Evaluating measurements with MATLAB : a doctoral student's course at Building Materials, Lund University, Sweden

Wadsö, Lars

2006

LUND INSTITUTE OF TECHNOLOGY
LUND UNIVERSITY

Division of Building Materials

Evaluating measurements with MATLAB

A doctoral student's course at Building Materials, Lund University, Sweden

Lars Wadsö

```
x=0:10;
y=x+rand(1,11);
plot(x,y,'*b')
x2=0:0.1:10;
y2=spline(x,y,x2);
hold on
plot(x2,y2,'g')
y3=interp1(x,y,x2,'linear');
plot(x2,y3,'m')
y4=interp1(x,y,x2,'nearest');
plot(x2,y4,'c')
hold off
```

Report TVBM-7190 Lund 2006

ISRN LUTVDG/TVBM--08/7190--SE(1-63)
ISSN 0348-7911 TVBM

Lund Institute of Technology
Division of Building Materials
Box 118
SE-221 00 Lund, Sweden

Telephone: 46-46-2227415
Telefax: 46-46-2224427
www.byggnadsmaterial.lth.se

Evaluating measurements with MATLAB

A doctoral student's course at Building Materials, Lund University, Sweden

Autumn 2006

Lars Wadsö, Building Materials, Lund University, Sweden
lars.wadso@byggtek.lth.se

The course consists of 14 lectures with exercises.

MATLAB was developed as a convenient computer tool to work with matrices (MATrix LABoratory), but it has found a very wide use in the scientific and engineering community. In this course we will concentrate on what is useful when we have measured data that we want to evaluate. We will not go into any detail on MATLABs matrix functions.

This course was written while I had MATLAB 7.0.1 installed. Although there are significant changes to MATLAB almost every year the basic functionality is the same. Usually no changes are needed to run a program with an earlier or later version than it was written in. However, many new commands are added each year and some nomenclature rules have been changed slightly, so you may in some cases need to make some changes to your programs.

This compendium is a course material; it is not a MATLAB manual. However, it gives in a rather concentrated form the most common MATLAB functions with examples on how to use them, so it can be used as a companion to the built-in MATLAB help functions and manuals.

The following nomenclature is used here.

MATLAB statements are written in a Courier format:

```
hp=plot(t,P,'r*');
```

Outputs from MATLAB statements (in the workspace) are also written in Courier, but in a more compact way than MATLAB does it:

```
sin(2*pi)
ans = -2.4493e-016
```

in the MATLAB workspace it will normally look like this:

```
>> sin(2*pi)

ans =

 -2.4493e-016
```

Table of contents

0. Examples of use of MATLAB
1. Introduction to MATLAB
2. More MATLAB commands and catalog structure
3. Curve fitting and interpolation
4. Logical operators, program control and string handling.
5. Functions, script files and the eval-function
6. Advanced plotting and file I/O
7. String handling and data structures
8. Efficient MATLAB programs, debugging and the MATLAB Notebook
9. General curve fit, optimization and linear algebra
10. Simple forward differences
11. Noise reduction (filtering)
12. Graphical User Interfaces (GUIs)
13. Monto-Carlo Simulations
14. Example of clever ways of using MATLAB

0. Examples of use of MATLAB

I will here give a number of examples of how MATLAB can be used.

cd c:\measure2\matlabcourse\ht05\0

1. Plot variable and make plot look nice.

```
t=1:100;
q=2+t/20+rand(size(t));
plot(t,q)

hp=plot(t,q,'.')
hx=xlabel('Time / s');
hy=ylabel('q-var');
set([gca hx hy],'FontSize',14)
set(hp,'Color',[0.2 0.4 1],'MarkerSize',20)
```

2. We can make this into a program/macro called an m-file (copy all into test.m).

3. It is convenient to make m-files that evaluate measurement files.

I have a file biofuel30nov04.txt in the present directory. Let is look at that:

load biofuel30nov04.txt (using MATLABs load function)

This file contains time, voltage output from an instrument and four temperatures (6 columns). I would like to look at the voltage output (column 2) as a function of time (column 1):

```
t=biofuel30nov04(:,1);
P= biofuel30nov04(:,2);
t=t/60/24; %t in days
P=P*0.144; % apply calibration coeff.
plot(t,P)
```

Lets put this into an m-file instead. Then we can run it again if we like, we can have it as a memory of what calibration coefficient we used etc.

4. ISOCAL6 toolbox for evaluating calorimetric results

I have written a toolbox with convenient functions for evaluating calorimetric data (you can do the same for the type of measurements you are doing):

newmea	load new data
plotmea	plot data
tian	Tian correction
blcorr	baseline correction
powers	find thermal powers

heats	find integrals (heats)
etc.	

Let's test it on the biofuel-file:

```
newmea
plotmea(1)
plotmea(2:5)
tian(1,0.144)
blcorr(1)
P=powers
m=[0.2 0.2 0.1 0.1 0.1 0.1];
P./m
```

5. Automatic evaluation

One very interesting (and advanced) possibility with MATLAB is to automate the evaluation of large amounts of data. I have one example here:

```
cd c:\measure2\kvalster
```

HDMeval is a master evaluation program that A. Finds all measurement files. B. Finds which samples that have been measured (in the files). C. Evaluates all measurements and plots them.

```
HDMeval
```

The nice thing with this is that when I add new data files I only need to type "HDMeval" to get updated curves.

Another example is a Round Robin test that I coordinated. There I have several hundreds of cement hydration measurements, each containing at least 10000 data. They are automatically evaluated by a system of several programs guided by a master program that starts by checking what files there are to evaluate.

6. Graphical User Interfaces (GUI)

The ISOCAL6 toolbox is made with MATLAB GUI with Windows appearance. Here is one more example that I have made for fun: a tool to manipulate photos:

```
imagex
```

7. Using a MATLAB toolbox: the Image Processing Toolbox

I have used functions in the Image Processing Toolbox to quantify how fast mould spread on wood samples. This was done by comparing the intensity of a part of the sample with the intensity of a reference surface. I wrote a program that automatically does this evaluation for a series of photos taken during an experiment. The final result is a plot of how much darker the wood becomes during the experiment. Almost ready to be used in a paper!

1. Introduction to MATLAB

The MATLAB window can contain a number of different parts: the command window (the MATLAB work space), the launch window, the command history etc. The layout can be changed with Desktop - Desktop Layout -.

This course is about MATLAB as a command based program, both because this is the main way that MATLAB works and because this is the way you can make MATLAB do much more than is possible in any, e.g., spreadsheet program. MATLAB is in quite similar to programming languages like BASIC, FORTRAN and PASCAL, but it also has an interactive workspace, which makes it very powerful to work with.

Some notes on how to write MATLAB commands:

1. MATLAB is by default case sensitive, i.e. 'a' is not the same as 'A'.
2. Variables can be called almost anything, e.g. 'RhU_fd' or 'LongVariableName', but do not use special symbols like +-/*^.
3. Spaces do not count (except in variable or function names). "5+xvar" is the same as "5 + xvar".
4. If you want to have many commands in one line you can separate them with ";" (semicolon), e.g. "a=sin(x);b=cos(y)".
5. If you have very long lines you can wrap them with "...", e.g. "xvar2=23+y+ ..." on one line and the rest on the next line: "tvar-rr*sin(a);".
6. Matrices have two indices: M(row, column). Vectors also have these indices; one of them being 1 (i.e., vectors can be row vectors or column vectors). However, you need not state the 1-index when indexing vectors.
7. When you are working in the workspace you can recall any previously written command by pressing ↑ (and ↓). If you first give one or more letters, only those commands starting with that letter (those letters) will be found.
8. Esc clears the command line.

