M. Introduction to MATLAB[†]

MATLAB is a powerful high-level programming language for scientific computations. It is very easy to learn and use in solving numerically complex engineering problems. The exercises in this book have been written assuming you are not proficient in MATLAB. However, some basic concepts of MATLAB are included here for a quick review to facilitate your understanding of the programs and for performing the exercises.

MATLAB consists of functions that are either built into the interpreter or available as M-files, with each containing a sequence of program statements that execute a certain algorithm. A completely new algorithm can be written as a program containing only a few of these functions and can be saved as another M-file.

MATLAB works with three types of windows on your computer screen. These are the *Command window*, the *Figure window* and the *Editor window*. The Command window has the heading `Command`, the Figure window has the heading `Figure No. 1`, and the Editor window has the heading showing the name of an opened existing M-file or `Untitled` if it is a new M-file under construction. The Command window also shows the prompt `>>` indicating it is ready to execute MATLAB commands. Results of most printing commands are displayed in the Command window. This window can also be used to run small programs and saved M-files. All plots generated by the plotting commands appear in a Figure window. Either new M-files or old M-files are run from the Command window. Existing M-files can also be run from the Command window by typing the name of the file.

In the remaining part of this appendix we illustrate the use of some of the most commonly used functions and review some fundamental concepts associated with MATLAB.

M1 Number and Data Representation

MATLAB uses conventional decimal notations to represent numbers with a leading minus sign for negative numbers. The approximate range of numbers that can be represented is from 10^{-308} to 10^{308}. Very large or very small

[†]Reproduced from *Digital Signal Processing Laboratory Using MATLAB®*, Sanjit K. Mitra, ©1999, McGraw-Hill by permission of the author and the publisher.

numbers can be represented using exponents. Typical examples of valid number representations are as follows:

```
1234.56789        123456.789E-2        1.23456789e3        -1234.56789
```

There should be no blank space before the exponent.

The data in MATLAB are represented in the form of a rectangular matrix that does not require dimensioning. Its elements can be either real or complex numbers. Thus, a one-dimensional discrete-time signal can be represented either as a row or a column vector. For example the row vector data representation in

```
x = [3.5+4*j     -2.1-7.4*j     1.05-0.8*j     0     9.2*j];
```

denotes a complex-valued signal `x` of length `5`. Note the use of square brackets to indicate that `x` is a rectangular matrix. Note also that the imaginary part of a complex number is represented using the operator `*` and the letter `j`. An alternate form of representation of the imaginary part uses the letter `i` instead of the letter `j`. The real and imaginary parts of a complex number should be entered without any blank spaces on either side of the `+` or $-$ sign as indicated above. The elements in the row of a matrix can also be separated by commas as indicated below:

```
x = [3.5+4*j,     -2.1 - 7.4*j,     1.05-0.8*j,     0,     9.2*j];
```

The semicolon `;` at the end of the square brackets ensures that the data are not printed in the command window after they have been entered. If the above data were entered without the semicolon, MATLAB would print in the Command window

```
x =
 Columns 1 through 4
 3.5000 + 4.0000i     -2.1000 - 7.4000i     1.0500 - 0.8000i     0
 Column 5
 0 + 9.2000i
```

Alternately, if needed, the actual value of `x` can be printed by typing `x` in the Command window.

The elements of a matrix can be entered in two different ways. The rows can be typed on a single line separated with semicolons or on different lines.

For example, the 3×4 matrix A

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 9 & 11 & 13 & 15 \end{bmatrix}$$

can be entered either as

```
A = [1    3    5    7;    2    4    6    8;    9    11    13    15];
```

or as

```
A = [1  3  5  7
     2  4  6  8
     9  11 13 15];
```

The indexing of vectors and matrices in MATLAB begins with 1. For example, x(1) in the above vector x is 3.5000 + 4.0000i, x(2) is -2.1000 - 7.4000i, and so on. Similarly, the first element in the first row of a matrix A is given by A(1,1), the second element in the first row is given by A(1,2), and so on. The index cannot be less than 1 or greater than the dimension of the vector or matrix under consideration.

