Estratto da Kermit Sigmon "MATLAB Primer", Third Edition, Department of Mathematics, University of Florida

The name MATLAB is derived from MATrix LABoratory.

MATRICES

MATLAB works with essentially only one kind of object: a rectangular numerical **matrix** with possibly complex entries; all variables represent matrices. In some situations, 1-by-1 matrices are interpreted as scalars and matrices with only one row or one column are interpreted as vectors.
For example, either of the statements

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

and

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

creates the obvious 3-by-3 matrix and assigns it to a variable A.
The elements within a row of a matrix may be separated by commas as well as a blank.
A matrix can be contained in a file. If this file is named, say, *data.ext* (where *.ext* is any extension), the MATLAB command **load data.ext** will read this file to the variable *data* in your MATLAB workspace.
The built-in functions **rand** and **magic**, for example, provide an easy way to create matrices. The command **rand(n)** will create an nxn matrix with randomly generated entries distributed uniformly between 0 and 1, while **rand(m,n)** will create an mxn one. **magic(n)** will create an integral nxn matrix which is a magic square (rows, columns, and diagonals have common sum).
Individual matrix and vector entries can be referenced with indices inside parentheses in the usual manner:
A(2,3) denotes the entry in the second row, third column of matrix A; *x(3)* denotes the third coordinate of vector x.

MATRIX OPERATIONS, ARRAY OPERATIONS

The following matrix operations are available in MATLAB:
+ addition
- subtraction

* multiplication
^ power
′ conjugate transpose
\ left division
/ right division
These matrix operations apply, of course, to scalars (1-by-1 matrices) as well. If the sizes of the matrices are incompatible for the matrix operation, an error message will result, except in the case of scalar-matrix operations (for addition, subtraction, and division as well as for multiplication) in which case each entry of the matrix is operated on by the scalar.

ARRAY OPERATIONS

The matrix operations of addition and subtraction already operate entry-wise; the other operations *, ^, \, and /, can be made to operate entry-wise by preceding them by a period. For example, either [1,2,3,4].*[1,2,3,4] or [1,2,3,4].^2 will yield [1,4,9,16].

STATEMENTS, EXPRESSIONS, AND VARIABLES

MATLAB is an *expression* language; the expressions you type are interpreted and evaluated. MATLAB statements are usually of the form

variable = *expression*, or simply
expression

Expressions are usually composed from operators, functions, and variable names. Evaluation of the expression produces a matrix, which is then displayed on the screen and assigned to the variable for future use. If the variable name and = sign are omitted, a variable *ans* (for *answer*) is automatically created to which the result is assigned.
A statement is normally terminated with the carriage return. However, a statement can be continued to the next line with three or more periods followed by a carriage return. On the other hand, several statements can be placed on a single line if separated by commas or semicolons.
<u>If the last character of a statement is a semicolon, the printing is suppressed, but the assignment is carried out. This is essential in suppressing unwanted printing of intermediate results.</u>

The command **who** (or **whos**) will list the variables currently in the workspace. A variable can be cleared from the workspace with the command **clear** *variablename*. The command **clear** alone will clear all nonpermanent variables.
The permanent variable *eps* (epsilon) gives the machine unit roundoff - about 10^{-16} on most machines. It is useful in specifying tolerences for convergence of iterative processes.
A runaway display or computation can be stopped on most machines without leaving MATLAB with CTRL-C (CTRL-BREAK on a PC).

SAVING A SESSION

The command **save** before exiting MATLAB causes all variables to be written to a non-human-readable diskfile named *matlab.mat*. When one later reenters MATLAB, the command **load** will restore the workspace to its former state.

MATRIX BUILDING FUNCTIONS

Convenient matrix building functions are

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	create or extract diagonals
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square

For example, **zeros(m,n)** produces an m-by-n matrix of zeros and **zeros(n)** produces an n-by-n one. If A is a matrix, then **zeros(size(A))** produces a matrix of zeros having the same size as A.
If x is a vector, **diag(x)** is the diagonal matrix with x down the diagonal; if A is a square matrix, then **diag(A)** is a vector consisting of the diagonal of A.
Matrices can be built from blocks. For example, if A is a 3-by-3 matrix, then

```
B = [A, zeros(3,2); zeros(2,3), eye(2)]
```

will build a 5-by-5 matrix.

