

Matlab Manual

It is forbidden to copy or abuse software and documentation supplied by the Faculty of
Electrical Engineering, Mathematics and Computer Science

Contents

0	Matlab version	5
1	Matlab session	6
1.1	<i>Getting started with Matlab</i>	6
1.2	<i>Matlab and matrices, a general remark</i>	7
1.3	<i>The Matlab Editor/Debugger</i>	7
1.4	<i>The Workspace Browser</i>	7
1.5	<i>The property editor</i>	8
2	Lay-out	9
3	Common Commands	10
4	Numbers and strings	11
5	Variables	12
6	Complex variables	13
7	Matrices and Vectors	14
8	Matrix operation, matrix functions, array operation	17
9	Elementary mathematical functions and constants	19
10	Conditional statements	20
11	Loop Statements	22
12	Output	23
13	Input	24
13.1	<i>The import wizard</i>	24
14	Graphical Output	25
15	Script files, function files	28
15.1	<i>Script files</i>	28
15.2	<i>Function files</i>	28
15.3	<i>Passing a function</i>	29
15.4	<i>Collecting multiple Matlab files</i>	30
15.5	<i>Nested Functions</i>	31

16	Solving a system of equations	32
17	Tracking down errors	33
18	Symbolic Computing.....	34
19	Example program, time integration	36
20	Example program, filling a penta-diagonal matrix	38
21	Reference and index	39

0 Matlab version

This manual describes (some) elements of Matlab Version 7.0 (or higher).

1 Matlab session

The way to start Matlab differs from computer to computer. You may type the command 'matlab' in a command window of the operating system. Often, though, you will have to click on a specific icon in order to run the program.

1.1 Getting started with Matlab

Once you have started Matlab a Matlab command window will appear, showing the command prompt:

```
» % The Matlab command prompt.
```

The line after the prompt is called the command line. On this line you can give Matlab commands. After you have pressed <return>, Matlab will execute the command.

```
» pause(5) % Wait 5 seconds before showing the plot.  
» plot(x,y) % Plot vector y versus vector x.
```

Besides the command window Matlab has graphical windows. Output of plot commands is directed to the graphical window.

The **quit** command enables you to leave Matlab. To terminate a running Matlab command you may use **[Ctrl]+[c]** (Press both the Ctrl button and the c button simultaneously).

By using the **!** symbol you can use the original operating system

```
» ! printer command % Execute the printer command belonging to the  
% original operating system.
```

Only for short computations it is useful to execute Matlab straightaway from the command line. In general the next procedure is much more practical:

1. Make a script file (see section 15) by means of your favorite text editor or the Matlab Editor/Debugger (see Section 1.3). A script file consists of a sequence of Matlab commands. In general script files will contain the main program and subprograms.
2. If necessary make the additional function files, using the same editor. Through these files we are able to define the functions which play a role in script files.
3. Matlab executes the commands in the script file after you have typed the name of the script file on the command line. Note, however, that the script file should be in the current (working) directory, indicated in the box above the command window.

From the command line background information can be obtained using

1. help

```
» help plot % gives information on the Matlab command plot.
```

2. demo

» **demo** % presents multiple examples of the usage of Matlab.

1.2 Matlab and matrices, a general remark

Suppose that we define vectors x , y and a matrix z by

$$\begin{aligned} x(i) &= i && , i = 1, \dots, 10, \\ y(i) &= i^2 && , i = 1, \dots, 10, \\ z(i, j) &= \sin(x(i) * y(j)) && , i, j = 1, \dots, 10. \end{aligned}$$

In most programming languages a computer implementation will use nested loops:

```
» for i = 1:10
    x(i) = i; y(i) = i^2;
end
for i = 1:10
    for j = 1:10
        z(i, j) = sin (x(i) * y(j));
    end
end
```

In Matlab this can be done quite differently, because matrices are basic objects:

```
» x = 1:10; y = 1:10; y = y.^2;            % Mark the period (see Section 8)
» z = sin(x'*y);                          % Mark the apostrophe (see Section 8)
```

Both programming styles are possible in Matlab. However, the latter is far more efficient. Therefore we prefer the latter and all examples will be given in this style.

1.3 The Matlab Editor/Debugger

It is advantageous to use the Matlab Editor/Debugger when creating or editing script files. You invoke this editor by typing **edit** at the command prompt or from the File-New or File-Open menu. The Matlab editor has various features to aid in editing script files so that most typing errors can be recognized. For example, text strings, reserved words (if, else, for, end, ...) and expressions are all shown in different colours. Saving and running the script is easily done using the Debug menu.

1.4 The Workspace Browser

The Workspace browser is invoked by the View-Workspace menu, giving a list of current variables (scalars, vectors, matrices), just as **whos** (Section 3) does. By double clicking on a variable in the Workspace window the values of this variable are shown, in a separate window (the array editor), enabling inspection and interactive adaptations.

1.5 *The property editor*

Matlab directs graphical output to the graphical window. In this window the so-called property editor is available. Access is possible via the Edit or View button. Each graphical window contains several graphical objects such as axes and lines. One can select the different objects by clicking on them. Next, using the property editor one may inspect, make changes or add objects. This is in particular handsome in the final stage when it is needed to prepare the plot for inclusion in a report. It is the easy to add a title, label, etc.

2 Lay-out

When you use Matlab's default configuration, the program distinguishes upper case and lower case characters. One says that Matlab is *case sensitive*.

If the default configuration is used, Matlab will also print the result after every command. Typing ; after the command will suppress this.

```
» x = 2    %    Matlab prints the result
```

```
x =  
    2
```

```
» x = 2;   %    Matlab does not print the result
```

The symbol % (*comment*) is used to give comments.

```
» x = 2    %    gives the value 2 to x and prints the result  
           %    printing the result can be suppressed with;
```

The symbol ... (*continuation*) denotes that the command continues on the next line

```
» x = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 ...  
    + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20;  
           %    this command does not fit on one line
```

3 Common Commands

quit	:	leave Matlab
help command name	:	gives information about a command
arrow up / down	:	retrieves preceding and following commands
pause	:	pauses execution, Matlab will proceed after <return>
whos	:	gives a list of Matlab variables stored in the memory
clear	:	clears the memory
clc	:	clears the command window
clf	:	clears the graphical window
shg	:	brings the graphical window to the foreground
close	:	closes the graphical window
cputime	:	determines the elapsed cpu time
tic, toc	:	stopwatch timers for the elapsed real time
demo	:	activates Matlab demonstrations

4 Numbers and strings

Numbers can be entered in Matlab in the usual way; however, spaces inside a number should be avoided.

```
» (52/4 - 0.01) * 1e-3
```

```
ans =  
1.2990e-02
```

Matlab automatically assigns a type for every number you enter. Depending on the type, Matlab chooses an internal representation for these numbers and the corresponding computations. The external representation of a number (e.g. on the screen) can be altered with the format command.

format long e	:	16 digits, (exponential) floating point	
		3.141592653589793e-02	
format short e	:	5 digits, (exponential) floating point	3.1416e-02
format short	:	5 digits, fixed point	0.0314
format long	:	15 digits, fixed point	0.03141592653590
format	:	returns to the default configuration	

The default configuration is 'format short'. It might be possible that the local system manager has changed this into short e.

