

It is possible to operate on characters and strings in MATLAB. Strings can be displayed on the screen and can be used to construct commands which are evaluated or executed within other commands.

5.1 Assignment

A string in MATLAB is defined by apostrophes:

```
nameofvariable = 'text'
```

where 'text' could be letters, numbers and special characters. This text is stored in a row vector in which each entry represents a character. In reality, the elements contain the internal code of the characters, that is the ASCII code. When the value of a string variable is displayed on the screen the text is shown, not the ASCII numbers. Since strings are stored as vectors, we can refer to any element in them by giving its index.

Matrices of characters are also allowed but have to have the same number of characters in each row.

■ Example 5.1

- (a) A simple assignment, `name = 'John Smith'`, gives the following display on the screen:

```
name =  
      John Smith
```

- (b) If just 'John Smith' is typed, the variable `ans` is assigned the string:

```
ans =  
      John Smith
```

- (c) Assignment of a character. If `name` from (a) exists then `name(3) = 'a'` gives:

```
name =  
      Joan Smith
```

- (d) To reverse the order of the components for the string `name` from previous examples we type:

```

for i = length(name):-1:1
    eman(i) = name(length(name)+1-i);
end

```

This will give us the string **eman** with the value:

```

eman =
    htimS naoJ

```

See Section 12.2 for more about for-loops.

- (e) The length of a string: `namelen = size(name)` gives:

```

namelen =
    1    10

```

- (f) The apostrophe character is typed in a string by doubling it:
`whoscat = 'Joan''s cat'` becomes:

```

whoscat =
    Joan's cat

```

- (g) Strings can be composed just like numerical matrices:

```

name1 = 'Joan'; name2 = 'John'; heart = 'is in love with';
sentence = [name1 ' ' heart ' ' name2] give:

```

```

sentence =
    Joan is in love with John

```

- (h) It is also possible to use colon notation just like in numerical matrices:

```

name = 'Charles Johnson'; firstname = name(1:7) give:

```

```

firstname =
    Charles

```

- (i) `Text1 = 'John'; Text2 = 'Joan'; Couple = [Text1; Text2]`
give:

```

Couple =
    John
    Joan

```

5.2 String commands

There are several commands for converting strings to other forms of representation.

Command 41 TO CONVERT A STRING

`abs(str)`

returns a vector with the ASCII codes for the characters in the string `str`.

`setstr(x)`

returns a string composed of the integer vector `x`. The elements in `x` are translated to characters according to the ASCII table, therefore the integers must be in the interval $[0, 255]$ to give a correct result.

`num2str(f)`

converts the scalar `f` into a string representation in floating-point format, consisting of four digits and exponent if required. The command is often used together with `disp`, `x` label and other output commands.

`num2str(f, k)`

converts the scalar `f` into a string representation in floating-point format, consisting of `k` digits.

`int2str(n)`

converts the integer `n` into a string representation of the integer.

`rats(x, strlen)`

converts the floating point number `x` into a string representation containing the rational approximation of `x`. The integer `strlen` is the number of positions the result is allowed to have, default is 13.

`hex2num(hstr)`

converts the hexadecimal number in the string `hstr` to the corresponding floating point number (IEEE double precision).

`hex2dec(hstr)`

converts the hexadecimal number in the string `hstr` to an integer.

`dec2hex(n)`

converts the integer `n` to the corresponding hexadecimal number. The result is a string.

`sprintf(formatstr, A)`

returns the elements of the matrix `A` in a string of the format defined in the format string `formatstr`, similar to format control in the programming language C. The command performs just like `fprintf` but the result is a string instead of a file. (See Section 15.4, especially Table 15.2).

Command 41 TO CONVERT A STRING (continued)

<code>[Str,E]=sprintf(...)</code>	returns the string <code>str</code> as above, and a matrix <code>E</code> , containing an error message string if an error occurred. If the conversion is correct the empty matrix <code>E</code> is returned.
<code>sscanf(str, formatstr,mn)</code>	returns a matrix whose elements are read from the string <code>str</code> according to the string <code>formatstr</code> . The maximum number of elements read is <code>mn</code> , but this parameter is optional. Performs like <code>fscanf</code> , but operates on a string instead of a file (see Section 15.4).
<code>[A,nm,E,next] = sscanf(str, formatstr,mn)</code>	returns a matrix <code>A</code> just like <code>sscanf</code> , but does also return the number of correct converted elements <code>nm</code> , and the errors in the matrix <code>E</code> . The scalar <code>next</code> is index to the next element in the case when all the elements are not read.