The size of an array in the workspace of MATLAB can be determined by using the function size. For example, by typing size(x) we obtain the result

```
ans =
    1    5
```

The length of a vector can also be found by using the function length. For example, typing length(x) yields the result

```
ans =
    5
```

The array transpose operation is implemented using the operator .′. Thus the transpose of X is given by the expression X.′. If X is a matrix with complex-valued elements, X′ is the complex conjugate transpose of X, whereas if X is a matrix with real-valued elements, X′ is the transpose of X.

The data vectors and matrices in MATLAB can be labeled by a collection of characters including the numbers, such as x, x1, X, X1, XY, and so on.

It should be noted that MATLAB normally differentiates between lowercase and uppercase letters.

Example M.1
Let `X` denote the `3×4` real-valued matrix entered by typing

```
X = [1    2    3    4;    5    6    7    8;    9    10    11    12];
```

Then typing `X` in the Command window results in the display of

```
ans =
   1    2    3    4
   5    6    7    8
   9   10   11   12
```

and typing `X'` we get

```
ans =
   1    5    9
   2    6   10
   3    7   11
   4    8   12
```

Consider next a 2×3 complex-valued matrix `Y` entered as

```
Y = [1+2*i, 3-4*i, 5+6*i; 7-8*i, 9+10*i, 11-12*i];
```

Typing of `Y` yields

```
Y =
   1.0000 + 2.0000i   3.0000 - 4.0000i    5.0000 + 6.0000i
   7.0000 - 8.0000i   9.0000 + 10.0000i  11.0000 - 12.0000i
```

whereas typing `Y'` we get

```
ans =
   1.0000 - 2.0000i    7.0000 + 8.0000i
   3.0000 + 4.0000i    9.0000 - 10.0000i
   5.0000 - 6.0000i   11.0000 + 12.0000i
```

To obtain the transpose of `Y` we type `Y.'` resulting in

```
ans =
   1.0000 + 2.0000i     7.0000 - 8.0000i
   3.0000 - 4.0000i     9.0000 + 10.0000i
   5.0000 + 6.0000i    11.0000 - 12.0000i
```

M2 Arithmetic Operations

Two different types of arithmetic operations are available in MATLAB for the manipulation of stored data, as indicated below where `X` and `Y` denote two different matrices. If `X` and `Y` are of the same dimensions, the *addition* of `X` and `Y` is implemented by the expression `X + Y`. The addition operation `+` can also be used to add a scalar to a matrix. Likewise, the *subtraction* of `Y` from `X` is implemented by the expression `X - Y`. The subtraction operation − can also be used to subtract a scalar from a matrix.

If the number of columns of `X` is the same as the number of rows of `Y`, the *matrix multiplication* `X*Y` yields the linear algebraic product of `X` and `Y`. The multiplication operation `*` can also be used to multiply a matrix by a scalar. If `X` and `Y` have the same dimensions, `X.*Y` is an *array multiplication* forming the element-by-element product of `X` and `Y`.

If `Y` is a square matrix and `X` is a matrix with the same number of columns as that of `Y`, then the *matrix right division* `X/Y` is equivalent to `(Y'\X')'` where \ denotes the matrix left division. The right division operation `X/Y` can also be carried out if one of them is a scalar. If `Y` is a square matrix and `X` is a matrix with the same number of rows as that of `Y`, then the *matrix left division* `W=Y\X` is equivalent to solving `YW=X` by Gaussian elimination. If `X` and `Y` are of the same dimension, the *array right division* is implemented using the expression `X./Y`, resulting in a matrix whose (`r,s`)-th element is given by `X(r,s)/Y(r,s)`.

If multiple operations are employed in a statement, the usual precedence rules are followed in evaluating the expression. However, parentheses can be used to change the precedence of operations.

Arithmetic operations on matrices are illustrated in the following example.