Remark: The **format** command influences only the external representation of real numbers. This command has no influence on the internal representation used by Matlab to process the program and its computations.

Remark: The **format** command is not able to alter the external representation of integer numbers. This can result for example in jumps in tables.

The command **vpa** from the Symbolic Toolbox is helpful for displaying numbers and variables:

```
» x = pi/10000; d=8; disp(vpa(x,d))  
.31415927e-3
```

In the above the variable **d** in the call of **vpa** refers to the number of digits to be displayed.

To manipulate text, Matlab uses strings.

```
» disp('give the spring constant a')  
give the spring constant a
```

5 Variables

A variable's name has to start with a letter, but may not contain more than 31 letters, digits, or underscores. Matlab is *case sensitive* in its default settings. This means that *j* and *J* do not have the same meaning. Matlab automatically reserves space in the computer's memory to store the variable. Variables do not need to be declared before use; Matlab derives the type of the variables by looking at the stored data. So it is possible that the type of a variable changes while a session is in progress.

The basic element of Matlab is the matrix. Depending on the size of the matrix we distinguish scalars (1 x 1 - matrix), vectors (1 x m -, or m x 1 – matrix), etc. Depending on the context Matlab also assigns the type of the variables in the matrix, e.g. real or complex.

The operators +, -, *, /, ^ can be used for all Matlab variables ($x \wedge y = x$ to the power of y). In the scalar case these operations will reduce to the usual computations. At every operation step Matlab checks if the dimensions of the matrices involved are correct.

```
» a = 1; c = 1 + i; v(1) = 1; v(2) = 2; word = 'text';
```

The command **whos** (see section 3) gives:

Name	Size	Class
a	1 x 1	double array
c	1 x 1	double array (complex)
v	1 x 2	double array
word	1 x 4	char array

Multiplying a vector *v* with itself is not possible. If we try this anyhow we get:

```
» w = v * v;
```

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

6 Complex variables

A complex number can be defined using the imaginary number i .

$$\gg c = 1 + i$$

$$c = \\ 1.0000 + 1.0000 i$$

The operators $+$, $-$, $*$, $/$, $^$ also work for complex numbers. With the symbol $'$ we conjugate a complex variable:

$$\gg c_{\text{gec}} = c'$$

$$c_{\text{gec}} = \\ 1.0000 - 1.0000 i$$

The (square of the) modulus of c can be computed in the following way:

$$\gg \text{modc2} = c' * c$$

$$\text{modc2} = \\ 2.0000$$

An alternative method is:

$$\gg \text{modc2} = \mathbf{abs}(c) ^2$$

$$\text{modc2} = \\ 2.0000$$

Imaginary and real parts of a variable can be obtained with the functions **real** and **imag**:

$$\gg a = \mathbf{real}(c)$$

$$a = \\ 1.0000$$

$$\gg b = \mathbf{imag}(c)$$

$$b = \\ 1.0000$$

7 Matrices and Vectors

Matlab stores its variables in matrices of size $n \times m$. If $m = 1$, we are dealing with a column vector, and if $n = 1$, with a row vector. When $n = m = 1$ the matrix represents a scalar. The size of a matrix does not have to be given; Matlab determines the size from the data given by the user. Matlab does not recognize a more general array structure, for example $v(-10:100)$; the lower bound in Matlab is always equal to 1.

We can define matrices in different ways, e.g. by giving values to every element separately. We separate row entries by a space or comma and column entries by a semicolon or a `<return>`.

```
» A = [1 2 3;4,5,6;7 8 9]    %   generating matrix A

A =
    1    2    3
    4    5    6
    7    8    9

» A = [1 2 ...
      3 4;
      5 6 7 8]              %   generating matrix A; ... means continuation

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

Vectors can also be made using the colon symbol `:`. Here the first value stands for the initial value, the second for the step size.

```
» x = 0 : 0.2 : 1           %   generating row vector x

x =
    0    0.2000    0.4000    0.6000    0.8000    1.0000
```

The default vector in Matlab is a row vector as can be seen here. If a column vector is needed, extra measures are needed (see later on).

Sometimes it is good to use one of the following Matlab functions:

```
zeros(n,m) :   gives an n x m matrix with zeros
ones(n,m)  :   gives an n x m matrix with ones
eye(n)     :   gives the n x n identity matrix

» A = ones(3,3) + 2*eye(3)    %   generating matrix A

A =
    3    1    1
```

```

1 3 1
1 1 3

```

Matrices can also be built from smaller variables

```

» v = ones(3,1); A = [-v 2*v -v] % generating matrix A

```

```

A =
-1 2 -1
-1 2 -1
-1 2 -1

```

Sometimes it is useful to construct matrices by concatenation.

```

» v1 = [1 2]; v2 = [3 4];
» v = [v1 v2] % or v = cat(v1, v2)

```

```

v =
1 2 3 4

```

```

» E = eye(3); C = zeros(3,2); D = [E C]

```

```

D =
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0

```

Large diagonal band matrices can be made by using the function

diag(v,k) : returns a square matrix with on the k-th upper diagonal the entries of the vector v.

```

» v = 1 : 2 : 9 % generating vector v

```

```

v =
1 3 5 7 9

```

```

» A = diag(v,-1) + eye(6) + 2*diag(v,1)

```

```

A =
1 2 0 0 0 0
1 1 6 0 0 0
0 3 1 10 0 0
0 0 5 1 14 0
0 0 0 7 1 18
0 0 0 0 9 1

```

Matrix elements can be used separately or in groups:

$A(i, j)$	=	A_{ij}
$A(:, j)$	=	j^{th} column of A
$A(i, :)$	=	i^{th} row of A
$A(i, j1:j2)$	=	vector existing of the entries, from column j1 to j2, of row i
$A(i1:i2, j1:j2)$	=	matrix existing of the entries, from column j1 to j2, of the rows i1 to i2.

» *plot* ($x(75 : 125)$, $y(325 : 375)$);

In case one of the dimensions equals one we are dealing with a vector. We can refer to this vector with just one index. This index is either a row index or a column index, depending on the type of the vector.

Note:

Matrix elements can also be addressed using one index only. This might sometimes be handy. One-dimensional references are taken column wise by Matlab:

```

» A=[1 2 3; 4 5 6; 7 8 9];
» x=A(6)
x=
      8
» x=A(2,3)
x=
      6

```

However, in many cases such a reference is the result of a programming error, and then one must be aware of the fact no error message will be given.

8 Matrix operation, matrix functions, array operation

The operators +, -, *, /, ^ can also be used for matrices. For every operation Matlab checks if the dimensions of the matrices involved are correct.

