

Interdisciplinary Center
for Neural Computation
Hebrew University, Jerusalem
Israel

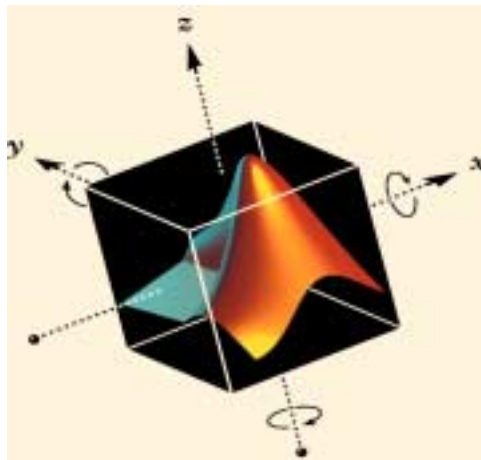
MATLAB tutorial

Assembled, edited and written by:

Oren Shriki, orens@fiz.huji.ac.il

and

Oren Farber, orenf@cc.huji.ac.il



MATLAB is a general-purpose mathematical software that is user friendly and very useful for data analysis, simulations, presentation of results, and more. The software runs on both UNIX and WINDOWS.

MATLAB is an interpreter, which means that it performs every command line immediately after it is entered. A script file for MATLAB is called an m-file and it must have the extension 'm' (i.e. something like xxx.m). To run an m-file you just enter the name of the file without the extension at the MATLAB prompt.

Getting started

If you are a first-time user, you should invoke MATLAB now and follow along. The main tool of the MATLAB is the matrix (The name MATLAB stands for matrix laboratory). A matrix of one element is called a scalar and we can create one by entering:

```
>> s = 3
```

A matrix of one row or one column is called a vector. We can create a row vector by entering:

```
>> v = [8 3 5]
```

To create a column vector we will use the ';' symbol to separate between the elements

```
>> u = [4.2; 5; 7]
```

We can create a 3 by 3 matrix simply by entering

```
>> A = [3 4 0; 1 4 6; 9 7 2]
```

Note the fact that you don't have to define a variable type - you just enter it.

We can now calculate quantities like the transpose of A (which is defined by $A^T(i,j)=A(j,i)$).

```
>> A'
```

or the transpose of v which will be a column vector:

```
>> v'
```

For example we can create another 3 by 3 matrix by entering

```
>> B=[v; u'; 1 0 4]
```

If we have several matrices, we can add them

```
>> A+B
```

or multiply them

```
>> A*B
```

but we have to take care that the dimensions are appropriate. The multiplication operation we just used obeys the rules of ordinary matrix multiplication. There is another kind of multiplication, which is multiplying each element in a matrix by the corresponding element in another matrix which has to be the same size (this is called array multiplication). To do this you enter

```
>> A.*B
```

This dot rule is general, so if you enter

```
>> A.^2
```

it will raise all the elements of A to the power of 2 (which is of course different from A^2).

MATLAB has some commands for the construction of special matrices.

```
>> ones(3,4)
```

will create a matrix of 1's with 3 rows and 4 columns. And you can figure what

```
>> zeros(5,7)
```

does.

To create an identity matrix of order 5 you type

```
>> eye(5)
```

If you type

```
>> x=1:10
```

This will give a vector with all the integer numbers from 1 to 10, and

```
>> y=0:0.1:8
```

will give a vector of all the numbers between 0 and 8 with a 0.1 step.

While working with MATLAB, you can always list the variables in the workspace by entering 'who' or 'whos' ('whos' shows also the size of the variables).

If you would like to learn about the syntax and the different uses of a command just type 'help' and the name of the command. For example try this:

```
>> help rand
```

If you want to search for commands related to some topic, say complex numbers, you enter

```
>> lookfor complex
```

You can also use

```
>> helpdesk
```

for a comprehensive hypertext documentation.

Subscripts

After we created a matrix we might want to extract some elements from it. To extract the i 'th element of a vector you type

```
>> v(i)
```

Typing the command 'A(i, j)' will extract the element of A that sits in the i 'th row in the j 'th column. For example:

```
>> A(2,3)
```

will extract the element in the 2nd row in the 3rd column.
Typing the command:

```
>> A(:,1:2)
```

will extract the first 2 columns of A. and

```
>> A(end,:)
```

shows the last row of A.

Suppose we want to know the sum of the 3rd row. We will use the colon (:) and the function sum like this:

```
>> sum(A(3,:))
```

for the sum of the third column we would type:

```
>> sum(A(:,3))
```

The command

```
>> B(1:2,2:3) = 0
```

zeros out a portion of B, namely the 2nd and 3rd columns of the first two rows.

