MATLAB® Handbook

L. Scuderi

English translation v.1.0 by A. Tibaldi

1 Starting commands

In Windows environments, MATLAB can be started by selecting with the mouse the corresponding icon and double-clicking on it. With MsDOS or Unix/Linux operating systems it is sufficient to write on the shell `matlab` and press enter. The symbol `>>` that will appear indicates the MATLAB prompt. An expression or a command can be executed by writing it and pressing enter. The working session can be terminated by executing the expressions `exit` or `quit`.

Table 1. Some useful commands during a MATLAB working session.

Command	**Description**
`help`	visualize all the available help topics
`help arg`	visualize information about topic `arg`
`doc arg`	visualize detailed information about topic `arg`
`clc`	clear the command window display, giving you a clean screen
`;`	after an expressions, avoid to print on screen its result
`...`	continue to write an expression in the successive row
`who`	visualize the variables in the MATLAB workspace (*i.e.*, in memory)
`whos`	visualize information about the variables in the MATLAB workspace
`clear`	clear all the variables from the workspace
`clear var1 var2`	clear the variables `var1` and `var2` from the workspace

2 Variables in MATLAB

The maximum number of characters allowed in a variable name is 32. Allowed characters are letters (capital or not), numbers, and the character "`_`" (underscore). The name of a variable must begin with a letter (`a-z`, `A-Z`). MATLAB is case-sensitive (*e.g.,* the names `a1` and `A1` are associated to different variables). A variable is automatically created by assigning a value or the result of an expression to it. The assignment is performed by using the symbol `=` as in the following example:

```
>> variable_name=expression
```

In order to create a string-type variable, the `expression` must be enclosed within a couple of inverted commas. Table 2 reports some default scalar variables.

MATLAB works with sixteen significant digits (double precision). In the command window, an output integer variable is usually printed without any decimal digit, whereas a real (non-integer) variable is plotted with only four decimal digits. A different output format can be adopted, by using one of the commands from Table 3. For example, in order to print all the sixteen digits of the MATLAB working precision, it is possible to use the command `format long e`.

Table 4 reports the main operations between scalar variables. In addition to the basic operations, MATLAB features the function listed in Table 5.

In order to define a vector, its elements should be introduced between square parentheses; the elements of a row vector should be separated by a space or a comma, whereas those of a column

Table 2. Examples of predefined MATLAB variables.

Variable	Description
`ans`	temporary variable containing the result from the last operation
`1i,j`	imaginary unit
`pi`	π, 3.14159265...
`eps`	*epsilon* of the machine
`realmax`	maximum positive machine number
`realmin`	minimum positive machine number
`Inf`	∞, *i.e.,* a number greater than `realmax`, or the result of 1/0
`NaN`	Not a Number (for example, the result of 0/0)

Table 3. Some possible output formats in MATLAB.

Format	Description
`format`	default format, equivalent to `format short`
`format short`	fixed-point representation with 4 decimal digits
`format long`	fixed-point representation with 14 decimal digits
`format short e`	floating-point representation with 4 decimal digits
`format long e`	floating-point representation with 15 decimal digits
`format rat`	irreducible fraction representation

Table 4. Operations between scalar variables.

Operation	Description
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	exponentiation

vector by a semicolon or by pressing enter after the introduction of each element. In MATLAB it is not possible to use null or negative indexes to identify the components of a vector. The command `x(i)` identifies the i-th component of the vector `x`, `x(end)` is the last element of `x` and `length(x)` returns the length of `x`. Table 6 reports some commands aimed at generating and manipulating vectors.

Table 7 reports few operations that can be applied on the vector $x = (x_i)_{i=1,\dots,n}$ to compute some quantities associated to it.

The function `diag` can be used also as `diag(x,k)`, with k integer positive or negative; in this case it generates a square matrix with dimension $n + |k|$ with all-zero entries except those of the k-th diagonal above ($k > 0$) or below ($k < 0$) the main diagonal, which are set to be the components of the vector x.

The elements of a matrix should be inserted between square parentheses by rows, where each row is terminated by a semicolon or by pressing enter. The command `A(i,j)` identifies the (i, j) element of the matrix `A`, `size(A)` generates a row vector containing the number of rows and of columns of `A`, and `length(A)`, applied to a matrix, is equivalent to `max(size(A))`.

Table 8 reports some MATLAB functions that generate particular square and/or rectangular matrices. It is to be remarked that, in Table 8, when $m \equiv n$ it is sufficient to insert only the parameter n.

Table 9 reports some commands that, applied to the matrix $A = (a_{ij})_{i,j=1,\dots,n}$, return particular

Table 5. Examples of predefined MATLAB functions.

