MATLAB Programming
(Lecture 5)

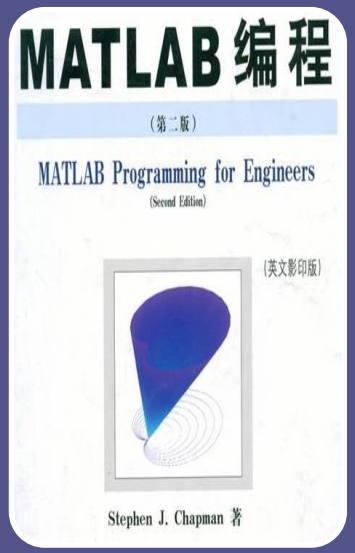
Dr. Sun Bing

School of EIE

Beihang University

Contents

Input and Output



- Review
- Import Wizard
- Text Files Processing
- Data Files Processing
- Comparing Formatted and Binary I/O Function
- File Positioning and Status Functions
- Some functions about path
- Live script

5.1 Review

- We have already introduced :

1. `var = input('prompt string')`
2. `disp('the string to be displaied')`
3. `save` or `load` matlab.mat file.
4. `fprintf()` formatted print out.

5.1.1 save

- Command format:

  Save filename [list of variables] [options]

- The default filename is ***matlab.mat***

- Option :

  `mat` Save data in MAT-file format (default)

  `ascii` Save data in space-separated ASCII format.

  `append` Adds the specified variables to an existing MAT-file.

See example chpt5_save.m.

5.1.2 load

- Command format :

  Load filename [list of variables] [options]

- Option : `mat` Treat file as a MAT_file (default, if file extent is mat)

  `ascii` Treat file as a space-separated ASCII file.

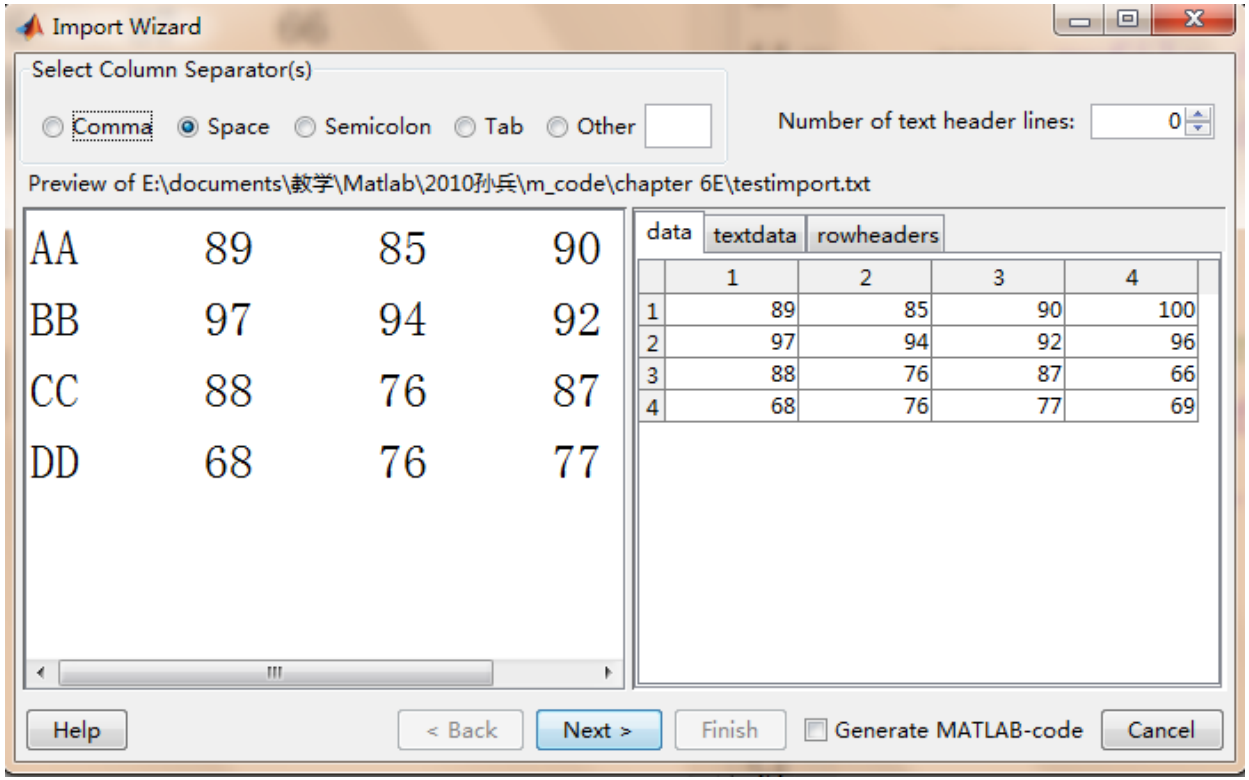
- `load` can import `ascii` data file to workspace or to a variable. ASCII files must be organized as a rectangular table of numbers, with each number in a row separated by a blank, comma, or tab character, and with an equal number of elements in each row.

```
Load datafile.dat
Var = load('datafile.dat')
```

See example chpt5_load.m

5.2 Import Wizard

- Click the selection ‘File’->’import data …’ to open import wizard window, and follow the wizard to specify the data file and step by step operate it.