Logical Operations

Type the following command:

```
>> A>3
```

What you got is a new matrix, with ones and zeros. Every place with one means that the corresponding place in A fulfills the condition (bigger than 3). Zeros mean that the opposite is true, that is, the number is smaller or equal to 3. We now go further by typing :

```
>> (A>3) .* A
```

can you explain the result ?

hint: notice we used the element by element multiplier (.*)

Until this stage, every result was displayed on the screen. If you don't want to see the result, just add a ';' at the end of the command. You can use the ';' also to separate different commands on the same line like this

```
>> x=3; y=[1 x ones(1,5); 5*rand(1,7); 1:7];
```

Try it and then type

```
>> y
```

to see what we got.

Logical subscripting

The logical vectors created from logical and relational operations can be used to reference subarrays. Suppose X is an ordinary matrix and L is a matrix of the same size that is the result of some logical operation. Then X(L) specifies the elements of X where the elements of L are nonzero.

This kind of subscripting can be done in one step by specifying the logical operation as the subscripting expression. Suppose you have the following set of data.

```
>> x = [2.1 1.7 1.6 1.5 NaN 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8]
```

The NaN is a marker for a missing observation, such as a failure to respond to an item on a questionnaire. To remove the missing data with logical indexing, use `finite(x)`, which is true for all finite numerical values and false for NaN and Inf.

```
>> x = x(finite(x))
```

gives:

```
x =
```

```
2.1 1.7 1.6 1.5 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8
```

Now there is one observation, 5.1, which seems to be very different from the others. It is an *outlier*. The following statement removes outliers, in this case those elements more than three standard deviations from the mean.

```
>> x = x(abs(x-mean(x)) <= 3*std(x))
```

gives:

```
x =  
2.1 1.7 1.6 1.5 1.9 1.8 1.5 1.8 1.4 2.2 1.6 1.8
```

The *find* function

The find function determines the indices of array elements that meet a given logical condition. Here is the help text on this function:

```
FIND Find indices of nonzero elements.  
I = FIND(X) returns the indices of the vector X that are  
non-zero. For example, I = FIND(A>100), returns the indices  
of A where A is greater than 100. See RELOP.
```

```
[I,J] = FIND(X) returns the row and column indices of  
the nonzero entries in the matrix X. This is often used  
with sparse matrices.
```

```
[I,J,V] = FIND(X) also returns a vector containing the  
nonzero entries in X. Note that find(X) and find(X~=0)  
will produce the same I and J, but the latter will produce  
a V with all 1's.
```

```
See also SPARSE, IND2SUB.
```

The *diary* function

To log everything you type in the workspace (and the results), use the 'diary' command. To start, enter something like:

```
>> diary my_log.txt
```

and to stop saving, enter:

```
>> diary off
```

Type 'help diary' to see exactly what it does.

The *type* function

To watch a script file inside MATLAB's workspace you enter for example

```
>> type randperm
```

To save the script in a file, which you can later edit, enter

```
>> diary randperm1.m; type randperm; diary off
```

Multivariate Data

MATLAB uses column-oriented analysis for multivariate statistical data. Each column in a data set represents a variable and each row an observation. The (i,j)th element is the ith observation of the jth variable.

As an example, consider a data set with three variables:

Heart rate •

Weight •

Hours of exercise per week •

For five observations, the resulting array might look like:

```
D =  
    72    134    3.2  
    81    201    3.5  
    69    156    7.1  
    82    148    2.4  
    75    170    1.2
```

The first row contains the heart rate, weight, and exercise hours for patient 1, the second row contains the data for patient 2, and so on. Now you can apply many of MATLAB's data analysis functions to this data set. For example, to obtain the mean and standard deviation of each column:

```
>> mu = mean(D), sigma = std(D)
```

```
mu =  
    75.8    161.8    3.48
```

```
>> sigma =  
    5.6303    25.499    2.2107
```

For the maximal element in each column type

```
>> max(D)
```

For the maximal element of D you have to type

```
>> max(max(D))
```

or

```
>> max(D(:))
```

For a list of the data analysis functions available in MATLAB, type

```
>> help datafun
```

Here we list few functions, which are very useful for data analysis:

- max - maximum value
- min - minimum value
- mean - mean value
- median - median value
- std - standard deviation
- sort – sorting
- sum -sum of the elements
- prod - product of elements
- cumsum - cumulative sum of elements
- cumprod - cumulative product of elements
- diff - approximate derivatives
- corrcoef - correlation coefficients
- cov - covariance matrix

Typing help on each of these functions will give you the full description of them. Note that, although it is very easy to write most of these functions yourself, it is nice to already have them and it makes the scripts more compact.

