Chapter 1
Basic Operations

This chapter gives an overview of MATLAB and compares MATLAB with a scientific calculator. The chapter gives also an overview of basic arithmetic operations and functions as well as a short introduction to matrices and matrix manipulation.

It is supposed that you have already installed MATLAB on your computer. When you start MATLAB, the MATLAB desktop opens, as shown in Fig. 1.1 (or something similar, depending on your MATLAB version). In this first chapter we refer only to the Command Window, where the special >> prompt appears. The other windows have the following meaning:

- The Workspace Window contains a list of variables that are in use in the working session.
- The Command History contains the list of all commands you have typed in the command window.
- The Current Folder window shows the list of the files contained the folder you are working on.

When the prompt >> is visible, this means that MATLAB is waiting for a command. You can quit MATLAB at any time in either of the following ways:

1. Select Exit MATLAB from the desktop File menu.
2. Enter `quit` or `exit` after the command window prompt >>, and press the Enter key.

Alternatively, select File with the mouse from the top menu bar, and then exit MATLAB.

Observe that the tab above the Workspace shows the Current Directory Window. For example, in the Windows operating system, the path might be as follows: C:\MATLAB\Work, indicating that directory "Work" is a subdirectory of the main directory "MATLAB," which is installed in drive C. Clicking on the arrow in the Current Directory Window shows a list of recently used paths. Clicking on the button to the right of the window allows the user to change the current directory. Knowing which is the current path is fundamental: from the Command Window you

M. Borgo et al., *MATLAB for Psychologists*,
DOI 10.1007/978-1-4614-2197-9_1, © Springer Science+Business Media, LLC 2012

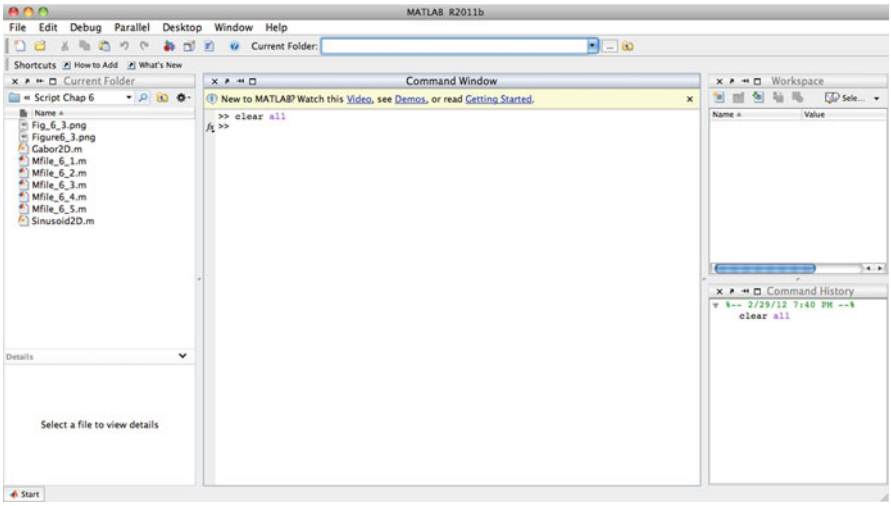

Fig. 1.1 The MATLAB desktop. MATLAB release 2011b

have access to the file stored in the given directory. It is, of course, possible to change your working directory.

Before continuing our introduction to MATLAB, we want to highlight a very useful window: the HELP Window. This window is the most useful window for beginning MATLAB users—and for expert users as well: select Help▶ PRODUCTHELP from the top bar menu. The Help Window has most of the features you would see in any web browser, including clickable links, a back button, and a search engine. All MATLAB commands and functions are explained with examples: you have simply to search for the desired word.

Now let us begin with a description of the MATLAB language. The word "MATLAB" is the concatenation of the words MATrix LABoratory, meaning that MATLAB is an interactive software system for numerical computation, especially designed for computations with matrices. Before going into the details of matrix computations, let us first see how to use MATLAB to do simple arithmetic operations: Type 1+1 after the >> prompt, followed by Enter; that is, press the Enter key, as indicated by <ENTER>

```
>> 1+1                    <ENTER>
```

MATLAB gives its quick answer by displaying the following message:

```
ans =
     2
```

You can perform other arithmetic operations, such as multiplication, subtraction, and division, and MATLAB always returns the correct result. If such is not the case, there is certainly something wrong with what you typed. For example, you can try the following operations (type the operation after the >> prompt followed by Enter):

To TYPE after prompt >> followed by Enter	MATLAB answer	Meaning of the operation
`35*12`	`ans = 420`	Multiplication
`2/45`	`ans = 0.0444`	Division
`4-1`	`ans = 3`	Subtraction
`2^3`	`ans = 8`	Exponentiation

Note that to type numbers such as the Avogadro's number 6.023×10^{23}, you can either write the *expression* `6.023*10^23` or you can *represent* the number in *scientific notation*. To enter Avogadro's number in scientific format, type `6.023e23`, where 6.023 is the mantissa and 23 is the exponent. Mantissa and exponent must be separated by the letter e (or E):

```
>> 6.023*10^23        <ENTER>
ans =
   6.0230e+023
```

```
>> 6.023e23           <ENTER>
ans =
   6.0230e+023
```

Such numbers are also defined to be *floating point.*

MATLAB warns you in the case of in invalid operation or "unexpected" results. What do you think MATLAB will show us if we type 12/0 or 0/0? Let's try it:

To TYPE after prompt >> followed by Enter	MATLAB answer	Meaning of the answer
`12/0`	`ans = Inf`	You should not divide by zero, but if you do, the result is **Infinity**
`0/0`	`ans = NaN`	Unable to find the answer, so the result is **NaN = Not a Number**

`11+`	`??? 11+`
	`  |`
	`Error: Expression or statement is incomplete or incorrect`
		If you want to perform this operation, you must complete the expression with another term

As you can see, MATLAB is unable to "stay quiet." It quickly answers your commands by displaying something in the command window. In the previous cases, the answer was a special value such as `Inf` *(Infinity)* or `NaN` *(Not a Number)*. You can use these special values on their own, typing, for example:

```
>> 12/Inf          <ENTER>
ans =
     0
```

MATLAB can be used as a scientific calculator, combining several operations in one expression. The computation of the expression is based on well-known mathematical rules. In particular, some operations are performed before others, based on precedence rules, which are given in the following table:

Precedence	Operator
1	Parentheses (round brackets)
2	Exponentiation, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

If you want to know the result of the operation $\{2+[5*3/(7-5)^2]/3\}$ you have to type:

```
>> (2+(5*3/(7-5)^2)/3)          <ENTER>
ans =
    3.2500
```

In this example, MATLAB first calculates (7−5) = 2, then it squares 2^2 = 4, then it performs the multiplication 5*3 = 15 (multiplication left to right), and then divides the result by the previously computed result, i.e., 15/4 = 3.75. The result in brackets is divided by 3 (3.75/3 = 1.25) and then added to 2, giving the result. Note that in MATLAB, parentheses are always given by round brackets.

MATLAB was developed for scientists, and for this reason you can find built-in operations and functions that are more advanced than the ones we have just looked at. Considering MATLAB as a sort of scientific calculator, you can engage the "cosine button" simply by typing:

```
  >> cos(36)          <ENTER>
ans =
    -0.1280
```

Other common functions are reported in the following table. Type the operation after the >> prompt followed by Enter:

To TYPE after prompt >> followed by Enter	MATLAB answer	Meaning of the operation
`cos(12)`	`ans = 0.8439`	Cosine of the element in parentheses
`sin(12)`	`ans = -0.5366`	Sine of the element in parentheses
`tan(4)`	`ans = 1.1578`	Tangent of the element in parentheses
`exp(3)`	`ans = 20.0855`	Exponential of the element in parentheses
`log(10)`	`ans = 2.3026`	Natural logarithm of the element in parentheses
`log10(12)`	`ans = 1.0792`	Base-10 logarithm of the element in parentheses

We will see in the rest of the book the possibility using many other functions. Just to introduce some: statistical functions, interpolation functions, linear-algebraic functions, functions for images and sound elaboration, and last but not least, your own custom-created functions!

We conclude by giving some hints on creating and editing command lines:

- You can select (and edit) previous commands you have entered using the up-arrow and down-arrow keys. Remember to press Enter to execute the command.
- MATLAB has a useful editing feature called *smart recall*. Just type the first few characters of the command you want to recall, e.g., type the characters `lo` and press the up-arrow key—this recalls the most recent command starting with `lo`. The result might be, for example, `log(10)` or `log10(12)`.

Variables

Thus far, we have seen the use of MATLAB as a scientific calculator. However, MATLAB is much more than a calculator, and the main difference is the possibility to use "variables." In a scientific calculator we can save and recall a single number only. In MATLAB, in contrast (as in other programming languages), we can store and recall virtually an infinity of different values called *variables*. A *variable* is a sort of box, having a certain shape, a certain dimension, with a label naming it. In such a box you can put the (virtual) item you need, for example a number, an image, and so on.

Suppose you want to save a number representing your age. You can create your own variable and store it by simply typing the following command:

```
>> age=22          <ENTER>
age =
    22
```

The symbol `age` is the variable name (the box name), which contains the number 22. Each time you recall (type) such a name, the content of the variable is used; in this simple case, it is displayed. Type again the *variable* name:

```
>> age          <ENTER>
age=
     22
```

You can define other variables, for example the number of your friends. Just type:

```
>> Nfriends = 132          <ENTER>
Nfriends =
   132
```

At this stage, you may wonder about the shape of the box or its volume. The answer is not straightforward. However, by typing the `whos` command, MATLAB prompts all the variables currently active in the working session:

```
>> whos          <ENTER>

Name        Size   Bytes  Class    Attributes
age         1×1        8  double
Nfriends    1×1        8  double
```

The `whos` command gives you a list of all the variables created in the workspace together with their characteristics. In order to understand the meaning of such characteristics, consider the analogy between variable and box, as presented in the following table:

Variable	Box	Visual interpretation
Name	Name of the box	
Size	Number of objects you have put in (in the previous case, just one object, i.e., 1×1).	
Bytes	Total volume of the box. This is the number of objects multiplied by the dimension of each (to store a number you need 8 bytes)	
Class	Type of object you can put inside the box (in the previous case, it is a number stored with *double precision*)	
Attributes	Other information	

Note that the variables list obtained using the command `whos` can readily be seen in the Workspace Window (see Fig. 1.1).

One nice thing that MATLAB does when you create a variable is that it automatically selects the most suitable type of box for the variable. You need, however, to know a few simple rules about variable names:

1. The variable name must start with a letter.
2. It may consist only of the letters a–z, the digits 0–9, and the underscore (_). You cannot have a name with spaces or others symbols (such as +, ^, *).
3. MATLAB is case-sensitive, which means that it distinguishes between upper- and lowercase letters. So `age` is different from `AgE` or `Age`.