Example M.2

Let `X = [1 2 3; 4 5 6]` and `Y = [12 11 10; 9 8 7]`. Then `X+Y` yields

```
ans =
   13    13    13
   13    13    13
```

and `X-Y` yields

```
ans =
   -11    -9    -7
   -5     -3    -1
```

The result of the operation `X+3` is given by

```
ans =
   4    5    6
   7    8    9
```

whereas the result of the operation `X*3` yields

```
ans =
   3    6    9
   12   15    18
```

The statement `X.*Y` develops the answer

```
ans =
   12    22    30
   36    40    42
```

Typing `X*Y'` we obtain the result

```
ans =
   64    46
   163    118
```

and typing `X'*Y` we arrive at

```
ans =
   48    43    38
   69    62    55
   90    81    72
```

Consider the two matrices `X = [1 2 3; 4 5 6; 7 8 9]` and `Y = [1 1 2; 2 2 3; 1 3 4]`.

Then `X/Y` yields

```
ans =
  0.5000         0    0.5000
 -2.5000    3.0000    0.5000
 -5.5000    6.0000    0.5000
```

and `Y\X` results in

```
ans =
  0     0     0
  5     4     3
 -2    -1     0
```

M3 Relational Operators

The relational operators in MATLAB `<`, `<=`, `>`, `>=`, `==`, and `=`, represent the comparison operations *less than*, *less than or equal to* ($\leq$), *greater than*, *greater than or equal to* ($\geq$), *equal to*, and *not equal to* ($\neq$), respectively. Element-by-element comparisons between two matrices of the same size are carried out using these operators with the result appearing as a matrix of the same size whose elements are set to `1` when the relation is TRUE and set to `0` when the relation is FALSE. In the case of complex-valued matrices, the operators `<`, `<=`, `>`, and `>=` are applied to compare only the real parts of each element of the matrices, whereas the operators `==` and `=` are applied to compare both real and imaginary parts.

We illustrate the use of these operators in the following example.

Example M.3

Consider the two matrices `C = [1 2 3; 4 5 6]` and `D = [1 7 2; 6 5 1]`. Then the results of applying the above relational operators on `C` and `D` are indicated below:

```
C < D =   0   1   0
          1   0   0

C > D =   0   0   1
          0   0   1

C <= D =  1   1   0
          1   1   0

C >= D =  1   0   1
          0   1   1

C == D =  1   0   0
          0   1   0

C = D =   0   1   1
          1   0   1
```

M4 Logical Operators

The three logical operators in MATLAB, `&`, `|`, and `~`, perform the logical AND, OR, and NOT operations. When applied to matrices, they operate element-wise, with FALSE represented by a `0` and TRUE represented by a `1`. We illustrate the use of these operators in the following example.

Example M.4

Consider the two matrices `A = [1 1 0 1]` and `B = [0 1 0 0]`. The results of applying the above logical operators on `A` and `B` are illustrated below:

```
A & B   =   0   1   0   0

A | B   =   1   1   0   1

    A   =   0   0   1   0
```

M5 Control Flow

The control flow commands of MATLAB are `break`, `else`, `elseif`, `end`, `error`, `for`, `if`, `return`, and `while`. These commands permit the conditional execution of certain program statements. The command `for` is used to repeat a group of program statements a specific number of times. The command `if` is used to execute a group of program statements conditionally, and the command `while` can be used to repeat program statements an indefinite number of times. The statements following the commands `for`, `while`, and `if` must be terminated with the command `end`. The command `break` is used to terminate the execution of a loop. The commands `else` and `elseif` are used with the command `if` to provide conditional breaks inside a loop. The command `error` is employed to display error message and abort functions.

The use of these commands is illustrated in the following examples.