To learn about the interpolation capabilities of MATLAB, try the following demo

```
>> census
```

If you have access to the Statistics Toolbox, type

```
>> help stats
```

Graphics and Visualization

MATLAB is indeed a great tool for visualization of data of all sorts. The first command you should learn how to use is the 'plot'. If you want to plot vector y versus vector x just type

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

MATLAB will produce a figure window and will plot the graph.

For example, type the following:

```
>> x = 0:0.01:pi*4
>> plot(x, sin(x))
```

To learn more about the 'plot' command, try the help on it. To present several plots on the same figure, use the 'subplot' command. To create another figure window, use the 'figure' command.

Another useful command is 'hist'. With 'hist' you get nice histograms very quickly. Try this:

```
>> x = randn(1000,1) ;
```

this will create a vector with 1000 elements drawn from a normal (Gaussian) distribution and store it in x.

The command

```
>> hist(x,30)
```

will produce a histogram of these random numbers with 30 bins.

The following commands will help you control the way your graphs look: axis, xlabel, ylabel, title, hold, grid, get, set, gca,(gcf

Here is a list of some specialized 2-D functions:

- loglog - Log-log scale plot.
- semilogx - Semi-log scale plot.
- semilogy - Semi-log scale plot.
- polar - Polar coordinate plot.
- bar - creates a bar graph.
- errorbar - creates a plot with error bars.

The best way to learn about MATLAB visualization capabilities is to see the demo and then learn about the commands that are most useful for you. Note that the demos are written in MATLAB so you can "type" the files and just copy the commands you need. For instance,

```
>> lorenz
```

is a demo program that plots an orbit around the Lorenz chaotic attractor. To see the script enter

```
>> type lorenz
```

A nice 3D demo you can use to show off if a friend of yours walks by is

```
>> spharm2
```

Type 'help graph2d' and 'help graph3d' for a list of all relevant commands.

Writing scripts

A script file is an external file that contains a sequence of MATLAB statements. In fact, we now have everything we need to create MATLAB scripts, but there are some important commands that we did not talk about yet. Here is the help text on these commands:

IF IF statement condition.

The general form of the IF statement is

```
IF expression
    statements
ELSEIF expression
    statements
ELSE
    statements
END
```

The statements are executed if the real part of the expression has all non-zero elements. The ELSE and ELSEIF parts are optional.

Zero or more ELSEIF parts can be used as well as nested IF's. The expression is usually of the form `expr rop expr` where `rop` is `==`, `<`, `>`, `<=`, `>=`, or `~=`.

Example

```
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```

See also RELOP, ELSE, ELSEIF, END, FOR, WHILE, SWITCH.

FOR Repeat statements a specific number of times.

The general form of a FOR statement is:

```
FOR variable = expr, statement, ..., statement END
```

The columns of the expression are stored one at a time in the variable and then the following statements, up to the END, are executed. The expression is often of the form `X:Y`, in which case its columns are simply scalars. Some examples (assume `N` has already been assigned a value).

```
FOR I = 1:N,
    FOR J = 1:N,
        A(I,J) = 1/(I+J-1);
```

```
END
END
```

```
FOR S = 1.0: -0.1: 0.0, END steps S with increments of -0.1
FOR E = EYE(N), ... END sets E to the unit N-vectors.
```

Long loops are more memory efficient when the colon expression appears in the FOR statement since the index vector is never created.

The BREAK statement can be used to terminate the loop prematurely.

See also IF, WHILE, SWITCH, BREAK, END.

WHILE Repeat statements an indefinite number of times.
The general form of a WHILE statement is:

```
WHILE expression
    statements
END
```

The statements are executed while the real part of the expression has all non-zero elements. The expression is usually the result of `expr rop expr` where `rop` is `==`, `<`, `>`, `<=`, `>=`, or `~=`.

The BREAK statement can be used to terminate the loop prematurely.

For example (assuming A already defined):

```
E = 0*A; F = E + eye(size(E)); N = 1;
while norm(E+F-E,1) > 0,
    E = E + F;
    F = A*F/N;
    N = N + 1;
end
```

See also FOR, IF, SWITCH, BREAK, END.

In addition to writing regular scripts, we can write functions that accept arguments. Here is the help text on functions:

FUNCTION Add new function.

New functions may be added to MATLAB's vocabulary if they are expressed in terms of other existing functions. The

commands and functions that comprise the new function must be put in a file whose name defines the name of the new function, with a filename extension of '.m'. At the top of the file must be a line that contains the syntax definition for the new function. For example, the existence of a file on disk called STAT.M with:

```
function [mean,stdev] = stat(x)
%STAT Interesting statistics.
n = length(x);
mean = sum(x) / n;
stdev = sqrt(sum((x - mean).^2)/n);
```

defines a new function called STAT that calculates the mean and standard deviation of a vector. The variables within the body of the function are all local variables. See SCRIPT for procedures that work globally on the workspace.

A subfunction that is visible to the other functions in the same file is created by defining a new function with the FUNCTION keyword after the body of the preceding function or subfunction.

For example, avg is a subfunction within the file STAT.M:

```
function [mean,stdev] = stat(x)
%STAT Interesting statistics.
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);

%-----
function mean = avg(x,n)
%MEAN subfunction
mean = sum(x)/n;
```

Subfunctions are not visible outside the file where they are defined.

Normally functions return when the end of the function is reached.

A RETURN statement can be used to force an early return.

See also SCRIPT, RETURN, VARARGIN, VARARGOUT, NARGIN, NARGOUT, INPUTNAME, MFILENAME.

As we mentioned in the beginning, to run an m-file you just enter its name without the extension. In addition to this, you have to be in the directory of this file, and this can be done using the UNIX command 'cd' inside MATLAB or before you invoke MATLAB.

Another possibility is to use the 'addpath' command to add a directory to MATLAB's search path.

Function functions

(this means function that takes other functions as their arguments).

Suppose you want to investigate the following formula:

$$f(x) = x^2 + 42\sin(x)$$

1. Type:

```
>> fplot('x^2+42*sin(x)', [-3*pi 3*pi])
```

As you guessed, this plots a function whose definition is specified by the first argument (a string). The second argument is the x range of interest. Type 'help fplot' to learn more.

2. Try to plot the function over larger and smaller ranges. Try to use the zoom tool to inspect the function in details.
3. In order to use the function functions of MATLAB we first need to define our function like this:

```
>> f = inline('x^2 + 42*sin(x)')
```

You will get this answer:

```
f =
```

```
Inline function:
```

```
f(x) = x^2+42*sin(x)
```

Which means that MATLAB now recognizes our function.

4. Add grids to the plot by : 'grid on' (you can turn it off by 'grid off')
5. If you want to know for what x between -2 and -4 the function zeros. Type:

```
>> t = fzero (f, [-2 -4])
```

6. If you want to test this, simply type: f(t).

Is this good enough ?

7. Use fzero to find other intersection with zero.

8. To find the minimum points of f type:

```
>> m = fmin(f, -8, 0)
```

This means that m will be assigned with the x that gives a local minimum such that $-8 < m < 0$.

9. Now we might like to plot the derivative of f . We shall use some symbolic math for that. Type:

```
>> s = diff('x^2 + 42*sin(x)')
```

The output of s is a mathematical expression. If we want to plot it, we need to use the string equivalent to s . This will do it

```
>> ss = char(s)
```

and now:

```
>>
```

figure

(this will create another figure)

```
>> fplot(ss, [-3*pi 3*pi])
```

10. If we wanted to plot the integral f we would do

```
>> s = int('x^2 + 42*sin(x)')
```

and repeat the other commands as before.

Final Tips:

1. Try to avoid using loops when you can use matrix operations.
2. Learn many commands. In many cases, what you are looking for already exists.
3. Use the web site: <http://www.mathworks.com>

MATLAB tutorials on the web:

Introduction:

- <http://eewww.eng.ohio-state.edu/~juodvalk/ee281/matlab/matlab.html> •
- <http://www.engin.umich.edu/group/ctm/basic/basic.html> •

Neural Networks Toolbox:

- <http://carol.wins.uva.nl/~portegie/matlab/nnt/> •
- <http://matlab.kimhua.co.kr/product/neuralnet/> •

MATLAB in Neuroscience Courses:

- Foundations of Vision: Perception and Computation •
<http://www.cvs.rochester.edu/~knill/psyc311.html>
- Cochlear Implants: A Tutorial from Sound to Stimulus •
<http://www-personal.engin.umich.edu/~ueda/cochlear/>

Ex1: Matrix manipulations

1. Create a matrix with 10 rows and 4 columns. The numbers in the matrix should be random integers within 10 to 100.
2. Find the maximal values in each column.
3. Find the maximal values in each row.
4. Find the maximal entry in the matrix
5. Find the index of this entry (row and column)
6. Create a matrix with the same dimensions in which all entries equal the maximal value that you found in 4.
7. Calculate the difference between the matrix from 6 and the original matrix.
8. Find all the entries that are less than 40 and more than 20. Can you do it in one line?

Ex2: Basic statistics

1. Create a vector x with 100 random entries uniformly distributed between 0 to 1
2. Create a vector y with 100 random entries uniformly distributed between 0.1 to 1.1
3. Create a figure with two subplots. The upper one should show an histogram of x and the lower an histogram of y. Make sure that both histograms show the same range.
4. Calculate the mean, range and standard deviation for each vector.
5. Find how many members of x are greater than their corresponding members in y.
6. Find the greatest difference between x and y.