

MATLAB NOTES

These notes are intended as a quick introduction to MATLAB. Mathworks, the company behind MATLAB, offers a *Getting Started* document. This document can be opened by clicking on the question mark symbol on the MATLAB toolbar. A pdf version is available on the Mathworks website at

http://www.mathworks.com/access/helpdesk/help/pdf_doc/MATLAB/getstart.pdf

Page numbers used in these notes refer to the *Getting Started* document.

At RU you can get into MATLAB by going through the sequence
Start > Programs > Radford University Course Software > Math-Statistics >
The Mathworks > MATLAB

- MATLAB is an interactive expression evaluator.
- Algebraic operations work as usual with $+$ $-$ $*$ $/$ $^$ for single variables. But you need to be careful. MATLAB assumes that all variables are matrices, and you must make sure that the matrices have the correct dimension. In addition, the multiplication symbol is mandatory, i.e. you need to write $3*x$. The expression $3x$ is meaningless.
- Assign values to variables using $=$.
Example $a = 4.5$, compared to maple's $a := 4.5$;
- Interrupt a computation with Control + C
- You can write a transcript of your MATLAB session to a text file with the `diary` command. This file can be read, edited and printed, but it cannot be executed, i.e. you cannot run a MATLAB program with a diary file.
This is different from your experience with maple files, which can be edited and executed. But then, these are not simple text files either.
- Important symbols

>>	MATLAB prompt
%	Comments
;	Suppresses output, new lines
:	Many uses, see below.

- The `format` command controls the display of MATLAB output. Important options:


```
>> format compact % suppresses extra line feeds
>> format long
>> format long e % scientific notation, exponents
>> format long g % mix of fixed and exponential
>> format long eng % engineering notation
```

 Any of the longs can be replaced by short.

Help and Documentation:

- The `help` command explains how a MATLAB command works, its syntax and limitations, and so on. Example:


```
>> help inv
```
- With `lookfor` you can search for MATLAB functions containing certain keywords. Example:


```
>> lookfor Fourier
```

Built-in Variables are `pi`, `i`, `j`, `eps`, `realmin`, `realmax`, `Inf` among others (see p. 2-13).

Bookkeeping Commands:

- `clear` clears the memory
- `who` lists all defined variables
- `whos` same, with more information
- `what` lists the m files in the current directory
- `save` saves all variables to a file
- `load` loads saved variables into memory

Entering Matrices

A **matrix** is a rectangular array of numbers. More specifically, an m by n matrix is an array of m rows and n columns. Enter matrices using brackets, begin new lines using the ENTER key, or a semicolon. A **vector** is a 1 by n matrix (row vector) or an m by 1 matrix (column vector).

This process is explained on p. 2-4. Built-in functions can be used to create special matrices: `diag`, `eye`, `ones`, `zeros`, `rand`, `linspace`, `logspace`, `magic`, and many more. You can use small matrices to build larger ones (see p. 2-16, Concatenation).

The section on subscripts (p. 2-7) shows how you can extract or reset a specific value in a matrix. The commands `size` and `length` let retrieve information about a matrix or a vector.

Operations with matrices:

- `+`, `-` and scalar multiplication work the usual way (matrices have to be of the same size for addition/subtraction).
- `*`, `/` and `^` are executed in the sense of matrix multiplication, etc.
- For **element-by-element operations** use `.*`, `./` or `.^`. See p. 2-24.
- Matrix block operations are built in. For example, if `A` is a given 4 by 7 matrix, `[A, eye(4); zeros(5,7), rand(5,4)]` generates a 9 by 11 matrix.
- A host of matrix functions is available. Some functions perform term-by-term operations, for example `cos(A)` evaluates the cosine of each entry in `A`. Other functions are strictly matrix operations using linear algebra techniques, for example `B = lu(A)`.

The Colon Operator

The colon `:` has a wide variety of uses. Try the following sequence in MATLAB.

```
>> 3:6
>> A = magic(4)
>> A(:,3)
>> A(2,:)
>> A(:)
>> A(:, [3 2 4 1])
>> A(:, [2 4])
```

Reshaping of a matrix can be done with `'` (transpose), `rot90`, `flipud`, `fliprl` and `reshape`.

Graphing

A basic graph can be generated with the `plot` command. If `x` and `y` are vectors of the same length, the command `plot(x, y)` will plot the points (x_k, y_k) in the plane, and connect with line segments.