Example M.5

Consider the generation of a length-`N` sequence `x` of integers beginning with a specified first element `x(1)` and with each succeeding element increasing linearly by a specified positive integer `D`. The MATLAB program generating and displaying this sequence is given below:

```
N = 10;
D = 3;
x = [5 zeros(1,N-1)];
for k = 2:N
     x(k) = x(k-1) + D;
end
disp('The generated sequence is');disp(x)
```

Example M.6

Now consider the generation of a length-`N` sequence `x` of integers beginning with a specified first element `x(1)` and with each succeeding element increasing linearly by a specified positive integer `D` until an element is equal to `R*D + x(1)`, where `R` is a positive integer, and then each succeeding element decreasing linearly by an amount `D` until an element is equal to `x(1)`, and then repeating the process. A MATLAB program generating this sequence is as follows:

```
N = 15; D = 3;
```

```
x = [5 zeros(1,N-1)];
for k = 2:N
    x(k) = x(k-1) + D;
    if x(k) == 3*D + x(1)
    D = -abs(D);
    elseif x(k) == x(1)
    D = abs(D);
    end
end
disp('The generated sequence is');disp(x)
```

Example M.7

The following program illustrates the use of the command `break`. The program develops the sum of a series of numbers beginning with a specified initial value `y`, with each succeeding number increasing by a fixed positive increment `D`; stops the addition process when the total sum exceeds `100`; and then displays the total sum.

```
y = 5; D = 3;
while 1
    y = y + D;
    if y > 100, break, end
end
disp('y is');disp(y)
```

M6 Special Characters and Variables

MATLAB uses a number of special characters and words to denote certain items exclusively. These characters and words should not be used for any other purpose. For example, if the letter `i` or the letter `j` is used as a variable, it cannot be used to represent the imaginary part of a complex number. Hence, it is a good practice to restrict either the letter `i` or the letter `j` exclusively for the representation of the imaginary part of complex numbers.

There are several permanent variables that cannot be cleared by the user and should not be used to denote any other quantities. The word `pi` is used to denote π. Thus, `sin(pi/4)` yields 0.70710678118655, which is equal to $1/\sqrt{2}$. The variable `eps` is equal to 2^{-52} and is a tolerance for determining

precision of certain computations such as the rank of a matrix. It can be set to any other value by the user. `NaN` represents *Not-a-Number*, which is obtained when computing mathematically undefined operations such as $0/0$ and $\infty - \infty$. `inf` represents $+\infty$ and results from operations such as dividing by zero, for example, $2/0$, or from overflow, for example, e^{1000}. The variable `ans` stores the results of the most recent operation.

The square brackets `[]` are used to enter matrices and vectors. The elements of a matrix can be separated by spaces or commas. A semicolon `;` indicates the end of a row in a matrix. It is also used to suppress printing. The precedence in arithmetic expressions can be indicated using the parentheses `()`. The parentheses are also employed to enclose the indices of an array and arguments of functions. The operator notation for transpose of an array is `'`. However, two such symbols can be used to denote a quote. For example, `'dsp program'` is a vector containing the ASCII codes of the characters inside the quotation marks. Any text following a percent symbol % denotes a comment and is not treated as a program statement.

The colon symbol `:` has many different applications in MATLAB. It is used to generate vectors, subscript matrices, and perform iterations of a block of commands. For example, `x = M:N` generates the vector

```
x = [M   M+1   M+2   ...   N],
```

if `M < N`. However `x = M:N` is an empty matrix, denoted by `[]`, if `M > N`. The command `x = M:k:N` generates the vector

```
x = [M   M+k   M+2k   ...   N],
```

where `k` can be a positive or a negative integer. Note that `x = M:k:N` generates the empty matrix `[]` if `k > 0` and `M > N` or if `k < 0` and `M < N`.

The colon can also be employed to select specific rows, columns, and elements of a matrix or a vector. For example, `Y(:,N)` represents the Nth column of `Y`. Likewise, the Mth row of `Y` is represented by `Y(M,:)`. `Y(:,M:N)` is equivalent to `Y(:,M), Y(:,M+1), ... , Y(:,N)`. Finally, `Y(:)` is equivalent to a column vector formed by concatenating the columns of `Y`.