Example 5.2

Let these variables be defined: `str = 'ABC'; float = 1.25;`

(a) The command `x = abs(str)` then returns:

```
x =
    65    66    67
```

that is ASCII codes for the characters A, B and C.

(b) If we type `Number = hex2dec(str)`, MATLAB gives:

```
Number =
    2748
```

(c) The line `numstr = num2str(float)` results in:

```
numstr =
    1.2500
```

To show that this in fact is a string, we type `Char = numstr(4)` and MATLAB gives:

```
Char =
```

(d) The command `numinfo = sprintf('The Number = %5.2e',float)` gives:

```
numinfo=  
The Number = 1.25e+00
```

(e) The command `rational = rats(0.979796)` will give us a string containing the rational approximation of the floating point number 0.979796:

```
rational =  
4995/5098
```

The optional number 5 in the command `littlerat = rats(0.979796,5)` will restrict the string length to be 5.

```
littlerat =  
48/49
```

In Section 2.4 the command `rat` is defined. ■

There are also logical functions that operate on strings and functions to extract substrings. In Command 42 below, let `str` be a string.

Command 42 FUNCTIONS OF STRINGS

<code>blanks(n)</code>	returns a string with n blanks.
<code>deblank(str)</code>	returns the string <code>str</code> without trailing blanks.
<code>lower(str)</code>	changes all letters in <code>str</code> to lower-case letters.
<code>upper(str)</code>	changes all letters in <code>str</code> to capital letters.
<code>isstr(x)</code>	returns a 1 if <code>x</code> is a string, otherwise 0.
<code>isletter(str(i))</code>	returns a 1 if the i th character in <code>str</code> is a letter.
<code>isspace(str)</code>	returns a vector of the same size as <code>str</code> , whose elements are 1 if the corresponding character in <code>str</code> is a blank, tabular or line feed, otherwise 0.
<code>strcmp(str1,str2)</code>	compares <code>str1</code> and <code>str2</code> . If they are equal, 1 is returned otherwise 0.
<code>str2mat(str1,str2,...)</code>	creates a string matrix with <code>str1</code> , <code>str2</code> , and so on. If <code>str_i</code> are of different sizes, MATLAB completes the shorter ones with blanks in the end. The function handles at most 11 arguments but <code>str_i</code> can also be string matrices.
<code>findstr(str1,str2)</code>	returns a vector containing the start positions for the substring <code>str2</code> in <code>str1</code> .

Command 42 FUNCTIONS OF STRINGS (continued)

<code>strrep(str1, str2,str3)</code>	returns a modified string str1 with all existences of str2 replaced with str3 .
<code>strtok(str1,str2)</code>	returns the part of str1 that remains when the trailing part after the first appearance of str2 has been removed. If str2 is not specified MATLAB uses blanks, that is selects the first sequence in str1 that does not contain a blank.
<code>[outstr,rstr] = strtok(str1,str2)</code>	returns the string outstr as above but also the removed part in the string rstr .
<code>lasterr</code>	returns a string containing the last error message.

■ **Example 5.3**

(a) `name = upper('matlab')` gives:

```
name =
    MATLAB
```

(b) `fun = strrep('hahaha','a','i')`

```
fun =
    hihih
```

(c) Let the string variables

`greet = 'Welcome'`, `where = 'to Joan's'`, `party = 'birthday party!'` be defined. Then the command `str2mat(greet,where,party)` gives:

```
ans =
    Welcome
    to Joan's
    birthday party!
```

(d) It is possible to extract information from a string with the contents separated by commas by using the command `strtok`. We define a string `text` as:

```
text = 'Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday'
[day,rest] = strtok(text,',')
```

```
day =
    Monday
```

```
rest =
    ,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday
```

By calling `strtok` with the string `rest` we can find next day and so on:
`[day2,rest] = strtok(rest,',')` gives:

```
day2 =
    Tuesday
rest =
    ,Wednesday,Thursday,Friday,Saturday,Sunday
```

Note that the command reads until the second comma, since the first is in the first position of the string and therefore does not separate any part of the string. ■

5.3 Display and input

To display the contents of a numerical matrix or a string vector one can simply type the variable name (see Section 2.2). The result is that both the variable name and the contents are shown.