Here follows some basic MATLAB commands.

		example
`help`	find help on a function	`help sin`
`helpwin`	start MATLAB's help function with a catalogue of all MATLAB functions	`helpwin`
	enter a scalar	`s=8`
	enter a row vector	`r=[0 1 5 8]`

	enter a column vector	`c=[9;8;6;4]` *or* `c=[9`
		`8`
		`6`
		`4]`

	enter a matrix	`M=[1 3 4`
		`4 5 7`
		`3 5 2]`

`;`	no echo (after line)	

`ans`	MATLAB uses this variable if you do not give an argument	

`:`	range, n:m gives numbers from n to m in steps of one, n:k:m gives numbers from n to m in steps of k (k can be negative).
	":" by itself means "the whole range". "r(:)" is the same as "r", "M(:,2)" is the second column of vector M.
		`1:10`
		`100:-10:50`
		
		`M(:,2)`

`'`	transpose (turn 90^o) vector or matrix	`r=r'`
`+ -`	addition and subtraction of scalars, vectors or matrices (must have the same size)	`r+r` `r+c'`

`* /`	multiplication and division of scalars *and* vector/matrix
	multiplication and division of vectors and matrices
		`5*7`
		`M2=c*r;`
		`scalar1=r*c;`

`\`	see help slash	

`^`	raised to (scalars and matrices)	`34^1.4`
		`M^3`

`.*  ./  .^`	element-vise multiplication, division and raised to; can be used on vectors and matrices	`c.*r'` `M.^3`
`ones`	variable with ones	`ovect= ones(2,3);`
`zeros`	variable with zeros	`zvect= zeros(3,3);`
`rand`	variable with random numbers	`rv= rand(100,1)`
`size`	size of a variable	`size(rvect)`
`length`	length (largest dimension) of a variable	`length(rvect)`
`numel`	number of elements in variable	`numel(Qvect)`
`sqrt`	square root*	`sqrt(2)`
`sin etc.`	trigonometric functions (in radians)*	`sin(pi/2)`
`log`	natural logarithm*	`log(2)`
`log10`	10-logarithm*	`log10(Pmea)`
`mod`	remainder after division	`mod(19,4)`
`NaN`	Not a Number (very useful for marking missing data)**	`q=[1 3 NaN 5 7];`
`eps`	the smallest number that MATLAB can handle	
`Inf`	infinity**	
`end`	the last index in a variable	`r(2:end)`
`[ ]`	empty variable (it exists, but it does not have any content)	`a=[]`
`pi`	3.14159...	
`cd`	change directory, e.g. cd c:\measurem\sorp8	
`dir`	contents of current directory	`dir`
`path`	a list of directories in which MATLAB looks for functions (MATLAB-functions, toolbox-functions or your own functions)	
`ctrl^C`	stop execution	
`plot`	plots variable	`plot(X,Y)`

* Most functions work as expected on scalars, vectors *and* matrices.

** These special “numbers” will in most cases give expected results in calculations and in MATLAB-functions such as plot.

With size you can find out if a vector is a column or row vector, e.g.

```
a=[1 2 3 4];
b=[2;3;4;6;7];
size(a)
ans =     1     4
size(b)
ans =     5     1
```

Size gives number of rows and numbers of columns (in that order). Indexing a matrix is made by giving the number of the row and the number of the column (in that order):

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

Memorize this order: Row-Column! Also use size to make one variable the same size as another vector, e.g. q=ones(size(p));

When you work with vectors, note that it is often important to differ between row and column vectors and that it is important that vectors are of equal lengths *and* direction when they are to be used together. The following lines show a rather typical trial-and-error-way of entering MATLAB commands that is the result of *not* thinking:

```
>> x=0:10;
>> y=x+rand(10,1);
??? Error using ==> plus
Matrix dimensions must agree.
                       %  There are two errors here:
                       %  1: x has length 11, but rand(10,1) has length 10
                       %  2: x is a row vector, but rand(10,1) is a column vector

>> y=x+rand(10,1)';
??? Error using ==>plus
Matrix dimensions must agree.
                       % ψ&#ℵ..., I thought that a transpose would solve the problem

>> y=x+rand(1,11);
??? Error using ==> +
Matrix dimensions must agree.
                       % ψ&#ℵℜξψψξℑ..., I thought that correcting the
                       % length of one of the vectors would solve the problem

>> y=x+rand(1,11);
                       % At last correct.
```

This shows that one should always think (at least a little) first.

EXERCISE 1a: Plot the following function:

$$y = \sin(\frac{x}{10})\sin(\frac{x}{5})$$

in the interval from x=-100 to x=100. What is its maximum? (0.77)

EXERCISE 1b: Make two vectors X and Y with 1000 random numbers between 0 and 1 in each. Plot Y as a function of X with a green dot in each data-point (check help plot)!

EXERCISE 1c (optional): How many of the data points are at a distance of less than 1.00 from the origin? How can you use this to calculate an approximate value of pi?

---

2. More MATLAB commands and catalog structure

There are essentially two ways of working with a computer: either you give commands that are executed one by one (interactively) or you write a program (some people call programs macros) that is executed after you have finished writing it. In MATLAB you can do both: you can make both simple and advanced calculations directly in the workspace and you can write a program that can be executed later.

Many people find it easy to work interactively, but there is one important advantage with writing programs: a program can be run many times, performing the same tasks each time. By writing a MATLAB-program that evaluates your measured data you do not need to repeat the evaluation manually each time you have made a measurement. Writing programs in MATLAB is relatively easy.

Note that the use of the colon ":" to indicate ranges is very powerful in MATLAB,for example like in the following examples:

- `2:2:12` gives 2 4 6 8 10 12
- `M(:,2)` gives the second column of matrix M

MATLAB programs are called m-files because they have the extension m. In an m-file you write the same MATLAB commands as you do in the workspace. There are different reasons for writing m-files:

- You may want to add new functions to MATLAB, e.g. a function that converts from Fahrenheit to Celsius, or a new plot function that has two x-axes.
- You may write specialized programs so that they can be used many times, e.g. to evaluate a certain type of measurements.
- You may want to write a program that makes a certain evaluation or plots data in a certain format for a paper for documentation purposes, e.g. you will not save the figure, but only the data and the program that makes the figure.
- You may want to send your evaluation procedures to someone else who has MATLAB.

Note that most MATLAB functions are very flexible and can be used in many different ways. As an example, consider the text-function given below:

- `text(3,5,'end-point')` Here everything is given directly ('end-point' is placed in a plot at coordinates x=3, y=5)
- `text(x(1),5,'y=35')` Here the x-value is given as a variable
- `text(x(n)+2, y(n),['temp=',num2str(T(n)),'{\circ}C']);`
  Here almost everything is given indirectly

Be creative; look in help and experiment.

Here are some new commands:

		example
`linspace`	equally spaced numbers	`linspace(1,5,8)`
`logspace`	logarithmically spaced numbers	`logspace(1,1000,100);`
`min`	lowest value	`min(c)`
`max`	largest value	`max([3 5 6 8])`
`mean`	mean value	`mean(randn(1 1000))`
`std`	standard deviation	`mean(randn(1 1000))`
`sum`	sum	`sum(M(:,1))`
`cumsum`	cumulative sum	`cumsum(1:10)`
`prod`	product	`prod(r)`
`diff`	differences	`diff([1 3 5 7])`
`sort`	sort data in ascending order (default)	`sort(y)`
`clear`	delete selected variables or all variables (“clear” only)	`clear r c` `clear`
`%`	notes in m-file	
`disp`	display line in workspace	`disp('hej')`
`input`	input variables	`x=input('give a number : ');`
`plot`	2D plot command	`plot([1 2 3],[4 5 6])`
`plotyy`	2D plot with two y-axes	
`plot3`	3D plot command	
`fill`	same as plot, but area inside plot is filled	`fill([1 2  2 1 1], ... [1 1 2 2 1],’g’)`
`figure`	chose figure to work with	`figure(3)`
`hold on`	next plot object wlil be added to old plots	`hold on`
`hold off`	next plot command will first clear old plot	`hold off`
`clf`	clear figure	`clf`
`xlabel`	label on x-axix	`xlabel('Time / h')`
`ylabel`	label on y-axis	`ylabel('Voltage / mV')`
`title`	make title on plot (not for sci. paper)	`title('13 May 2002')`
`text`	put a text in a plot	`text(10,5+y,'x=23')`
`gtext`	input of text where you click	`gtext('note this!')`
`ginput`	click and get coordinates for points in plot*	`[x y]=ginput(2)`
`axis`	scale plot, axis([xmin xmax ymin ymax])	`axis([0 10 0 100])`
`axis auto`	default axis scaling (makes all points fit into plot)	`axis auto`
`zoom`	zoom function	
`zo`	Lars W:s zoom function	
`num2str`	make a number into a string	`num2str(r(5))`
`int2str`	make an integer into a string	`int2str(25+M(1,1))`
`[ ]`	make one variable (strings or numbers)	`[r c']` `['temp=',num2str(T)]`
`load`	load data from mat-file	`load data1`
`save`	save specified variables to mat-file	`save data2 t temp`

* Always give the number of points wanted. If you do not do that you have to stop the ginput function by pressing the Enter-key (Return). The example above gives the x and y coordinates in x and y.

