

INTRODUCTION TO MATLAB FOR ENGINEERING STUDENTS

based on the original version by

David Houcque Northwestern
University

(version 1.2, August 2005)

with small adaptations by

Gianni Pedrizzetti University of
Trieste

(version 1.3, August 2018)

Contents

1. Introduction

1.1	Introduction.....	1
1.2	Basic features.....	2
1.3	A minimum MATLAB session.....	2
1.3.1	Starting MATLAB.....	2
1.3.2	Using MATLAB as a calculator.....	3
1.3.3	Quitting MATLAB.....	4
1.4	Getting started	5
1.4.1	Creating MATLAB variables	5
1.4.2	Overwriting variable.....	5
1.4.3	Error messages.....	5
1.4.4	Making corrections	6
1.4.5	Controlling the hierarchy of operations or precedence.....	6
1.4.6	Controlling the appearance of floating point number.....	7
1.4.7	Managing the workspace	8
1.4.8	Keeping track of your work session	9
1.4.9	Entering multiple statements per line.....	9
1.4.10	Miscellaneous commands	10
1.4.11	Getting help.....	10

2. Basic Elements

2.1	Mathematical functions	11
2.1.1	Examples.....	12
2.2	Basic plotting.....	13
2.2.1	overview.....	13
2.2.2	Creating simple plots	13
2.2.3	Adding titles, axis labels, and annotations.....	14
2.2.4	Multiple data sets in one plot.....	15
2.2.5	Specifying line styles and colors.....	16
2.3	Introduction to the concept of Matrix	17
2.3.1	Entering a vector.....	17
2.3.2	Entering a matrix.....	19
2.3.3	Matrix indexing.....	20
2.3.4	Colon operator	20
2.3.5	Linear spacing	20
2.3.6	Colon operator in a matrix.....	21
2.3.7	Creating a sub-matrix.....	22
2.3.8	Deleting row or column.....	23
2.3.9	Dimension.....	24
2.3.10	Continuation.....	24
2.3.11	Transposing a matrix.....	25
2.3.12	Concatenating matrices	25
2.3.13	Matrix generators.....	26
2.3.14	Special matrices	27

3	Array operations and linear equations	
3.1	Array operations	28
3.1.1	Matrix arithmetic operations	28
3.1.2	Array arithmetic operations	28
3.2	Solving linear equations	30
3.2.1	Matrix inverse	31
3.2.2	Matrix functions	32
4	Introduction to programming in MATLAB	
4.1	Introduction	33
4.2	M-File Scripts	33
4.2.1	Examples	33
4.2.2	Script side-effects	35
4.3	M-File functions	35
4.3.2	Input and output arguments	37
4.4	Input to a script file	37
4.5	Output commands	38
5	Control flow and operators	
5.1	Introduction	40
5.2	Control flow	40
5.2.2	Relational and logical operators	42
5.2.5	Other flow structures	43
5.2.6	Operator precedence	43
5.3	Saving output to a file	44

Appendix: Summary of commands

1. Introduction

1.1 Introduction

The primary objective of this document is to help you learn *quickly* the first steps. The emphasis here is “learning by doing”. Therefore, the best way to learn is by trying it yourself. Working through the examples will give you a feel for the way that MATLAB operates. In this introduction we will describe how MATLAB handles simple numerical expressions and mathematical formulas.

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

MATLAB is a high-performance language for technical computing. It integrates *computation, visualization, and programming* environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated *data structures*, contains built-in editing and *debugging tools*, and supports *object-oriented programming*. These factors make MATLAB an excellent tool for teaching and research.

MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an *array* that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.

It has powerful *built-in* routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as *toolbox*. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

In addition to the MATLAB documentation which is mostly available online, there are innumerable books and online tutorials that can be useful for specific applications. Nevertheless gaining a basic knowledge from direct classes represents a fundamental support for starting exploration of the MATLAB potential.

1.2 Basic features

As we mentioned earlier, the following tutorial lessons are designed to get you started quickly in MATLAB. The lessons are intended to make you familiar with the basics of MATLAB. We urge you to perform yourself some exercises at the end of each lesson.

1.3 A minimum MATLAB session

The goal of this *minimum* session (also called *starting* and *exiting* sessions) is to learn the first steps:

- How to log on
- Invoke MATLAB
- Do a few simple calculations
- How to quit MATLAB

1.3.1 Starting MATLAB

After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut *icon* (MATLAB 7.0.4) on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains *other* windows. The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser
- The Startbutton

When MATLAB is started for the first time, the screen looks like the one that shown in the Figure 1.1. This illustration also shows the default configuration of the MATLAB desktop. You can customize the arrangement of tools and documents to suit your needs.

Now, we are interested in doing some simple calculations. We will assume

that you have sufficient understanding of your computer under which MATLAB is being run.

You are now faced with the MATLAB desktop on your computer, which contains the prompt (`>>`) in the Command Window

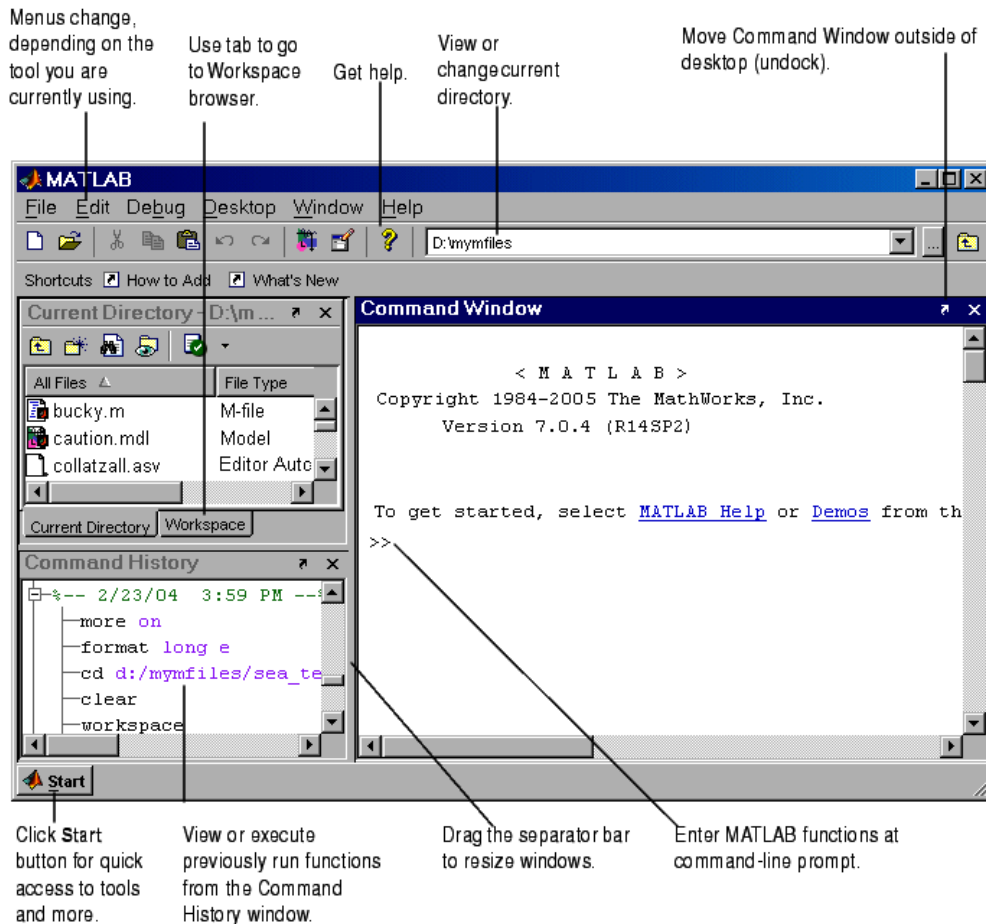


Figure 1.1: The graphical interface to the MATLAB workspace

1.3.2 Using MATLAB as a calculator

As an example of a simple interactive calculation, just type the expression you want to evaluate. Let's start at the very beginning. For example, let's suppose you want to calculate the expression, $1 + 2 \times 3$. You type it at the prompt command (`>>`) as follows,

```
>> 1+2*3
ans =
    7
```

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable `ans`, short for answer, to store the results of the current calculation. Note that the variable `ans` is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example,

```
>> x = 1+2*3
x
=
7
```

will result in `x` being given the value $1 + 2 \times 3 = 7$. This variable name can always be used to refer to the results of the previous computations. Therefore, computing `4x` will result in

```
>
>
4
*
x
a
n
s
=
28.0000
```

Before we conclude this minimum session, Table 1.1 gives the partial list of arithmetic operators.