Try to create the following variables by typing them after the >> prompt followed by Enter: `N-friends=12` , `$aDay=60`, `3rd_classified=11`. What happens, and why? MATLAB gives you the following error:

```
??? $aDay=60
    |
Error: The input character is not valid in MATLAB statements or expressions.
```

Obviously, in these examples we didn't follow the aforementioned rules (use of the character—and $, beginning the name with a number).

MATLAB has a few predefined variable names. Some of these are presented in the following table:

To TYPE after prompt >> followed by Enter	MATLAB answer	Value contained in the variable
`Pi`	`ans=3.1416`	π
`Esp`	`ans=2.2204e-016`	Floating-point relative accuracy, i.e., the distance from 1.0 to the next largest double-precision number
`j`	`ans=0+1.0000i`	Imaginary unit, i.e., sqrt(−1), used to enter complex numbers
`I`	`ans=0+1.0000i`	Imaginary unit, i.e., sqrt(−1), used to enter complex numbers
`NaN`	`ans=NaN`	Not a number
`Inf`	`ans=inf`	Infinity

You can redefine a variable by simply assigning it a new value:

```
>> pi=12                    <ENTER>
pi =
    -12
>> pi                       <ENTER>
pi =
    -12
```

Once you have inserted a new value, you cannot recall the previous one. However, in the special case of predefined variables, you can clear the redefined variable, and the predefined variable is restored. To clear variables you use the command `clear` followed by the variable name (or a list of them). Let's try:

```
>> clear pi                 <ENTER>
```

MATLAB doesn't give you an answer. However, the command has been executed. Type the *pi* variable again, and MATLAB will return the value of π:

```
>> pi                       <ENTER>
ans =
    3.1416
```

The command `clear` can be followed by the specification `all`, and all the variables stored in the workspace are deleted. To test whether this is indeed the case, type the `whos` command:

```
>> clear all                    <ENTER>
>> whos                         <ENTER>
>>
```

Note that you receive no answer from MATLAB because there is nothing to display. At the same time, you can see that the Workspace Window (Fig. 1.1) is empty.

With variables you can type complex expressions and store the result. Let's try:

```
>> number=13;                   <ENTER>
>> a=14;                        <ENTER>
>> c=pi*((number+a/2)/10);      <ENTER>
```

MATLAB doesn't give an answer because you ended the command with the semicolon (;) which prevents the value of `number` from being echoed on the screen. However, `number` still has the value 13, as you can see by entering its name without a semicolon (or looking at the Workspace Window):

```
>> number                       <ENTER>
number =
    13
>> c                            <ENTER>
c =
   6.2832
```

Thinking in a Matrix Way

Our first question about matrices is, "What is a Matrix?" We are not talking about the film, the sequels, the comic books, or the video games. For us, a matrix isn't a complex computer simulation that you have to do battle with to save humanity. However, you can choose to continue to read the book (in analogy to the blue pill in the film that allows you to lead your normal life) to learn how use MATrix LABoratory to create innovative experiments, thereby changing the world with your discoveries.

In MATLAB, a matrix is a rectangular array of numbers, as shown in the following Fig. 1.2.

The horizontal lines of a matrix are called rows, and the vertical lines are called columns. The numbers in the matrix are called entries. A matrix with m rows and n columns is called an $m \times n$ matrix. A matrix one of whose dimensions equals one is often called a vector. An $m \times 1$ matrix (one column and m rows) is called a column vector, and a $1 \times n$ matrix (one row and n columns) is called a row vector.

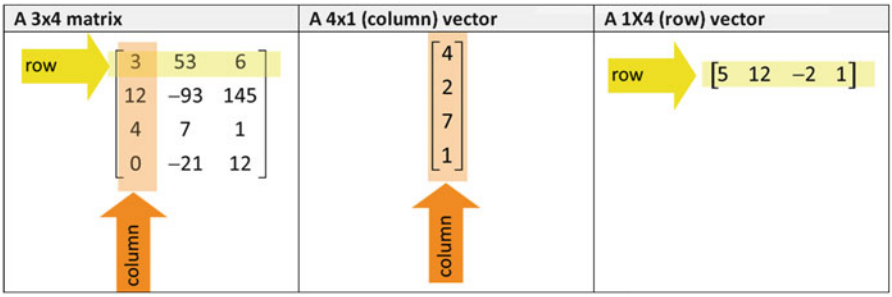

Fig. 1.2 Matrices of various dimensions

If you are familiar with the spreadsheet software Excel, you can imagine each Excel worksheet as a matrix, with rows and columns.

Now let's try to define some matrices and vectors in MATLAB. Type the following statements as written:

```
>> a=[3,5,7,8]                    <ENTER>
a =
      5    12    -2    1
>> b=[4;2;7;1]                    <ENTER>
b =
      4
      2
      7
      1
>> c=[3, 53,6;12,-93,145;4,7,1;0,-21,12]        <ENTER>
c =
      3   53      6
     12   -93     145
      4   7       1
      0   -21     12
>> whos                           <ENTER>
   Name     Size          Bytes      Class     Attributes

   a        1x4           32         double
   b        4x1           32         double
   c        4x3           96         double
   number   1x1           8          double
```

As you can see, after the command `whos`, each variable is displayed together with its size expressed in rows × columns. (Note: if you have used other variables previously, your list of variables could be different). Note that a scalar value, like the variable `number`, can be considered a 1×1 matrix.