Here is a small program that makes a plot:

```
temp=0:5:50;
resistance=23.45*sqrt(temp)+1370;
plot(temp,resistance,'-r')
xlabel('Temperature / {\circ}C')
ylabel('Resistance / {\Omega}')
```

If you write plot(x,y) y will be plotted as a function of x, but if you write plot(y) with only one argument y will be plotted as a function of index, e.g. 1, 2, 3, etc.

In the plot command you can specify if you want markers or lines, and what color these shall be. The following is taken from the plot help file:

Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star		
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

The commands *save* and *load* are convenient for storing data. MATLAB uses a compressed format and the data is stored in a so-called mat-file that can only be opened by MATLAB (we will come to other possibilities later). When you issue a load command the saved variables are loaded with the names they had when they were saved.

Note the difference between the value of a variable and its index. If we have y=[1 3 6 4 3 7 3 6], then max(y) is 7 and the index of y-max is 6. The max and min functions will give both value and index if you give two output arguments, e.g. [val ind]=min(y);

It is important to have a good structure in your directories (catalog). Below is an example with some directories on the hard disk (this example does not use the standard Windows My Documents etc.):

```
c:          \matlab701    \bin...
                          ...
                          \toolbox     \isocal6
                          \matlab
                          \signal

            \measurem   \nordtest   \2001
                                    \2002
                        \sorp8      \cement
                                    \cellulose
                                    \calibration
                                    \sorpeval
                        \tamair     \paper02
                                    \tests
                                    \xtra
```

Note that:

1. MATLAB makes the directory \matlab701 (it can also be called something else, depending on which version you have) and the subdirectories under it. Under \toolbox are directories that contain MATLAB functions. Under \toolbox\matlab are the functions that come with MATLAB. Never change anything there.
2. If you buy MATLAB toolboxes these will be placed under \toolbox, like the \signal (Signal Processing Toolbox) in the example above. Never change anything there.
3. If you make your own toolbox you can place it under \toolbox. In the example above is my isocal6 toolbox that contains functions for manipulating calorimetric data. You can also make a toolbox-directory for the other files that you use often.
4. I have found it convenient to place my other MATLAB-files in the same directories as I keep other files (data, figures, texts). In the example above the \measurem-directory contains a number of subdirectories, one for each project. For example, in the \tamair\paper02 (a subdirectory which contains all files for a paper I have written) I have the following files:

cal020201.txt	measurement file
tamea1.txt	measurement file
tamea2.txt	measurement file
evalcal.m	an m-file which evaluates the data in cal020201
t2psat.m	a MATLAB-function
fig1.m	an m-file that makes Fig. 1 for the paper
fig2.m	an m-file that makes Fig. 2 for the paper
fig3.m	an m-file that makes Fig. 3 for the paper
paperta.doc	a Word document

There are principally two ways to access m-files: either you put them on the path so they can be reached from wherever you are or you place yourself in the directory where the m-file is. For the above four cases I would do like this:

1. The standard MATLAB functions are automatically placed on the path so that they always can be accessed from MATLAB.
2. Any added toolboxes are also automatically placed on the path.
3. If you make your own toolboxes or directories where you place functions that you use often, place these on the path (either using File – Set Path or with the function path).
4. MATLAB files that are placed where they are used (e.g. in c:\measurem\tamair\paper01 in the example above) are best used by placing yourself in that directory (cd c:\measurem\tamair\paper01 in example above or using the MATLAB browser). If you have a file in one of these directories that you need in another directory you can A. Copy it to the other directory if it is a small file (m-files are usually small). B. Change to that other directory to access that file (e.g. if it is a large data file) and then change back, or C. Include the path when the file is called (how this is done depends on what file type it is);e.g. load('c:\measurem\tamair\paper02\cal020201.txt').

Note that many functions is MATLAB can be used in different forms; e.g.:

```
load data3            %loads all variables in data3.mat in the current directory
load data3 xvar       %loads only xvar
load('c:\measurem\tamair\paper02\cal020201.txt').
                      %loads a txt-file from another directory
load(filename)        %load data file whose name is in filename
S=load(filename);     % contents of filename are placed in S
```

Which form of a function you should use depends on the circumstances. If you, for example, have a filename with a spaces (e.g. “kalles data.txt”) you have to use the functional form load(filename).

Also note that many MATLAB functions take different numbers of input and output arguments depending on how you use them, e.g.

```
plot(y)               %draws one line with indexes as x-values
plot(x,y)             %draws one line
plot(x1,y1,x2,y2) %draws two lines
plot(x,y,'*')         %makes a plot with *
hp=plot(x,y,'--');    %hp is a handle, see lecture 6.
```

The help texts give all possible options.

MATLAB can work with complex numbers just as easily as it works with ordinary numbers:

```
c=1+2i;
d=4-8i;
sin(c/d)
```

MATLAB has two functions for integration: `quad` (integrates functions, more about that later in the course) and `trapz` that uses trapezoidal integration to integrate vectors. Use `trapz` like this:

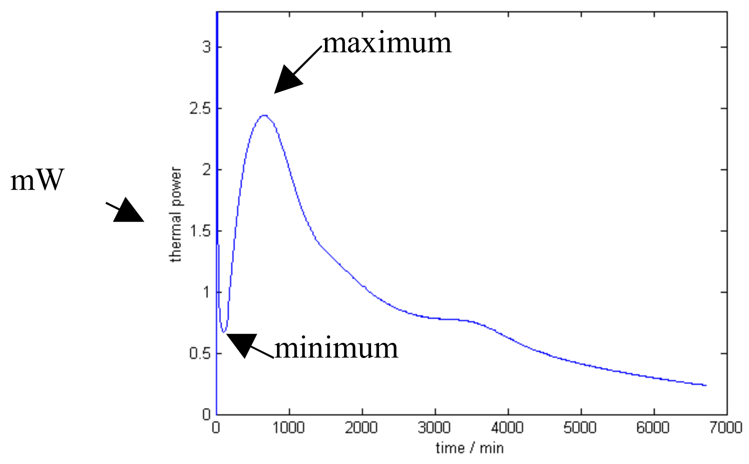
```
x=0:1:10;
y=[0.21 0.34 0.56 0.78 0.90 0.91 0.86 0.77 0.71 0.70 0.71];
integralxy=trapz(x,y)
```

For integrals with equally spaced data one can also integrate by summing (`sum` function) and multiplying the result with the time step.

EXERCISE 2: Write an m-file (=write a program) that finds the minimum and the maximum on a calorimetric cement hydration curve and calculates the integral between the minimum and the maximum. A calorimetric cement hydration measurement gives the thermal power as a function of time. Write an m-file cementnn (where nn are your initials) that does the following:

1. Find the maximum and minimum of variable P (thermal power / mW) as a function of variable t (time / min). Note the note to the min and max functions above. You may assume that the main hydration peak (maximum) never comes before 60 minutes and that the minimum never comes after 1000 minutes
2. Calculates the integral Q (heat, J) of the curve from the minimum to the maximum (for example by the sum function). Note the units: the integral should be in units of joules.
3. Plot the curve with plot, xlabel, and ylabel.
4. Give the integral (e.g. 'Q=3.45 J') somewhere in the figure with the text command.
5. Mark the part of the diagram that you integrated with the fill-command (fill the integrated area of the diagram down to y=0 with a color).

The diagram below shows which minimum and maximum you are to find in the data!



The mat-file exercise2.mat contains an example of t and P for a cement hydration. You should use 'load exercise2' before you run your program. The program should use variables t and P in the workspace.