Table 1.1: Basic arithmetic operators

Symbol	Operation	Example
+	Addition	$2 + 3$
-	Subtraction	$2 - 3$
*	Multiplication	$2 * 3$
/	Division	$2/3$

1.3.3 Quitting MATLAB

To end your MATLAB session, type **quit** in the Command Window, or select **File** → **Exit MATLAB**

1.4 Getting started

After learning the minimum MATLAB session, we will now learn to use some additional operations.

1.4.1 Creating MATLAB variables

MATLAB variables are created with an assignment statement. The syntax of variable assignment is

$$\text{variable name} = \text{a value (or an expression)}$$

For example,

```
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls. In other words, expression can involve:

- manual entry
- built-in functions
- user-defined functions

1.4.2 Overwriting variable

Once a variable has been created, it can be reassigned. In addition, if you do not wish to see the intermediate results, you can suppress the numerical output by putting a semicolon (;) at the end of the line. Then the sequence of commands looks like this:

```
>> t = 5;  
>> t = t+1  
t =  
6
```

1.4.3 Error messages

If we enter an expression incorrectly, MATLAB will return an error message. For example, in the following, we left out the multiplication sign, *, in the following expression

```
>> x = 10;
>> 5x
??? 5x
|
Error: Unexpected MATLAB expression.
```

1.4.4 Making corrections

To make corrections, we can, of course retype the expressions. But if the expression is lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key . When the command is displayed at the command prompt, it can be modified if needed and executed.

1.4.5 Controlling the hierarchy of operations or precedence

Let's consider the previous arithmetic operation, but now we will include *parentheses*. For example, $1 + 2 \times 3$ will become $(1 + 2) \times 3$

```
>> (1+2)*3
ans =
     9
```

and, from previous example

```
>> 1+2*3
ans =
     7
```

By adding parentheses, these two expressions give different results: 9 and 7.

The order in which MATLAB performs arithmetic operations is exactly that taught in high school algebra courses. *Exponentiations* are done *first*, followed by *multiplications* and *divisions*, and finally by *additions* and *subtractions*. However, the standard order of precedence of arithmetic operations can be changed by inserting *parentheses*. For example, the result of $1+2 \times 3$ is quite different than the similar expression with parentheses $(1+2) \times 3$. The results are 7 and 9 respectively. Parentheses can always be used to overrule *priority*, and their use is recommended in some complex expressions to avoid ambiguity.

Therefore, to make the evaluation of expressions unambiguous, MATLAB has established a series of rules. The order in which the arithmetic operations are evaluated is given in Table 1.2. MATLAB arithmetic operators obey the same *precedence* rules as those in

Table 1.2: Hierarchy of arithmetic operations

Precedence	Mathematical operations
First	The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward.
Second	All exponentials are evaluated, working from left to right
Third	All multiplications and divisions are evaluated, working from left to right
Fourth	All additions and subtractions are evaluated, starting from left to right

most computer programs. For operators of *equal* precedence, evaluation is from *left* to *right*. Now, consider another example:

$$\frac{1}{2+3^2} + \frac{4}{5} \cdot \frac{6}{7}$$

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7
ans =
    0.7766
```

or, if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7
ans =
    10.1857
```

So here what we get: two different results. Therefore, we want to emphasize the importance of precedence rule in order to avoid ambiguity.

1.4.6 Controlling the appearance of floating point number

MATLAB by default displays only 4 decimals in the result of the calculations, for example 163.6667, as shown in above examples. However, MATLAB does

numerical calculations in *double* precision, which is 15 digits. The command `format` controls how the results of computations are displayed. Here are some examples of the different formats together with the resulting outputs.

```
>> format short
>> x=-163.6667
```

If we want to see all 15 digits, we use the command `format long`

```
>> format long
>> x= -1.636666666666667e+002
```

To return to the standard format, enter `format short`, or simply `format`.

There are several other formats. For more details, see the MATLAB documentation, or type `help format`.

Note - Up to now, we have let MATLAB repeat everything that we enter at the prompt (`>>`). Sometimes this is not quite useful, in particular when the output is pages en length. To prevent MATLAB from echoing what we type, simply enter a semicolon (`;`) at the end of the command. For example,

```
>> x=-163.6667;
```

and then ask about the value of `x` by typing,

```
>> x
x    =
-163.6667
```

1.4.7 Managing the workspace

The contents of the workspace persist between the executions of separate commands. Therefore, it is possible for the results of one problem to have an effect on the next one. To avoid this possibility, it is a good idea to issue a `clear` command at the start of each new independent calculation.

```
>> clear
```

The command `clear` or `clear all` removes all variables from the workspace. This frees up system memory. In order to display a list of the variables currently in the memory, type

```
>> who
```

while, whos will give more details which include size, space allocation, and class of the variables.

It is sometime useful to use the command pack to pack the memory space after clearing it when we want to use memory for totally different target variable with respect to previous ones.

1.4.8 Keeping track of your work session

It is possible to keep track of everything done during a MATLAB session with the diary command.

```
>> diary
```

or give a name to a created file,

```
>> diary FileName
```

where FileName could be any arbitrary name you choose.

The function diary is useful if you want to save a complete MATLAB session. They save all input and output as they appear in the MATLAB window. When you want to stop the recording, enter diary off. If you want to start recording again, enter diary on. The file that is created is a simple text file. It can be opened by an editor or a word processing program and edited to remove extraneous material, or to add your comments. You can use the function type to view the diary file or you can edit in a text editor or print. This command is useful, for example in the process of preparing a homework or lab submission.

1.4.9 Entering multiple statements per line

It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to enter more than one statement at once. Commas (,) allow multiple statements per line without suppressing output.

```
>> a=7; b=cos(a),  
c=cosh(a) b =  
    0.6570  
c =  
    548.3170
```

1.4.10 Miscellaneous commands

Here are few additional useful commands:

- To clear the Command Window, type `clc`
- To abort a MATLAB computation, type `ctrl-c`
- To continue a line, type `...`

1.4.11 Getting help

To view the online documentation, select [MATLAB Help](#) from Help menu or [MATLAB Help](#) directly in the Command Window. The preferred method is to use the *Help Browser*. The Help Browser can be started by selecting the ? icon from the desktop toolbar. On the other hand, information about any command is available by typing

```
>> help Command
```

Another way to get help is to use the `lookfor` command. The `lookfor` command differs from the `help` command. The `help` command searches for an exact function name match, while the `lookfor` command searches the quick summary information in each function for a match. For example, suppose that we were looking for a function to take *the inverse of a matrix*. Since MATLAB does not have a function named `inverse`, the command `help inverse` will produce nothing. On the other hand, the command `lookfor inverse` will produce detailed information, which includes the function of interest, `inv`.