5.2 Import Wizard

- Using **uiimport function**

```
>> uiimport;
```

OR

```
>> var = uiimport;
```

- Import data from data file or clipboard selected by user.
- The first case, The imported data is directly inserted into the current MATLAB workspace.
- The second case, the data is converted into a structure and saved in the variables var.
- The data file can be a .mat file or a text data file which can have a column or row header.

5.3 Text Files Processing

csvread	Read comma-separated value file
csvwrite	Write comma-separated value file
dlmread	Read ASCII-delimited file of numeric data into matrix
dlmwrite	Write matrix to ASCII-delimited file
fileread	Read contents of file into string
textread	Read data from text file; write to multiple outputs
textscan	Read formatted data from text file or string

5.3.1 csvread() function

- **Read comma-separated value file**

```
M = csvread(filename)
M = csvread(filename, row, col)
M = csvread(filename, row, col, range)
```

- M = csvread(filename) reads a comma-separated value formatted file, filename. The filename input is a string enclosed in single quotes. The result is returned in M. The file can only contain numeric values.
- M = csvread(filename, row, col) reads data from the comma-separated value formatted file starting at the specified row and column. The row and column arguments are zero based, so that row=0 and col=0 specify the first value in the file.
- M = csvread(filename, row, col, range) reads only the range specified. Specify range using the notation [R1 C1 R2 C2] where (R1,C1) is the upper left corner of the data to be read and (R2,C2) is the lower right corner. You can also specify the range using spreadsheet notation, as in range = 'A1..B7' .

See example chpt5_csvread.m

5.3.2 csvwrite() function

- **Write comma-separated value file**

```
csvwrite(filename,M)
csvwrite(filename,M,row,col)
```

- `csvwrite(filename,M)` writes matrix `M` into filename as comma-separated values. The `filename` input is a string enclosed in single quotes.

- `csvwrite(filename,M,row,col)` writes matrix `M` into `filename` starting at the specified row and column offset. The `row` and `column` arguments are zero based, so that R=0 and C=0 specify the first value in the file.

See example chpt5_csvwrite.m

5.3.3 dlmread() function

- **Read ASCII-delimited file of numeric data into matrix**

```
M = dlmread(filename)
M = dlmread(filename, delimiter)
M = dlmread(filename, delimiter, R, C)
M = dlmread(filename, delimiter, range)
```

- `M = dlmread(filename)` reads from the ASCII-delimited numeric data file filename to output matrix `M`. The `filename` input is a string enclosed in single quotes. The `delimiter` separating data elements is inferred from the formatting of the file. Comma (,) is the default delimiter.
- `M = dlmread(filename, delimiter)` reads numeric data from the ASCII-delimited file `filename`, using the specified delimiter. Use \t to specify a tab delimiter.

help dlmread

5.3.4 dlmwrite () function

- **Write matrix to ASCII-delimited file**

```
dlmwrite(filename, M)
dlmwrite(filename, M, 'D')
dlmwrite(filename, M, 'D', R, C)
dlmwrite(filename, M, 'attrib1', value1, 'attrib2',
                                         value2, ...)
dlmwrite(filename, M, '-append')
dlmwrite(filename, M, '-append', attribute-value list)
```

- dlmwrite(filename, M) writes matrix M into an ASCII format file using the default delimiter (,) to separate matrix elements. The data is written starting at the first column of the first row in the destination file, filename. The filename input is a string enclosed in single quotes.

help dlmwrite

5.3.5 fileread () function

- **Read contents of file into string**

  `text = fileread(filename)`

- `text = fileread(filename)` returns the contents of the file `filename` as a MATLAB string.

5.3.6 textread() function

- **Read data from text file; write to multiple outputs**

```
[A,B,C,...] = textread(filename,format)
[A,B,C,...] = textread(filename,format,N)
```

- `[A,B,C,...] = textread(filename,format)` reads data from the file `filename` into the variables `A,B,C`, and so on, using the specified format, until the entire file is read. The filename and format inputs are strings, each enclosed in single quotes. `textread` is useful for reading text files with a known format. `textread` handles both fixed and free format files.
- The format conversion specifiers are shown in the next slide.

See example chpt5_textread.m

Parts of format conversion specifier

Format	Action	Output
%d	Read a signed integer value.	Double array
%u	Read an unsigned integer value.	Double array
%f	Read a floating-point value.	Double array
%s	Read a white-space or delimiter-separated string.	Cell array of strings
%c	Read characters, including white space.	Character array
%*...	instead of % Ignore the matching characters specified by *	No output

5.3.7 textscan() function

- **Read formatted data from text file or string**

```
C = textscan(fid, 'format')
C = textscan(fid, 'format', N)
C = textscan(fid, 'format', param, value, ...)
C = textscan(fid, 'format', N, param, value, ...)
```

- `C = textscan(fid, 'format')` reads data from an open text file identified by file identifier `fid` into cell array `C`. The format input is a string of conversion specifiers enclosed in single quotation marks. The number of specifiers determines the number of cells in the cell array `C`.
- `C = textscan(fid, 'format', N)` reads data from the file, using the format `N` times, where `N` is a positive integer.

help textscan

5.4 Data Files Processing

MATLAB has a very flexible methods for reading and writing disk data file. The file processing procedure is

(1) open file and get file id;

(2) read data from or write data to file;

(3) close file.

File id(or `fid`) is a mechanism which is a number assigned to a file when it is opened. For example `fid=1` is standard output device (stdout).

5.4.1 file opening

- Before reading or writing a text or binary file, you must open it with the `fopen` command.

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

- If successful, fopen returns a nonnegative integer fid
- The permission string specifies the kind of access to the file you require.