3. Curve fitting and interpolation

It is useful to be able to do curve fitting. At this lecture we will discuss curve fitting with polynomial equations and how to transform other simple equations to polynomial equations. We will also go through a number of different ways to import and export data to and from MATLAB.

		example
`polyfit`	make curve fit to polynomial of order n	`k=polyfit(x,y,n)`
`polyval`	evaluate polynomial with coefficients in k	`y=polyval(k,x);`
`spline`	cubic interpolation	`yy=spline(x,y,xx)`
`interp1`	interpolation	`yy=interp1(x,y,xx,’method’)`
`interp2`	interpolation	
`interp3`	interpolation	
`demo`	starts demonstration window	
`who`	check which variables you have in the workspace	
`whos`	information on variables in workspace	
`lookfor`	search all m-files for keyword	
`which`	locate functions and files	

Polyfit and polyval

You can use the polyfit function to fit a polynomial to a set of data-pairs, for example to fit a second order polynomial to x and y as in the following example:

```
x=0:6;
y=[0 1.2 2.5 4.1 6.8 8.8 12.1];
k=polyfit(x,y,2)                          % k are the polynomial coefficients
k =  0.1929    0.8357    0.0571
```

To show both the original data and the fitted curve it is convenient to use polyval:

```
xplot=0:0.1:7;
yplot=polyval(k,xplot);
plot(xplot,yplot,'r-')
hold on
plot(x,y,'*')
hold off
```

The polyfit function is sometimes (but not always) useful for other simple functions that are not polynomials. Consider for example the following equation:

$$m = a * \sqrt{t} + b$$

Here, the mass *m* is a linear function of the square root of time *t* (*a* and *b* are constants). We can transform this into a first order polynomial by rewriting the equation:

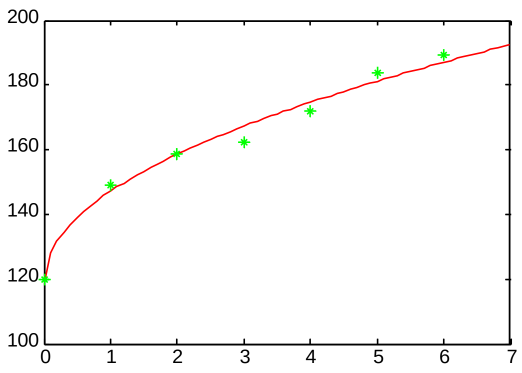
$$m = a * T + b$$

Here $T = \sqrt{t}$. The general curve fitting procedure is like this:

1. Find a transformation.
2. Transform input data.
3. Use polyfit to find coefficients.
4. Use the coefficients in the *original* equation.

Let us test the above equation on the following data set:

```
t=[0 1 2 3 4 5 6];
m=[119.9  149.2  158.9  162.2  172.0  183.9  189.3];
T=sqrt(t);
k=polyfit(T,m,1);
tplot=0:0.1:7;
mplot=polyval(k,sqrt(tplot)
plot(tplot,mplot,'-r')
hold on
plot(t,m,'*g');
hold off
```



The same procedure can also be used on more complex functions as long as they can be transformed to polynomials (usually of order 1), for example the equation:

$$y = (a * \ln x + b)^2$$

can be written like:

$$\sqrt{y} = a * \ln x + b$$

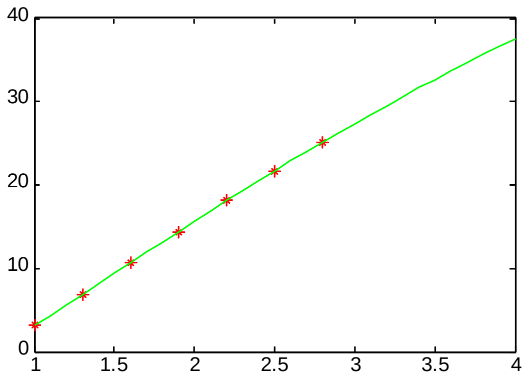
Here we can make the above equation into a polynomial by two transformations X=log(x) and Y=sqrt(y) (written in the MATLAB language, remember that the natural logarithm is ‘log’ in MATLAB):

$$Y = a * X + b$$

A MATLAB example with the above equation:MAKE ARRHENIUS EXAMPLE

```
x=1:0.2:3;          %generation of example data with noise
y=(3.14*log(x)+1.78).^2+rand(size(x));   %a=3.14 and b=1.78
plot(x,y,'r*')      %plot the original noisy data x and y
```

```
X=log(x);
Y=sqrt(y);
k=polyfit(X,Y,1)
hold on
xplot=1:0.1:4;
yplot=(polyval(k,log(xplot
plot(xplot,yplot,'g-')
hold off
```



Interpolation

The following shows the use of the spline and the interp1 functions to generate more data points (with different methods) between given data points. Note that interp1(x,y,xx,'spline') is the same as spline (interp1 is a more general function with more methods):

```
x=0:10;
y=x+rand(1,11);
plot(x,y,'*b')
x2=0:0.1:10;
y2=spline(x,y,x2);
hold on
plot(x2,y2,'g')
y3=interp1(x,y,x2,'linear');
plot(x2,y3,'m')
y4=interp1(x,y,x2,'nearest');
plot(x2,y4,'c')
hold off
```

Repetition

Text strings are written like 'string'. There are a large number of functions that manipulate strings (next lecture). Here is an example of the use of strings:

```
firstname='Karl';
lastname='Anka';
fullname=[firstname,' ',lastname];
text(34,45,['NAMN: ',fullname)
```

Note the use of [ ] to connect (put together) strings into larger strings. The same method is used to put together vectors and matrices, for example:

```
A=[1 3 4 6];
```

```
   B=[A  7:10];  %B=[1 3 4 6 7 8 9 10]
   C=[1 2; 3 4];
   D=[5; 6];
   E=[C D]
E=[1 2 5
   3 4 6]
```

The colon ':' gives a range, for example (continuing with the above examples):

```
   B(1:3)
ans=[1 3 4]
   B([3 5:end])
ans=[4 7 8 9 10]
```

Note also the difference between a number in a vector and its index (plural indices):

```
   P=sin(1:10)'
P = 0.8415
    0.9093
    0.1411
   -0.7568
   -0.9589
   -0.2794
    0.6570
    0.9894
    0.4121
   -0.5440
   min(P)
ans=-0.9589
   [Pmin,imin]=min(P)
Pmin= -0.9589
imin= 5
```

There are a number of more exotic plot functions in MATLAB: `mesh`, `slice`, `hist` etc. Look in the demos for some examples of these. There are a large number of other demos and built-in data files that one can use to see what MATLAB can do and test your ideas. You can reach the demos from Help or by issuing the command `demo`.

EXERCISE 3a. Write a program that curve-fits vapor pressure data from the Handbook of Chemistry and Physics. You will get a mat-file data3.mat that contains variables *T* and *substances*. Just run 'load data3' with data3.mat in your current directory (or on the path) to get the variables into the workspace.

The variable T contains the temperatures (ºC) at which the saturation vapor pressures are 1, 10, 40, 100, 400 and 760 mmHg for the following three organic substances found in indoor air: nicotine, *d*-limonene and α-pinene (data from Handbook of Chemistry and Physics):

```
   T=[61.8 107.2 142.1 169.5 219.8 247.3
   14.0 53.8 84.3 108.3 151.4 175.0
   -1 37.3 66.8 90.1 132.3 155.0];
```

The names of the substances are in a variable called ’substances’:

```
   substances=['nicotine    ','d-limonene  ','alpha-pinene'];
```

Your task is to write a program called psatnn.m (nn are your initials) that uses T and substances in the workspace. The program shall do the following:

1. It shall find the number of substances for which there is data (we want the program to be useful for other sets of data later).

2. It shall make a least square curve fits of the data for each substance to the following equation that is good for his type of data (psat is saturation vapor pressure in mmHg and T is temperature in K):

$$\ln(psat)=A-B/T$$

Note that T should go into this equation in kelvins. Transform psat and T so that you can use polyfit (that does a least square fit). Note also that the natural logarithm is ‘log’ in MATAB.

3. Plot the data points *and* the fitted curve (saturation vapor pressure as a function of temperature) for all substances in one diagram.

4. Name the curves (automatically) with the names of the substances (it is not easy to make a routine that finds a perfect place for each name, but make something...).

Note also the following:
- The diagram should have T / ºC as x-axis and psat / mmHg as y-axis.
- You need to make a curve fit (and find parameters A and B); otherwise you cannot draw the line in the figure.
- You can cut the diagram at psat=760 mmHg (atmospheric pressure) as there can not be any higher saturation pressures at one atmosphere (the temperature at which the saturation pressure is 760 mmHg for a substance is the boiling point (boiling temperature)).
- There is also a mat-file data3x with other substances and other vapor pressure data. Test your program with this data too!

---

4. Logical operators, program control and string handling.

In MATLAB true is 1 and false is 0 (true can actually be represented by any non-zero number). The logical functions given below are the most common, but it is also possible to use Boolean algebra with or, and, xor etc.