Sometimes we need to know the size of the variables only, instead of the full list of their properties. In this case, the function `size(c)` returns the number of rows and columns of the variable `c`:

```
>> size(c)
ans =
     4    3
```

Another useful function is `length(c)`, which returns the length of the vector `c`. If `c` is a matrix, the function `length` returns the number of rows only:

```
>> length(c)
ans=
    4
```

To put data into a matrix, you must type the values within square brackets, separated by *spaces* or *commas* for different elements in a row, while the *semicolon* (;) is used to indicate the end of the row. Note that the number of elements must be the same in each row:

```
>> x=[ 1 2 3; 2 5 7]          <ENTER>
x =
     1     2     3
     2     5     7
```

If you have not put the same number of elements in each row, MATLAB displays an error:

```
>> x = [2 3; 2 5 7];
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

As you can see, MATLAB is not a wizard who tries putting the missing element in the right place. MATLAB does not know whether you want to put the element 2 and 3 in the first and second columns or in the second and third columns respectively.

You can use a matrix or a vector to implement another variable. For example, type in the following statements:

```
>> x = [3 2 1];               <ENTER>
>> y = [6,7,8];               <ENTER>
>> z1 = [x -y];               <ENTER>
>> z2 = [x; -y];              <ENTER>
```

Can you work out what `z1 and z2` will look like before displaying them? In the following table we present other examples showing how to use variables already implemented to create new variables:

Mathematical representation	MATLAB (type after the prompt >> followed by Enter)	Dimension
$M = \begin{bmatrix} 3 & 12 & \pi \end{bmatrix}$	`M=[3,12,pi];`	1×3 Row vector

$N = \begin{bmatrix} 3 & 12 & \pi \\ 8 & 9 & 10 \end{bmatrix}$	`N=[3,12,pi; 8,9,10];`
	Or equivalently, if you have already inserted M:
	`N=[M; 8,9,10];`
		2×3 Matrix

$P = \begin{bmatrix} 4 \\ 2 \\ -1 \end{bmatrix}$	P=[4;2;−1]	3×1 Column vector
$Q = \begin{bmatrix} 4 & -4 \\ 2 & -2 \\ -1 & 1 \end{bmatrix}$	Q=[4, −4;2, −2;−1, 1]; Or equivalently, if you have already inserted P: Q=[P;−P];	3×2 Matrix

If you do not specify any variable content (i.e., any values inside the square brackets), MATLAB creates a variable of size zero with no value, or more precisely, a matrix of dimension 0×0 with no value in it.

```
>> y = [ ];                    <ENTER>
>> whos y                     <ENTER>
Name        Size        Bytes      Class      Attributes

y           0x0         0                     double
```

The Workspace Browser in the desktop provides a handy visual representation of the workspace. By clicking a variable in the Workspace Browser, we open the Array Editor, which can be used to view and change values.

The entry that lies in the *i*th row and the *j*th column of a matrix is typically referred to as the (*i*,*j*), or (*i*,*j*)th entry of the matrix. For example, the (3,2) entry of matrix Q in the table above is 1. In mathematical format, it is usually written as $Q_{3,2}$, while in MATLAB you can access to the matrix entries in this way:

```
>> Q(3,2)                     <ENTER>
ans =
     1
```

Note the use of parentheses. For indexing you use parentheses, whereas to define a matrix, you use square brackets; otherwise, you get an error:

```
>>Q[2,3]
??? Q[2,3]
     |
Error: Unbalanced or unexpected parenthesis or bracket.
```

In the following table you can find other examples:

Mathematical representation	MATLAB (type after the prompt >> followed by Enter)

$Q_{3,1}$ is equal to −1	`>>Q(3,1)`
	`ans=`
	`-1`

$N_{2,2}$ is equal to 9	`>> N(2,2)`
	`ans=`
	`9`

$M_{1,3}$ is equal to π	`>> M(1,3)`
	`ans=`
	`3.1416`

$P_{2,1}$ is equal to 2	`>> N(2,1)`
	`ans=`
	`2`

If P and M are two vectors, it is possible to refer to their entries by referencing only their single dimension, i.e., you can type M(3) instead of M(1,3), and N(2) instead of N(2,1).

If you refer to an element in a nonexistent position, MATLAB gives you an alert:

```
>> Q(3,3)                    <ENTER>
??? Index exceeds matrix dimensions.
```

What happens if you want to address more than one element at time? This is possible in MATLAB using a vector (or a matrix) in the indexing place to express the selected rows or columns:

```
>> Q([1,3],2)                <ENTER>
ans =
    -4
     1
```

How many values do you expect MATLAB to display when you type Q([1,3],[1,2]) ? Two or Four? Let's try:

```
>> Q([1,3],[1,2])            <ENTER>
ans =
     4    -4
    -1     1
```

The answer is four, because MATLAB shows the values in the positions given by each combination of the specified rows and columns, i.e., Q_{11}, Q_{12}, Q_{31}, Q_{32}.

Now suppose you have a large matrix from which you want to extract elements going from the ith row to the jth row in the second column. MATLAB offers a very efficient way to this, namely the colon (*:*) operator. Before seeing how it works, let us generate a new matrix:

```
>> x=[1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]     <ENTER>
```

```
x =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15
```

Now type:

```
>> i=2; j=4;                    <ENTER>
>> x(i:j,2)                     <ENTER>
ans =
     5
     8
    11
```

Note that more than one command has been typed on the first line. This can be done by separating commands with a semicolon. In addition, note that MATLAB displays exactly the values from the second row to the fourth row in the second columns. This is equivalent to:

```
>> x([2 3 4],2)                 <ENTER>
ans =
     5
     8
    11
```

As a matter of fact, using the colon operator is equivalent to generating a vector going from a given value to another one, possible using a prescribed increment (step). The rule is:

Start:Step:Stop

Type the following commands:

To TYPE after prompt >> followed by Enter	MATLAB answer	Meaning of the operation
2:5:25	`ans = 2 7 12 17 22`	Generate a vector going from 2 to 25 incremented by 5. Note that 22 + 5 = 27, which is greater than 25. MATLAB will generate numbers until it reaches or exceeds the Stop value (i.e., 25)
`i:j`	`ans = 2 3 4`	Generate a vector going from 2 to 4. Here the step value is not specified, and MATLAB uses the default value 1
`10:-3:-5`	`ans = 10 7 4 1 -2 -5`	Generate a vector going from 10 to −5, increasing the first value by −5. This is equivalent to generating a vector of decreasing values

You now have three equivalent ways of accessing the third-row entries of x:

```
>> x(3,[1 2 3])              <ENTER>
ans =
     7    8    9
>> x(3,1:3)                  <ENTER>
ans =
     7    8    9
>> x(3,:)                    <ENTER>
ans =
     7    8    9
```

In the last case, you do not need to specify the start and stop values when you use the colon operator. MATLAB assumes that you mean the entire row. Analogously, if you need the first column only, you can type either `x([1,2,3,4,5],1)` or `x([1:5],1)` or `x(:,1);`

By using the colon operator together with the empty array, we are able to delete entire rows or entire columns. For example, to delete the entire second column of x and then its third and fourth rows, type:

```
>> x(:,2) = [ ]              <ENTER>
x =
    1     3
    4     6
    7     9
   10    12
   13    15
>> x([3,4],:) =[ ]           <ENTER>
x =
    1     3
    4     6
   13    15
```

Note that you cannot delete a single entry in a matrix because that would lead to an ambiguity in its dimensions. So a statement like `x(1,2)=[ ]` returns an error:

```
>> x(1,2)=[ ]
??? Subscripted assignment dimension mismatch.
```

We conclude this paragraph by mentioning a way to create a matrix using indexing. In contrast to other computer languages, in MATLAB we do not need to `declare a variable` (i.e., tell to MATLAB what type of variable, how large it is, etc.) before using it. MATLAB creates the variable on the fly. So if you want to insert the response time to the stimulus number 10, you can simply type:

```
>> AnsTime(10)=1.34
AnsTime =
        0      0      0      0      0      0      0      0
0    1.3400
```

MATLAB automatically creates the variable called AnsTime and put the number you have entered in position 10. The unspecified values are filled with zeros by default.

Operations

There are operations that can be applied to modify the contents of a matrix without changing the number of elements. These operations are *matrix addition*, *scalar multiplication*, and *transposition*. These form the basic techniques for dealing with matrices, as displayed in the following table:

Operation	Definition	Math example	Matlab example

Addition (subtraction)	The result of **A**+**B** or (**A**−**B**) is calculated entrywise, i.e., the element $B_{i,j}$ is added to (subtracted from) the element in $A_{i,j}$	$A=\begin{bmatrix}1 & 5\\ 2 & 3\end{bmatrix}, B=\begin{bmatrix}2 & 3\\ 4 & 1\end{bmatrix}$ $A+B=\begin{bmatrix}3 & 8\\ 6 & 4\end{bmatrix}$ $A-B=\begin{bmatrix}-1 & 2\\ -2 & 2\end{bmatrix}$	`>>A=[1,5;2,3];`
			`>>B=[2,3;4,1];`
			`>>A+B`
			`ans =`
			`3 8`
			`6 4`
			`>> A-B`
			`ans =`
			`-1 2`
			`-2 2`

Scalar multiplication	The multiplication of a scalar (= number) ***s*** by a matrix **C** is obtained by multiplying every entry of **C** by ***s***	$C=\begin{bmatrix}3 & 2\\ 4 & 1\end{bmatrix}, s=4$ $s\cdot C=\begin{bmatrix}12 & 8\\ 16 & 4\end{bmatrix}$	`>>C=[3,2;4,1];`
			`>>s=4;`
			`>>s*C`
			`ans =`
			`12 8`
			`16 4`

Transposition	The transpose of an m×n matrix **D** is an n×m matrix denoted by $\mathbf{D}^{\mathrm{T}}$ obtained by turning rows into columns and columns into rows	$D=\begin{bmatrix}3 & 12 & 2\\ 8 & 9 & 10\end{bmatrix}$ $D^T=\begin{bmatrix}3 & 8\\ 12 & 9\\ 2 & 10\end{bmatrix}$	`>> D=[2,12,2;8,9,10];`
			`>> D'`
			`ans =`
			`3 8`
			`12 9`
			`2 10`

Multiplication of a matrix by another matrix is more complicated. *Multiplication* of two matrices is well defined only if the number of columns of the left-hand matrix is the same as the number of rows of the right-hand matrix. Matrix multiplication may seem complex, and perhaps you will not use it very often. However, it turns out to be useful when one of the matrices is a vector, so we give you the following definition:

If **A** is an $m\times n$ matrix and **B** is an $n\times p$ matrix, then their *matrix product* **AB** is the $m\times p$ matrix whose entries are given by the following equation.

$$(AB)_{i,j}=\sum_{r=1}^{n}A_{i,r}B_{r,j}$$

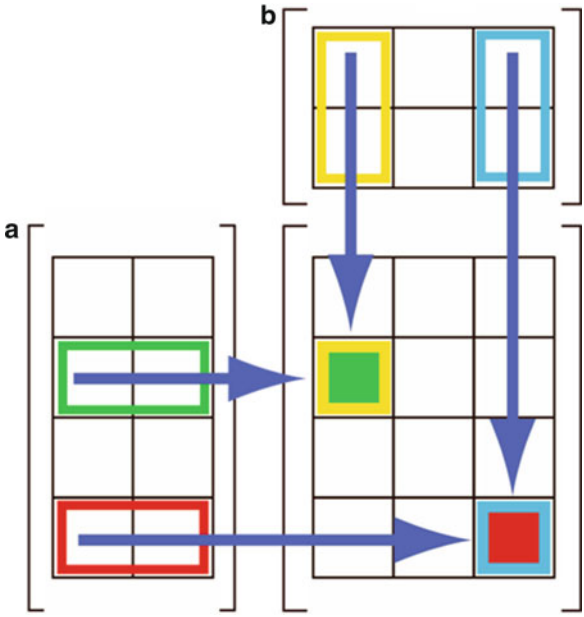

Fig. 1.3 Visual interpretation of matrix product

The equation means that each element of the ith row of **A** is multiplied successively by each element of the jth column of **B**. In Fig. 1.3 you can see a visual interpretation of the previous equation.

We reiterate that a matrix product can be performed only if the number of columns in the first matrix (**A**) is equal to the number of rows of the second matrix (**B**). For example, using the matrices entered in the previous table, it is not possible to perform the product **DC** (try to verify this with MATLAB):