`'r'` for reading only

`'w'` for writing only

`'a'` for appending only

`'r+'` for both reading and writing

5.4.2 file closing

- When finish reading or writing, use fclose to close the file.

- The form of Close file function

```
stature = fclose(fid)
stature = fclose('all')
```

  - Both forms return 0 if the file or files were successfully closed

5.4.3 Binary I/O functions

- There two functions: `fwrite()` & `fread()`
- The `fwrite()` function

```
count = fwrite(fid,array,precision)
```

Where :

`fid` : file id of the file opened with the fopen() function.

`array` : the array of values to write out to the file.

`precision` : the string specifies the format in which the data will be output.(see below Table)

`count`: the number of values written to the file in column order.

Part of Precision String

- ‘int8’ 8 bits integer
- ‘int16’ 16 bits integer
- ‘int32’ 32 bits integer
- ‘int64’ 64 bits integer
- ‘float32’ 32 bits floating point
- ‘float64’ 64 bits floating point

See example bin_ascii.m

The fread() function

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

Where:

`count` : number of values read from the file.

`array` : an array to contain the data.

`size`:

`n`: read exactly n values. After reading, array is a column vector containing n values read from the file.

`inf` : read until the end of the file.

`[n m]` : read n×m values, and format the data as an n × m array.

See example fileio_w.m & file_r.m

The fread() & fwrite() function

See example fileio_fread.m

5.4.4 Formatted I/O Functions

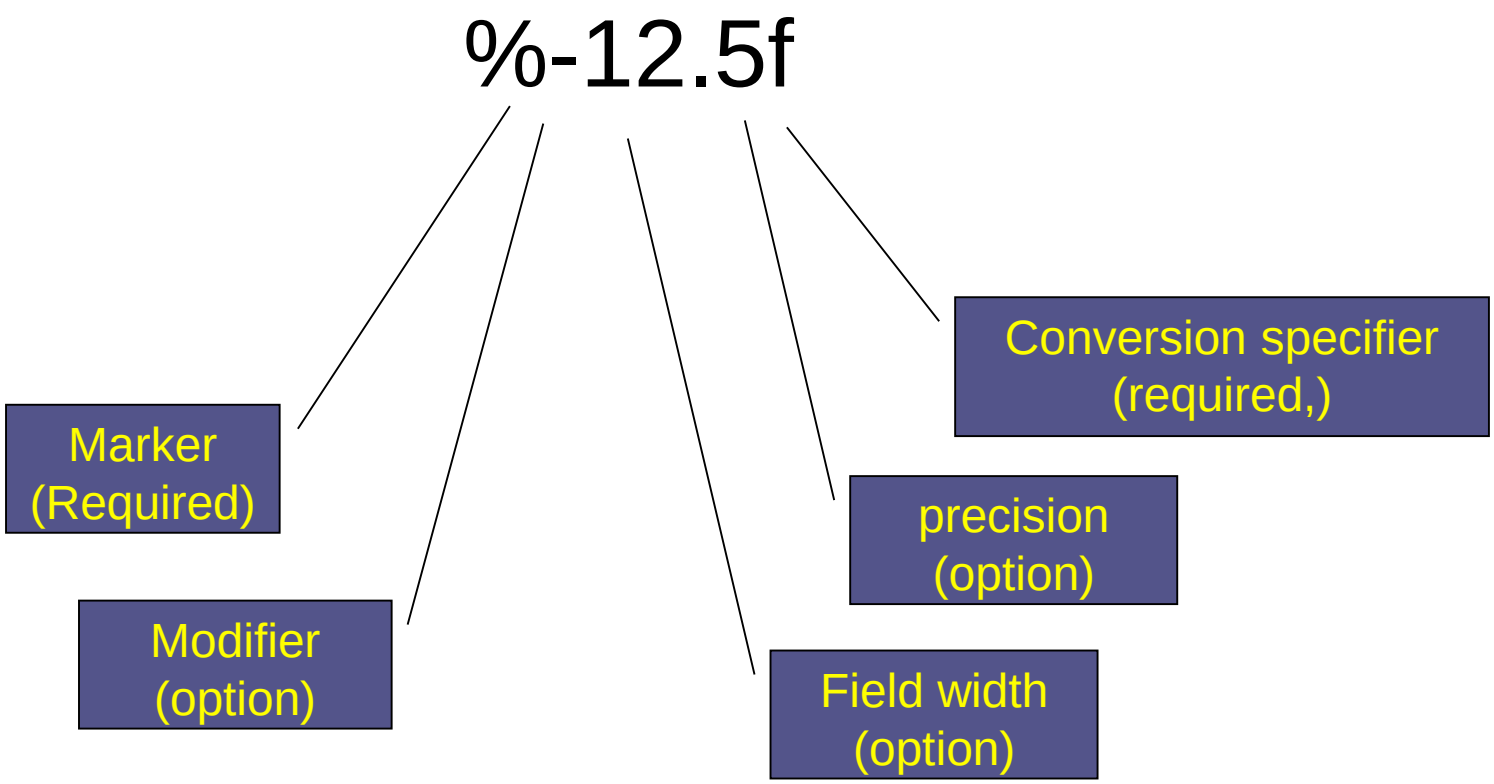
```
count = fprintf(fid, format,varlist)
fprintf(formt,varlist)
```

Where:

`count:` the number of bytes that are successfully written to the file.

`format :` the format string specifies the output format.