LOGICAL OPERATORS		
`==`	logical equal to	`if (a==2)`
`>`		
`<`		
`>=`		
`<=`		
`~=`	not equal to	`if (a~=2)`
`~`	negation	`if ~(a==0)`
`&`	logical AND	`if (a==2)&(b==1)`
`|`	logical OR	`if (a==2)|(b==1)`
`find`	find indices	`ind=find(a==5)`
PROGRAM CONTROL		
`if...` `else...` `end`	if statement*	
`for...` `end`	program control: loops*	
`while...` `end`	program control: while statement*	
`switch...` `case...` `end`	program control: switch*	
`break`	jump out of current for, while or case statement	
`pause`	make pause**	`pause(5)`
STRING HANDLING		
`strcmp`	compare strings	`strcmp(str,'hej')`
`findstr`	find one string within another	`findstr(A,'p')`
`deblank`	remove trailing blanks	`deblank(s2)`
ROUNDING ETC.		
`abs`	absolute value (remove sign)	
`round`	round to nearest integer	
`ceil`	round towards Inf (ceiling)	
`floor`	round towards -Inf	
`fix`	round towards zero	
`mod`	modulus (signed remainder after division)	`mod(x,5)`

* examples of the use of these program control statements are given in the files logik.m and loops.m

** pause(5) makes a 5 second pause; pause by itself pauses until any button is pressed (Ctrl^C terminates both the pause and the program).

The function `find` is very powerful. `find(x)` finds the indices of x that are logically true (1) It can be used in many cases to find where in a vector something has taken place. Here are some examples:

```
find([1 3 5 7 8]==5)       %find the index/indices where the variable has the value 5
find(min(t>1000))          %find the index of the first time t that is higher than 1000.
find((RH>70)&(RH<75))      %find the indices where RH>70 and RH<75.
find(diff(RH)==0)          %find  indices where the RH the same as the following RH.
find(max(diff(P)./diff(t)))%find the index where the slope of P vs. t is highest.
```

Note that find always gives indices as output, not the value of the variable.

EXAMPLES:

- logik.m gives examples of the use of logical operators.
- hitta.m plots a curve with many min/max and then marks each max with a red star, each min with a green star and each passing of zero with a yellow line. Note that new curves are generated each time that hitta.m is run as there is rand in the function, and that 'find' is used to find max/min/zeros.
- loops.m gives examples of the use of program control commands.
- strex.m finds all filenames in a certain directory that contains a certain string. The dir-command is used to generate a list of all filenames. Some manipulation is needed to get the correct string-formats. Note that the original directory is stored so that the program returns to that at the end.

EXERCISE 4. The Excel-datafile cottoncloth.xls contains the result from a run with a DVS sorption balance to measure the sorption isotherm of a cotton textile. The sorption isotherm is the relation between the air relative humidity (RH) and the moisture content absorbed by a solid, i.e. (water gain)/(dry mass), as a function of RH (at equilibrium).

You can import the data in this rather large file into the workspace with the MATLAB function xlsread that reads all xls-files. If you do like this the data will be imported to a variable a in the workspace:

```
a=xlsread('cottoncloth.xls');
```

In this large matrix you need only use columns 1 (=time in min), 2 (=mass in mg) and 9 (=RH in %).

During the cottoncloth-measurement the RH was first kept at zero to dry the sample. When the dry mass was constant the balance is zeroed and the RH was then stepwise increased to higher and higher values. After the maximal RH it also stepped back to zero RH. At each RH-level the DVS waited until it had a constant mass.

Your task is to write a program that calculates the absorption and desorption isotherms from the data (the sorption isotherm is moisture content (=(water gain)/(dry mass)) as a function of RH at equilibrium). The program should both plot the sorption isotherms and give a list (in the workspace) of the relative humidity *and* the moisture content, e.g.:

Note that:

- When I write “constant mass”, I mean “almost constant mass” as I otherwise would have had to wait for a very long time for the measurement to be completed.
- The sorption isotherm should be calculated as close to equilibrium as possible, i.e. you shall use the mass data points just before the DVS changes RH to a new value. Use the find-function to find the indices of the equilibrium masses.
- The dry mass is found at the end of the initial period when RH=0;
- Call your program sorpnn.m where nn are your initials.

---

5 Functions, script files and the eval-function

During the previous lessons we have written programs in m-files just as if the same commands had been issued from the workspace. Now we will start to use “functions” that have their own workspace and that need to be written and called in a certain way.

		example
`eval`	evaluation of command given as string	`eval(['sin(',num2str(pi),')'])`
`function`	see below	
`nargin`	number of input arguments to function	`if nargin==1;b=0;end`
`nargout`	number of output arguments to function	`if nargout==2;b=2;end`
`varargin`	variable number of input arguments	

The function eval is very powerful. One instance where eval is useful is when we have a number of variables called var1, var2, var3 etc. that we want to work with. Then the eval function can be used to “open” these variables one by one and call them something that the evaluation program can accept, e.g.

```
for k=1:n
   y=eval(['var',int2str(k),';']);
   myprog(y)
end
```

There are two ways of writing a MATLAB-program: script-file or function:

	script	function
Definition	A script file is an external file that contains a sequence of MATLAB statements. By typing the filename, subsequent MATLAB input is obtained from the file. Script files have a filename extension of ".m" and are often called "M-files".	A new function added to MATLAB's vocabulary expressed in terms of other existing functions

Example	`%script file stat.m`
	`m = sum(x) / n;`
	`s= sqrt(sum((x - m).^2)/n);`
		`function [m,s] = stat(x)`
		`n = length(x);`
		`m = sum(x) / n;`
		`s= sqrt(sum((x - m).^2)/n);`

Syntax	Just like you would write in the workspace	Starting with a function command
Variables	Uses the same variables as the workspace	Has its own workspace with its own variables.
Input/output	The input should be in the workspace when the script file is started (or asked for in the script file). The output will be in the workspace when the script file end (and/or displayed by the script file)	The input must be passed to the function (x above; or asked for in the function). The output must be passed from the function (m and s above)
Calling	stat	[m,s]=stat(x) (note that one does not need to use the same variable name when calling the function as in the function)

A note on using apostrophes in strings

Using apostrophes within strings can be difficult and confusing. The general idea is that two apostrophes within an apostrophed string means one apostrophe, i.e. disp('sad''asd') will display sad'asd. Sometimes even more apostrophes are needed. e.g. when one wants to display '*' (with the apostrophes), one has to write disp('''*'''). Below is an example where the plot symbol syntax '*' (with apostrophes) is incorporated into an eval-string:

```
a1=[3 5 6 8];
a2=[5 4 3 2];
a3=[5 7 8 9];
clf;
for k=1:3
   eval(['plot(a',int2str(k),',''*'')']);
   hold on
end
hold off
```

EXAMPLES:

circ.m is a function that draws a circle. `circ(2,4,1,100)` draws a circle with its center at x=2 and y=4, with a radius 1 and uses 100 segments.

porer.m is a function that uses circ.m to draw a number of circles (pores) with their centers x and y of 0 and 10.

Note that the above two functions have input arguments, but no output arguments.

EXERCISE 5: The function porer plots a number of circles (for example pores in a porous material). Rewrite the program porer.m so that is does not plot any pores on top of other pores and so that no parts of any pore are outside the 10x10 box. Your function should have the following call-syntax: porernn(n,d) where nn are your initials, n is the number of pores, and d is the smallest distance allowed between two pores (d=0 means that two pores can touch each other).

Hints:

- You should start by generating n pore-diameters and then try to place each of them randomly into the area until you find a place for them.
- To get as many pores as possible into the 10x10 area you shall sort the pores so that you place the larger pores first (check MATLAB function sort.m) (it is difficult to place the larger pores when there are smaller pores already placed).
- You will have to set a limit to the number of tries to place each pore as it is impossible to place all pores if one enters a too high n (or a too high d). Give some kind of warning that the program was not successful in placing all the pores.