FOR

For example, for a given n, the statement

```
x = []; for i = 1:n, x=[x,i^2], end
```

or

```
x =[];
for i = 1:n
    x = [x,i^2]
end
```

will produce a certain n-vector and the statement

```
x = []; for i = n:-1:1, x=[x,i^2], end
```

will produce the same vector in reverse order.
The statements

```
for i = 1:m
    for j= 1:n
        H(i, j) = 1/(i+j-1);
    end
end
H
```

will produce and print to the screen the m-by-n hilbert matrix. The semicolon on the inner statement is essential to suppress printing of unwanted intermediate results while the last H displays the final result.
The **for** statement permits any matrix to be used instead of 1:n. The variable just consecutively assumes the value of each column of the matrix. For example,

```
s =0;
for c = A
s = s + sum(c);
end
```

computes the sum of all entries of the matrix A by adding its column sums (Of course, *sum(sum(A))* does it more efficiently).

WHILE

The general form of a while loop is

while *relation*
statements

```
end
```

For example, for a given number a, the following will compute and display the smallest nonnegative integer n such that 2^n >= a:

```
n =0;
while 2^n <a
    n = n+ 1;
end
n
```

IF

The general form of a simple **if** statement is

if *relation*
statements
end

Multiple branching is also possible, as is illustrated by

```
if n < 0
    parity = 0;
elseif rem(n,2) == 0
    parity = 2;
else
    parity = 1;
end
```

In two-way branching the elseif portion would, of course, be omitted.

RELATIONS

The relational operators in MATLAB are

< less than
> greater than
<= less than or equal
>= greater than or equal
== equal
~= not equal.

Relations may be connected or quantified by the logical operators

& and
¦ or
~ not.

When applied to scalars, a relation is actually the scalar 1 or 0 depending on whether the relation is true or false. Try entering 3 < 5,3 > 5,3 == 5,and 3 == 3. When applied to matrices of the same size, a relation is a matrix of 0's and 1's giving the value of the relation between corresponding entries.

A relation between matrices is interpreted by **while** and **if** to be true if each entry of the relation matrix is nonzero. Hence, if you wish to execute *statement* when matrices A and B are equal you could type

if A == B
    statement
end

but if you wish to execute statement when A and B are not equal, you would type

if A == B else
statement
end

Note that the seemingly obvious

if A ~= B, *statement*, end

will not give what is intended since *statement* would execute only if *each* of the corresponding entries of A and B differ.

SCALAR FUNCTIONS

Certain MATLAB functions operate essentially on scalars, but operate element-wise when applied to a matrix. The most common such functions are

sin	asin	exp	abs	round
cos	acos	log (natural log)	sqrt	floor
tan	atan	rem (remainder)	sign	ceil

VECTOR FUNCTIONS

Other MATLAB functions operate essentially on a vector (row or column), but act on an m-by-n matrix (m >= 2) in a column-by-column fashion to produce a row vector containing the results of their application to each column. Row-by-row action can be obtained by using the transpose; for example, **mean**(A')'. A few of these functions are

max	sum	median	any
min	prod	mean	all
sort		std	

For example, the maximum entry in a matrix A is given by **max**(**max**(A)) rather than **max**(A).

MATRIX FUNCTIONS

Useful matrix functions are

eig	eigenvalues and eigenvectors
inv	inverse
poly	characteristic polynomial
det	determinant
size	size
norm	1-norm, 2-norm, F-norm, 1-norm
rank	rank
……..	

MATLAB functions may have single or multiple output arguments. For example,

y = eig(A), or simply eig(A)

produces a column vector containing the eigenvalues of A while

[U,D] = eig(A)

produces a matrix U whose columns are the eigenvectors of A and a diagonal matrix D with the eigenvalues of A on its diagonal.

SUBMATRICES AND COLON NOTATION

Vectors and submatrices are often used in MATLAB to achieve fairly complex data manipulation effects. “Colon notation" (which is used both to generate vectors and reference submatrices) and subscripting by integral vectors are keys to efficient manipulation of these objects.

The expression 1:5 is actually the row vector [1 2 3 4 5]. The numbers need not be integers nor the increment one. For example,

0.2:0.2:1.2

gives [0.2, 0.4, 0.6, 0.8, 1.0, 1.2], and

5:-1:1 gives [5 4 3 2 1].

The following statements will, for example, generate a table of sines.

```
x = [0.0:0.1:2.0]′;
y = sin(x);
[x y]
```


Note that since **sin** operates entry-wise, it produces a vector y from the vector x.

The colon notation can be used to access submatrices of a matrix. For example,

A(1:4,3) is the column vector consisting of the first four entries of the third column of A.

A colon by itself denotes an entire row or column:

A(:,3) is the third column of A, and A(1:4,:) is the first four rows.

Arbitrary integral vectors can be used as subscripts:

A(:,[2 4]) contains as columns, columns 2 and 4 of A.

M-FILES

MATLAB can execute a sequence of statements stored in diskfiles. Such files are called “M-files" because they must have the file type of “.m" as the last part of their filename.

There are two types of M-files: *script files* and *function files*.