The components of a format specifiers



The Conversion specifiers for fprintf

%c	Single character
%d	Decimal notation(signed)
%e	Exponential notation (Using lower case e)
%E	Exponential notation (Using Upper Case E)
%f	Fixed-point notation
%g	The more compact of %e and %f; insignificant zeros do not print
%s	String of characters
%u	Decimal notation (unsigned)
%x	Hexadecimal notation (Using lower case letters a-f)
%X	Hexadecimal notation (Using upper case letters A-F)

Format Flags

- **Minus sign (-)**

  Left-justifies the converted arguments in its field. If this flag is not present, the argument is right-justified

- **+** Always print a + for positive number
- **0** Pad argument with leading zero instead of blanks.

Examples:

```
fprintf('%-12.5f\n',pi)
3.14159
```

```
fprintf('%+12.5f\n',pi)
   +3.14159
```

```
fprintf('%12.5f\n',pi)
    3.14159
```

```
fprintf('%012.5f\n',pi)
000003.14159
```

Escape characters in format strings

\n	New line
\t	Horizontal tab
\\	Print an ordinary backslash (\) symbol
’ ’	Print an apostrophe or single quote
%%	Print an ordinary percent (%) symbol

The fscanf() function

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

Size: same as previous.

See examples :

file_fw.m & file_fr.m

chpt5_fscanf.m (reading a data file text_fsacn.dat)

The fgetl() function

Read line from file, removing newline characters

`tline = fgetl(fileID)`

- `fileID` is an integer file identifier obtained from fopen.
- `tline` is a text string unless the line contains only the end-of-file marker. In this case, *tline* is the numeric value -1.

- `fgetl` reads characters using the encoding scheme associated with the file. To specify the encoding scheme, use `fopen`.

See example : fgetl_fgets.m

The fgets() function

- Read line from file, keeping newline characters

  ```
  tline = fgets(fileID)
  tline = fgets(fileID, nchar)
  ```

- *fileID* is an integer file identifier obtained from fopen.
- *tline* is a text string unless the line contains only the end-of-file marker. In this case, *tline* is the numeric value -1.
- `fgets` reads characters using the encoding scheme associated with the file. To specify the encoding scheme, use fopen.
- `tline = fgets(fileID, nchar)` returns at most *nchar* characters of the next line.

5.4.5 Example programs

- Binary I/O

(1) Write to file in Binary. (fileio_w.m)

(2) Read in from Binary file. (file_r.m)

- Formatted I/O

(1) Write to the file with formatted function `fprintf()` (file_fw.m)

(2) read in from the file with formatted function `fscanf()` (file_fr.m).

5.5 Comparing Formatted and Binary I/O Function

Formatted Files	Unformatted Files
Can display data on output devices.	Cannot display data on output devices.
Can easily transport data between different computers.	Cannot easily transport data between computers with different internal data representations.
Require a relatively large amount of disk space.	Require relatively little disk space.
Slow: requires a lot of computer time.	Fast: requires little computer time.
Truncation or rounding errors possible in formatting.	No truncation or rounding errors.

5.6 File Positioning and Status Functions

- The `exsit` function
- The `ferror` function
- The `feof` function
- The `ftell` function
- The `frewind` function
- The `fseek` function

5.6.1 The exsit () function

- Check existence of variable, function, directory, or class
  exist name *kind*
  `A = exist('name','kind')`

The *kind* argument

builtin	Checks only for built-in functions.
class	Checks only for classes.
dir	Checks only for directories.
file	Checks only for files or directories.
var	Checks only for variables.

Order of Evaluation	Return Value	Type of Entity
1	1	Variable
2	5	Built-in
3	7	Directory
4	3	MEX or DLL-file
5	4	MDL-file
6	6	P-file
7	2	M-file
8	8	Class

5.6.2 The ferror () function

- **Information about file I/O errors**

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

- `message = ferror(fileID)` returns the error message for the most recent file I/O operation on the specified file. If the operation was successful, `message` is an empty string. `fileID` is an integer file identifier obtained from fopen, or an identifier reserved for standard input (0), standard output (1), or standard error (2).
- `[message, errnum] = ferror(fileID)` returns the error number. If the most recent file I/O operation was successful, `errnum` is 0. Negative error numbers correspond to MATLAB error messages. Positive error numbers correspond to C library error messages for your system.
- `[...] = ferror(fileID, 'clear')` clears the error indicator for the specified file.

5.6.3 The feof () function

- **Test for end-of-file**

```
status = feof(fileID)
```

- `status = feof(fileID)` returns 1 if a previous operation set the end-of-file indicator for the specified file. Otherwise, `feof` returns 0. `fileID` is an integer file identifier obtained from fopen.
- Opening an empty file does *not* set the end-of-file indicator. Read operations, and the `fseek` and `frewind` functions, move the file position indicator.

5.6.4 The ftell() function

- **Position in open file**

```
position = ftell(fileID)
```

`position = ftell(fileID)` returns the current position in the specified file. `position` is a zero-based integer that indicates the number of bytes from the beginning of the file. If the query is unsuccessful, `position` is `-1`. `fileID` is an integer file identifier obtained from fopen.

5.6.5 The frewind () function

- Move file position indicator to beginning of open file

  `frewind(`*`fileID`*`)`

- `frewind(`*`fileID`*`)` sets the file position indicator to the beginning of a file. *`fileID`* is an integer file identifier obtained from fopen.
- If the file is on a tape device and the rewind operation fails, `frewind` does not return an error message.