Another way of displaying the value of a variable is to use the command `disp`. In this case only the contents of the variable are shown on the screen. This can be of special use in string applications.

Command 43 DISPLAY

`disp(A)` displays the contents of the matrix **A**. If **A** is a string, the text is displayed.

Input from the terminal is received with the `input` command. This also displays text and a prompter. See example in Appendix A, Section A.6. The command `disp` in combination with `int2str` is shown in Example 13.7.

Command 44 INPUT

`input(str)` displays the text in the string **str** on the screen and waits for an input from the terminal. The read value is returned by `input`.
`input(str,'s')` prints the text in **str** and returns the input as a string.

■ Example 5.4

(a) Read a real number `x`:

```
x = input('Give a number x: ') results in
```

Give a number x : 2.0944

`x =`

2.0944

- (b) Read a matrix A.

A semicolon after the command `input` suppresses the printing of the result.

`A = input('Give the matrix A row by row: ');` gives:

Give the matrix A row by row: [1 2; 3 5]

- (c) It is also possible to use arithmetical expressions and functions:

`A = input('Please give me a matrix: ');` gives:

*Please give me a matrix: rand(4) * hilb(4)*

- (d) The input command may have more than one argument:

`[m n] = input('Give the size of A: ');`

Give the size of A: size(A)

- (e) To read a string:

`name = input('What is your last name? ');`

What is your last name? 'Smith'

Note that the string in this case must be enclosed by ' '. If the command looks like:

`strname = input('What is your last name? ','s');`

the name can be written directly:

What is your last name? Smith ■

It is also possible to give a menu of options in MATLAB. A menu is a good method for the user to input choices (see Chapter 12). An explanation of the menu command is shown in Command 45.

Command 45 MENUS

```
menu(titlestr,
      optstr1,optstr2,
      ...)
```

writes a menu with the title in **titlestr** and with options from **optstr1**, **optstr2**, and so on, on the screen and waits for an input from the terminal. The menu command returns an integer.

Command 45 MENUS (continued)

<code>errordlg(errStr, dlgName)</code>	creates an error dialog box which displays the string <code>errStr</code> in a window named <code>dlgName</code> . A button must be pushed to make the window go away.
<code>warndlg(warn, dlgName)</code>	creates a warning dialog box which displays the string <code>warn</code> in a window named <code>dlgName</code> . A key must be pressed to make the window disappear.
<code>helpdlg(helpStr, dlgName)</code>	creates a help dialog box which displays the string <code>helpStr</code> in a window named <code>dlgName</code> . A key must be pressed to make the window disappear.
<code>questdlg(quest, ... op1, op2, op3)</code>	creates a question dialog box which displays the string <code>quest</code> and buttons marked <code>op1</code> , <code>op2</code> and <code>op3</code> . There can be up to three buttons. A key must be pressed to make the window disappear. Returns the chosen string.

Example 5.5

To create a simple menu with three choices enter:

```
choice = menu('Choose:', 'Enter', 'Wait', 'Leave')
```

On an Xterminal a window containing the menu in Figure 5.1 is shown.

By clicking the desired option the choice is made. If we choose the Wait-box MATLAB returns:

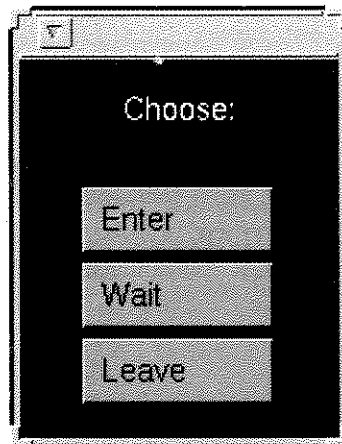


Figure 5.1 A MATLAB menu.

```
choice =
```

```
2
```

since the second option was selected. ■

The menus look different depending on the computer used. If there is no access to graphics, MATLAB gives the options in the command window and waits for an answer from the terminal.