```
>> lookfor inverse
```

Note - At this particular time of our study, it is important to emphasize one main point. Because MATLAB is a huge program; it is impossible to cover *all the details* of each function one by one. However, we will give you information how to get help. Here are some examples:

- Use on-line help to request info on a specific function

```
>> help sqrt
```

2. Basic Elements

2.1 Mathematical functions

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.

Typing `help elfun` and `help specfun` calls up full lists of *elementary* and *special* functions respectively.

There is a long list of mathematical functions that are *built* into MATLAB. These functions are called *built-ins*. Many standard mathematical functions, such as $\sin(x)$, $\cos(x)$, $\tan(x)$, e^x , $\ln(x)$, are evaluated by the functions `sin`, `cos`, `tan`, `exp`, and `log` respectively in MATLAB.

Table 2.1 lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

Table 2.1: Elementary functions

<code>cos(x)</code>	Cosine	<code>abs(x)</code>	Absolute value
<code>sin(x)</code>	Sine	<code>sign(x)</code>	Signum function
<code>tan(x)</code>	Tangent	<code>max(x)</code>	Maximum value
<code>acos(x)</code>	Arc cosine	<code>min(x)</code>	Minimum value
<code>asin(x)</code>	Arc sine	<code>ceil(x)</code>	Round towards $+\infty$
<code>atan(x)</code>	Arc tangent	<code>floor(x)</code>	Round towards $-\infty$
<code>exp(x)</code>	Exponential	<code>round(x)</code>	Round to nearest integer
<code>sqrt(x)</code>	Square root	<code>rem(x)</code>	Remainder after division
<code>log(x)</code>	Natural logarithm	<code>angle(x)</code>	Phase angle
<code>log10(x)</code>	Common logarithm	<code>conj(x)</code>	Complex conjugate

In addition to the elementary functions, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 2.2.

Table 2.2: Predefined constant values

<code>pi</code>	The π number, $\pi = 3.14159\dots$
<code>i, j</code>	The imaginary unit i
<code>Inf</code>	The infinity, ∞
<code>NaN</code>	Not a number

2.1.1 Examples

We illustrate here some typical examples which related to the elementary functions previously defined.

As a first example, the value of the expression $y = e^{-a} \sin(x) + 10^{\sqrt{y}}$, for $a = 5$, $x = 2$, and $y = 8$ is computed by

```
>> a = 5; x = 2; y = 8;
>> y = exp(-
a)*sin(x)+10*sqrt(y) y      =
28.2904
```

The subsequent examples are

```
>> log(142)
ans      =
4.9558
```

```
>> log10(142)
ans      =
2.1523
```

Note the difference between the natural logarithm $\log(x)$ and the decimal logarithm (base 10) $\log_{10}(x)$.

To calculate $\sin(\pi/4)$ and e^{10} , we enter the following commands in MATLAB,

```
>>sin(pi/4)
ans      =
0.7071
```

```
>> exp(10)
ans      =
2.2026e+004
```

Notes:

- Only use built-in functions on the right hand side of an expression. Reassigning the value to a built-in function can create problems.

- There are some exceptions. For example, i and j are pre-assigned. However, one or both of i or j are often reassigned and used as loop indices.
- To avoid any possible confusion, it is suggested to use instead ii or jj as loop indices or redefine the variable ci as the complex i as
- ```
.>> ci=sqrt(-1)
 ci =
 0.0000 + 1.0000i
```

## 2.2 Basic plotting

### 2.2.1 overview

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning mathematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

### 2.2.2 Creating simple plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of  $x$ -coordinates,  $\mathbf{x} = (x_1, \dots, x_N)$ , and a vector of  $y$ -coordinates,  $\mathbf{y} = (y_1, \dots, y_N)$ , locate the points  $(x_i, y_i)$ , with  $i = 1, 2, \dots, n$  and then join them by straight lines. You need to prepare  $x$  and  $y$  in an identical array form; namely,  $x$  and  $y$  are both row arrays or column arrays of the *same* length.

The MATLAB command to plot a graph is `plot(x,y)`. The vectors  $\mathbf{x} = (1, 2, 3, 4, 5, 6)$  and  $\mathbf{y} = (3, -1, 2, 4, 5, 1)$  produce the picture shown in Figure 2.1.

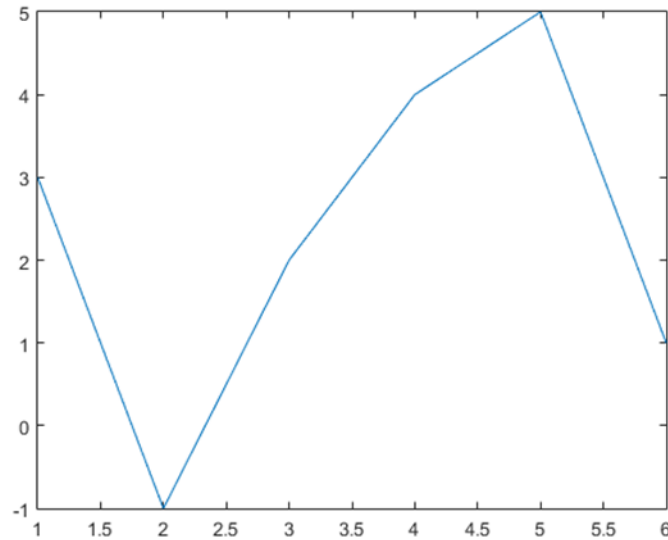
```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

Note: The plot functions has different forms depending on the input arguments. If  $y$  is a vector `plot(y)` produces a piecewise linear graph of the elements of  $y$  versus the index of the elements of  $y$ . If we specify two vectors, as mentioned above, `plot(x,y)` produces a graph of  $y$  versus  $x$ .



For example, to plot the function  $\sin(x)$  on the interval  $[0, 2\pi]$ , we first create a vector of  $x$  values ranging from 0 to  $2\pi$ , then compute the *sine* of these values, and finally plot the result:

Figure 2.1: Plot for the vectors  $x$  and  $y$



```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
```

#### Notes:

- $0:\pi/100:2*\pi$  yields a vector that
  - starts at 0,
  - takes steps (or increments) of  $\pi/100$ ,
  - stops when  $2\pi$  is reached.
- If you omit the increment, MATLAB automatically increments by 1.

### 2.2.3 Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add an  $x$ - and  $y$ -axis labels.

Now label the axes and add a title. The character `\pi` creates the symbol  $\pi$ . An example of 2D plot is shown in Figure 2.2.

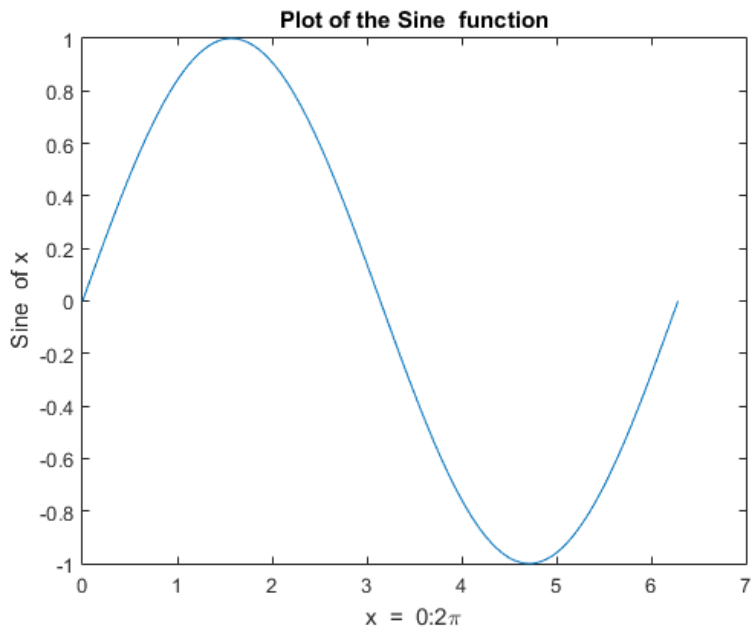


Figure 2.2: Plot of the Sine function

```
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
```

The color of a single curve is, by default, **blue**, but other colors are possible. The desired color is indicated by a third argument. For example, **red** is selected by `plot(x,y,'r')`. Note the single quotes, ' ', around `r`.

### 2.2.4 Multiple data sets in one plot

Multiple  $(x, y)$  pairs arguments create *multiple* graphs with a single call to `plot`. For example, these statements plot three related functions of  $x$ :  $y_1 = 2 \cos(x)$ ,  $y_2 = \cos(x)$ , and  $y_3 = 0.5 * \cos(x)$ , in the interval  $0 \leq x \leq 2\pi$ .