```
>> D*C
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

However, you can compute the product CD. The result is a matrix having the same number of rows as the first matrix (**C**) and the same number of columns as the second matrix (**D**). We see, then, that in general, for two matrices $\mathbf{CD} \neq \mathbf{DC}$, and indeed, one of these products might not even be defined. But even if **C** and **D** are square matrices, it is generally the case that $\mathbf{CD} \neq \mathbf{DC}$:

```
>> C*D                          <ENTER>
ans =
    22    54    26
    16    57    18
```

An element-by-element operation similar to matrix summation is also available in MATLAB. Let's say that we want to multiply the elements of **A** in position i,j by the elements of **B** in the same i,j position using the *element-by-element operators*, as shown in the following table:

Description	MATLAB operator	Example

Element-by-element Multiplication	`.*`	`>> A.*B`
		`ans=`
		`2 15`
		`8 3`

Element-by-element Right division	`./`	`>> A./B`
		`ans =`
		`0.5000 1.6667`
		`0.5000 3.0000`

Element-by-element Left division	`.\`	`>> A.\B`
		`ans =`
		`2.0000 0.6000`
		`2.0000 0.3333`

Element-by-element Exponentiation	`.^`	`>> A.^B`
		`ans =`
		`1 125`
		`16 3`

For those who are familiar with matrix equations, MATLAB has a huge number of other possible operations. Here are some of the basic functions: The inverse function *inv(A)*, the determinant function *det(A)*, the eigenvalue function *eig(A)*, the singular value decomposition function *svd(A)*, the LU factorization *lu(A)*.

Summary

- MATLAB can be thought of as a scientific calculator: you can perform operations from simple to complex, simply by typing them into the command line of the command window; operations are calculated immediately.
- The six arithmetic operators for scalars are `+ - * \ / and ^`. They operate according to rules of precedence. Parentheses have the highest precedence.
- To store numbers or operation results you need variables.
- Variable names consists only of letters, digits, and underscores, and must start with a letter. MATLAB interprets uppercase and lowercase as different letters (e.g., `AgE` is different from `age`).
- The command `whos` lists the variable in the workspace. To delete variables use `clear` followed by the name of variables, or alternatively, `clear all` to clear every variable.

- MATLAB refers to all variables as matrices:
  - An N×M matrix is an array having N rows and M columns.
  - A vector can be a 1×N matrix (row vector) or an N×1 matrix (column vector)
  - A scalar is a 1×1 matrix
- Vectors and matrices are entered in *square brackets*. Elements are separated by spaces or commas. Rows are separated by semicolons.
- An element of a matrix is referred to by a pair of numbers in *parentheses* indicating its position. An element of a vector can be referenced using simply a number. A range of elements can be referred to using vectors instead numbers in parentheses.
- The Colon operator is equivalent to generating a vector going from one value to another, possibly using a specified increment (step): start:step:stop
- Pay attention when using Matrix operations especially to the dimensions of the matrices involved.
- The basic matrix operations are addition (+), scalar multiplication (*), and transposition (‘).
- Element-by-element operations between matrices of the same dimension can be carried out using the operators `./ .\ .* and .^ .`

Exercises

1. Evaluate the following expressions:

Mathematical expression	Solution (to type after prompt >> followed by Enter)	Result
`2·4/12*2`	`2*4/12*2^2`	`ans=2.6667`
`2·3+5/2-7.`	`2*3+5/2-7.5`	`ans=1`
$\sqrt{16}$	`sqrt(16)`	`ans=4`
$\frac{13\cdot 3}{12-4}$	`(13*3)/(12-4)`	`ans=4.875`
$\frac{(2\pi+4)-2\cdot\sin\left(\frac{\pi}{2}\right)}{12-3}$	`((2*pi+4)-2*sin(pi/4))/(12-3)`	`ans=0.9854`
$\frac{22+12}{12/3+4}$	`Try by yourself. The result must be the same`	`ans=4.2500`
$\sqrt{(2+3)^2-2}$	`Try by yourself. The result must be the same`	`ans=4.7958`