---

6. Advanced plotting

MATLAB has very powerful plotting functions. Here we will only look at some of these, but you can run the MATLAB demos to see more possibilities.

		example
`plot3`	plot 3D	
`grid`	grid lines	`grid on`
`view`	change view on 3D plot	`view(30,60)`
`mesh`	plot 3D surface with a mesh	
`surf`	plot 3D surface	
`view`	set position from where you view 3D plot	`view(-30,100)`
`colormap`	set colormap (colors of plot)	`colormap(spring)`
`caxis`	set color	
`errorbar`	get errorbars on plots	
`legend`	get legends on plots	
`set`	sets (changes) a HandleGraphics property, set only retrieves possible values	`set(h,'FontSize',12)`
`get`	retrieves the value of a HandleGraphics property, get only retrieves present values	`val=get(h,'FontSize')`
`delete`	deletes files or graphical objects	`delete(h)`
`axis`	set axis	`axis([0 10 0 10])`
`axis auto`	get back default axis setting	
`axis square`	make square axis	

Try

```
plot3(1:10,1:10,(1:10)^2)
[X,Y,Z]=sphere(30);
surf(X,Y,Z)
mesh(X,Y,Z)
surf(1:10,1:10,(1:10)'*(1:10))
```

MATLAB figures are made with Handle Graphics. Each part of a plot (a line, an axis etc.) has a handle (a name in the form of a number). To access a part of a plot one used the handle. An example:

```
hp=plot(x,y,'o');               %plots circles standard size
ms=get(hp,'MarkerSize');        %gets the size of the markers (the circles)
set(hp,'MarkerSize',ms*2)       %makes the markers twice the size
```

Note that in both set and get the name of the property you like to retrieve or change is written as a string. To know all properties that you can work with for a certain handle h you can write

```
set(h)   %to get a list of all properties and all possible values of these (not numeric)
get(h)   %to get a list of all properties and their present values
```

Here is an example of the use of handles. You have a figure and the user should be able to place a text in the figure with gtext. To make sure he or she knows that he or she must click one can write a message in the plot that is taken away after the click:

```
plot(rand([1,10]),rand([1,10]));
ax=axis;
tx=mean(ax(1:2));
ty=mean(ax(3:4));
ht=text(tx,ty,'Click where you want the text');
set(ht,'HorizontalAlignment','Center','VerticalAlignment','Middle')
gtext('New text')
set(ht,'Visible','Off')
```

In this example ht is a handle to the text and the ax, mean and set commands on the first three lines are used to get the text in the middle of the plot.

It is also possible to access the HandleGraphics properties by activating a figure with Edit Plot (arrow facing up-left).

Normally, when you for example plot a graph, you either chose a color or get one by default. In other more complex graphical objects mesh and surf color-ranges can be used as a way to show information. For these graphics objects the colors are taken from the current colormap linearly from the lowest to the highest values of the graphical object. The color of each part of a plot are determined by the values in the plot and which colormap you are using. The following colormaps are built into MATLAB:

hsv	- Hue-saturation-value color map.
hot	- Black-red-yellow-white color map.
gray	- Linear gray-scale color map.
bone	- Gray-scale with tinge of blue color map.
copper	- Linear copper-tone color map.
pink	- Pastel shades of pink color map.
white	- All white color map.
flag	- Alternating red, white, blue, and black color map.
lines	- Color map with the line colors.
colorcube	- Enhanced color-cube color map.
vga	- Windows colormap for 16 colors.
jet	- Variant of HSV.
prism	- Prism color map.
cool	- Shades of cyan and magenta color map.
autumn	- Shades of red and yellow color map.
spring	- Shades of magenta and yellow color map.
winter	- Shades of blue and green color map.
summer	- Shades of green and yellow color map.

A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0. For example, [0 0 0] is black, [1 1 1] is white, [1 0 0] is pure red, [.5 .5 .5] is gray, and [127/255 1 212/255] is aquamarine.

EXAMPLES:

The rand-function generates randomly distributed numbers (distributed like a rectangle between 0 and 1). What does the distribution of the difference between such random numbers look like? Well test the following:

```
hist(rand(1,100000)-rand(1,100000),50)
```

There are a number of nice data files in MATLAB that you can use to test 3-D plots (look in demo). Try

```
load penny
pcolor(P)
view(160,70)
colormap(copper)
```

Test Handle Graphics with this sequence:

```
x=1:10;y=sqrt(x);
hp=plot(x,y,'*')
hold on
hp(2)=plot(y,x,'*')
hold off
set(hp,'Marker','o')
set(hp(1),'Color','r')
legend(hp,'curve 1','curve 2')
```

...and this...

```
x=1:12;y=[1 3 4 6 7 6 4 5 3 4 2 0];
plot(x,y)
xlabel('Month')
ylabel('Temp')
se(gca,'Xtick',1:12,XTickLabelMode','manual','XTickLabel' ...
,
         ['Jan';'Feb';'Mar';'Apr';'May';'Jun';'Jul';'Aug'; ...
         'Sep';'Oct';'Nov';'Dec']);
```

File I/O

Save and load are the principal input-output (I/O) functions in MATLAB. However, there are also low-level possibilities using sequences like open file, write to file, read from file, close file. Such low level operations may be necessary if you want to write to a file in a certain format or read a file generated by a measurement instrument. One example of the latter is PLW2ML.m used in the exercise in the next chapter. For more information on different possibilities of low level file I/O look in the MATLAB help and in the files iotest, iotes2 and iotest3. Note that with all these three programs you have to write (1) before you read (0) the first time you use them!

EXERCISE 6: The file called ex6.m contains a short m-file that generates a plot. Change that file to a new file ex6nn.m (where nn are your initials) and make the following changes:

1. The figure should have a width of 75 mm and a height of 50 mm when printed (including labels) suitable for a two-column journal.
2. All text and figures (numbers) shall be with font size 14 and in font ‘Roman’ (or ‘TimesRoman’, ‘TimesNewRoman’).
3. Add a legend-box calling the solid line ‘yellow brick’ and the dashed line ‘red brick’.
4. Add circles on both lines at the data points so that one can see how many data there are.
5. Add uncertainties (standard deviations) to both lines using the function errorbar. The solid line has an uncertainty of 1 and the dashed line has an uncertainty of 2.
6. Scale the figure so that there is a minimum of unused space.

---

7. String handling and data structures

String handling

ASCII is the system by which each letter has a number so that computers can work with text strings. The MATLAB function `char` gives the letter corresponding the integer x and `abs` give the number corresponding to a certain letter

```
  char(103)
ans = g
  abs('g')
ans = 103
```

Note that the ASCII-system has more characters than the letters and the digits. It also includes characters such as "CR" (Carriage Return, i.e. new line).

MATLAB has two different ways of handling text variables containing more than one string: character array and cell array. How this works is not trivial. Trial and error is may be the best method. Here are a few lines from the MATLAB help files:

A collection of strings can be created in two ways: 1) as the rows of a character array via STRVCAT or 2) as a cell array of strings via the curly braces. The two are different but can be converted back and forth with CHAR and CELLSTR. Most string functions (but not other functions) support both types.

Examples

```
      msg = 'You''re right!'
      name = ['Thomas' ' R. ' 'Lee']
      name = strcat('Thomas',' R.',' Lee')
      C = strvcat('Hello','Yes','No','Goodbye')
      S = {'Hello' 'Yes' 'No' 'Goodbye'}
```

Note the use of curly braces {}; `S(4)` is the whole string 'Goodbye'. More examples:

```
a={'hej','kalle','sven'}       %a cell array
a(2)                           %kalle
[a(2),'f']                     %error! (one cannot combine cellstr with char)
[char(a(2)),'f']               %kallef (charcter array)
a(2)=cellstr((([char(a(2)),'f'])        %converted back to cell array
```

Cell arrays are convenient to use when one wants to have many strings with different sizes in one vector. Another possibility is to use an character array and pad the individual strings with blanks so that they are the same length. Padded blanks can be removed with `deblank`:

```
a=['hej  ';'kalle';'sven '];
deblank(a(1,:))             %gives hej without trailing blanks
```

		example
`abs`	the symbol of an ASCII number	`abs('d')`
`char`	the ASCII number of a symbol	`char(76)`
`[...]`	makes one string of two or more strings	`['my name is ',namestr]`
`{...}`	makes a cell array with more than one string	
`datevec`	separates components of date-string	`see help datevec`
`datestr`	sets the way dates are represented as strings	`see help datestr`
`datenum`	calculates the number of the day	`see help datenum`
`tic`	start stopwatch	`tic`
`toc`	stop stopwatch and display result	`toc`
`isempty`	true for empty variable	`if isempty(x);x=0;end`
`ischar`	true for character array (string)	
`isletter`	true for letters of the alphabet	
`isspace`	true for white space characters	

`Tic` and `toc` are useful for timing your programs to optimize them.