Function	Description
`sin`	sine
`cos`	cosine
`asin`	arcsine
`acos`	arccosine
`tan`	tangent
`atan`	arctangent
`exp`	exponential
`log`	natural logarithm
`log2`	logarithm with base 2
`log10`	logarithm with base 10
`sqrt`	square root
`abs`	absolute value
`real`	real part
`imag`	imaginary part
`sign`	sign function
`factorial`	factorial
`round`	round to the closest integer number
`floor`	round to the closest lower integer number
`ceil`	round to the closest upper integer number
`chop(x,t)`	round `x` with `t` significant digits

Table 6. Some commands aimed at generating and manipulating vectors.

Command	Description
`x.'`	generate the transposed vector of `x`
`x=[]`	generate the empty vector `x`
`sort(x)`	sort in ascending order the components of `x`
`x=[a:h:b]`	generate the row vector $x = (x_i)_{i=1,\dots,m+1}$ where $x_i = a + (i-1)h$, where m is the floor of $(b-a)/h$
`x=linspace(a,b, n)`	generate the row vector $x = (x_i)_{i=1,\dots,n}$ where $x_i = a + (i-1)h$, where $h = (b-a)/(n-1)$
`x=logspace(a,b,n)`	generate the row vector $x = (10^{x_i})_{i=1,\dots,n}$ where $x_i = a + (i-1)h$, where $h = (b-a)/(n-1)$
`x(r)`	return the components of `x` which indexes are specified in the vector `r`
`x(r)=z`	assign to the components of `x` (which indexes are specified in the vector `r`) the respective values of `z`
`x(r)=[]`	remove from the vector `x` some components (which indexes are specified in `r`)
`x([i j])=x([j i])`	switch the components i and j of the vector `x`

quantities associated to it. The functions described in Table 9 can be applied to rectangular matrices as well. Moreover, the functions `sum`, `max` and `min` can be applied also as `sum(A,k)`, `max(A,[],k)` and `min(A,[],k)`, with $k = 1, 2$. For $k = 1$ they act as in Table 9. For $k = 2$ they generate a column vector $x = (x_i)_{i=1,\dots,n}$, with $x_i = \sum_{j=1}^{n} a_{ij}$, $x_i = \max_j a_{ij}$ e $x_i = \min_j a_{ij}$, respectively. The function `diag` can be used also as `diag(A,k)`, with `k` positive or negative integer; in this case it generates a column vector containing the elements of the k-th diagonal above $(k > 0)$ or below $(k < 0)$ the main diagonal. Also the function `tril` (`triu`) can be used as `tril(A,k)` (`triu(A,k)`),

Table 7. Some predefinite MATLAB functions acting on a vector x.

Command	Description
a=sum(x)	generate the scalar $a = \sum_{i=1}^{n} x_i$
a=prod(x)	generate the scalar $a = \prod_{i=1}^{n} x_i$
a=max(x)	generate the scalar $a = \max_i x_i$
a=min(x)	generate the scalar $a = \min_i x_i$
a=norm(x)	generate the scalar $a = \|\|x\|\|_2$
a=norm(x,1)	generate the scalar $a = \|\|x\|\|_1$
a=norm(x,inf)	generate the scalar $a = \|\|x\|\|_\infty$
A=diag(x)	generate the diagonal matrix $A = (a_{ij})_{i,j=1,\dots,n}$, con $a_{ii} = x_i$

Table 8. Some commands aimed at generating and manipulating matrices.

Command	Description
A=[]	generate the empty matrix A
A.'	generate the transposed of the matrix A
A=eye(n)	generate the identity matrix $A = (a_{ij})_{i,j=1,\dots,n}$, with $a_{ij} = \delta_{ij}$
A=zeros(n,m)	generate the matrix $A = (a_{ij})_{i=1,\dots,n,j=1,\dots,m}$, with $a_{ij} = 0$
A=ones(n,m)	generate the matrix $A = (a_{ij})_{i=1,\dots,n,j=1,\dots,m}$, with $a_{ij} = 1$
A=rand(n,m)	generate the matrix $A = (a_{ij})_{i=1,\dots,n,j=1,\dots,m}$, with pseudo-casual $0 < a_{ij} < 1$
A=hilb(n)	generate the Hilbert matrix $A = (a_{ij})_{i,j=1,\dots,n}$, with $a_{ij} = 1/(i+j-1)$
A=vander(x)	generate the Vandermonde matrix $A = (a_{ij})_{i,j=1,\dots,n}$, with $a_{ij} = x_i^{n-j}$
A(r,c)	return the elements of A belonging to the intersections of the rows and columns specified in r and c respectively
A(r,c)=C	assigns to the elements of A (whose row and column indexes are specified in r e in c) the respective values defined in C.
A(r,c)=[]	removes from the matrix A some elements (whose row and column indexes are specified in r and c)
A([i j],c)=A([j i],c)	switch the elements of the rows i and j of A belonging to the columns specified in c
A(r,[i j])=A(r,[j i])	switch the elements of the columns i e j di A belonging to the rows specified in r

with k positive or negative integer; in this case it returns the upper (lower) triangular part starting from the k-th diagonal above ($k > 0$) or below ($k < 0$) the main diagonal.