```
» v(1) = 1; v(2) = 2; v(3) = 3; A = eye(3);
» w = A * v;
```

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

Matlab assumes that the index of every new vector is a column index, i.e. unless explicitly specified otherwise (see example below), the new vector is a row vector. Hence in the example above v is a row vector, which results in an error message.

```
» v = zeros(3,1); % v is now a column vector
» v(1) = 1; v(2) = 2; v(3) = 3; A = eye(3);
» w = A * v;
```

Furthermore the next functions and operations are often quite useful

'	:	returns the transpose of a matrix
size(A)	:	returns the size of the matrix A.
length(v)	:	returns the size of the vector v.
norm(v)	:	returns the Euclidean norm of the vector v, i.e. $\text{norm}(v) = \sqrt{\text{sum}(v.^2)}$.
norm(v,inf)	:	returns the infinity norm of the vector v, i.e. $\text{norm}(v, \text{inf}) = \max(\text{abs}(v))$.
det(A)	:	returns the determinant of the matrix A.

```
» B = A'; % transpose A
» v(1) = 1; v(2) = 2; v(3) = 3; A = eye(3);
» v = v'; % change v into a column vector
» w = A * v;
» inprod = v' * v % inner product if v is a column vector.
```

```
inprod =
    14
```

```
» normv2 = norm(v)
```

```
normv2 =
    3.7417
```

```
» normvinf = norm(v, inf)
```

```
normvinf =
    3
```

If we use . (dot) as a prefix, we can do some of the operations element by element (sometimes referred to as array operations or operations in arraysense).

```
» v = v1.*v2;      % multiply v1 and v2 element by element.
                   % v(1) = v1(1) * v2(1), . . . , v(n) = v1(n) * v2(n).

» y=v.^2;         % y(1)=v(1)^2, ..., y(n)=v(n)^2.
                   % y=v^2 generates an error message
```

9 Elementary mathematical functions and constants

Some of the predefined constants in Matlab are:

pi : the constant pi
i : the imaginary unit

The following mathematical functions operate on each element of the variable separately:

abs : absolute value or modulus (complex case)
sqrt : square root
exp : exponential function
log : natural logarithm
log10 : logarithm with base 10

sin : sine
cos : cosine
tan : tangent
atan : arctangent

round : round off to the nearest integer
rem : remainder after division

For vectors are available:

max : maximal element
min : minimal element
sum : sum of the elements
norm : the norm

» *max(abs(v))* % *computing the largest element of v, irrespective of its sign*

For matrices we mention

eig : determines the eigenvalues (and eigenvectors) of a matrix
lu : determines the LU-factorization of a matrix (using permutations if necessary).
chol : determines the Cholesky factorization of a symmetric matrix

10 Conditional statements

In Matlab six relational operators can be used to compare variables or evaluate expressions

<	:	smaller than
<=	:	smaller than, or equal to
>	:	greater than
>=	:	greater than, or equal to
==	:	equal to
~=	:	not equal to

Furthermore there are three logical operators:

&	:	and
	:	or
~	:	not

An example of a logical expression is

$$x > 1 \ \& \ (y <= 5 \ | \ z = 1)$$

Conditional statements are, for example:

Syntax conditional statement:

```
if condition
    commands
end
```

Syntax single conditional statement:

```
if condition
    commands [1]
else
    commands [2]
end
```

Syntax multiple conditional statement:

```
if condition
    commands [1]
elseif condition
    commands [2]
else
    commands [3]
end
```

The condition has to be a logical expression.

An example:

```
» if (x >= 1 & y >= 1)
    a = 1;
elseif ~(x >= 1) & ~(y >= 1)
    a = -1;
else
    a = 0;
end
```

11 Loop Statements

FOR - loop

Syntax:

```
for variable = start value : increment : end value  
    commands  
end
```

The increment can be positive as well as negative. If not defined, Matlab will use increment 1 as a default.

```
» for j = 0 : 10  
    v(j + 1) = j * 0.1; % the row vector v needs to start at index one!  
end
```

WHILE - loop

Syntax:

```
while condition  
    commands  
end
```

The condition needs to be a logical expression.

```
» x = 0;  
» while (x < 1)  
    x = x + 0.1;  
end
```

Note that this is not a proper method when we want to execute the loop statement exactly ten times. In Matlab 10 times 0.1 is just a bit less than 1, because Matlab makes round-off errors. This means that the loop will be executed 11 times instead of 10 times, and therefore the final value of x will be too big:

```
» x  
  
x =  
    1.1000
```

12 Output

Output can be directed to the screen with the command `disp`.

```
disp (x)           :   print the value of the variable x
disp ('text')      :   print the string 'text'
disp (A(:,5))     :   print the 5-th column of matrix A
```

With the command **format** or **vpa** we can define how a number is displayed (see section 4).

Remark: Advanced control mechanisms for printing are available with the command **fprintf**. (see the example program in section 19 or use **help fprintf**)

We can write data to a file.

```
save data varlist           :   the variables in varlist are written to the file
                                data.mat in binary format. This file can be used in
                                a subsequent session.
save output varlist - ascii :   the variables in varlist are written legible (8 digits,
                                floating point) to the file output. If necessary this
                                file can be printed.
diary output                :   makes a diary of everything which happens
                                during the session and writes this to the file
                                output. If necessary this file can be printed.
diary off                   :   stops monitoring the session.