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
```

```

>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)', 'cos(x)', '0.5*cos(x)')

>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])

```

The result of multiple data sets in one graph plot is shown in Figure 2.3.

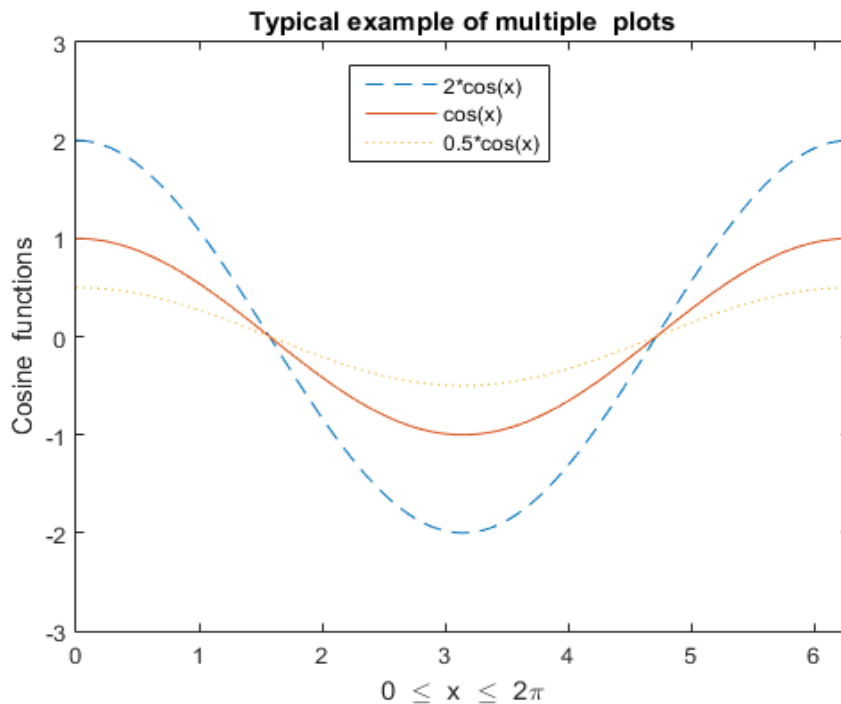


Figure 2.3: Typical example of multiple plots

By default, MATLAB uses *line style* and *color* to distinguish the data sets plotted in the graph. However, you can change the appearance of these graphic components or add annotations to the graph to help explain your data for presentation.

### 2.2.5 Specifying line styles and colors

It is possible to specify *line styles*, *colors*, and *markers* (e.g., circles, plus signs, ...) using the plot command:

```
plot(x,y,'style_color_marker')
```

where *style\_color\_marker* is a *triplet* of values from Table 2.3.

To find additional information, type `help plot` or `doc plot`.

Table 2.3: Attributes for plot

| Symbol | Color   | Symbol | Line Style | Symbol | Marker    |
|--------|---------|--------|------------|--------|-----------|
| k      | Black   | —      | Solid      | +      | Plus sign |
| r      | Red     | --     | Dashed     | o      | Circle    |
| b      | Blue    | :      | Dotted     | *      | Asterisk  |
| g      | Green   | -.     | Dash-dot   | .      | Point     |
| c      | Cyan    | none   | No line    | ×      | Cross     |
| m      | Magenta |        |            | s      | Square    |
| y      | Yellow  |        |            | d      | Diamond   |

## 2.3 Introduction to the concept of Matrix

Matrices are the basic elements of the MATLAB environment. A matrix is a two-dimensional array consisting of  $m$  rows and  $n$  columns. Special cases are *column vectors* ( $n = 1$ ) and *row vectors* ( $m = 1$ ).

In this section we will illustrate how to apply different *operations* on matrices. The following topics are discussed: vectors and matrices in MATLAB, the inverse of a matrix, determinants, and matrix manipulation.

MATLAB supports two types of operations, known as *matrix operations* and *array operations*. Matrix operations will be discussed first.

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation. Matrices can be generated in several ways.

### 2.3.1 Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension  $1 \times n$  is called a *row vector*, whereas an array of dimension  $m \times 1$  is called a *column vector*. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, `v`, type

```
>> v = [1 4 7 10 13]
v =
```

1      4      7      10      13

Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

```
>> w = [1;4;7;10;13]
w =
 1
 4
 7
 10
 13
```

On the other hand, a *row* vector is converted to a *column* vector using the *transpose* operator. The *transpose* operation is denoted by an apostrophe or a single quote (').

```
>> w = v'
w =
 1
 4
 7
 10
 13
```

Thus,  $v(1)$  is the first element of vector  $v$ ,  $v(2)$  its second element, and so forth.

Furthermore, to access *blocks* of elements, we use MATLAB's colon notation (:). For example, to access the first three elements of  $v$ , we write,

```
>> v(1:3)
ans =
 1 4 7
```

Or, all elements from the third through the last elements,

```
>> v(3:end)
ans =
 7 10 13
```

where end signifies the *last* element in the vector. If  $v$  is a vector, writing

```
>> v(:)
```

produces a column vector, whereas writing

```
>> v(1:end)
```

produces a row vector.

### 2.3.2 Entering a matrix

A matrix is an array of numbers. To type a matrix into MATLAB you must

- begin with a square bracket, [
- separate elements in a row with spaces or commas (,)
- use a semicolon (;) to separate rows
- end the matrix with another square bracket, ].

Here is a typical example. To enter a matrix A, such as

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

type

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

MATLAB then displays the  $3 \times 3$  matrix as follows,

```
A =
 1 2 3
 4 5 6
 7 8 9
```

Note that the use of semicolons (;) here is different from their use mentioned earlier to suppress output or to write multiple commands in a single line.

Once we have entered the matrix, it is automatically stored and remembered in the *Workspace*. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write,

```
>> A(2,1)
ans =
 4
```

$A(2,1)$  is an element located in the second row and first column. Its value is 4.

### 2.3.3 Matrix indexing

We select elements in a matrix just as we did for vectors, but now we need two indices. The element of row  $i$  and column  $j$  of the matrix  $A$  is denoted by  $A(i,j)$ . Thus,  $A(i,j)$  in MATLAB refers to the element  $A_{ij}$  of matrix  $A$ . The *first* index is the *row* number and the *second* index is the *column* number. For example,  $A(1,3)$  is an element of *first* row and *third* column. Here,  $A(1,3)=3$ .

Correcting any entry is easy through indexing. Here we substitute  $A(3,3)=9$  by  $A(3,3)=0$ . The result is

```
>> A(3,3) = 0
A =
 1 2 3
 4 5 6
 7 8 0
```

Single elements of a matrix are accessed as  $A(i,j)$ , where  $i$  and  $j$  are strictly positive integers. Zero or negative subscripts are not supported in MATLAB.

### 2.3.4 Colon operator

The colon operator will prove very useful and understanding how it works is the key to efficient and convenient usage of MATLAB. It occurs in several different forms.

Often we must deal with matrices or vectors that are too large to enter one element at a time. For example, suppose we want to enter a vector  $x$  consisting of points  $(0, 0.1, 0.2, 0.3, \dots, 5)$ . We can use the command

```
>> x = 0:0.1:5;
```

The row vector has 51 elements.

### 2.3.5 Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: `linspace`. It is similar to the colon operator (`:`), but gives direct control over the number of points. For example,

```
y = linspace(a,b)
```

generates a row vector  $y$  of 100 points linearly spaced between and including  $a$  and  $b$ .

