches to find script and function M-files stored on disk. Commands to manage this search path:
matlab path:	Display search path
addpath *dir*:	Add directory *dir* to commencement of Matlab path. If the user construct a directory to store his/her script and function M-files, he/she will need to add this directory to the search path.
rmpath *dir*:	Eliminate directory *dir* from the matlab path
path(p1,p2):	Varies the path to the concatenation of the two path strings **p1** and **p2**. Therefore path (path,p) attaches a novel directory to the present path and **path (p, path)** prepends a novel path. If **p1** or **p2** are previously on the path, they are not included.

For instance, the following statements include an additional directory to Matlab's search path:

- Unix: path(path,'/home/ford/matlabm')
- DOS: path(path,'TOOLS\GOODSTUFF')

editpath:	Edit matlab path by means of Matlab editor, similar as the **Set Path** in **File** menu.
which test:	Exhibit the directory path to **test.m.** It informs the user which M-file script will be implemented while he/she enter the command **test**.

2.5.1 Matlab at Startup

Two files are implemented at startup: **matlabrc.m** and **startup.m**

matlabrc.m:	Comes with Matlab, should not be modified. Sets default Figure window size and placement, and several other default characteristics.
startup.m:	An optional M-file to be produced by the user, usually including commands that attach personal default features. It is general to put add path or path commands in **startup.m** to add supplementary directories to the Matlab search path. As **startup.m** is a standard script M-file, then there are no limitations as to what commands can be positioned in it. For exemplar, the user may desire to enclose the command **format compact in startup.m** such that the compact outcomes display the format that turns out to be the default.