Example: Graph the function $f(x) = x^2 e^{-x}$ for $-1 < x < 4$.

```
>> x = linspace(-1,4,100); % create x-values
>> y = x.^2.*exp(-x); % define the y-values
>> plot(x,y)
```

The looks of such a graph can be improved with commands such as `axis`, `grid`, `legend`, `text`, `title`, `xlabel` or `ylabel`. These commands are explained in the Getting Started manual. You can also modify the looks of your graph interactively

from the figure window. Activate the plot tools (last icon) and experiment with the features.

If you want to display several functions in the same figure, just list the data sets separately. For example

```
>> x=linspace(0,2*pi,200);
>> y = sin(x);
>> z = cos(x);
>> plot(x,y,x,z,x,y./z)
>> axis([0 2*pi -3 3]), grid
>> title('sine cosine and tangent')
```

will display a graph with sine, cosine and the tangent in a common figure.

You can display several graphs on a single page (plot) using the subplot command. This might save you lots of paper. Basically you define a matrix consisting of individual graphs with the subplot command. Continuing the example above, we could create four plots (one each for sine, cosine and tangent plus a common graph) in this way

```
>> subplot(2,2,1), plot(x,y,x,z,x,y./z), axis([0 2*pi, -3
3]), grid
>> subplot(2,2,2), plot(x,y), axis([0 2*pi, -3 3]), grid
>> subplot(2,2,3), plot(x,z), axis([0 2*pi, -3 3]), grid
>> subplot(2,2,4), plot(x,y./z), axis([0 2*pi, -3 3]), grid
```

Three dimensional graphs visually display a matrix. The row and the column indices represent the xy-coordinates in the plane, and the actual matrix entry gives the z-coordinate. For example, try

```
>> surf(eye(20,25)) % or
>> mesh(eye(20,25))
```

If you want to plot a function $f(x,y)$ of two variables you need to first create a matrix representing the x and y values, as well as a matrix which represents the function. This step can get a little tricky. Let's do this by for an example.

Example: Graph the function $f(x,y) = 1 - xy$ on the rectangle $[-1,1] \times [-1,2]$.

```
>> x = linspace(-1,1,20); % x values
>> y = linspace(-1,2,30); % y -values
>> [X,Y] = meshgrid(x,y); % matrices with coordinate
information
>> Z = 1-X.*Y; % define the function
```

```

>> % Now we display the function in several ways
>> mesh(X,Y,Z);
>> surf(X,Y,Z);
>> contour(X,Y,Z);
>> imagesc(x,y,Z);

```

Depending on your artistic taste, you might prefer `surf` over `mesh`, or vice versa. `contour` and `imagesc` may also prove useful for functions of two variables. Again you can also experiment with the options in the figure window to improve the looks of your graph.

Programming

Programming in MATLAB is done with so-called **m-files**. In order to run such a program, the file has to be stored in the **current directory**, or in a directory which is in the **search path**.

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

Script files contain a sequence commands as you would enter them in an interactive MATLAB session. Within the MATLAB session you just type the name of the m-file, and the commands will be executed in the order given by the file.

Example: Write an m-file that will graph the function $f(x) = \sin^2 x$ on the interval $[0, 2\pi]$. First we type the necessary commands into a file called `sinedemo.m`. This can easily be done with the MATLAB editor. Our file could look like this:

```

% sine squared function
clf
x = linspace(0,2*pi,300);
y = sin(x);
y = y.^2;
plot(x,y), axis([0 2*pi -0.2 1.2]), grid
title('f(x) = sin^2x')

```

When in MATLAB, just type `sinedemo`, and the plot should appear.

Be careful using scripts, since they may delete or overwrite existing data!

Function m-files are the main programming tool. Like any function in math, they are used for input-output relations. The first line in a function m-file must be of the form

```
function [output list] = functionname(input list)
```

Here `functionname` should be identical with the name of the m-file.

Input and output variables should be separated by commas. All variables inside a function m-file are **local variables**; they will not conflict with names in the calling program or the main MATLAB session. The variables from the input list have assigned values already. The value that was last assigned to an output variable will be communicated to the calling environment.

Example: Assign to each angle θ the matrix which represents a counterclockwise rotation by θ degrees in the xy-plane. Note we are working in degrees and not with the radian measure. The content of the file `rotationmatrix.m` could look like this:

```
function A = rotationmatrix(t)
% This function returns a 2x2 matrix
% corresponding to a rotation by t degrees
% Syntax: A = rotationmatrix(theta)

t = t*pi/180; % convert degrees to radians
A = [cos(t) -sin(t) ; sin(t) cos(t)];
```

If this file is in the current directory, we might do the following

```
>> B60 = rotationmatrix(60)

B60 =
    0.5000    -0.8660
    0.8660     0.5000

>> help rotationmatrix
This function returns a 2x2 matrix
corresponding to a rotation by t degrees
Syntax: A = rotationmatrix(theta)
```

It is possible to call functions from within functions. The function which is being called must be on the current search path, or it can be part of the m-file. (One function m-file may contain many functions).