```
y = linspace(a,b,n)
```

generates a row vector  $y$  of  $n$  points linearly spaced between and including  $a$  and  $b$ . This is equivalent to writing

```
y = a+(0:n-1)/(n-1)*(b-a)
```

the former is more practical, the second is more fundamental. This is useful when we want to divide an interval into a number of subintervals of the same length. For example,

```
>> theta = linspace(0,2*pi,101)
```

divides the interval  $[0, 2\pi]$  into 100 equal subintervals, then creating a vector of 101 elements.

Which is equivalent to the explicit statement

```
>> theta = (0:100)*2*pi/100
```

### 2.3.6 Colon operator in a matrix

The colon operator can also be used to pick out a certain row or column. For example, the statement  $A(m:n,k:l)$  specifies rows  $m$  to  $n$  and column  $k$  to  $l$ . Subscript expressions refer to portions of a matrix. For example,

```
>> A(2,:)
ans =
 4 5 6
```

is the second row elements of  $A$ .

The colon operator can also be used to extract a sub-matrix from a matrix  $A$ .

```
>> A(:,2:3)
ans =
 2 3
 5 6
 8 0
```



$A(:,2:3)$  is a sub-matrix with the last two columns of  $A$ .

A row or a column of a matrix can be deleted by setting it to a *null* vector,  $[\ ]$ .

```
>> A(:,2)=[]
ans =
 1 3
 4 6
 7 0
```

### 2.3.7 Creating a sub-matrix

To extract a *submatrix*  $B$  consisting of rows 2 and 3 and columns 1 and 2 of the matrix  $A$ , do the following

```
>> B = A([2 3],[1 2])
B =
 4 5
 7 8
```

To interchange rows 1 and 2 of  $A$ , use the vector of row indices together with the colon operator.

```
>> C = A([2 1 3],:)
C =
 4 5 6
 1 2 3
 7 8 0
```

It is important to note that the *colon operator*  $(:)$  stands for *all columns* or *all rows*. To create a vector version of matrix  $A$ , do the following

```
>> A(:)
ans =
 1
 2
 3
 4
 5
 6
 7
 8
 0
```

The submatrix comprising the intersection of rows  $p$  to  $q$  and columns  $r$  to  $s$  is denoted by

$A(p:q,r:s)$ .

As a special case, a colon ( $:$ ) as the row or column specifier covers all entries in that row or column; thus

- $A(:,j)$  is the  $j$ th column of  $A$ , while
- $A(i,:)$  is the  $i$ th row, and
- $A(\text{end},:)$  picks out the last row of  $A$ .

The keyword `end`, used in  $A(\text{end},:)$ , denotes the last index in the specified dimension. Here are some examples.

```
>> A
A =
 1 2 3
 4 5 6
 7 8 9
```

```
>> A(2:3,2:3)
ans =
 5 6
 8 9
```

```
>> A(end:-1:1,end)
ans =
 9
 6
 3
```

```
>> A([1 3],[2 3])
ans =
 2 3
 8 9
```

### 2.3.8 Deleting row or column

To delete a row or column of a matrix, use the *empty vector* operator, `[]`.

```
>> A(3,:) = []
A =
1 2 3
4 5 6
```

Third row of matrix A is now deleted. To restore the third row, we use a technique for creating a matrix

```
>> A = [A(1,:);A(2,:);[7 8 0]]
A =
1 2 3
4 5 6
7 8 0
```

Matrix A is now restored to its original form.

### 2.3.9 Dimension

To determine the *dimensions* of a matrix or vector, use the command `size`. For example,

```
>> size(A)
ans =
3 3
```

means 3 rows and 3 columns. Or more explicitly with,

```
>> [m,n]=size(A)
```

### 2.3.10 Continuation

If it is not possible to type the entire input on the same line, use consecutive periods, called an ellipsis ..., to signal continuation, then continue the input on the next line.

```
B = [4/5 7.23*tan(x) sqrt(6); ...
 1/x^2 0 3/(x*log(x)); ...
 x-7 sqrt(3) x*sin(x)];
```

Note that *blank* spaces around +, -, = signs are optional, but they improve readability.

### 2.3.11 Transposing a matrix

The *transpose* operation is denoted by an apostrophe or a single quote ('). It flips a matrix about its main diagonal and it turns a row vector into a column vector.

Thus,

```
>> A'
ans =
 1 4 7
 2 5 8
 3 6 0
```

By using linear algebra notation, the transpose of an  $n \times m$  real matrix  $A$  is the  $m \times n$  matrix that results from interchanging the rows and columns of  $A$ . The transpose matrix is denoted  $A^T$ .

### 2.3.12 Concatenating matrices

Matrices can be made up of sub-matrices. Here is an example. First, let's recall our previous matrix  $A$ .

```
A =
 1 2 3
 4 5 6
 7 8 9
```

The new matrix  $B$  will be,

```
>> B = [A 10*A; -A [1 0 0; 0 1 0; 0 0 1]]
B =
 1 2 3 10 20 30
 4 5 6 40 50 60
 7 8 9 70 80 90
 -1 -2 -3 1 0 0
 -4 -5 -6 0 1 0
 -7 -8 -9 0 0 1
```

### 2.3.13 Matrix generators

MATLAB provides functions that generates elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions `zeros`, `ones`, and `eye`, respectively.

Table 2.4: Elementary matrices

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>eye(m,n)</code>   | Returns an m-by-n matrix with 1 on the main diagonal |
| <code>eye(n)</code>     | Returns an n-by-n square identity matrix             |
| <code>zeros(m,n)</code> | Returns an m-by-n matrix of zeros                    |
| <code>ones(m,n)</code>  | Returns an m-by-n matrix of ones                     |
| <code>diag(A)</code>    | Extracts the diagonal of matrix A                    |
| <code>rand(m,n)</code>  | Returns an m-by-n matrix of random numbers           |

For a complete list of *elementary matrices* and *matrix manipulations*, type `help elmat` or `doc elmat`. Here are some examples:

```
1. >> b=ones(3,1)
 b
 =
 1
 1
 1
```

Equivalently, we can define b as `>> b=[1;1;1]`

```
2. >> eye(3)
 ans
 =
 1 0 0
 0 1 0
 0 0 1
```

```
3. >> c=zeros(2,3)
 c
 =
 0 0 0
 0 0 0
```

In addition, it is important to remember that the three elementary operations of *addition* (+), *subtraction* (-), and *multiplication* (\*) apply also to matrices whenever the dimensions are *compatible*.

Two other important matrix generation functions are `rand` and `randn`, which

generate matrices of (pseudo-)random numbers using the same syntax as eye.

In addition, matrices can be constructed in a block form. With C defined by  $C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ , we may create a matrix D as follows

```
>> D = [C zeros(2); ones(2) eye(2)]
D =
 1 2 0 0
 3 4 0 0
 1 1 1 0
 1 1 0 1
```

### 2.3.14 Special matrices

MATLAB provides a number of special matrices (see Table 2.5). These matrices have interesting properties that make them useful for constructing examples and for testing algorithms. For more information, see MATLAB documentation.

Table 2.5: Special matrices

|                  |                                    |
|------------------|------------------------------------|
| <b>hilb</b>      | Hilbert matrix                     |
| <b>invhilb</b>   | Inverse Hilbert matrix             |
| <b>magic</b>     | Magic square                       |
| <b>pascal</b>    | Pascal matrix                      |
| <b>toeplitz</b>  | Toeplitz matrix                    |
| <b>vander</b>    | Vandermonde matrix                 |
| <b>wilkinson</b> | Wilkinson's eigenvalue test matrix |

## 3 Array operations and Linear equations

### 3.1 Array operations

MATLAB has two different types of arithmetic operations: matrix arithmetic operations and array arithmetic operations. We have seen matrix arithmetic operations in the previous lab. Now, we are interested in array operations.

#### 3.1.1 Matrix arithmetic operations

As we mentioned earlier, MATLAB allows arithmetic operations:  $+$ ,  $-$ ,  $*$ , and  $^$  to be carried out on matrices. Thus,

|                          |                                                                  |
|--------------------------|------------------------------------------------------------------|
| $A+B$ or $B+A$           | is valid if $A$ and $B$ are of the same size                     |
| $A*B$                    | is valid if $A$ 's number of column equals $B$ 's number of rows |
| $A^2$                    | is valid if $A$ is square and equals $A*A$                       |
| $\alpha*A$ or $A*\alpha$ | multiplies each element of $A$ by $\alpha$                       |

#### 3.1.2 Array arithmetic operations

On the other hand, array arithmetic operations or *array operations* for short, are done *element-by-element*. The period character,  $.$ , distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition ( $+$ ) and subtraction ( $-$ ), the character pairs  $(.+)$  and  $(.-)$  are not used. The list of array operators is shown below in Table 3.2. If  $A$  and  $B$  are two matrices of the same size with elements  $\mathbf{A} = [a_{ij}]$  and  $\mathbf{B} = [b_{ij}]$ , then the command

|      |                                   |
|------|-----------------------------------|
| $.*$ | Element-by-element multiplication |
| $./$ | Element-by-element division       |
| $.^$ | Element-by-element exponentiation |

Table 3.1: Array operators