The elementary operations acting on scalars can be naturally extended (when well-defined) to vectors and matrices, exception made for division and power. The operation * returns the row-column product. For square matrices the command A^k with integer positive k returns the row-column product of the matrix A with itself, for k times; A^(-1) generates the inverse of A, if it is nonsingular.

In addition to the classic sum and product operations between vectors or in general matrices, MATLAB supports element-wise operations, which act directly element by element. These operations can be performed by writing a point before the symbol defining the operation.

Given the row (column) vectors $x = (x_i)_{i=1,\dots,n}$, $y = (y_i)_{i=1,\dots,n}$, the matrices $A = (a_{ij})_{i,j=1,\dots,n}$, $B = (b_{ij})_{i,j=1,\dots,n}$ and the positive real number e, Table 10 reports a list of element-wise operations.

Table 9. Some predefined MATLAB functions acting on a matrix `A`.

Command	Description
`a=norm(A)`	generate the scalar $a = \|\|A\|\|_2$
`a=norm(A,1)`	generate the scalar $a = \|\|A\|\|_1$
`a=norm(A,inf)`	generate the scalar $a = \|\|A\|\|_\infty$
`x=sum(A)`	generate the scalar $x = (x_j)_{j=1,\ldots,n}$, with $x_j = \sum_{i=1}^n a_{ij}$
`x=max(A)`	generate the row vector $x = (x_j)_{j=1,\ldots,n}$, with $x_j = \max_i a_{ij}$
`x=min(A)`	generate the row vector $x = (x_j)_{j=1,\ldots,n}$, with $x_j = \min_i a_{ij}$
`x=diag(A)`	generate the column vector $x = (x_i)_{i=1,\ldots,n}$, with $x_i = a_{ii}$
`B=abs(A)`	generate the matrix $B = (b_{ij})_{i,j=1,\ldots,n}$, with $b_{ij} = \|a_{ij}\|$
`B=tril(A)`	generate the lower triangular matrix $B = (b_{ij})_{i,j=1,\ldots,n}$, with $b_{ij} = a_{ij}$, $i = 1,\ldots,n$, $1 \leq j \leq i$
`B=triu(A)`	generate the upper triangular matrix $B = (b_{ij})_{i,j=1,\ldots,n}$, with $b_{ij} = a_{ij}$, $i = 1,\ldots,n$, $i \leq j \leq n$

Table 10. Element-wise operations in MATLAB.

Operation	Description
`z=x.*y`	generate the row (column) vector $z = \{z_i\}_{i=1,\ldots,n}$, with $z_i = x_i * y_i$
`z=x./y`	generate the row (column) vector $z = \{z_i\}_{i=1,\ldots,n}$, with $z_i = x_i/y_i$
`z=x.^y`	generate the row (column) vector $z = \{z_i\}_{i=1,\ldots,n}$, with $z_i = x_i^{y_i}$
`z=x.^e`	generate the row (column) vector $z = \{z_i\}_{i=1,\ldots,n}$, with $z_i = x_i^e$
`C=A.*B`	generate the matrix $C = (c_{ij})_{i,j=1,\ldots,n}$, with $c_{ij} = a_{ij} * b_{ij}$
`C=A./B`	generate the matrix $C = (c_{ij})_{i,j=1,\ldots,n}$, with $c_{ij} = a_{ij}/b_{ij}$
`C=A.^B`	generate the matrix $C = (c_{ij})_{i,j=1,\ldots,n}$, with $c_{ij} = a_{ij}^{b_{ij}}$
`C=A.^e`	generate the matrix $C = (c_{ij})_{i,j=1,\ldots,n}$, with $c_{ij} = a_{ij}^e$

3 Graphics in MATLAB

In order to plot a function f with variable x it is possible to use the function `fplot`

```
fplot('f',[xmin xmax])
```

where `f` is the expression to be plotted and `[xmin xmax]` is a vector of two variables, containing the boundaries of the x axis for the plot. It is possible to specify also the boundaries of the representation for the y axis, by using the vector `[xmin xmax ymin ymax]` as second argument of the function `fplot`.