Input and Output

This applies to script files and function files.

- The `input` command will prompt the user for input. This can be very annoying when you run a program frequently, and I would generally advise against using input prompts.

- The `disp` command lets you display comments (as strings) or values of variables. This is very useful when intermediate results are desired, for example when you want to observe the progress of a routine, or you try to debug an m-file.
- `fprintf` can be used to format the output. Example,

```
>> x = sqrt(7);
>> fprintf(' the square root of seven is %12.8f\n',x)
the square root of seven is 2.64575131
```

Use `\n` to start a new line. The statement `%12.8f` indicates that you reserve 12 spaces for the output, with 8 digits after the decimal point. Use

- `%f` for floating points
- `%e` for scientific notation
- `%s` for strings

Flow Control

First of all, keep in mind that relational operators return a 1 if the relation is true, else they return a 0. Example:

```
>> 3^2 == 9
ans =
    1
>> 3^2 == 10
ans =
    0
```

- **if statements:**
the `if` has to be followed by a logical expression. If its value is non-zero, the next statements will be executed, until an `elseif`, an `else` or `end` is encountered. If the value of the logical expression is zero, the program will continue after the `elseif`, `else` or the `end` statement.
 - The simplest form is a single `if`. For example
`>> if 4, x=8, end;`
 - The general form of the `if` statement is

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

- **for loops**

In MATLAB's own words:

```
FOR      Repeat statements a specific number of times.
         The general form of a FOR statement is:
         FOR variable = expr, statement, ..., statement END
```

Example:

```
for k = 1:10
    x = ln(k)
    y = sqrt(x)
end
```

- **while loops.**

Again, let's look at MATLAB's definition:

```
WHILE   Repeat statements an indefinite number of
times.
```

The general form of a WHILE statement is:

```
WHILE expression
    statements
END
```

In a while loop the number of repeats is not specified and it could be infinite if the programmer messes up.

Example:

```
factorial = 1, k=2
while factorial < 1.0e10
    factorial = k*factorial
    k = k+1;
end
```

- **break and return**

A `break` statement lets you jump out of the current loop (`for`, `while`), while with the `return` statement you stop the execution of the function routine and go back to the calling environment.

- **error**

This command works like `return`, in that it stops the execution of the m-file. It also lets you display an error message. Example:

```
if x<0
    error('please do not use negative numbers today')
end
```

Other useful programming tools:

- **nargin** and **nargout** tests the number of input and output arguments. This can be used to define defaults. Example: Make 0.001 the default tolerance

```
function y = sowieso(x,tol)
if nargin < 2, tol = 0.001, end
```

- **feval** and calling functions within an m-file. This is best illustrated by an example

```
function y = sowieso(fun,x,tol)
.....
y1 = feval(fun,x); %this will evaluate the function
.....
```

- **pause** This interrupts the execution of the program temporarily (until a key is pressed, or for a given amount of time).
- **Keyboard** : Gives control to the command window until return is entered.

Vectorization and Preallocation

Always preallocate space. Don't just create matrices on the fly or append to existing variables as needed!

Many loops can be avoided if you utilize MATLAB's vectorization environment.

Example: The statements

```
>> x = zeros(1,10); % Preallocate memory
>> for k = 1:10, x(k) = k^2 ; end
and
>> x=1:10; x =x.^2;
```

both generate a vector with the square numbers from 1 to 100; the second statement uses vectorization and is more efficient.

Programming Advice

A typical m-file structure should have these parts:

- Prologue: Contains instructions and information on how to use the file. This is linked to the MATLAB help feature.
- Process input data: Check defaults, verify that inputs are admissible.
- Main Program

- Process results for output.

Suggestions:

- Use blank rows to make code readable.
- Indent to identify program blocks
- Use meaningful variable names.
- Comments: Explain the steps/loops. This should contain additional information, should make code more readable. Develop comments as you write the code. Comments should help in discovery of mistakes.

In the development of your code you should use **stepwise refinement**. While writing the code you should have print statements for intermediate results. **Always test** your code in straightforward examples. In addition

- Do not assume that input data are correct.
- Guard against occurrence of an impossible condition.
- Catch unlikely events with an `else` statement when using `if`, or `elseif` commands.
- Provide diagnostic error messages.