5.6.6 The fseek () function

- **Move to specified position in file**

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

- `fseek(fileID, offset, origin)` sets the file position indicator `offset` bytes from `origin` in the specified file.
- `status = fseek(fileID, offset, origin)` returns 0 when the operation is successful. Otherwise, it returns -1.

origin Starting location in the file:

'bof' or -1	Beginning of file
'cof' or 0	Current position in file
'eof' or 1	End of file

5.7 Some functions about path

- **pathsep:** path separator for this platform
- **filesep:** directory separator for this platform
- **fullfile:** build full file name from parts
- **fileparts**: returns the path, file name, and file name extension
- **which:** locate functions and files
- **dir:** directory_name lists the files in a directory
- **cd:** change current working directory
- **pwd:** Show (print) current working directory
- **what:** List MATLAB-specific files in directory
- **addpath**： add directory to search path
- **rmpath**： remove directory from search path
- **mkdir**：make new directory
- **rmdir**： remove directory
- **copyfile**：copy file or directory
- **delete**：delete file or graphics object

5.7 Some functions about path

- pathsep: path separator for this platform

  Windows： ‘;’

  Linux: ‘:’

- filesep: directory separator for this platform

  Windows： ‘\’

  Linux: ‘/’

5.7 Some functions about path

- fullfile

Examples:

```
% To build platform dependent paths to files:
   fullfile(matlabroot,'toolbox','matlab','general','Contents.m')

% To build platform dependent paths to a folder:
   fullfile(matlabroot,'toolbox','matlab',filesep)

% To build a collection of platform dependent paths to files:
   fullfile(toolboxdir('matlab'),'iofun',{'filesep.m';'fullfile.m'})
```

5.7 Some functions about path

- fileparts : [pathstr,name,ext] = fileparts(file) returns the path, file name, and file name extension for the specified file.

- dir: D = dir('directory_name') returns the results in an M-by-1 structure with the fields:

  name -- Filename

  date -- Modification date

  bytes -- Number of bytes allocated to the file

  isdir -- 1 if name is a directory and 0 if not

  datenum -- Modification date as a MATLAB serial date number.

5.7 Some functions about path

- which

Examples:

% Return the results of which in the string S. (If which is called on a variable, then S is the string 'variable'.):

S = which(...)

% Return the results of the multiple search version of which in the cell array C. Each row of cell array C identifies a function and the functions are in order of precedence:

C = which(...,'-ALL')

% Display the pathname to function FUN1 in the context of program file FUN2. While debugging FUN2, which FUN1 does the same thing:

D = which FUN1 IN FUN2

5.7 Some functions about path

- cd: change current working directory

cd directory-spec sets the current directory to the one specified

cd .. moves to the directory above the current one

- pwd: show (print) current working directory

  pwd = cd, by itself

5.7 Some functions about path

what

W = what('directory') returns the results of what in a structure array with the fields:

path -- path to directory
m -- cell array of MATLAB program file names.
mat -- cell array of mat-file names.
mex -- cell array of mex-file names.
mdl -- cell array of mdl-file names.
slx -- cell array of slx-file names.
p -- cell array of p-file names.
classes -- cell array of class directory names.
packages -- cell array of package directory names.

5.7 Some functions about path

- addpath: add directory to search path

addpath DIR1 DIR2 DIR3 ... prepends all the specified directories to the path.

- rmpath: remove directory from search path

rmpath DIR1 DIR2 DIR3 ... removes all the specified directories from the path.

5.7 Some functions about path

- mkdir: make new directory

[SUCCESS,MESSAGE,MESSAGEID] = mkdir(PARENTDIR,NEWDIR)

SUCCESS: defining the outcome of mkdir.

1 : mkdir executed successfully. 0 : an error occurred.

MESSAGE: defining the error or warning message.

MESSAGEID: defining the error or warning identifier.

- rmdir: remove directory

[SUCCESS,MESSAGE,MESSAGEID] = rmdir(DIRECTORY)

5.7 Some functions about path

- copyfile: copy file or directory

[SUCCESS,MESSAGE,MESSAGEID] = copyfile(SOURCE,DESTINATION,MODE)

SOURCE: defining the source file or directory.

DESTINATION: defining destination file or directory.The default is the current directory.

MODE: character scalar defining copy mode.

'f' : force SOURCE to be written to DESTINATION. If omitted, copyfile respects the current writable status of DESTINATION.

5.7 Some functions about path

- delete: delete file or graphics object

delete file_name deletes the named file from disk:

```
% deletes all P-files from the current directory
    delete *.p
```

delete(H) deletes the graphics object with handle H. If the object is a window, the window is closed and deleted without confirmation.

5.8 Some useful functions

- uigetdir-Standard open directory dialog box
- uigetfile-Standard open file dialog box.
- uiputfile-Standard save file dialog box
- xlswrite
- xlsread
- xmlwrite
- xmlread

[FileName,PathName,FilterIndex] = uigetfile(FilterSpec,DialogTitle,DefaultName)
[FileName,PathName,FilterIndex] = uigetfile(...,'MultiSelect',*selectmode*)