Alternatively, it is possible to use the command `plot` according to the syntax

```
plot(x,y)
```

where `x` is a vector containing a set of coordinates in the x axis, and `y` is obtained by evaluating the function f in the points specified in `x`.

Both `plot` and `fplot` commands allow to customize the color and type (solid, dashed, dash-dotted...) of the line. Moreover, `plot` allows to introduce markers correspondingly to the points (x_i, y_i) contained in the vectors `x` and `y`. Tables 11 and 12 list some of the possible options and existing commands aimed at commenting a plot.

In order to use a logarithmic scale on the x, y or both axis, instead of `plot` it is possible to use the commands `semilogx`, `semilogy` or `loglog`, respectively. The syntax and options of these commands are the same as those of `plot`.

Table 11. List of options for commands `plot` and `fplot`.

Color	Description	Symbol	Description	Line	Description
`w`	white	`.`	dot	`-`	solid line
`y`	yellow	`o`	circle	`:`	dotted line
`r`	red	`x`	cross	`-.`	dash-dotted line
`g`	green	`+`	plus	`--`	dashed line
`b`	blue	`*`	star		
`k`	black	`s`	square		

Table 12. List of commands for commenting a plot.

Command	Description
`title`	introduce a title to the plot
`xlabel`	introduce a label for the x axis
`ylabel`	introduce a label for the y axis
`grid`	introduce a grid for the x and y axis
`legend`	introduce a legend to describe the different curves in the plot
`text`	introduce a text string in a specific location
`gtext`	introduce a text string in a location selected with the mouse

At every call of `plot`, Matlab erases by default the previous plot and traces the new one. In order to draw more graphs in the same figure, it is possible to use the command `hold on` before calling the command `plot` to trace the second curve. Alternatively, it is possible to use the command `plot` with the following syntax:

```
plot(x_1,y_1,'-',x_2,y_2,':')
```

It is possible to draw several plots in the same graphical window, in different sub-plots, by using the command `subplot`

```
subplot(rows,columns,subfigure)
```

where `rows` and `columns` indicate the number of rows and columns of the matrix of subfigures of the window, and `subfigure` indicates the subfigure to be considered for plotting a graph. Subfigures are enumerated from left to right and from top to bottom. It is possible to open several graphical windows (or "figures"). The figure `n` can be activated by means of the command `figure(n)`; to close this figure, it is possible to use the command `close(n)`. In order to close all the figures, it is possible to use the command `close all`.

4 MATLAB programs

A file containing several Matlab instructions is generally referred to as *m-file*. This name derives from its extension "`.m`". A *m-file* name can contain letters and/or numbers, and the character `_` (underscore). In a *m-file*, commands should be introduced on different lines or on the same line (separated by comma or semicolon). Comments can be introduced after the special character `%`. There are two categories of *m-files*: *scripts* or *functions*.

- *Scripts* are given by a sequence of Matlab commands. A *script m-file* can be executed by selecting the directory where it is saved, and inserting its name (without extension) in the Matlab prompt. *Script* files do not allow input or output passing parameters. Moreover, all the variables defined in a *script* are kept in memory, in the workspace, just like those defined directly in the command prompt.

- The *function*-type *m-files* must start with the syntax

  ```
  function [y_1,y_2,...,y_n]=function_name(x_1,x_2,...,x_m)
  ```

  where `y_1,y_2,...,y_n` are the output parameters, and `x_1,x_2,...,x_m` are the input parameters. The string `function_name` is the name of the *function*; this must coincide with the *m-file* name (appending, of course, the file extension `.m`) where the *function* is saved. A *function* can be executed in the command prompt, o from a *script*, or by another *function*, by writing

  ```
  [y_1,y_2,...,y_n]=function_name(x_1,x_2,...,x_m)
  ```

  or

  ```
  function_name(x_1,x_2,...,x_m)
  ```

  In the latter case the *function* returns only the first output parameter, which is then saved in the variable `ans`.

  It is to be remarked that a value must be assigned for each output parameter; otherwise, MATLAB returns the following error message

  ```
  ??? One or more output arguments not assigned during call to 'function_name'
  ```

  Unlike *scripts*, *function m-files* require input and output parameters, and the variables defined during their execution are local, and erased from the MATLAB memory workspace after the execution of the *function*.

4.1 MATLAB coding: syntax constructs

The MATLAB programming language supports classical syntax constructs. Table 13 lists the main MATLAB relational operators.