M7 Output Data Format

All arithmetic operations in MATLAB are performed in double precision.[1] However, different formats can be used to display the result of such operations

[1] MATLAB 7 has added single-precision and integer arithmetic.

in the Command window. If all results are exact integers, they are displayed as such without any decimal points. If one or more data elements are not integers, the results can be displayed with various precision. `format short` displays five significant decimal digits and is the default format. `format short e` displays five significant decimal digits with two positive or negative decimal exponents. `format long` shows results in 15 significant decimal digits, while `format long e` adds two positive or negative decimal exponents to 15 significant decimal digits. There are three other formats for displaying results. However, these are not that useful in signal processing applications.

M8 Graphics

MATLAB includes high-level graphics capability for displaying the results of a computation. In most situations, we shall be concerned with two-dimensional (2-D) graphics and will use three-dimensional (3-D) graphics in some special cases. For 2-D graphics, plotting can be done in various forms with either linear or logarithmic scales for one or both axes of the plots. Grid lines can be added to the plots along with labels for the two axes and a title on top of the plot. Text can be placed anywhere on the graph using a mouse or specifying the starting position through commands in the program. Moreover, by including appropriate symbols in the argument of the plotting command, specific line styles, plot symbols, and colors can be displayed in the graph.

For 3-D data, plotting can also be done in various forms with either linear or logarithmic scales for one or two or all three axes of the plots. For example, lines and points can be plotted in three dimensions. Contour plots, 3-D perspective plots, surface plots, pseudocolor plots, and so forth can also be generated.

The M-file in the following section illustrates the use of several graphics commands.

M9 M-Files: Scripts and Functions

An M-file is a sequence of MATLAB statements developed using a word processor or a text editor and saved with a name that must be in the form `filename.m`. The names of M-files must begin with a letter followed by at

most 18 letters and/or digits (or underscores). However certain characters, such as hyphen - and decimal point ., are not allowed in the names. Also, do not use the names of existing M-files. An M-file can include references to other existing M-files.

Each statement of a new program is typed in the Editor window line by line as ASCII text files and can be edited using the text editor or the word processor of your computer. The complete program can then be saved as an M-file.

There are two types of M-files: *scripts* and *functions*. A function file must contain the word `function` in the first line of all program statements. Arguments in a function file may be passed from another M-file, and all variables used inside the function file are local.

The script file makes use of workspace data globally. The first line of a function file contains the word `function` and does not make use of workspace data globally. The variables defined inside a function file are manipulated locally, and the arguments of the file may be passed. When a function is completed, all local variables are lost. Only values specifically passed out are retained.

A simple example of a function file `runsum` is given below.

```
function y = runsum(x)
% Computes the mean of a vector x
L = length(x);
y = sum(x)/L;
```

A simple example of a script file `lowpass.m` follows.

```
% Script M-file lowpass.m
% Program to Perform Lowpass Filtering
% Using Three-Point Averaging of a Random Signal
% Program uses the function file runsum
z = zeros(1,11);data = randn(size(z));
u = [zeros(1,3) data];
N = 3; % N is the filter length
for k = 1:10;
    w = u(k:k+N);
    z(k) = runsum(w);
end
```

```
n = 0:10;
% Plot the noise in solid line and
% the smoothed version in dashed line
plot(n,data,'r-',n,z,'b--');grid
xlabel('Time index n');
ylabel('Amplitude');
gtext('Noisy data');gtext('Smoothed data');
```

The plot generated by executing the M-file `lowpass.m` is shown in Figure M9.1

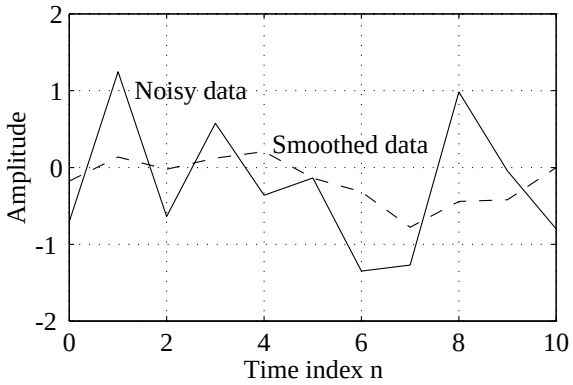

Figure M9.1: Signal smoothing example.