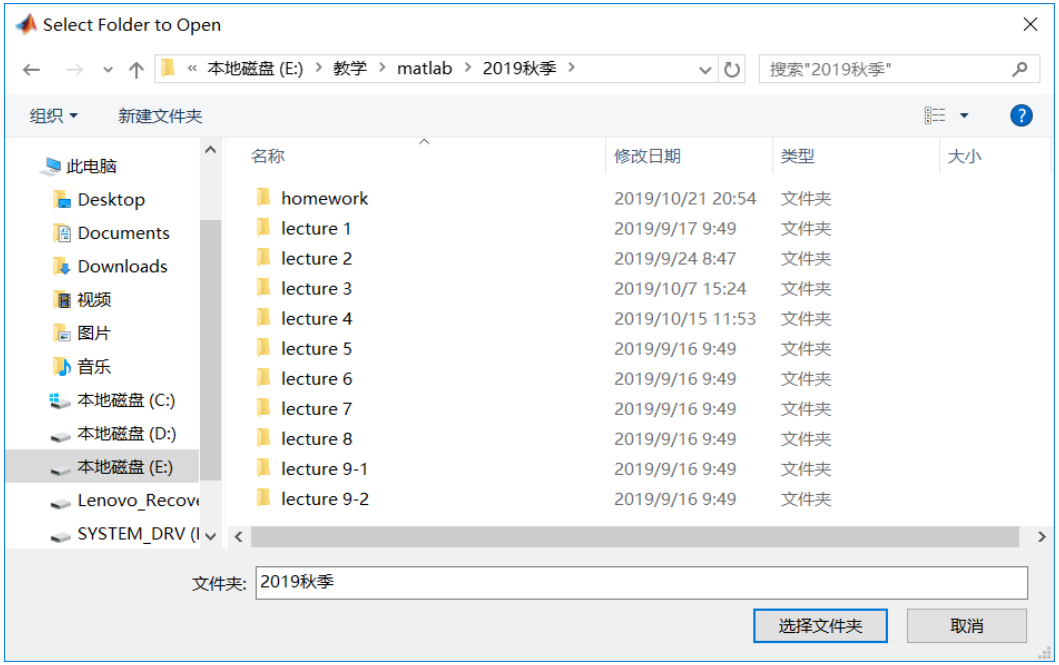
5.8 Some useful functions

✈ uigetdir: folder selection dialog box

folder_name = uigetdir

folder_name = uigetdir(start_path)

folder_name = uigetdir(start_path,dialog_title)



5.8 Some useful functions

✈ uigetfile: Open file selection dialog box

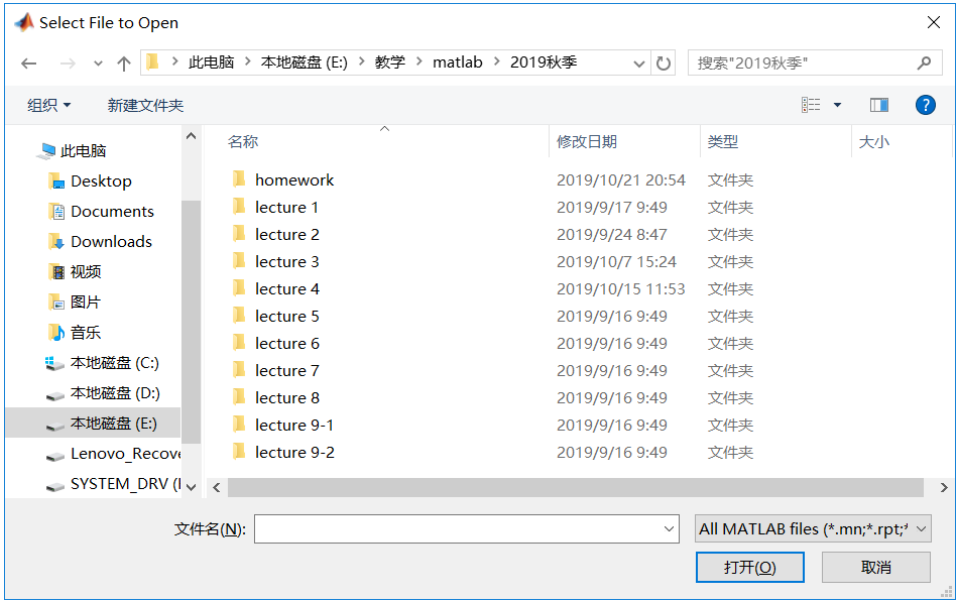
filename = uigetfile

[FileName,PathName,FilterIndex] = uigetfile(FilterSpec)

[FileName,PathName,FilterIndex] = uigetfile(FilterSpec,DialogTitle)

[FileName,PathName,FilterIndex] = uigetfile(FilterSpec,DialogTitle,DefaultName)

[FileName,PathName,FilterIndex] = uigetfile(...,'MultiSelect',*selectmode*)



[FileName,PathName,FilterIndex] = uigetfile(FilterSpec,DialogTitle,DefaultName)
[FileName,PathName,FilterIndex] = uigetfile(...,'MultiSelect',*selectmode*)

5.8 Some useful functions

- uiputfile: Open dialog box for saving files.