```
>> C = A.*B
```

produces another matrix  $C$  of the same size with elements  $c_{ij} = a_{ij} b_{ij}$ . For example, using the same  $3 \times 3$  matrices,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \text{ and } B = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

we have,

```
>> C = A.*B
C =
 10 40 90
 160 250 360
 490 640 810
```

To raise a scalar to a power, we use for example the command  $10^2$ . If we want the operation to be applied to each element of a matrix, we use  $.^2$ . For example, if we want to produce a new matrix whose elements are the square of the elements of the matrix  $A$ , we enter

```
>> A.^2
ans =
 1 4 9
 16 25 36
 49 64 81
```

The relations below summarize the above operations. To simplify, let's consider two vectors  $U$  and  $V$  with elements  $U = [u_j]$  and  $V = [v_j]$ .

$U.*V$  produces  $[u_1v_1 \ u_2v_2 \dots \ u_nv_n]$

$U./V$  produces  $[u_1/v_1 \ u_2/v_2 \dots \ u_n/v_n]$

$U.^V$  produces  $[u_1^{v_1} \ u_2^{v_2} \dots \ u_n^{v_n}]$



| Operation      | Matrix | Array |
|----------------|--------|-------|
| Addition       | +      | +     |
| Subtraction    | -      | -     |
| Multiplication | *      | .*    |
| Division       | /      | ./    |
| Left division  | \      | .\    |
| Exponentiation | ^      | .^    |

Table 3.2: Summary of matrix and array operations

## 3.2 Solving linear equations

One of the problems encountered most frequently in scientific computation is the solution of systems of simultaneous linear equations. With matrix notation, a system of simultaneous linear equations is written

$$Ax = b$$

where there are as many equations as unknown.  $A$  is a given square matrix of order  $n$ ,  $b$  is a given column vector of  $n$  components, and  $x$  is an unknown column vector of  $n$  components.

In linear algebra we learn that the solution to  $Ax = b$  can be written as  $x = A^{-1}b$ , where  $A^{-1}$  is the inverse of  $A$ .

For example, consider the following system of linear equations

$$\begin{cases} x + 2y + 3z = 1 \\ 4x + 5y + 6z = 1 \\ 7x + 8y = 1 \end{cases}$$

With matrix notation, a system of simultaneous linear equations is written

$$Ax = b$$

In this case, the coefficient matrix is  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$  and the vector of known terms is

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

This equation can be solved for  $x$  using linear algebra. The result is  $x = A^{-1}b$ .

There are typically two ways to solve for  $x$  in MATLAB:

1. The first *naïve* method is to use the matrix inverse, `inv`.

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> b = [1; 1; 1];
>> x =
inv(A)*b
x =
-1.0000
 1.0000
-0.0000
```

2. The second and more correct approach is to use the *backslash* (`\`) operator. The numerical algorithm behind this operator is computationally efficient. This is a numerically reliable way of solving system of linear equations by using a well-known process of Gaussian elimination.

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> b = [1; 1; 1];
>> x
=
A\b
x =
-1.0000
 1.0000
-0.0000
```

This problem is at the heart of many problems in scientific computation. Hence it is important that we know how to solve this type of problem efficiently.

Now, we know how to solve a system of linear equations. In addition to this, we will see some additional details which relate to this particular topic.

### 3.2.1 Matrix inverse

Let's consider the same matrix  $A$ .

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Calculating the inverse of  $A$  manually is probably not a pleasant work; but for a  $3 \times 3$  matrix it is feasible. Here the hand-calculation of  $A^{-1}$  gives as a final result:

$$A^{-1} = \frac{1}{9} \begin{bmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

In MATLAB, however, it becomes as simple as the following commands:

```
>> A = [1 2 3; 4 5 6; 7 8 0];
>> inv(A)
ans =
 -1.7778 0.8889 -0.1111
 1.5556 -0.7778 0.2222
 -0.1111 0.2222 -0.1111
```

which is numerically the same of what expected.

And the determinant of A is computed as

```
>> det(A)
ans =
 27
```

Similarly, most application of numerical linear algebra turns out to be immediate.

### 3.2.2 Matrix functions

MATLAB provides many matrix functions for various matrix/vector manipulations; see Table 3.3 for some of these functions. Use the online help of MATLAB to find how to use these functions.

|      |                                                |
|------|------------------------------------------------|
| det  | Determinant                                    |
| diag | Diagonal matrices and diagonals of a matrix    |
| eig  | Eigenvalues and eigenvectors                   |
| inv  | Matrix inverse                                 |
| norm | Matrix and vector norms                        |
| rank | Number of linearly independent rows or columns |

Table 3.3: Matrix functions

## 4 Programming in MATLAB

### 4.1 Introduction

So far in these examples, all the commands were executed in the Command Window. The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to *create* a file with a list of commands,
2. *save* the file, and
3. *run* the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or *scripts* for short.

This section covers the following topics:

- M-File Scripts
- M-File Functions

### 4.2 M-File Scripts

A *script file* is an external file that contains a sequence of MATLAB statements. Script files have a filename extension `.m` and are often called M-files. M-files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that can accept arguments and can produce one or more outputs.

#### 4.2.1 Examples

Here are two simple scripts.

##### Example 1

Consider the following system of linear equations

$$\begin{cases} x + 2y + 3z = 1 \\ 3x + 3y + 4z = 1 \\ 2x + 3y + 3z = 2 \end{cases}$$

that corresponds to the coefficient matrix is  $A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 3 & 4 \\ 2 & 3 & 3 \end{bmatrix}$  and the vector  $b = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$

Find the solution  $x$  to the system of equations.

**Solution:**

- Use the MATLAB *editor* to create a file: **File** → **New** → **M-file**.
- Enter the following statements in the file:

```
A = [1 2 3; 3 3 4; 2 3 3];
b = [1; 1; 2];
x = A\b
```

- Save the file, for example, example1.m.
- Run the file, in the command line, by typing:

```
>> example1
```

```
x =
-0.5000
 1.5000
-0.5000
```

When execution completes, the variables (A, b, and x) remain in the workspace. To see a listing of them, enter whos at the command prompt.

Note: The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger. The MATLAB editor has numerous menus for tasks such as *saving*, *viewing*, and *debugging*. Because it performs some simple checks and also uses color to differentiate between various elements of codes, this text editor is recommended as the tool of choice for writing and editing M-files.

## Example 2

Plot the following cosine functions,  $y_1 = 2 \cos(x)$ ,  $y_2 = \cos(x)$ , and  $y_3 = 0.5 \cos(x)$ , in the interval  $[0, 2\pi]$ .

This example has been presented in a previous Chapter. Here we put the commands in a file.

- Create a file, say `example2.m`, which contains the following commands:

```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
plot(x,y1,'--',x,y2,'-',x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)', 'cos(x)', '0.5*cos(x)')
title('Typical example of multiple plots')
axis([0 2*pi -3 3])
```

- Run the file by typing `example2` in the Command Window.

### 4.2.2 Script side-effects

All variables created in a script file are added to the workspace. This may have undesirable effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code any complicated applications defining a function in the M-file.

## 4.3 M-File functions

As mentioned earlier, functions are programs (or *routines*) that accept *input* arguments and return *output* arguments. Each M-file function (or *function* or *M-file* for short) has its *own* area of workspace, separated from the MATLAB base workspace.

### 4.3.1 Anatomy of a M-File function

This simple function shows the basic parts of an M-file.