2. Create in MATLAB the following variables:

Mathematical expression	Solutions (to type after prompt >> followed by Enter)
$Mat1 = \begin{bmatrix} 3 & 4.56 & 8 & 7 \\ 12 & 2 & 1 & 3 \\ \pi & 34 & 2 & 3 \end{bmatrix}$	`Mat1=[3, 4.56, 8, 7; 12, 2, 1, 3; pi, 34, 2, 3];`

$Mat2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}$	`Mat2=[1, 0, 0; 0, 0, 0; 0, 0, 0; 0, 0, 3];`
	or
	`Mat2=1;`
	`Mat2(4,3)=3;`

$Mat3 = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 & 14 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 \end{bmatrix}$	`Mat3=[2,4,6,8,10,12,14; 3,6,9,12,15,18,21];`
	or
	`Mat3=2:2:14;`
	`Mat3(2,:)=3:3:21;`

3. c is a variable containing the second and the third elements of the third row of Mat1.
   - b is a variable containing the first and second elements of the second row of Mat3.
   - Create a matrix called bbcc with b on top of c.
   - Use the first column of Mat1 and the transpose of the last row of Mat2 to create a new matrix called nice. Multiply the result by the third element in the first row of Mat3.
   - SubMat1 is a matrix obtained from the second and fourth columns of Mat3.
   - SubMat2 is a matrix obtained from the first and last columns of Mat3.
   - NewMat is a 2x4 matrix obtained by using SubMat1 in the first two columns, and SubMat2 in the second two columns.

Solutions:

To type after prompt >> followed by Enter	Display by MATLAB

`c=Mat1(1,2:3)`
or
`c=Mat1(1,[2,3])`
	`c =`
	`4.5600 8.0000`

`b=Mat3(2,1:2)`	`b =` `3 6`
`REsuLT=(b+c)*4`	`REsuLT =` `30.2400 56.0000`

`bbcc=[b;c]`	`bbcc =`
	`3.0000 6.0000`
	`4.5600 8.0000`

`nice=[Mat1(:,1), (Mat2(4,:))']*Mat3(1,3)`	`nice =`
	`18.0000 0`
	`72.0000 0`
	`18.8496 18.0000`

(continued)

(continued)

To type after prompt >> followed by Enter	Display by MATLAB

`SubMat1=Mat3(:,[2,4])`	`SubMat1 =`
	`4 8`
	`6 12`

Try by yourself	`SubMat1 =`
	`2 14`
	`3 21`

Try by yourself	`NewMat =`
	`4 8 2 14`
	`6 12 3 21`

4. Calculate the sum of b and c and multiply the result by 4. Put the result in the matrix REsuLT.
   - Create a matrix Mol obtained from element-by-element multiplication between SubMat1 and SubMat2.
   - Change the element in position (2,2) of Mol to 5;

To type after prompt >> followed by Enter	Display by MATLAB
`REsuLT=(b+c)*4`	`REsuLT =` `30.2400 56.0000`

`Mol=SubMat1.*SubMat2`
`PAY ATTENTION: MATLAB can also compute the product Mol=SubMat1*SubMat2. However, that is not the element-by-element product`
	`Mol =`
	`8 112`
	`18 252`

`Mol(2,2)=5`	`Mol =`
	`8 112`
	`18 5`

A Brick for an Experiment

In this section of the book we illustrate, step by step, a MATLAB program implementing a behavioral experiment together with the graphical interface for running the program and the statistical analysis for analyzing the data. The experiment we are implementing is a classic experiment in audiovisual perception by Sekuler et al. (1997). The effect showed by these authors is one of the most compelling examples of interaction between audition and vision. It can be observed by comparing the post coincidence trajectories of two moving objects. The objects are perceived as bouncing off each other or as streaming through each other according to whether a sound is presented (or not) when the objects overlap during the motion (Sekuler et al. 1997; Watanabe and Shimojo 2001; Remijn et al. 2004; Kawabe and Miura 2006; Kawachi and Gyoba 2006; Zhou et al. 2007; Grassi and Casco 2009, 2010; Grove and Sakurai 2009). The effect is based on a motion display originally proposed by Metzger (1934). Metzger's display shows two identical objects (e.g., two discs) that move along the azimuth with uniform rectilinear motion and opposite directions: discs start their motion, overlap and stop at the other disc's starting point with uniform

rectilinear motion. This simple two-dimensional display is a complex inverse optics problem for the visual system (Marr 1982). The display is equally representative of two different events in the real three-dimensional world. In both events the two objects are placed at different depths so that the retinal images of both have identical size. In one event, the objects start their motion, overlap (i.e., one object occludes the other), then stream past one another. In the other possible event, in contrast, after the occlusion, the objects reverse their motion and return to their original starting positions. In brief, the motion of the discs is bistable because both the streaming and bouncing percepts are compatible with the proximal stimulus. However, the streaming percept is usually predominant if the motion display is silent, whereas the bouncing is predominant when the sound is presented. We strongly suggest that the reader read the cited paper to have a clearer idea about the experiment we are going to implement in the "brick" section.

Here, we implement the original experiment by Sekuler et al. (1997). The experiment is a 2 (motion type) by 2 (sound condition) experiment. In the experiment, a disc's motion can be continuous, or it can stop for a certain number of frames when the discs overlap. In addition, the discs' motion can be accompanied (or not) by a brief sound that is presented when the discs overlap. The subject's task is to report whether s/he perceived the discs as streaming or as bouncing. Usually, in this type of experiment, the experimenter records the proportion of bounce responses as a function of the various experimental conditions (the number of streaming responses is, of course, the reciprocal of the number of bounce responses).