» format compact           %   suppress empty lines in output
» diary output             %   open the file output as output file
» disp ('x y')             %   display the header of the table
» for j = 1 : 10
    disp ([x(j) , y(j)]) %   display the values
end
» diary off
```

Remark: The **diary** command easily gives an unnecessary amount of output. Therefore: Use the **diary** command in a selective way and only at places where output is needed.

Remark: Files can be printed using the printer command of the original operating system, see section 1.

Remark: Depending on the operating system other printer commands might be available. Be careful in using these commands, because they can cause an unnecessarily long waiting time for other people who want to use the printer. Using the printer command (in the file menu) you can print the whole Matlab session.

13 Input

Variables can be entered in Matlab in different ways. The first method is using the command line. We already explained how to do this. Another way to enter it is:

```
» x = input ('x-coordinate =');
```

This command writes the text from the string to the screen, waits for input (until a <return> is given) and assigns the value given by the user to the variable *x*. The following command:

```
» load data
```

fetches the binary file *data.mat*. The file *data.mat* needs to be produced beforehand in Matlab and besides a list of variables it also contains the values of these variables.

The command

```
» load data.txt
```

fetches an ASCII file with the name *data.txt*. The content is stored in a double precision array with the name *data*. The file *data.txt* is only allowed to contain numbers, separated by blanks. Each line of the file corresponds with a row in the array *data*. Each number on a line corresponds with a column in the array *data*. The number of columns must be equal on each line.

13.1 The import wizard

Data can also be imported using a graphical user interface, called the Import Wizard. Choose in the File menu the option Import Data to start the Import Wizard. Then choose a file in the new window from which data should be read, preview the data, and if you are satisfied, use the next and finish buttons to import the variables in the Matlab workspace. You may check the imported variables afterwards, using **whos** (Section 3) or the Workspace Browser (Section 1.4).

14 Graphical Output

General commands affecting the graphical screen are:

clf	:	clear graphical window
shg	:	show graphical screen (bring graphical screen to the foreground)
close	:	close the graphical window
hold on	:	keep the contents of the graphical screen during new plot commands
hold off	:	erase the contents of the graphical screen before executing new plot commands
print	:	direct the graph to the printer
print filename	:	store the graph in the file 'filename'

Commands affecting the layout of the graph are:

axis ([xmin xmax ymin ymax])	:	sets the axes according to the given values
grid	:	adds a grid to the graph
title ('text ')	:	writes text above the plot
xlabel ('text ')	:	writes text underneath the x-axis
ylabel ('text ')	:	writes text next to the y-axis
text (x, y, 'text ')	:	writes text at the point (x,y) of the graphical screen
t = num2str (var)	:	makes text of the value of var, which can for example be written to the screen with text (x,y,t).

Some basic graphical commands are:

plot (y)	:	plots the vector y versus the indices 1, ..., n.
plot (x,y)	:	plots the vector y versus the vector x; x and y must be of the same size.
plot (x,y, 'symbol')	:	plots the vector y versus the vector x by using a symbol or color, e.g. ':' (dotted), '--' (dashed), 'o' (circles), 'x' (crosses), 'y' (yellow line), 'r' (red line), 'g' (green line)
fplot (@f,[xbegin xend])	:	plots the function f on the interval [xbegin, xend].
plot ([xbegin xend],[ybegin yend])	:	draws a line from (xbegin, ybegin) to (xend, yend).
zoom	:	makes it possible to zoom in at a specific point in the graph by clicking at that point.
ginput (n)	:	gets the coordinates of n points in the graph, which are located by positioning the cursor and thereupon clicking the mouse.

Remark: **help plot** shows all the possible arguments with **plot**.

```

» x = [0:20]/10; y = sin (pi * x);
» plot (x,y); % plot sin πx on [0,2], plot
% automatically assigns the right limits
% to the axes in the graphical screen.
» axis ([0 2 -1 1] ); % adjust the x-axis and the y-axis; it is
% preferable to call axis after plot
» xlabel ('x'); ylabel ('sin x');
» print % send the graph to the printer.

» clf
» hold on
» type = ['- ' ; ' : ' ; '- .' ; '- -'] % spaces to equalize the length of the
% character strings
» xgraph = [0:100]/100;
» for j = 1 : 4
    omega = j * pi ; % plot sin(ωx) on [0,1]
    ygraph = sin (omega * xgraph); % for ω = π, 2π, 3π, 4π,
    plot (xgraph,ygraph,type(j,:)); % distinguish the curves visually
end
» print figure % store the graph in a file named figure;
% use help print to see a list of
% available formats.

```

Some commands for 3 dimensional graphs are:

```

plot3(x,y,z) : plots a line through the coordinates of vectors x, y, z.
surf(x,y,Z) : plots the matrix Z versus the vectors y and x, creating a
surface.
surfc(x,y,Z) : same as surf, but adds a contour plot beneath the surface.
mesh(x,y,Z) : plots the matrix Z versus the vectors y and x, creating
a surface together with a mesh on it.
meshgrid(x,y) : creates matrices for a 2D grid from two 1D vectors.
rotate3d : enables the user to rotate the plot by mouse.

» t = [0:500] * pi / 50;
» x = sin(t); y = cos(t);
» plot3(x,y,t) % plots a spiral on the cylinder
%  $x^2 + y^2 = 1$ .

» x = [0:10] / 10;
» y = [0:20] / 10;
» [X,Y]=meshgrid(x,y) % build a 2D product grid by
% forming  $x \otimes y$ 
» Z=sin(X+Y); % compute function values on the grid
» mesh(x,y,Z); % or mesh(X,Y,Z) % plot sin(x+y) for x = 0 : 0.1 : 1
» % or surf(X,Y,Z) % and y = 0 : 0.1 : 2

```

Moreover, Matlab has some easy to use plotting routines with names starting with **ez**. Examples are:

ezplot : plots a function $f(x)$ on an interval

ezsurf : presents a surface of a function $f(x,y)$

As has been mentioned in section 1.5, the *property editor* can be used conveniently to manipulate the graphical window.

15 Script files, function files

15.1 Script files

A script file is a file that consists of a sequence of Matlab commands: a Matlab program. We make such a file with the help of the Editor. The standard extension for these files is .m, e.g. program.m. Every Matlab command is closed off with <return>. This also holds for the last line of the script file. The commands in the script file are executed after the filename is typed on the command line (without extension .m, i.e. in the example above you need to type 'program'). It is also possible to run a script file from Matlab programs (see section 18).

```
% plotsin.m           :   The script file creates a plot of  
%                   :   the function sin(x) on the interval [0, π]  
clear; clc; clf;    %   clear  
  
x = linspace(0, pi, 40); % create a grid on [0, π].  
y = sin(x);  
plot(x,y);  
axis([0 pi -1 1]);    %   set limits along axes  
title('sin(x)');    %   put text above the plot
```

15.2 Function files

By means of function files we can add new functions to Matlab. A function file contains a sequence of commands, like a script file, but by means of input and output variables it is possible to communicate with other files. The filename of a function file is used to execute the function.

A function file has the following form:

```
function output_variable = function_name(input_variable)  
commands
```

The word **function** on the first line implies that it is a function file and not a script file.

Remark: At the position of function_name you need to fill in your chosen name for the function. The filename of the file containing this function must be the same as this name with the standard extension .m. Similar to names of variables, the function name is not allowed to start with a number.

Both the input variable as well as the output variable may be matrices or vectors.