SCRIPT FILES

A script file consists of a sequence of normal MATLAB statements. If the file has the filename, say, **rotate.m**, then the MATLAB command **rotate** will cause the statements in the file to be executed. Variables in a script file are global and will change the value of variables of the same name in the environment of the current MATLAB session.

Script files may be used to enter data into a large matrix; in such a file, entry errors can be easily corrected. If, for example, one enters in a diskfile **data.m**

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

then the MATLAB statement **data** will cause the assignment given in **data.m** to be carried out. However, it is usually easier to use the MATLAB function **load**.

An M-file can reference other M-files, including referencing itself recursively.

FUNCTION FILES

Function files provide extensibility to MATLAB. You can create new functions specific to your problem which will then have the same status as other MATLAB functions. Variables in a function file are by default local. A variable can, however, be declared global (see **help global**).

We first illustrate with a simple example of a function file.

```
function a = randint(m,n)
%RANDINT Randomly generated integral matrix.
% randint(m,n) returns an m-by-n such matrix with entries
% between 0 and 9.
a = floor(10*rand(m,n));
```

A more general version of this function is the following:

```
function a = randint(m,n,a,b)
%RANDINT Randomly generated integral matrix.
% randint(m,n) returns an m-by-n such matrix with entries
% between 0 and 9.
% rand(m,n,a,b) return entries between integers a and b.
if nargin < 3, a = 0; b = 9; end
a = floor((b-a+1)*rand(m,n)) + a;
```

This should be placed in a diskfile with filename **randint.m** (corresponding to the function name). The first line declares the function name, input arguments, and output arguments; without this line the file would be a script file. Then a MATLAB statement **z = randint(4,5)**, for example, will cause the numbers 4 and 5 to be passed to the variables m and n in the function file with the output result being passed out to the variable z. Since variables in a function file are local, their names are independent of those in the current MATLAB environment.

Note that use of **nargin** ("number of input arguments") permits one to set a default value of an omitted input variable - such as a and b in the example.

A function may also have multiple output arguments. For example:


```
function [mean, stdev] = stat(x)
%    Mean and standard deviation
%    For a vector x, stat(x) returns the mean of x;
%    [mean, stdev] = stat(x) both the mean and
%    standard deviation.
……….
```

Once this is placed in a diskfile **stat.m**, a MATLAB command

```
[xm, xd] = stat(x)
```

for example, will assign the mean and standard deviation of the entries in the vector x to xm and xd, respectively. Single assignments can also be made with a function having multiple output arguments. For example, **xm = stat(x)** (no brackets needed around xm) will assign the mean of x to xm.

The % symbol indicates that the rest of the line is a comment; MATLAB will ignore the rest of the line. Moreover, the first few contiguous comment lines, which document the M-file, are available to the on-line help facility and will be displayed if, for example, **help stat** is entered. Such documentation should *always* be included in a function file.

As noted earlier, some of the input arguments of a function may be made optional through use of **nargin** ("number of input arguments"). The variable **nargout** can be similarly used.

Note that, when **while** or **if** evaluates a relation, "nonzero" means "true" and 0 means "false".

Finally, the MATLAB function **feval** permits one to have as an input variable a string naming another function. (Also see **eval**).

The actual listing of an M-file can be viewed with the MATLAB command **type** *functionname*.

TEXT STRINGS, ERROR MESSAGES, INPUT

Text strings are entered into MATLAB surrounded by single quotes. For example,

```
s = 'This is a test'
```

assigns the given text string to the variable s.

Text strings can be displayed with the function **disp**. For example:

```
disp('this message is hereby displayed')
```

Error messages are best displayed with the function **error**

```
error('Sorry, the matrix must be symmetric')
```

since when placed in an M-File, it aborts execution of the M-file.

In an M-file the user can be prompted to interactively enter input data with the function **input**. When, for example, the statement

```
iter = input('Enter the number of iterations: ')
```

is encountered, the prompt message is displayed and execution pauses while the user keys in the input data. Upon pressing the return key, the data is assigned to the variable iter and execution resumes.

MANAGING M-FILES

The command **pwd** will return the name of the present working directory and **cd** can be used to change the working directory. Either **dir** or **ls** will list the contents of the working directory while the command **what** lists only the M-files in the directory. The MATLAB commands **delete** and **type** can be used to delete a diskfile and print an M-file to the screen, respectively.

M-files must be in a directory accessible to MATLAB. M-files in the present working directory are always accessible. The current list of directories in MATLAB's search path is obtained by the command **path**. This command can also be used to add or delete directories from the search path.

GRAPHICS

MATLAB can produce planar plots of curves, 3-D plots of curves, …... The primary commands for these facilities are **plot**, **plot3**, …. An introduction to each of these is given below.