5.4 Evaluation of strings

MATLAB commands can be written and stored as strings. These command strings can be evaluated by the `eval` command.

Command 46 EVALUATION OF STRINGS

<code>eval(str)</code>	executes the MATLAB commands contained in str and returns the results.
<code>eval(str1, str2)</code>	executes the MATLAB commands in str1 . If the first string is correct it is equal to <code>eval(str1)</code> . If there is an error in evaluation of str1 , the string str2 will be evaluated, that is giving an error message or something else.

The `eval` command makes MATLAB a flexible programming language. The command is useful, for example, for calls to functions which are not predefined in MATLAB, see Section 12.4.

■ Example 5.6

- (a) Suppose that the scalars b , k and x are defined. If the string **str1** is defined by: `str1 = 'b.*sin(k.*x)'` then the command `value = eval(str1)` results in **str1** evaluated with the current values of the variables b , k and x .

If, for example, $b = 3$; $k = 2$; $x = 1.2$; then `eval(str1)` gives

```
value =
```

```
2.0264
```

It is also possible to give two strings to the `eval` command. Suppose that we have a string

```
str2 = 'disp(''Oops! Maybe the variables b, k and x are  
not all defined?'')
```

defined and that the variables *b*, *k* and *x* are not defined. Since the first string is not possible to evaluate the command `value = eval(str1, str2)` results in

Oops! Maybe the variables b, k and x are not all defined?

```
value =
```

```
 []
```

Note that the `'` sign has to be repeated twice in a string to become such a sign, otherwise MATLAB interprets it as end of string. The command `disp` is explained in Section 5.3.

- (b) String variables can be combined with strings. If we have a variable name `file = 'myfile.mat'` and wish to save the current session to this file, we type `eval(['save ', file])`. This is the same as typing `save myfile.mat`. (The `save` command is defined in Section 2.8.)

- (c) MATLAB can also evaluate vector functions. Let `str1` be the same as in (a). If **b**, **k** and **x** are vectors: `b = [1 2 3]; k = [2 2 2]; x = [1.2 1.5 1.2];`

The command `values = eval(str1)` gives:

```
values =
```

```
 0.6755  0.2822  2.0264
```

- (d) Let us create a string:

```
fcn = 'input(''Give a function'', 's')';
```

The `input` command is presented in Section 5.3. Now an optional function can be chosen from the terminal and evaluated.

The command `fplot(eval(fcn), [0, 4])` results in the following printed on screen:

Give a function

If we write `sin(x)` MATLAB opens a graphical window and displays the sine function.

Suppose an M-file (see Section 2.9) named `sinx2.m` is defined according to:

```
function y = sinx2(x)
```

```
y = sin(x.^2);
```

If we write `sinx2` after the text *Give a function* we get a graph of the function $\sin x^2$:

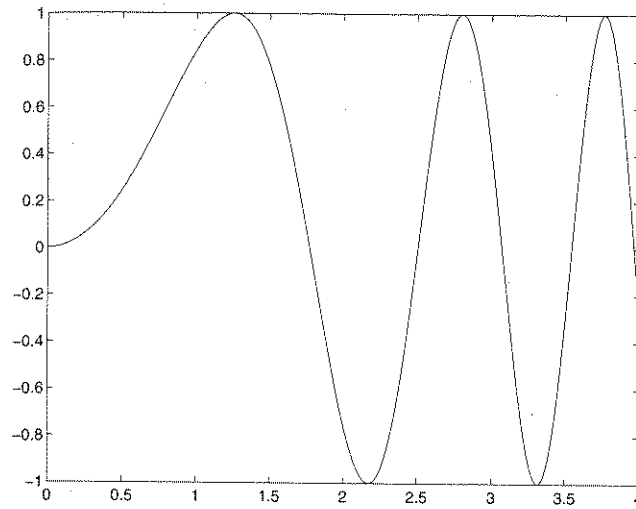


Figure 5.2 The function $\sin x^2$ between 0 and 4. ■

The strings which are evaluated can also contain MATLAB programming structures, such as *if*, *while*, *for* and others (see Chapter 12).