```
function y = average(v);  
% This function computes the average of the elements of vector v.  
% The function is stored in the file average.m
```

```
n = length(v);
y = sum(v)/n;
```

For the use of function files we refer to the example program (see section 18).

Variables used in a function file are local (i.e. only accessible inside the function), and therefore they are not stored in the Matlab memory.

```
function y = comp(x);
y = a * x;           % the variable a is defined locally and does not
                    % have a value
```

```
function y = comp(x);
a = 2;              % the variable a has the value 2, but only within this
y = a * x;         % function
```

Often it is necessary to use a variable in a function that gets its value outside the function. You can show Matlab you want to use such a variable by adding it to the list of parameters of that function. In the following example we add the variable `a` to the list of parameters of the function `comp`:

```
function y = comp(x,a);
y = a * x;         % the variable a gets its value outside the function
                    % and is passed to the function
```

This is not always possible. It may sometimes be necessary to define a variable globally, by using **global**. If you do so the variable is defined in all program parts that contain the same global declaration.

```
function y = comp (x);
global a;
y = a * x;         % the variable a is also defined globally at another
                    % location where a gets its value
```

Remark: The declaration with **global** should only be used when there is no other possibility. If you use it, it is wise to add some comments.

An example of a situation in which the occurrence of **global** might seem unavoidable is the usage of a library routine with a predefined function, where it is needed to supply the function with an additional parameter for a specific application. (See next Section)

15.3 Passing a function

A function can be passed as an argument by using a function handle or an inline object or a string. We recommend the use of function handles:

```
% program.m : script file to plot the function  $f(x) = x^m$ ;
% uses the function file f.m
```

```
global m;
m = 2;
fplot(@f,[0,1]);           % @ is the function handle
```

The function f is given in the following function file:

```
% f.m : Function file presenting the function  $x^m$ :
function fvalues = f(x);
global m;
fvalues = x.^m;           % note the usage of array operations
                        % the function  $f$  is called with an
                        % array as input variable. This
                        % array contains multiple  $x$ -values.
```

Remark : The usage of `global` is not really necessary, as the routine `fplot` could be called with a more elaborate argument list (see `help fplot`). Such variable argument lists however are outside the scope of this manual.

15.4 Collecting multiple Matlab files

Matlab does not allow for a mixture of scripts and functions in a single file. However, functions and subfunctions can be collected in a single file. As an example we present a single file containing both files from Section 15.3

```
function examplescript
% EXAMPLESCRIPT           This function executes the commands as contained in
%                          program.m (previous Section) only. No input and output
%                          arguments are used.

global m;
m = 2;
fplot(@f,[0,1]);           % @ is the function handle

function fvalues = f(x);
global m;
fvalues = x.^m;           % note the usage of array operations
                        % the function  $f$  is called with an
                        % array as input variable. This
                        % array contains multiple  $x$ -values.
```

Typing `examplescript` on the command line, excutes the script. However, it is a disadvantage that the workspace can not be accessed easily using the Workspace Browser because functions have local workspaces.

15.5 Nested Functions

In the context of the courses for which this manual is written it is often needed to vary parameters which are present in low-level routines. We can communicate parameters using the list of a function or by using the global command. A handsome alternative is available in the form of *nested functions*. This alternative is not free of dangerous aspects and can promote poor programming practice. Therefore, we can only recommend this as an alternative within the context of the present course. The underlying mechanism is that nested functions share the same workspace. This enables easy communication of parameters. If functions are nested, then it is obligatory to use **end** on the last line of the function

```
function solvede                                % level 0
% SOLVEDE solves a time-dependent DE          % top level

a = 2;
other commands;

function euler                                % level 1
% EULER performs Euler time stepping
% euler shares the workspace of solvede

commands, among others calling rhs;

function rhs                                  % level 2
% RHS contains the rhs of the DE
% rhs is called in euler
% rhs shares the workspace of solvede

commands    % these commands may involve
            % all that is in the workspace of
            % solvede, in particular the
            % parameter a can be used freely.

end      % end of rhs

end      % end of euler

end      % end of solvede
```

16 Solving a system of equations

To solve systems of equations $Ax = b$ we can use several methods:

- i) If a small system with a full matrix needs to be solved only once, we may use the black box option:

$$\gg x = A \backslash b;$$

Often it is necessary to be more specific to obtain an efficient program. As an example we mention:

- ii) If more than one system needs to be solved with the same matrix A , but different right-hand sides:

```

>> [L,U] = lu(A);           % Make an LU-decomposition of A
>> y = L\b;                % Solve lower triangular system
>> x = U\y;                % Solve upper triangular system
>> y2 = L\b2; x2 = U\y2;   % Solution for other right-hand side

```

- iii) If the matrix is symmetric and positive definite, we may use the Cholesky decomposition:

```

>> R = chol(A);           % Make a Cholesky decomposition of A: R' R = A
>> y = R\b;               % Solve lower triangular system
>> x = R\y;               % Solve upper triangular system

```

In all cases the command $A = \text{sparse}(A)$ can save a lot of runtime and memory space when A is a band matrix. As an example we give the number of floating-point operations for the case where A is a symmetric tridiagonal matrix of size 100×100 .

	without sparse (A)	with sparse (A)
method i	378350	2149
method ii	35250	1889
method iii	358750	1393

17 Tracking down errors

Matlab does not give specific directions for debugging programs. In any way it is useful to generate (intermediate) output, and if necessary to recalculate it manually. Some useful commands are:

- whos** : gives a table of the variables that are stored in memory. With this command you can check the size.
- size(A)** : gives the size of a matrix A.
- disp('labelj')** : directs the labeled text to the screen; can be used to find out at which line the program does not work properly.
- disp(var)** : prints the actual value of the variable var on the screen. Can be used to check values of intermediate results.

The Workspace Browser (Section 1.4) provides a very simple, but often effective, way to check names, types and sizes of variables. Moreover, variables can be opened in the array editor and then it is easy to check the values of the variables in use.

A more advanced feature can be found in the Matlab editor. After opening a Matlab file one can use the Debug menu, and, among others, set breakpoints.

18 Symbolic Computing

Matlab offers the Symbolic Math Toolbox, which is based upon the Maple kernel from Waterloo Maple, inc. This Toolbox allows symbolic manipulation of variables in a way very much similar to Maple, so it might be helpful to consult your Maple Manual. To obtain an overview of functions in the toolbox type **help symbolic** in the Matlab window. A short survey is given below.

sym x	:	declare x to be a symbolic variable
syms x y	:	declare x and y to be symbolic
subs (y,x,value)	:	substitute the 'value' for x into the symbolic expression y

As an example, observe the effect of following commands