The experiment by Sekuler et al. (1997) is a classic "fixed stimuli" experiment. In other words, we know before the subject participates in the experiment what and how many stimuli we are going to present.[1] This is an advantage, because we can prepare many things in advance. For example, a wise thing to do in such cases is to write an event table, i.e., a table in which we write the exact experimental condition we are going to present to the subject trial after trial. In the event table all events will be represented in numbers.

In this chapter we limit ourselves to writing few variables that will turn out to be useful later. First of all, let's write one variable containing the two factors we are manipulating in the experiment. We will symbolize each factor/condition in numbers.

```
>> conditions = [1, 1; 1, 2; 2, 1; 2, 2];
```

if you visualize the content of this variable, it looks like this:

```
>> conditions
conditions =

     1     1
     1     2
     2     1
     2     2
```

[1] This is not the case of adaptive psychophysical procedures in which the stimuli presented are selected trial by trial as a function of the subject's response.

The left column codes as a number the video display we are presenting (e.g., 1 = continuous motion; 2 = discontinuous motion), the right column codes as a number the sound associated with the display (1 = no sound; 2 = sound). However, usually in psychological experiments stimuli are presented more than once, i.e., they are presented with a certain number of repetitions (say, 20 times each).

```
>> repetitions = 20;
```

we can now begin to write the event table:

```
>> EventTable = [];
>> EventTable = [EventTable; conditions];
>> EventTable = [EventTable; conditions];
>> EventTable = [EventTable; conditions];
```

You should repeat the command `EventTable=[EventTable; conditions];` 20 times (i.e., the number of repetitions) in order to obtain the complete event table. In a later chapter we will show how to generate repetitive commands automatically.

Now let's add to the left of the `EventTable` a new column with the trial number (note the use of the apostrophe, which transposes the array we are creating, so that `EventTable` becomes a matrix of a few columns but several rows, one row for each trial of the experiment):

```
>> TotalNumberOfTrials = length(EventTable(:, 1));
>> TrialNumber = (1:TotalNumberOfTrials)';
>> EventTable = [TrialNumber, EventTable];
```

Now let's look at the content of the table matrix. At trial number 12, for example, we know that we are going to present the display 2 (the discontinuous motion) together with the sound (i.e., 2). At trial number 37, we will present the display 1 (the continuous motion) and no sound (i.e., 1), and so on. However, in the current experiment, as in the majority of experiments in psychology, we want the stimuli to be randomized within the block of trials the subject performs. In this way, we avoid possible unwanted effects such as the serial effects that could arise if we were using, for example, a fixed sequence of trials. We can do this by shuffling the trial sequence by means of `randperm`, which is a MATLAB function that generates a random permutation of integers from 1 to n, i.e., the input the user has to pass to the function. (Note that we will transpose the output of the `randperm` function so that we have an array with one column and several rows.) Now we substitute the trial sequence column by the shuffled trial sequence column:

```
>> RandomTrialSequence = randperm(TotalNumberOfTrials)';
>> EventTable(:, 1) = RandomTrialSequence;
```

Now let's sort the `EventTable` content by the shuffled trial number in the first column:

```
>> EventTable = sortrows(EventTable, 1);
```

If you now echo in the command window the content of the `EventTable` matrix, you will see that the stimuli presentation list is now nicely randomized.

A part of the commands we have shown will be included in a script that automatically generates the event table in the desired form. This will be shown in Chap. 3.

References

Grassi M, Casco C (2009) Audiovisual bounce-inducing effect: attention alone does not explain why the discs are bouncing. J Exp Psychol Hum Percept Perform 35:235–243

Grassi M, Casco C (2010) Audiovisual bounce-inducing effect: when sound congruence affects grouping in vision. Atten Percept Psychophys 72:378–386

Grove PM, Sakurai K (2009) Auditory induced bounce perception persists as the probability of a motion reversal is reduced. Perception 38:951–965

Kawabe T, Miura K (2006) Effects of the orientation of moving objects on the perception of streaming/bouncing motion display. Percept Psychophys 68:750–758

Kawachi Y, Gyoba J (2006) Presentation of a visual nearby moving object alters stream/bounce event perception. Perception 35:1289–1294

Marr D (1982) Vision: a computational investigation into the human representation and processing of visual information. W. H. Freeman, New York

Metzger W (1934) Beobachtungen über phänomenale Identität. Psychol Forsch 19:1–60

Remijn GB, Ito H, Nakajiama Y (2004) Audiovisual integration: an investigation of the "streaming-bouncing" phenomenon. J Physiol Anthropol Appl Human Sci 23:243–247

Sekuler R, Sekuler AB, Lau R (1997) Sound alters visual motion perception. Nature 385:308

Watanabe K, Shimojo S (2001) When sound affects vision: effects of auditory grouping on visual motion perception. Psychol Sci 12:109–116

Zhou F, Wong V, Sekuler R (2007) Multi-sensory integration of spatio-temporal segmentation cues: one plus one does not always equal two. Exp Brain Res 180:641–654

Suggested Readings

Some of the concepts illustrated in this chapter can be found, in an extended way, in the following books:

Gilat A (2008) MATLAB: an introduction with applications. Wiley, Mahwah, N.J.

Hunt BR, Lipsaman RL et al (2006) A guide to MATLAB: for beginners and experienced users, 2nd edn. Cambridge University Press, Cambridge, United Kingdom

Kattan PI (2008) MATLAB for beginners: a gentle approach, Revised edn. Lulu.com, Raleigh, NC, United States

Rosenbaum DA (2007) MATLAB for behavioral scientists. Lawrence Erlbaum Associates, Hoboken, N.J.