Note that the function file `runsum` uses the built-in function `sum`. Likewise, the script file `lowpass.m` uses the function file `runsum`.

M10 MAT-Files

Data generated by a MATLAB program can be saved as a binary file, called a MAT-file, for later use. For example, the noisy data generated by executing the program `lowpass.m` of the previous section can be saved using the command

```
save noise.mat data
```

Later it can be retrieved using the command

```
load noise
```

for use in another MATLAB session.

The data can also be saved in ASCII form. For example, after execution of the program `lowpass.m`, we can generate a 2×11 matrix containing the noisy data and the smoothed data using the command

```
result = [noise; z];
```

and then save the matrix `result` in ASCII form using the command

```
save tempo.dat result -ascii
```

The stored data can later be retrieved using the command

```
load tempo
```

M11 Printing

To develop a hardcopy version of the current Figure window, the command `print` can be used. There are many versions of this command. See the *MATLAB Reference Guide* for details. In a PC or a Mac environment, a figure can also be copied to the clipboard and then inserted into a word processing document. This approach permits generating a smaller size figure and also pasting the figure on to a text.

M12 Diagnostics and Help Facility

MATLAB has very good diagnostic capabilities, making it easier to correct any errors detected during execution. If any program statement has errors, the execution of the program will stop with a self-evident error message appearing in the Command window. For example, entering the real number `1.23456789e3` with a space before the exponent will result in the error message

```
???  1.23456789   e3
                    |
Missing operator, comma, or semi-colon.
```

Entering the real number `1.23456789e3` with a colon in place of the decimal point as `1:23456789e3` will cause the error message

```
??? Error using ==> colon
Maximum variable size allowed by the program is exceeded.
```

MATLAB provides online information for most topics through the command `help`. If `help` is typed in the Command window with no arguments, a list of directories containing MATLAB files are displayed in the window. For help on specific M-files or directories, the name of the file or the directory should be used as an argument. For example typing `help runsum` results in

```
Computes the mean of a vector x
```

Likewise, typing `help lowpass` yields

```
A Script M-File to Perform Lowpass Filtering
Using Three-Point Averaging
Program uses the function file runsum
```

A list of variables in the workspace can be obtained by typing `who`. To obtain information about the size of the variables, use the command `whos`. Other useful commands are `what`, `which`, `lookfor`, `echo`, and `pause`.

The command `what` lists all files in the current directory, whereas the command `what dryname` lists the files in the directory named `dryname` on MATLAB's search path. The command `which` is used to locate functions and files on MATLAB's search path. The command `lookfor abc` searches through all help entries on MATLAB's search path and looks for the string `abc` in the first comment line. The command `echo` is useful for debugging a new program and is used to list all M-files being invoked during the execution of a program. There are several versions of this command. See the *MATLAB Reference Guide* to determine the appropriate ones for you to use. The command `pause` stops program execution temporarily at the point it is invoked; the execution can be restarted at that point by pressing any key on the keyboard. This command is particularly useful when the program is generating a large number of plots and each plot can be examined or copied individually if the command `pause` is inserted after each plotting command.

M13 Remarks

Even though MATLAB uses double precision arithmetic, numerical approximations used in the computations may generate errors in the results. Care must be taken in such cases to interpret the results correctly. As an example, the computation of the expression `1 - 0.1 - 0.3 - 0.2 - 0.2 - 0.1 - 0.1` yields `5.551115123125783e-17` in the output `format long` when the result should have been ideally equal to `0`. On the other hand, a slight

change in the expression to `1 - (0.1 + 0.3 + 0.2 + 0.2 + 0.1 + 0.1)` yields the correct result `0`.