Data structures

A data structure can hold many different types of data under one name. Here is one example of a structure `exp`:

```
exp.number
exp.date
exp.filename
exp.data.t
exp.data.U
```

The structure `exp` contains information on experiments. Each experiment has a number, a date, a filename and two columns of data (t and U). All this is contained in the structure exp. Note that exp contains both numbers (e.g. exp.data.t) and strings (e.g. exp.filename). We can easily access different parts of the structure:

```
exp(5)                        %all fields of experiment 5
exp(5).date                   %the date of experiment 5
exp(5).data.t(100:200)%part of the t-vector of experiment 5
```

One advantage with structured data is that it is easy to move data from one function to another. You just put everything into a structure and pass that structure to the function (this is used by some Handle Graphics functions for which it is quite difficult to pass information to the functions). The following two forms of a function are equivalent, but the second one is often more convenient (and it is very easy to add a new parameter to the structure that will also be passed to the function):

Without structures:

```
t=1:1000;                     function hp=fplot(t,y,txt);
y=rand(size(t));              hp=fplot(t,y);
txt='random numbers'          legend(hp,txt)
hp=fplot(t,y);
```

With a structure:

```
s.t=1:1000;                 function hp=fplot(s)
s.y=rand(size(t));            hp=plot(s.t,s.y);
s.txt='random numbers'      legend(hp,s.txt)
hp=fplot(s);
```

MATLAB Help and Info

There are a large number of sources of information on MATLAB:

- MATLAB help
- The MATLAB Digest (email-letter)
- http://www.mathworks.com/support/
- MATLAB discussion forums, e.g. http://wwweng.uwyo.edu/matlab/discuss/wwwboard.html
- MATLAB tutorials, e.g. http://www.engin.umich.edu/group/ctm/basic/basic.html

Try a search on the web and you will get many interesting hits!

Exercise 7a Evaluating many files automatically

A large number of experiments have been made in which the temperature has been measured in cement pastes (cement powder + water). When cement and water are mixed heat is released and the temperature increases. In the present measurements temperature has been measured in two positions in reacting cement paste samples every second. All relevant data files have the form “cemtX_Y”, where “cemt” is the name of the experiment, X is the number of the cement, and Y is the number of the measurement on a specific cement.

Your task is to write an evaluation program that automatically evaluates all files of this type (with filenames starting with “cemt”) in the present directory (both the program and the data files should be in this directory) and generate a table of the cement and the result, for example like this:

```
cement 0: Delta T=1.805 K
cement 0: Delta T=1.815 K
cement 0: Delta T=1.6575 K
cement 1: Delta T=1.69 K
etc.
```

The temperature you shall evaluate is the temperature change from 60 s to 600 s during each measurements (use the mean of the two measurements for each sample).

The files that you are going to evaluate all have the extension .plw and you shall also use a MATLAB program called PLW2ML that reads these files (look in help PLW2ML to see how it works and get some hints on how to solve this exercise; you cannot read these files with load as they are in a special format). You will get PLW2ML and the measurement files by email (note that you will also get some dummy data files that your program shall not read!)

Call your program manyfilesNN.m, where NN are your initials.

EXERCISE 7b:

Write a MATLAB-program that deciphers a coded message. The system of the cipher is that each letter in the original text is exchanged for another letter which is n steps further on in the alphabet. If n=3, a ‘c’ becomes an ‘f’. When you come to ‘z’ you continue with ‘a’. The program need only work for lower case letters (“små bokstäver”) a-z and space. The program you should write should be run by the following command: decodenn(tx,n) where tx is the text to be deciphered (nn in the file name are your initials). Test your program on the text in the file secretxt.m which has n=6. Start by looking at the numbers associated with a-z (ASCII-codes). You will/may find the following functions useful: abs, char and mod. The space (blankslag) should not be changed; the same spaces found in the coded text should be found in the decoded text.

EXERCISE 7c:

Rewrite decodenn.m to codenn.m that codes text by the same procedure. Depending on how you have written decodenn, only very minor changes may be needed.

8. Efficient MATLAB programs, debugging and the MATLAB Notebook

A few rules to making efficient MATLAB programs that run quickly:

1. Pre-allocate vectors

It takes time each time MATLAB needs to expand a vector, so it is best to make vectors the size they should have instead of step-wise increasing their size. It is better to make it too large from the start than to have to increase its size many times: The time taken for the inefficient example below will be decreased if you pre-allocate the vector ind

```
r=rand([1 50000]);

disp('first time without pre-allocated vector')
tic
m=0;
for k=1:length(r)
  if r(k)>0.5
    m=m+1;
    ind(m)=k;
 end
end
toc

disp('second time when the vector already exists')
tic
m=0;
for k=1:length(r)
  if r(k)>0.5
    m=m+1;
    ind(m)=k;
 end
end
toc
```

2. Do not use loops where you can avoid using them

(NOTE: The JIT-acceleration technology introduced with MATLAB R13 improves loop performance dramatically so this rule is not as important anymore)

Write as much as you can without for, while, if etc. If you can write something in one line with :, find etc it will run quicker as the built-in MATLAB-functions are fast. For example, find the indices of all values that are greater than 0.5 in a vector:

```
r=rand([1 10000]);
disp('inefficient way')
tic
```

```
m=0;
for k=1:length(r)
  if r(k)>0.5
    m=m+1;
    ind(m)=k;
  end
end
toc

disp('more efficient way')
tic
ind=find(r>0.5);
toc
```

<u>3. The MATLAB debugger</u>

MATLAB also has a number of debugging functions that you can use to follow the execution of functions in which you normally cannot know what is going on (values of variables etc.). The names of all these functions start with db, e.g. `dbstop`. A typical debugging session with function weighsimple (whose variables normally are not accessible as it is a function):

```
dbstop in weighsimple at 26 %stop at line 26 *

weighsimple                 %start program to debug

K>>  numberofmea         %'K>>' is debugger prompt. Check value of numberofmea
numberofmea= 1

K>> dbstep               %step debugger one step in program

K>> numberofmea
numberofmea= 2

K>> dbcont               %continue program

K>> dbclear           %clear all breakpoints
```

(*) Breakpoints are normally set and removed by left-clicking the horizontal line in front of each line in the editor.

<u>4. M-Lint code checker</u>

I the programming world "lint" is a program that checks another program for style, language, usability and portability problems. MATLAB has such a tool called M-Lint that can either be called from the editor (Tools – Check code with M-Lint) or from the command window by the command mlint. M-Lint will give suggestions for improvements in your code.

5. The MATLAB profiler

The profiler is a way of making MATLAB count how much time it spends in different parts of a program or in different functions. It will give you information so that you can remove bottlenecks in your programs. Check help profile and help profreport. Normally you would issue the following command to check what goes on when you run a program, e.g., weighsimple:

```
profile on

porerlw

profile viewer %to see the result
```

When you have written an m-file it is often does not run as expected... . Here are a few ideas of to make it work properly:

1. Always write clear programs with a logical structure and many comments!

2. Divide large programs into smaller parts and check that each part works as expected before you connect them.

3. Before starting to execute the file at all MATLAB will check its syntax, so when you run a newly written m-file you will usually get a lot of error-messages because you have not used the correct syntax, e.g. written `axis(12 20 0 300)` instead of `axis([12 20 0 300])`, or simply made mistakes like not having the same number of left and right parentheses. Correct all such mistakes first.

4. When all syntax errors are gone MATLAB tries to run the program and then it is common that it either crashes or gives the wrong result. If it crashes (ends with an error) you will get some clues to what was wrong by reading the error message. Here are some ideas of how to find errors in script files and functions:

In script files (where you have access to the variables):

A. Instead of using the debugger you can put a pause in a program and stop the program with Ctrl+C when it comes to the pause (you may also need to type `disp('pause!')` on the line before the pause to know when it has come to the pause).

B. Remove some semicolons so that you get print-outs of the values of critical parameters.

C. Insert 'intelligent' lines that, e.g., stop the program at a critical point (i.e. when something goes wrong), like

```
if (n>6534)&(K(n)<K(n-1))
   disp(['K(n)=',num2str(K(n)),'at n=',int2str(n)])
end
```

In functions (where you normally do not have access to the variables):

A. Make the function into a script-file while you look for errors in it. Note that all input parameters must then be given in the workspace before you start the program.

B. Use the debugger.

5. With large programs it is a good practice to make it possible to run different parts of a program on their own (maybe with the help of special test programs). One example: A program has one large part where data for data input and initial trivial manipulation, and a second smaller part in which complex calculations are made. Each time the program is run the first part takes about 2 minutes of manual work and the second part only a few seconds of run-time. To be able to debug the second part it is then good to have a possibility to skip the first part. This can be done either by having a special routine that inputs test data by which the second part can be tested, or by making it possible to run the program a second, third time etc. with the data that was inputted during the first run.

If you get Out of Memory, read `help memory`. The functions `clear` and `pack` helps you get memory back.