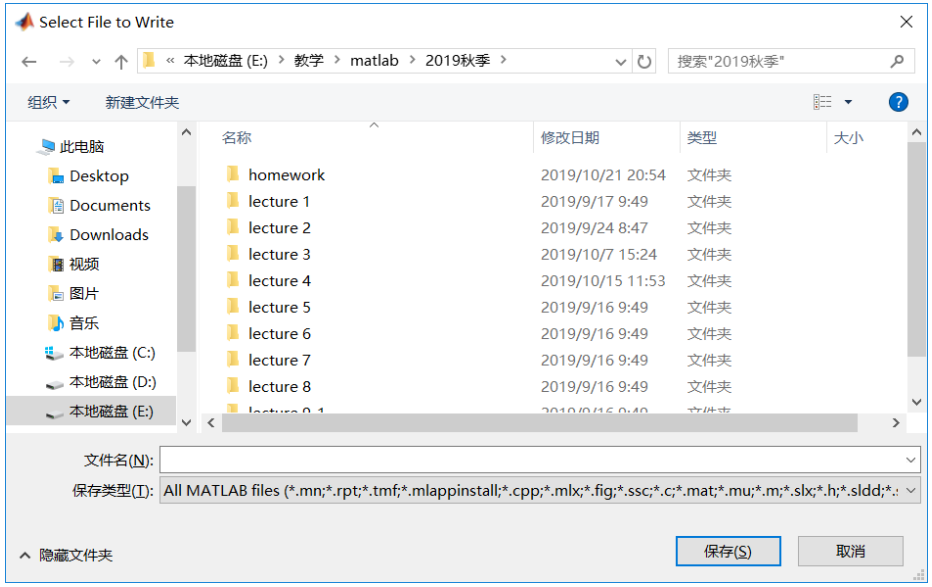
FileName = uiputfile

[FileName,PathName] = uiputfile

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec)

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec,DialogTitle)

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec,DialogTitle,DefaultName)



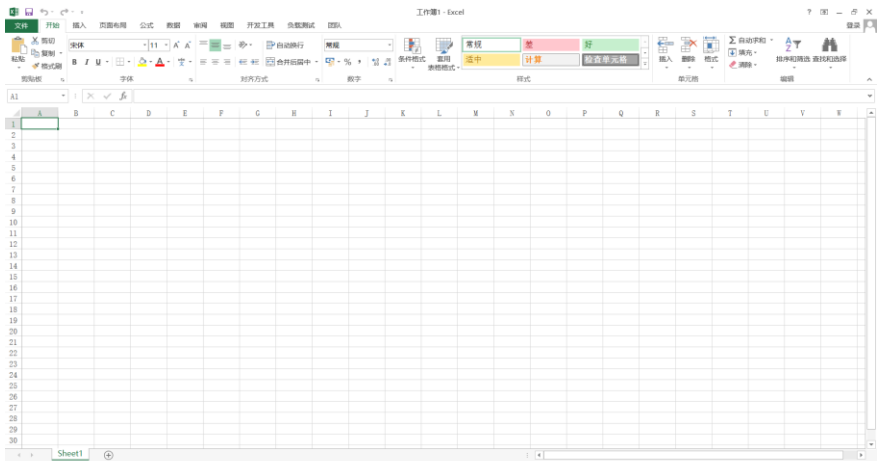
5.8 Some useful functions

- xlsread: Read Microsoft Excel spreadsheet file.

```
num = xlsread(filename)
num = xlsread(filename,sheet)
num = xlsread(filename,xlRange)
num = xlsread(filename,sheet,xlRange)
num = xlsread(filename,sheet,xlRange,'basic')
```

- xlswrite: Write Microsoft Excel spreadsheet file.

```
xlswrite(filename,A)
xlswrite(filename,A,sheet)
xlswrite(filename,A,xlRange)
xlswrite(filename,A,sheet,xlRange)
```



See example :

testxls.m

5.8 Some useful functions

- xmlread: Read Read XML document and return Document Object Model node.

DOMnode = xmlread(filename)

- xmlwrite: Write XML Document Object Model node

xmlwrite(filename,DOMnode)

chr = xmlwrite(DOMnode)

```
<?xml version="1.0" encoding="UTF-8"?>
- <root_element attribute="attribute_value">
<!--this is a comment-->
    <child_node>1</child_node>
    <child_node>2</child_node>
    <child_node>3</child_node>
    <child_node>4</child_node>
</root_element>
```

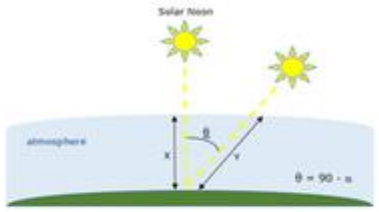
See example :

testxml.m

5.9 Live Script

- A MATLAB live script is an interactive document that combines MATLAB code with embedded output, formatted text, equations, and images in a single environment called the Live Editor.
- Live scripts are stored using the Live Script file format in a file with a .mlx extension.

Air Mass and Solar Radiation



The larger the air mass, the less radiation reaches the ground. The air mass can be calculated from the equation

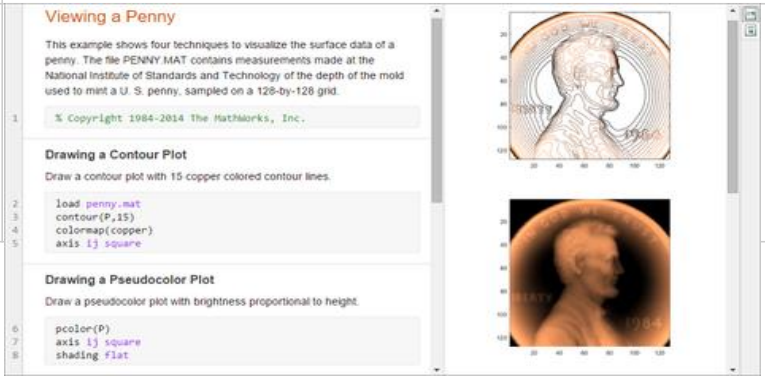
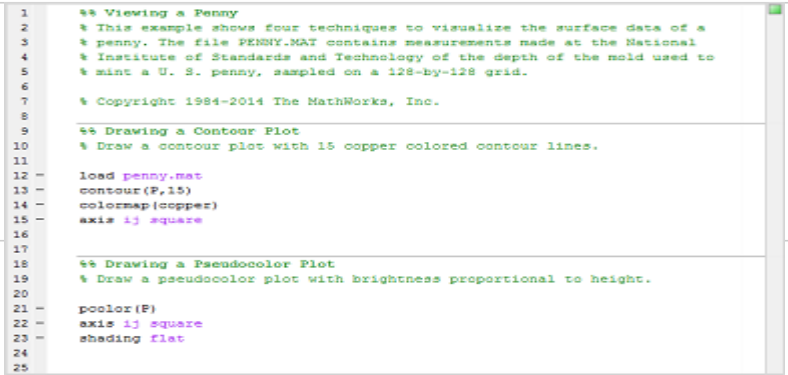
$$AM = \frac{1}{\cos(90-\alpha) + 0.5057(6.0799+\alpha)^{-1.6364}}$$