```

function f = factorial(n) (1)
% FACTORIAL(N) returns the factorial of N. (2)
% Compute a factorial value. (3)

f = prod(1:n); (4)

```

The first line of a function M-file starts with the keyword `function`. It gives the function *name* and order of *arguments*. In the case of function `factorial`, there are up to one output argument and one input argument. Table 4.1 summarizes the M-file function.

As an example, for  $n = 5$ , the result is,

```

>> f =
factorial(5)
f =
 120

```

Table 4.1: Anatomy of a M-File function

| Part no. | M-file element            | Description                                                                                                                              |
|----------|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| (1)      | Function definition line  | Define the function name, and the number and order of input and output arguments                                                         |
| (2)      | first lines and help text | A brief summary description of the program, and the instructions for using it. These are displayed when you request <a href="#">Help</a> |
| (3)      | Function body             | Program code that performs the actual computations                                                                                       |

Both *functions* and *scripts* can have all of these parts, except for the *function definition line* which applies to *function* only.

In addition, it is important to note that *function name* must begin with a letter, and must be no longer than the maximum of 63 characters. Furthermore, the name of the text file that you save will consist of the function name with the extension `.m`. Thus, the above example file would be `factorial.m`.

Table 4.2 summarizes the differences between *scripts* and *functions*.

Table 4.2: Difference between scripts and functions

| Scripts                                                                                                                                                                                                                                             | Functions                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>- Do not accept input arguments or return output arguments.</li> <li>- Store variables in a workspace that is shared with other scripts</li> <li>- Are useful for automating a series of commands</li> </ul> | <ul style="list-style-type: none"> <li>- Can accept input arguments and return output arguments.</li> <li>- Store variables in a workspace internal to the function.</li> <li>- Are useful for extending the MATLAB language for your application</li> </ul> |

### 4.3.2 Input and output arguments

As mentioned above, the input arguments are listed inside parentheses following the function name. The output arguments are listed inside the brackets on the left side. They are used to transfer the output from the function file. The general form looks like this

function [outputs] = function\_name(inputs)

Function file can have none, one, or several output arguments. Table 4.3 illustrates some possible combinations of input and output arguments.

Table 4.3: Example of input and output arguments

|                                |                                            |
|--------------------------------|--------------------------------------------|
| function C=FtoC(F)             | One input argument and one output argument |
| function area=TrapArea(a,b,h)  | Three inputs and one output                |
| function [h,d]=motion(v,angle) | Two inputs and two outputs                 |

## 4.4 Input to a script file

When a script file is executed, the variables that are used in the calculations within the file must have assigned values. The assignment of a value to a variable can be done in three ways.

1. The variable is defined in the script file.



2. The variable is defined in the command prompt.
3. The variable is entered when the script is executed.

We have already seen the two first cases. Here, we will focus our attention on the third one. In this case, the variable is defined in the script file. When the file is executed, the user is *prompted* to assign a value to the variable in the command prompt. This is done by using the input command. Here is an example.

```
% This script file calculates the average of points
% scored in three games.
% The point from each game are assigned to a variable
% by using the 'input' command.

game1 = input('Enter the points scored in the first game ');
game2 = input('Enter the points scored in the second game ');
game3 = input('Enter the points scored in the third game ');
average = (game1+game2+game3)/3
```

The following shows the command prompt when this script file (saved as example3) is executed.

```
>> example3
>> Enter the points scored in the first game 15
>> Enter the points scored in the second game 23
>> Enter the points scored in the third game 10

average =
 16
```

The input command can also be used to assign *string* to a variable. For more information, see MATLAB documentation.

## 4.5 Output commands

As discussed before, MATLAB automatically generates a *display* when commands are executed. In addition to this automatic display, MATLAB has several commands that can be used to generate displays or outputs.

Two commands that are frequently used to generate output are: disp and fprintf. The main differences between these two commands can be summarized as follows (Table 4.4).

Table 4.4: disp and fprintf commands

|                |                                                                                                                                                      |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>disp</b>    | <ul style="list-style-type: none"><li>. Simple to use.</li><li>. Provide limited control over the appearance of output</li></ul>                     |
| <b>fprintf</b> | <ul style="list-style-type: none"><li>. Slightly more complicated than disp.</li><li>. Provide total control over the appearance of output</li></ul> |

# 5 Control flow and operators

## 5.1 Introduction

MATLAB is also a *programming language*. Like other computer programming languages, MATLAB has some decision making structures for control of command execution. These decision making or *control flow* structures include for loops, while loops, and if-else-end constructions. Control flow structures are often used in script M-files and function M-files.

By creating a file with the extension `.m`, we can easily write and run programs. We do not need to *compile* the program since MATLAB is an interpretative (not compiled) language. MATLAB has thousand of *functions*, and you can add your own using m-files.

MATLAB provides several tools that can be used to control the *flow* of a program (*script* or *function*). In a simple program as shown in the previous Chapter, the commands are executed one after the other. Here we introduce the flow control structure that make possible to skip commands or to execute specific group of commands.

## 5.2 Control flow

MATLAB has four control flow structures: the if statement, the for loop, the while loop, and the switch statement.

### 5.2.1 The ‘‘if...end’’ structure

MATLAB supports the variants of ‘‘if’’ construct.

- `if ... end`
- `if ... else ... end`
- `if ... elseif ... else ... end`

The simplest form of the if statement is

```
if expression
 statements
```

end

Here are some examples based on the familiar quadratic formula.

1. 

```
discr = b*b - 4*a*c;
if discr < 0
 disp('Warning: discriminant is negative, roots are
 imaginary');
end
```
  
2. 

```
discr = b*b - 4*a*c;
if discr < 0
 disp('Warning: discriminant is negative, roots are
 imaginary');
else
 disp('Roots are real, but may be repeated')
end
```
  
3. 

```
discr = b*b - 4*a*c;
if discr < 0
 disp('Warning: discriminant is negative, roots are
 imaginary');
elseif discr == 0
 disp('Discriminant is zero, roots are repeated')
else
 disp('Roots are real')
end
```

It should be noted that:

- elseif has no space between else and if (one word)
- no semicolon (;) is needed at the end of lines containing if, else, end
- indentation of if block is not required, but facilitate the reading.
- the end statement is required

### 5.2.2 Relational and logical operators

A relational operator compares two numbers by determining whether a comparison is *true* or *false*. Relational operators are shown in Table 5.1.

Table 5.1: Relational and logical operators

| Operator | Description              |
|----------|--------------------------|
| >        | Greater than             |
| <        | Less than                |
| >=       | Greater than or equal to |
| <=       | Less than or equal to    |
| ==       | Equal to                 |
| ~=       | Not equal to             |
| &        | AND operator             |
|          | OR operator              |
| ~        | NOT operator             |

Note that the “equal to” relational operator consists of two equal signs (==) (with no space between them), since = is reserved for the *assignment* operator.

### 5.2.3 The ‘for...end’ loop

In the for ... end loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is

```
for variable = expression
 statements
end
```

Usually, expression is a vector of the form  $i:s:j$ . A simple example of for loop is

```
for ii=1:5
 x=ii*ii
end
```

It is a good idea to indent the loops for readability, especially when they are nested. Note that MATLAB editor does it automatically.

Multiple for loops can be nested, in which case *indentation* helps to improve the readability. The following statements form the 5-by-5 symmetric matrix A with  $(i,j)$  element  $i/j$  for  $j \geq i$ :

```

n = 5; A = eye(n);
for j=2:n
 for i=1:j-1
 A(i,j)=i/j;
 A(j,i)=i/j;
 end
end
end

```

### 5.2.4 The “while...end” loop

This loop is used when the number of *passes* is not specified. The looping continues until a stated condition is satisfied. The while loop has the form:

```

while expression
 statements
end

```

The statements are executed as long as expression is true.