```
» syms x y           % declare x, y to be symbolic
» y=sqrt(x)
» y1=subs(y,x,2)      % substitute x=2 into y= $\sqrt{x}$ 
» y2=subs(y,x,sym(2)) % y2 will be symbolic
» yv=subs(y,x,1:10)  % yv will be a vector
```

Expressions may contain several symbolic objects which can be substituted by a call to **subs** with lists (so-called cell arrays in Matlab) as second and third argument. Symbolic integration and differentiation is performed through the **int** and **diff** functions, as in Maple. The resulting expression sometimes appears to be quite complicated; then **simplify**, **factor** or conversion to numeric form, using **double** might help.

diff (f,x)	:	differentiate f with respect to x
int (f,x,a,b)	:	integrate f with respect to x from a to b
simplify (y)	:	simplify the symbolic expression y
factor (y)	:	factorize the symbolic expression y
double (y)	:	convert the symbolic expression into a floating point number

```
» syms x y
» f=sqrt(1+x^2+y^2)
» f12=subs(f,{x,y},{1,2}) % substitute x=1 and y=2 into f
» f12num=double(f12) % convert f12 to floating point
» fx=diff(f,x) % differentiate w.r.t. x
» fxsim=simplify(fx) % simplify
```

Expressions can be solved for a variable using **solve**, whereas **dsolve** tries to solve differential equations symbolically. The derivatives are denoted by D, D2, etc., and initial conditions can be passed as additional arguments.

solve (f,x)	:	solve f=0 for x
dsolve (deq,init)	:	solve the differential equation given by deq for the initial value init

```

»syms x a c
»f = a*x^2+c
»x0=solve(f,x) % find the zeros of f
»x0=solve('a*x+c=0') % solve ax+c=0
»y=dsolve('D2x+x=0') % solve the differential equation x''+x=0
»y=dsolve('D2x+x=0','x(0)=1') % solve the differential equation with initial
% value x(0) = 1

```

19 Example program, time integration

```

% This program computes the numerical solution of  $dy/dt=f(t,y)$  with Heun's method (the modified Euler method) .

% Filename      : exprog.m
% Function-file  : exf.m
% Script-files   : exheun.m, extabel.m          ex stands for example

% Screen management:
  clear;        % Clear work memory
  clc;          % Clear text window
  clf;          % Clear graphical screen

y0=[1; 0];      % Initial condition
t0=0;           % Start time
tend=5;         % End time
nrsteps=input('Give the number of steps:');
h=(tend-t0)/nrsteps; % Time step

exheun          % Call for the script file containing Heun's method (the modified Euler method)
extabel         % Call for the script file designed to print the results
                % The commands from a script file are substituted directly into this program.

pause(5);      % The program pauses a while before it shows the graph
hold on;
plot(tgraph,ygraph(1,:),'-',tgraph,ygraph(2,:),'--'); % components of y are distinguished by
                                                        % plotting the 2nd component with a broken line
axis([t0 tend -5 5]); % it is preferable to call axis after plot
title('Initial value problem: Name compulsory!');
xlabel(['Solution using Heun for h=' num2str(h)]);
hold off;

```

```

% Heun's method. Script file with filename exheun.m.

% Assuming (t0, y0) "nrsteps" steps are being executed
% The derivative of the diff. eq. is given in the function file exf.m.
% The results are put in tgraph and ygraph

tgraph=zeros(1,nrsteps+1);
ygraph=zeros(length(y0),nrsteps+1); % create matrix to save results
tgraph(1)=t0;
ygraph(:,1)=y0; % start saving results

y=y0; t=t0;
for j=1:nrsteps

  k1=h*exf(t,y);
  k2=h*exf(t+h,y+k1);
  ynew =y+(k1+k2)/2;
  tnew =t+h;

  tgraph(j+1)=tnew;
  ygraph(:,j+1)=ynew; % store the new results

  t=tnew;
  y=ynew;
end;

```

```
% Vectorfunction f. Function file with filename exf.m
```

```
function yacc = exf(t,y);  
  
yacc = [-2*y(1)-y(2)+t;  
        -y(1)-2*y(2)-t];
```

```
% Print result. Script file with file name extabel.m.
```

```
% The results which are stored in tgraph and ygraph are  
% printed in an 8-digit floating-point format in a table.
```

```
% In order to make a hardcopy of the tables you need to remove  
% the first and the last two comment symbols (%).
```

```
% fprintf is actually a C-command  
% In the format string of fprintf (the part between quotation marks),  
% text which needs to be printed is given, and for every number  
% which has to be printed the format is given.
```

```
% %5.1f means: fixed-point format with 1 decimal and 5 positions.  
% %15.7e means: floating point (exponential) format with 7 decimals and 15 positions.  
% \n means: continue printing on the next line
```

```
% After the format string the variables which (possibly) need to be printed follow.
```

```
% the actual file starts now:
```

```
% diary output
```

```
fprintf('Heun's method, h=%5.3f\n',h);
```

```
fprintf('step t y(1) y(2)\n');
```

```
for k=0 : 5 : nrsteps
```

```
fprintf('%4.0f %5.1f %15.7e %15.7e\n',k,tgraph(k+1),ygraph(1:2,k+1));
```

```
end;
```

```
% diary off
```

```
% print every 5th result
```

```
% the solution y has 2 components:
```

```
% they are both printed using the
```

```
% floating point format %15.7e
```

```
% Results of the example program
```

```
Heun's method, h=0.100
```

```
step t y(1) y(2)  
0 0.0 1.0000000e+00 0.0000000e+00  
5 0.5 5.2536330e-01 -2.9586400e-01  
10 1.0 5.7914644e-01 -5.2647651e-01  
15 1.5 8.4164231e-01 -8.2955459e-01  
20 2.0 1.2051207e+00 -1.2023466e+00  
25 2.5 1.6240001e+00 -1.6233635e+00  
30 3.0 2.0751573e+00 -2.0750112e+00  
35 3.5 2.5455986e+00 -2.5455650e+00  
40 4.0 3.0276755e+00 -3.0276678e+00  
45 4.5 3.5167996e+00 -3.5167979e+00  
50 5.0 4.0101983e+00 -4.0101979e+00
```

20 Example program, filling a penta-diagonal matrix