Then the solar radiation (in Kw/m^2) reaching the ground can be calculated from the empirical equation

$$sRad = 1.353 * 0.7^{AM^{0.678}}$$

```
AM = 1/(cosd(90-alpha) + 0.50572*(6.07955+alpha)^-1.6354);
sRad = 1.353*0.7^(AM^0.678);                                    % kW/m^2
disp(['Air Mass = ' num2str(AM) '   Solar Radiation = ' num2str(sRad) ' kW/m^2'])
```

5.9 Live Script

	Live Scripts and Functions	**Plain Code Scripts and Functions**
File Format	Live Code file format. For more information, see Live Code File Format (.mlx)	Plain Text file format
File Extension	.mlx	.m
Output Display	With code in the Live Editor (live scripts only)	In Command Window
Internationalization	Interoperable across locales	Non-7–bit ASCII characters are not compatible across all locales
Text Formatting	Add and view formatted text in the Live Editor	Use publishing markup to add formatted text, publish to view
Visual Representation		

Homework 4

HW4-1. Following is yearly average temperature of Beijing

Beijing Average Temperature			
Season	**Months**	**Temperature in Centigrade**	**Temperature in Fahrenheit**
Spring	Apr	13	55.4
	May	18	64.4
Summer	June	25	77
	July	28	82.4
	August	26	78.8
Autumn	Sept	26	78.8
	Oct	14	57.2
Winter	Nov	6	42.8
	Dec	-2	28.4
	Jan	-4	24.8
	Feb	-3	26.6
	Mar	5	41

1. Write a MATLAB program that creates a text file named BJAvgTemp.txt (table of the Beijing Average Temperatures). The text file should be organized as shown below
2. After creating the text file, read the text file 'BJAvgTemp.txt' and display all the data

```
                    Beijing Average Temperature

                  Month  Temperature in Centigrade Temperature in Fahrenheit
Spring :          Apr              13                      55.40
                  May              18                      64.40
Summer :          Jun              25                      77.00
                  July             28                      82.40
                  Aug              26                      78.80
Autumn :          Sept             26                      78.80
                  Oct              14                      57.20
Winter :          Nov               6                      42.80
                  Dec              -2                      28.40
                  Jan              -4                      24.80
                  Feb              -3                      26.60
                  Mar               5                      41.00
```

Submit homework online before Oct 29, 2019

Homework 4

HW4-2.

1. Create a new directory named 'Myfiles' by mkdir, in which all the following files will be stored.
2. Call the *Binomial_Coefficient function (from your last assignment) to generate values (where* k=0-5, n=5 and p=0.1, 0.2, 0.3, ... 0.9) *and store each array into seperate Binary files and name them as 'Binary_1.dat', 'Binary_2.dat', 'Binary_3.dat'... 'Binary_9.dat' [* for example: Probability values (for k=0-5, n=5, p=0.1) will be stored in Binary_1.dat. Table is shown below]*
3. Rename the nine Binary files named 'Binary_i.dat' (i=1, 2, ⋯, 9) with new names 'newfile_i.dat' (i=1, 2, ⋯, 9) .
4. Find the maximum value in each file ('newfile_i.dat' (i=1, 2, ⋯, 9)) and store the maximum value in 'Max_i.txt' (i=1, 2, ⋯, 9). [* Do not use built-in functions for finding the maximum value]
5. Read the result from 'Max_i.txt' and sort them (increasing order) into a 9*1 matrix. [* Do not use built-in functions for sorting]
6. Write the sorted matrix into an ASCII file named 'sort.dat' .

n=5									
k	p=0.1	p=0.2	p=0.3	p=0.4	p=0.5	p=0.6	p=0.7	p=0.8	p=0.9
0	0.5905	0.3277	0.1681	0.0778	0.0313	0.0102	0.0024	0.0003	0.0000
1	0.3281	0.4096	0.3601	0.2592	0.1563	0.0768	0.0284	0.0064	0.0005
2	0.0729	0.2048	0.3087	0.3456	0.3125	0.2304	0.1323	0.0512	0.0081
3	0.0081	0.0512	0.1323	0.2304	0.3125	0.3456	0.3087	0.2048	0.0729
4	0.0005	0.0064	0.0284	0.0768	0.1563	0.2592	0.3601	0.4096	0.3281
5	0.0000	0.0003	0.0024	0.0102	0.0313	0.0778	0.1681	0.3277	0.5905

Submit homework online before Oct 29, 2019

Thanks