```

x = 1
while x <= 10
 x = 3*x
end

```

It is important to note that if the condition inside the looping is not well defined, the looping will continue *indefinitely*. If this happens, we can stop the execution by pressing **Ctrl-C**.

### 5.2.5 Other flow structures

- . The break statement. A while loop can be terminated with the break statement, which passes control to the first statement after the corresponding end. The break statement can also be used to exit a for loop.
- . The continue statement can also be used to exit a for loop to pass immediately to the next iteration of the loop, skipping the remaining statements in the loop.
- . Other control statements include return, continue, switch, etc. For more detail about these commands, consult MATLAB documentation.

### 5.2.6 Operator precedence

We can build expressions that use any combination of *arithmetic*, *relational*, and *logical operators*. Precedence rules determine the order in which MATLAB evaluates an expression.

We have already seen this in the “Tutorial Lessons”.

Here we add other operators in the list. The precedence rules for MATLAB are shown in this list (Table 5.2), ordered from *highest* (1) to *lowest* (9) precedence level. Operators are evaluated from left to right.

Table 5.2: Operator precedence

| Precedence | Operator                                                                                                                                                                                       |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1          | Parentheses ()                                                                                                                                                                                 |
| 2          | Transpose ( $\prime$ ), power ( $\wedge$ ), matrix power ( $\wedge$ )                                                                                                                          |
| 3          | Unary plus (+), unary minus (−), logical negation ( $\sim$ )                                                                                                                                   |
| 4          | Multiplication ( $\ast$ ), right division ( $\./$ ), left division ( $.\backslash$ ), matrix multiplication ( $\ast$ ), matrix right division ( $\./$ ), matrix left division ( $\backslash$ ) |
| 5          | Addition (+), subtraction (−)                                                                                                                                                                  |
| 6          | Colon operator (:)                                                                                                                                                                             |
| 7          | Less than (<), less than or equal to ( $\leq$ ), greater (>), greater than or equal to ( $\geq$ ), equal to (==), not equal to ( $\sim=$ )                                                     |
| 8          | Element-wise AND, (&)                                                                                                                                                                          |
| 9          | Element-wise OR, ( )                                                                                                                                                                           |

### 5.3 Saving output to a file

In addition to displaying output on the screen, the command `fprintf` can be used for writing the output to a *file*. The saved data can subsequently be used by MATLAB or other softwares.

To save the results of some computation to a file in a text format requires the following steps:

1. Open a file using `fopen`
2. Write the output using `fprintf`
3. Close the file using `fclose`

Here is an example (script) of its use.

```
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op,'Thursday\nFriday\nSaturday\n');
fclose(op);
```

This file (weekdays.txt) can be opened with any program that can read .txt file.

# Appendix: Summary of commands

Table A.1: **Arithmetic operators and special characters**

| Character | Description                                                  |
|-----------|--------------------------------------------------------------|
| +         | Addition                                                     |
| -         | Subtraction                                                  |
| *         | Multiplication (scalar and array)                            |
| /         | Division (right)                                             |
| ^         | Power or exponentiation                                      |
| :         | Colon; creates vectors with equally spaced elements          |
| ;         | Semi-colon; suppresses display; ends row in array            |
| ,         | Comma; separates array subscripts                            |
| ...       | Continuation of lines                                        |
| %         | Percent; denotes a comment; specifies output format          |
| '         | Single quote; creates string; specifies matrix transpose     |
| =         | Assignment operator                                          |
| ()        | Parentheses; encloses elements of arrays and input arguments |
| []        | Brackets; encloses matrix elements and output arguments      |

Table A.2: **Array operators**

| Character | Description                     |
|-----------|---------------------------------|
| .*        | Array multiplication            |
| ./        | Array (right) division          |
| .^        | Array power                     |
| .\        | Array (left) division           |
| ./        | Array (nonconjugated) transpose |



Table A.3: **Relational and logical operators**

| <b>Character</b> | <b>Description</b>          |
|------------------|-----------------------------|
| <                | Less than                   |
| ≤                | Less than or equal to       |
| >                | Greater than                |
| ≥                | Greater than or equal to    |
| ==               | Equal to                    |
| ~=               | Not equal to                |
| &                | Logical or element-wise AND |
|                  | Logical or element-wise OR  |
| &&               | Short-circuit AND           |
|                  | Short-circuit OR            |

Table A.4: **Managing workspace and file commands**

| <b>Command</b> | <b>Description</b>                                |
|----------------|---------------------------------------------------|
| cd             | Change current directory                          |
| clc            | Clear the Command Window                          |
| clear (all)    | Removes all variables from the workspace          |
| clear x        | Remove x from the workspace                       |
| copyfile       | Copy file or directory                            |
| delete         | Delete files                                      |
| dir            | Display directory listing                         |
| exist          | Check if variables or functions are defined       |
| help           | Display help for MATLAB functions                 |
| lookfor        | Search for specified word in all help entries     |
| mkdir          | Make new directory                                |
| movefile       | Move file or directory                            |
| pwd            | Identify current directory                        |
| rmdir          | Remove directory                                  |
| type           | Display contents of file                          |
| what           | List MATLAB files in current directory            |
| which          | Locate functions and files                        |
| who            | Display variables currently in the workspace      |
| whos           | Display information on variables in the workspace |

Table A.5: **Predefined variables and math constants**

| <b>Variable</b> | <b>Description</b>                 |
|-----------------|------------------------------------|
| ans             | Value of last variable (answer)    |
| eps             | Floating-point relative accuracy   |
| <b>i</b>        | Imaginary unit of a complex number |
| Inf             | Infinity ( $\infty$ )              |
| eps             | Floating-point relative accuracy   |
| <b>j</b>        | Imaginary unit of a complex number |
| NaN             | Not a number                       |
| pi              | The number $\pi$ (3.14159 . . .)   |

Table A.6: **Elementary matrices and arrays**

| <b>Command</b> | <b>Description</b>                              |
|----------------|-------------------------------------------------|
| eye            | Identity matrix                                 |
| linspace       | Generate linearly space vectors                 |
| ones           | Create array of all ones                        |
| rand           | Uniformly distributed random numbers and arrays |
| zeros          | Create array of all zeros                       |

Table A.7: **Arrays and Matrices: Basic information**

| <b>Command</b> | <b>Description</b>                 |
|----------------|------------------------------------|
| disp           | Display text or array              |
| isempty        | Determine if input is empty matrix |
| isequal        | Test arrays for equality           |
| length         | Length of vector                   |
| ndims          | Number of dimensions               |
| numel          | Number of elements                 |
| size           | Size of matrix                     |

Table A.8: **Arrays and Matrices: operations and manipulation**

| <b>Command</b> | <b>Description</b>                        |
|----------------|-------------------------------------------|
| cross          | Vector cross product                      |
| diag           | Diagonal matrices and diagonals of matrix |
| dot            | Vector dot product                        |
| end            | Indicate last index of array              |
| find           | Find indices of nonzero elements          |
| kron           | Kronecker tensor product                  |
| max            | Maximum value of array                    |
| min            | Minimum value of array                    |
| prod           | Product of array elements                 |
| reshape        | Reshape array                             |
| sort           | Sort array elements                       |
| sum            | Sum of array elements                     |
| size           | Size of matrix                            |

Table A.9: **Arrays and Matrices: matrix analysis and linear equations**

| <b>Command</b> | <b>Description</b>                         |
|----------------|--------------------------------------------|
| cond           | Condition number with respect to inversion |
| det            | Determinant                                |
| inv            | Matrix inverse                             |
| linsolve       | Solve linear system of equations           |
| lu             | LU factorization                           |
| norm           | Matrix or vector norm                      |
| null           | Null space                                 |
| orth           | Orthogonalization                          |
| rank           | Matrix rank                                |
| rref           | Reduced row echelon form                   |
| trace          | Sum of diagonal elements                   |