```

% We will give two methods in this example in order to construct the N x N dimensional pentadiagonal matrix A:
%
%      [ 5 -4-b  1  0  0  0 ]
%      [ -4+b  6 -4-b  1  0  0 ]
%      [  1 -4+b  6 -4-b  1  0 ]
%      [  0  1 -4+b  6 -4-b  1 ]
%      [ . . . . . ]
%      A = c [ . . . . . ],
%      [ . . . . . ]
%      [  0  1 -4+b  6 -4-b  1  0 ]
%      [  0  0  1 -4+b  6 -4-b  1 ]
%      [  0  0  0  1 -4+b  6 -4-b ]
%      [  0  0  0  0  1 -4+b  5 ]
%
% where c and b are given constants. Note that the matrix A is symmetric for b = 0.
% Note that it is advisable to use sparse and spdiags for the construction of very large matrices (N>10000), see help sparse

% Method 1: Construction using a loop.
% In order to make the commands in the loop also valid for the first and the last column we first add extra columns.

B=zeros(N,N+4);      % Creates a N x (N+4) matrix, consisting of zeros

for j=1:N
    B(j,j+4)=c*[1 -4+b 6 -4-b 1];
end;                % Assigns values to the coefficients B(j,j), B(j,j+1), B(j,j+2), B(j,j+3) and B(j,j+4)

A=B(:,3:N+2);      % Copies B, without the first two and last two columns
clear B;           % Removes temporary matrix B

A(1,1)=5*c;
A(N,N)=5*c;       % Changes the upper-left and the lower-right coefficient of A

A=sparse(A);      % Only the non-zero elements of A are stored into the memory.
                  % When N is large it is better to use the non-zero elements of A only.
                  % For an LU-decomposition this saves computing time as well as memory
                  % space.

% Method 2: Construction on the basis of the diagonals of A
% Firstly we make 5 vectors, all containing the elements of a diagonal.

vm2 = c * ones(N-2,1); % This diagonal contains (N-2) elements, all equal to c
vp2 = c * ones(N-2,1);
vm1 = c*(b-4)*ones(N-1,1);
vp1 = c*(b-4)*ones(N-1,1);

v = c*[5; 6*ones(N-2,1); 5]; % The first and last element of the main diagonal have different values.
                             % We construct a column vector consisting of sixes, by 6 * ones(N-2,1),
                             % and add the fives at the beginning and the end. Note that we use
                             % the semicolons because we are making a column vector.

A=diag(vm2,-2)+diag(vm1,-1)+diag(v,0)+diag(vp1,1)+diag(vp2,2);
clear vm2 vm1 v vp1 vp2