Table 13. Relational operators in MATLAB.

Operator	Description
`<`	lesser
`>`	greater
`<=`	lesser or equal
`>=`	greater or equal
`==`	equal
`~=`	not equal

Relational operators allow to perform comparison between expressions. In the last MATLAB versions, the result of these comparisons is assigned to `logical`-type variables. Specifically, `0` is used to represent "false" and `1` represents "true".

Relational operators can be combined through the logical operators listed in Table 14. Table 15 lists the results of logical operators when applied to two conditions `a` e `b`.

In the following we summarize some programming structures available in MATLAB.

Table 14. MATLAB logical operators.

Operator	Description
&	and
|	or
∼	not
xor	exclusive or

Table 15. Logical operators applied to two conditions `a` and `b`.

a	b	a & b	a | b	∼a	xor(a,b)
0	0	0	0	1	0
1	0	0	1	0	1
0	1	0	1	1	1
1	1	1	1	0	0

- **unconditional loop controlled by a counter**

```
for index = expression
    instruction block
end
```

where `index` assumes the values defined by `expression`.

- **conditional loop**

```
while condition
    instruction block
end
```

where `condition` is an expression that MATLAB evaluates and interprets as true if different from `0`, and false if equal to `0`.

- **conditional structures**

```
if condition_1
    instruction block 1
elseif condition_2
    instruction block 2

      ⋮

else
    instruction block n
end
```

where `instruction block 1` will be executed only if `condition_1` is true, `instruction block 2` only if `condition_1` is false and `condition_2` is true, and so on. The `instruction block n` is executed only if no one of the previous conditions is true. The instructions `elseif` ande `else` can be omitted if unnecessary.

The command `return` allows to terminate the execution of a program before it reaches the last instruction.

Moreover, MATLAB supports the command `break`, which allows to terminate a cycle before it reaches its natural end. When this command is executed, MATLAB jumps directly to the `end` instruction, terminating the loop.

In order to estimate the efficiency of a program, it is possible to measure its execution time by means of the commands `tic` and `toc`. These commands return the number of seconds required to execute an `instruction block`, by means of the syntax

```
tic
     instruction block;
elapsed_time=toc
```

`tic` activates the timer, `toc` stops it and saves the time needed to execute the instructions between `tic` and `toc` in the variable '"`elapsed_time`".

5 Main MATLAB functions for numerical analysis

Table 16. Linear algebra.

Function	Description
`lu(A)`	generate the PA=LU (Gauss) factorization with partial pivoting
`chol(A)`	generate the Choleski factorization of the matrix A
`qr(A)`	generate the QR factorization of the matrix A
`x=A\b`	solve the linear system $Ax = b$
`cond(A)`	compute the spectral conditioning number (norm 2) of A
`cond(A,1)`	compute the norm 1 conditioning number of A
`cond(A,inf)`	compute the norm ∞ conditioning number of A
`rcond(A)`	compute the reciprocal of the norm 1 conditioning number of A
`rank(A)`	compute the rank of A
`det(A)`	compute the determinant of A
`inv(A)`	compute the inverse of A
`eig(A)`	compute the eigenvalues and eigenvectors of A

Table 17. Polynomials, functions and approximation.

Function	Description
`polyval`	evaluate a polynomial
`f=inline('expression','x_1',...,'x_n')`	define the function $f(x_1, \ldots, x_n) = expression$
`y=f(x_1,...,x_n)`	evaluate the function $y = f(x_1, \ldots, x_n)$ defined by `inline`
`y=feval(f,x_1,...,x_n)`	evaluate the function $y = f(x_1, \ldots, x_n)$ defined by `inline` or by a *function m-file*
`polyfit`	compute the coefficients of the polynomial interpolating or approximating a set of data according to the least squares criterion
`spline`	evaluate a cubic interpolating spline

Table 18. Equations and systems of nonlinear equations.

Function	**Description**
`fzero`	evaluate the zeros of a nonlinear function
`roots`	evaluate the zeros of a polynomial
`fsolve`	solve a system of nonlinear equations

Table 19. Integrals.

Function	**Description**
`quad`	adaptive Simpson formula
`quadl`	adaptive Gauss-Lobatto formula

Table 20. Equations and systems of ordinary differential equations (ODE).

Function	**Method**
`ode45`	Order 4 and 5 explicit Runge-Kutta
`ode113`	Variable-order Adams-Moulton
`ode23`	Order 4 and 5 explicit Runge-Kutta
`ode23t`	Trapezoidal method
`ode15s`	Variable-order implicit linear multi-step
`ode23s`	Order 2 implicit Runge-Kutta