Planar plots

The **plot** command creates linear x-y plots; if x and y are vectors of the same length, the command **plot(x,y)** opens a graphics window and draws an x-y plot of the elements of x versus the elements of y. You can, for example, draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
x = -4:.01:4; y = sin(x); plot(x,y)
```

The vector x is a partition of the domain with meshsize 0.01 while y is a vector giving the values of sine at the nodes of this partition (recall that sin operates entrywise).

One can have several graphics figures, one of which will at any time be the designated "current" figure where graphs from subsequent

plotting commands will be placed. If, for example, figure 1 is the current figure, then the command **figure(2)** (or simply **figure**) will open a second figure (if necessary) and make it the current figure. The command **figure(1)** will then expose figure 1 and make it again the current figure. The command **gcf** will return the number of the current figure.

As a second example, you can draw the graph of $y = e^{-x^2}$ over the interval -1.5 to 1.5 as follows:

```
x = -1.5:.01:1.5; y = exp(-x.^2); plot(x,y)
```

MATLAB supplies a function **fplot** to easily and efficiently plot the graph of a function. For example, to plot the graph of the function above, one can first define the function in an M-file called, say, **expnormal.m** containing

```
function y = expnormal(x)
y = exp(-x.^2);
```

Then the command

```
fplot('expnormal', [-1.5,1.5])
```

will produce the graph.

Plots of parametrically defined curves can also be made. For example,

```
t=0:.001:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)
```

The graphs can be given titles, axes labeled, and text placed within the graph with the following commands which take a string as an argument.

title	graph title
xlabel	x-axis label
ylabel	y-axis label
gtext	place text on the graph using the mouse
text	position text at specified coordinates

For example, the command

```
title('Best Least Squares Fit')
```

gives a graph a title. The command **gtext('The Spot')** allows one to interactively place the designated text on the current graph by placing the mouse pointer at the desired position and clicking the mouse. To place text in a graph at designated coordinates, one would use the command **text** (see **help text**).

The command **grid** will place grid lines on the current graph.

By default, the axes are auto-scaled. This can be overridden by the command **axis**.
Some features of axis are:

axis([x_{min} ,x_{max} ,y_{min} ,y_{max}])	set axis scaling to prescribed limits
axis(axis)	freezes scaling for subsequent graphs
axis auto	returns to auto-scaling
v = axis	returns vector v showing current scaling
axis square	same scale on both axes
axis equal	same scale and tic marks on both axes
axis off	turns off axis scaling and tic marks
axis on	turns on axis scaling and tic marks

The **axis** command should be given *after* the **plot** command.
Two ways to make multiple plots on a single graph are illustrated by

```
x=0:.01:2*pi;y1=sin(x);y2=sin(2*x);y3=sin(4*x);plot(x,y1,x,y2,x,y3)
```

and by forming a matrix Y containing the functional values as columns

```
x=0:.01:2*pi; Y=[sin(x)', sin(2*x)', sin(4*x)']; plot(x,Y)
```

Another way is with **hold**. The command **hold on** freezes the current graphics screen so that subsequent plots are superimposed on it. The axes may, however, become rescaled.
Entering **hold off** releases the "hold".

One can override the default linetypes, pointtypes and colors. For example,

```
x=0:.01:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
```

renders a dashed line and dotted line for the first two graphs while for the third the symbol + is placed at each node. The line- and mark-types are

Linetypes: solid (-), dashed (--). dotted (:), dashdot (-.)
Marktypes: point (.), plus (+), star (*), circle (o), x-mark (x)

Colors can be specified for the line- and mark-types.
Colors: yellow (y), magenta (m), cyan (c), red (r)
green (g), blue (b), white (w), black (k)
For example, **plot(x,y,'r--')** plots a red dashed line.
The command **subplot** can be used to partition the screen so that several small plots can be placed in one figure. See **help subplot**.

GRAPHICS HARDCOPY

A hardcopy of the current graphics figure can be most easily obtained with the MATLAB command **print**. Entered by itself, it will send a high-resolution copy of the current graphics figure to the default printer.
The command **print** *filename* saves the current graphics figure to the designated filename.

3-D LINE PLOTS

Completely analogous to **plot** in two dimensions, the command **plot3** produces curves in three dimensional space. If x, y, and z are three vectors of the same size, then the command **plot3(x,y,z)** will produce a perspective plot of the piecewise linear curve in 3-space passing through the points whose coordinates are the respective elements of x, y, and z. These vectors are usually defined parametrically. For example,

```
t=.01:.01:20*pi; x=cos(t); y=sin(t); z=t.^3; plot3(x,y,z)
```

will produce a helix which is compressed near the x-y plane.
Just as for planar plots, a title and axis labels (including **zlabel**) can be added. The features of **axis** command described there also hold for 3-D plots; setting the axis scaling to prescribed limits will, of course, now require a 6-vector.