A=sparse(A);      % See above

```

21 Reference and index

In this section the following notation holds:

n,m - scalar
 A - matrix
 v,w,b- vector
 x,y - arbitrary
 f - user supplied function file

<i>Command</i>	<i>Explanation</i>	<i>Page</i>
[]	are used for creating matrices and vectors	
()	are used to : - indicate the order of operations - embrace arguments of functions - embrace indices of matrices and vectors	
...	Three or more dots at the end of a line indicate that the line continues on the next line	9
,	symbol that separates between row elements in a matrix.	14
;	symbol that separates between distinct rows in a matrix. We can also use the semicolon to suppress the display of computations on the screen and to separate different commands on a line	9,14
%	All text on a line after the symbol % will be regarded as comment	9
!	used to insert operating system commands	9
:	used for the generation of variables in for -loops and used to select matrix elements: A(:,n) is the n th column of A, A(m,:) the m th row	
'	transposes matrices and vectors	17
.*	v .* w : multiplication of two vectors by multiplying element by element (operation in array sense)	18
\	A\b gives the solution of Ax = b	32
^	x ^ y = x to the power y	13
&	logical and	20
	logical or	20
~	logical not	20
abs	abs (x) is the absolute value of (the elements of) x	19
atan	atan (x) is the arctangent of (the elements of) x	19
axis	axis (v), with v = [xmin xmax ymin ymax] replaces the automatical scaling of a graph's axes by the configuration given by the vector v. axis ('square') switches over from a rectangular graphical screen to a square-shaped graphical screen, axis ('normal') switches back to the rectangular screen.	25
chol	chol (A) yields the Cholesky decomposition of the matrix A.	19
clc	clc clears the command window and moves the cursor to the upper left corner	10

clear	clear clears the variables from the Matlab memory. clear x {y} removes the variable x {and y}	10
clf	clf clears the graphical window	10,25
close	close closes the graphical window	10,25
cos	cos(x) is the cosine of (the elements of) x	19
cputime	cputime determines the cpu time	10
demo	demo starts a demonstration	6,10
diag	diag (v,k) returns a square matrix with the elements of the vector v on the k th upper diagonal	15
diary	diary <i>filename</i> puts all following commands and results in a file with the name <i>filename</i> . This stops after diary off is entered	23
diff	diff (f,x) differentiates f w.r.t. x	34
disp	disp ('text') writes <i>text</i> on the screen. disp (x) displays the value of the variable x, without variable name	11,23,33
double	double (expr) converts the symbolic expression to a number	34
dsolve	dsolve (deq) solves the differential equation deq	34,35
eig	eig (A) computes the eigenvalues of A and returns a vector containing these eigenvalues	19
else, elseif	see if	20
end	end is the command that ends loop statements and conditional statements	20,22
exp	exp (x) is the exponent of (the elements of) x with base e	19
eye	eye (n) is the nxn identity matrix. eye (m,n) is an mxn matrix with ones on the diagonal and zeros elsewhere	14
factor	factor (expr) factors the symbolic expression expr	34
for	loop statement: for variable = start value : increment : end value, commands end	22
format	formats the output: format short - 5 digits, fixed point format short e - 5 digits, floating point format long - 15 digits, fixed point format long e - 15 digits, floating point standard configuration is short . format compact suppresses extra empty lines in the output format loose adds empty lines	11
fplot	fplot (@f,[a,b]) plots the function f on [a,b]	25,30
fprintf	C-command for advanced formatting of output	23
function	user defined function: function outputvar = <i>functionname</i> (inputvars) commands Function files have to have the name <i>functionname.m</i> .	28
ginput (n)	gets the coordinates of n points in the graph, which are located by positioning the cursor and thereupon clicking the mouse	25
global	global x changes x into a global variable	29

grid	adds a grid (a lattice of horizontal and vertical lines) to a graph	25														
help	help shows the functions you can get information about. help <i>functionname</i> shows this information on the screen. helpwin generates an extra window with helptopics	6,10														
hold	hold on keeps the last plot in the graphical screen. A new graph will be plotted on top of the existing one. hold off restores the default settings. In that case, when a plot command has been given, the graphical screen will be cleared before the new graph is plotted	25														
i	the imaginary unit	19														
if	conditional statement: <table style="width: 100%; border: none;"> <tr> <td style="width: 50%;">if statement</td> <td style="width: 50%;">if statement</td> </tr> <tr> <td style="padding-left: 20px;">commands</td> <td style="padding-left: 20px;">commands</td> </tr> <tr> <td>else</td> <td>elseif statement</td> </tr> <tr> <td style="padding-left: 20px;">commands</td> <td style="padding-left: 20px;">commands</td> </tr> <tr> <td>end</td> <td>else</td> </tr> <tr> <td></td> <td style="padding-left: 20px;">commands</td> </tr> <tr> <td></td> <td>end</td> </tr> </table>	if statement	if statement	commands	commands	else	elseif statement	commands	commands	end	else		commands		end	20
if statement	if statement															
commands	commands															
else	elseif statement															
commands	commands															
end	else															
	commands															
	end															
imag	imag(c) returns the imaginary part of the complex number c	13														
input	input('text') displays <i>text</i> on the screen and waits for input by the user. Can be assigned to a variable	24														
int	int(f,x,a,b) integrates f w.r.t. x from a to b	34														
length	length(v) returns the number of elements of the vector v	17														
load	load filename loads the variables of the file <i>filename</i> into the memory. The file is of type .mat or of type .txt.	24														
log	log(x) is the natural logarithm of (the elements of) x	19														
log10	log10(x) is the logarithm with base 10 of (the elements of) x	19														
lu	lu(A) computes the lu factorization of the matrix A	19,32														
max	max(v) returns the largest element of the vector v	19														
mesh	mesh(x,y,Z) plots the matrix Z versus the vectors y and x, creating a mesh	26														
meshgrid	meshgrid(x,y) creates a 2D grid from the vectors x,y	26														
min	min(v) returns the smallest element of the vector v	19														
norm	norm(v) computes the Euclidean norm of v and norm(v, inf) computes the infinity norm	17														
num2str	num2str(var) converts the number <i>var</i> to a text string	25														
ones	ones(n) is an nxn matrix filled with ones, ones(m,n) is an mxn matrix	14														
pause	pause pauses the running programme and waits until the user presses any key. pause(n) pauses during n seconds	6,10														
pi	pi is the machine's representation of π	19														
plot	Drawing a graph: <table style="width: 100%; border: none;"> <tr> <td style="width: 50%;">plot(v)</td> <td style="width: 50%;">- plot the vector v versus its indices</td> </tr> <tr> <td>plot(v,w)</td> <td>- plot the vector w versus the vector v</td> </tr> <tr> <td>plot(m,n,'symbol')</td> <td>- put a symbol at position (m,n) in the graph. The following symbols can be used: +, *, o and x</td> </tr> </table>	plot(v)	- plot the vector v versus its indices	plot(v,w)	- plot the vector w versus the vector v	plot(m,n,'symbol')	- put a symbol at position (m,n) in the graph. The following symbols can be used: +, *, o and x	6,25								
plot(v)	- plot the vector v versus its indices															
plot(v,w)	- plot the vector w versus the vector v															
plot(m,n,'symbol')	- put a symbol at position (m,n) in the graph. The following symbols can be used: +, *, o and x															
plot3	plot3(x,y,z) plots a line through the coordinates of vectors x, y, z	26														

print	direct the graph to the printer. print filename stores the graph in the file <i>filename</i>	25
quad	library routine: quad(f,0,1) computes the integral of the function <i>f</i> on [0,1].	
quit	logout from Matlab	6,10
real	real(c) returns the real part of the complex vector <i>c</i>	13
rem	rem(m,n) returns the remainder after division of <i>m</i> by <i>n</i>	19
rotate3d	enables the user to rotate the plot by mouse	26
round	round(x) rounds the elements of <i>x</i> to the nearest integer	19
save	save filename x {y} saves the variable <i>x</i> {and <i>y</i> } into the file <i>filename.mat</i>	23
script file	a script file is a file consisting of a sequence of Matlab commands. After you have typed the file name at the command prompt these commands will be executed.	28
shg	shg shows the most recent graphical screen	10,25
simplify	simplify(expr) simplifies the symbolic expression <i>expr</i>	34
sin	sin(x) is the sine of (the elements of) <i>x</i>	19
size	[m,n] = size(A) returns the size of the matrix <i>A</i>	17,33
solve	solve(expr,x) solve $\text{expr}=0$ for <i>x</i>	34,35
sparse	sparse(A) saves computations as well as memory space. It can be used when <i>A</i> is a band matrix, and the computations only involve non-zero elements.	32
sqrt	sqrt(x) is the square root of (the elements of) <i>x</i>	19
subs	subs(y,x,value) substitutes $x=\text{value}$ into <i>y</i>	34
sum	sum(v) is the sum of the elements of the vector of <i>v</i>	19
surf	surf(x,y,Z) plots the matrix <i>Z</i> versus the vectors <i>y</i> and <i>x</i> , creating a surface	26
surfz	same as surf , but adds a contour plot beneath the surface	26
sym	sym x declares <i>x</i> to be symbolic	34
syms	syms x y declares <i>x</i> and <i>y</i> to be symbolic	34,35
tan	tan(x) is the tangent of (the elements of) <i>x</i>	19
text	text(m,n,'text') writes <i>text</i> at position (<i>m,n</i>) of the graphical screen	25
tic,toc	tic and toc are stopwatch timer for the elapsed time	10
title	title('text') writes <i>text</i> as a title above the graph in the graphical screen	25
while	conditional loop statement: while statement commands end	22
whos	whos shows the name, size and type of the variables in the Matlab memory	10,12,33
xlabel	xlabel('text') places <i>text</i> underneath the x-axis of the graphical screen	25
ylabel	ylabel('text') places <i>text</i> next to the y-axis of the graphical screen	25
zeros	zeros(n) returns an <i>n</i> x <i>n</i> matrix with zeros; zeros(m,n) returns an <i>m</i> x <i>n</i> matrix	14

zoom	zoom makes it possible to zoom in at a specific point in the graph by clicking the mouse at that point	25
-------------	---	----