MATLAB Notebook

MATLAB has a connection to Word called notebook. With the command notebook you will create a Word-file that you can write MATLAB-commands in. When in the Word-file you have to first write a command, then activate it (make it into a MATLAB-text) with Alt+D, and finally execute the command with Ctrl+Enter. Note that when you issue Alt+D the font etc of the text changes. Read more in the MATLAB Help. Example:

```
This is a magic square                    %Ordinary text

A=magic(4)                                %MATLAB command

A =                                       %Result
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1

sum(A)

ans =
   34   34   34   34

sum(A')

ans =
   34   34   34   34
```

EXERCISE 8:

In the file badprog.m there is a new function that you should debug so that it performs what it is supposed to do (see the first lines in the file).

---

9. General curve fit, optimization and linear algebra

`fminserch`	finds minimum of multi- variable function
`optimset, optimget`	used to tailor the `fminsearch` function
`fzero`	find zero of a function of one variable
`global`	makes variable global (accessible from all programs and functions in which they are made global)

General curve fit procedure (optimization)

Optimization is to find the best solution to something. In science it is often interpreted as finding the coefficients of an equation that gives the best fit to measured data. This can be done with the MATLAB- function `fminsearch` by searching for the coefficients that minimizes the least-square difference between the equation-curve and the measured data. Here is one example:

A function:

```
function S=mfunc(C)
%S is the variable to minimize; it is the sum of
%the squares of the differences between the measured
%values M and the calculated values using T and the present
%guess of the coefficients a and b.
global T M
a=C(1);
b=C(2);
S=sum((M-a*T+b*sqrt(T)).^2);
```

To run `mfunc` issue the following commands in the workspace:

```
global T M %these
[a,b]=fminsearch('mfunc',[1 2]);
```

The function fminsearch is called to minimize the function mfunc; 1 and 2 are starting values; the output result is placed in a and b

Note the use of global variables; a convenient way to make a variable usable in both a function and the workspace. However, global variables are usually not considered to be good programming practice and there are usually other more reliable ways of charing variables across functions. In the `fminsearch` function it is possible to pass extra arguments along to the function, i.e. arguments that are needed to calculate the square sum, but which the square sum shall not be minimized with respect to. Below is one example Hhfit_short that calculates the best fit of sorption data to the Hailwood-Horrobin equation:

$$M=\frac{H}{A+B*H+C*H^2}$$

Here, *H* is the relative humidity and *M* is the moisture content.

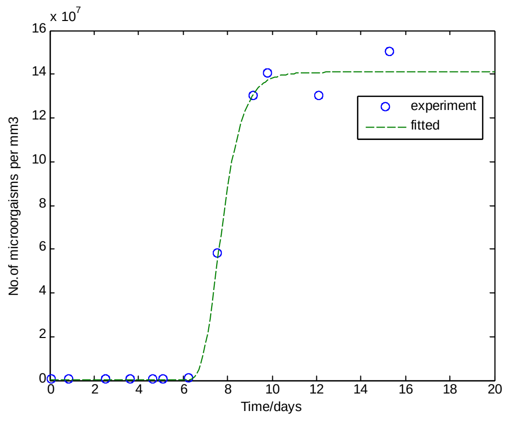
```
function [A,B,C]=HHfit_short(H,M)
% [A,B,C]=HHfit_short(H,M)
% Calculates the best fit A, B and C to given H and M.

% Lars Wadsö 19 Aug 2006

optHH=optimset('MaxIter',100000,'TolX', ...
                              1e-10,'MaxFunEvals',20000);
ABCguess=[1 1 1];
ABC=fminsearch(@(ABC) HailwoodHorrobinSquaresum(ABC,H,M) ...
                                         ,ABCguess,optHH);
A=ABC(1);B=ABC(2);C=ABC(3);

function S=HailwoodHorrobinSquaresum(A,H,M);
S=sum((M-H./(A(1)+A(2)*H+A(3)*H.^2)).^2);
```

Note a few things with the above program:

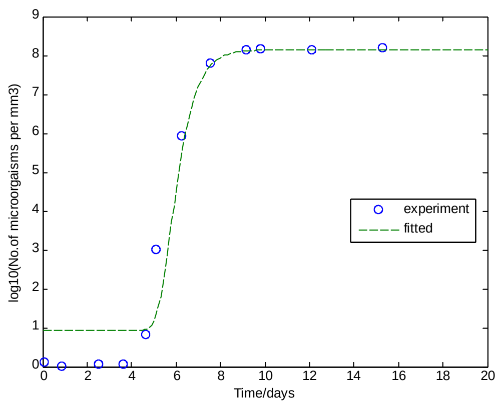
- The first comments lines (after a possible function command) are the help text that is given when you type help and the name of the function, e.g., `help Hhfit_short`.
- After the help text one usually makes one empty line and then a comment line showing who wrote the program, when it was written, version number etc. This is not shown in the help text.
- `optHH` contains information for `fminsearch` on how to run (more information in help files). The function `optimset` is used also for `fzero` and other iterative functions.
- `ABCguess` is a start value for the optimization. In some cases it will improve the speed and the solution if you can give good such guesses, i.e., in the same order of magnitude and with the same sign as the solution.
- The `@(ABC)`-part should be interpreted as "using `ABC` as the variable to change to find the minimum of the function (`HailwoodHorrobinSquaresum`)". The other variables `H` and `M` are just passed to the function.
- The same variable may have different names in a pragiram that calls a function and in the function. In the above program `ABC` in passed to the function `HailwoodHorrobinSquaresum`, but in this function (which uses another set of variables) it is called `A`.
- The function `HailwoodHorrobinSquaresum` is included as a subroutine at the end of the same file as the main function `HHfit_short`. This is a good way to make sure that the subroutine is always connected to the main program (if they were in two separate files, maybe you will send only the main file to someone who then cannot run the program). However, it is not possible for any other functions to assess the subroutine when it is hidden inside another function.

The program Hhfit is a longer version of Hhfit_short that contains tests of the inputs and plot possibilities.

Note that it is sometimes not optimal to just minimize the whole data set. For example, if there are many more values in one interval, this interval will become more heavily weighted that other intervals. Similarly, if the data has very different values (for example exponentially increasing data) some kind of data transformation may be needed to give a good curve fit. One example is exercise 9b, below. If the data is optimized as it is one will get the following result (which looks good):



However, if one instead looks at the log10 of the result one will see that the optimization procedure has not at all cared about the initial values as they were so small:



One result of this problem is that the initial concentration is overestimated. By using the above method on the log10 of the data, the result will be better.

Linear equation systems

Linear equation systems are easily solved by MATLAB with matrix division. An example:

```
2x+y-z=3
-3x+4y-3z=-1
x-y-z=-5
```

This can be written `A*X=B` where

```
A=[2 1 -1;-3 4 -3;1 -1 -1];B=[3;-1;-5];
```

To solve for X we write

```
X=B'/A';     %or X=A\B;
            %note the use of a backslash (see help slash)
```

and get

```
X =    1.4737    3.2632    3.2105
```

Iterative solutions

Sometimes it is necessary to solve equations iteratively, e.g. x^1.7=sqrt(x)+2 can be solved by

```
x=1; %start value;
xold=0;
while abs(xold-x)>1e-8
  xold=x
  x=(sqrt(x)+2)^(1/1.7);
end
x
```

Note that iterative solutions will not always converge and that there sometimes are many solutions. The one you get will then depend on your start values. Check different start values.

EXERCISE 9a (only for those using the DVS sorption instrument):

In this exercise you will rewrite the program sorpnn you wrote in exercise 4. As you may have noted, the mass in the data-file was not completely constant when an RH-step was finished. If each step had continued for longer time somewhat higher values in absorption and lower values in desorption would have been reached. In some commercial sorption balances there is an built-in correction that, at the end of each RH-step, makes a curve-fit to try to determine the final water gain value that would have been reached if the measurement had continued for a longer time. The most common function to use is an exponential one:

$$m(t)=m_0+(m_f-m_0)\exp(-kt^n)$$

Use this function and rewrite the program so that it makes a curve-fit of the mass-curve for each RH-step. To get the best fit possible one needs to let the optimization also find where to start the exponential function. One can do this by rewriting the above equation:

$$m(t)=m_0+(m_f-m_0)\exp(-k(t-t_0)^n)$$

Here m_0, m_f, k, t_0 and n and fitting parameters.

Some details:

- Usually the first part of a sorption step is quite fast and will not be well fitted with the above equation. You should therefore only use the last part of the measured curve, e.g. the last 25%.
- The curve fit should be made by using the `fminsearch`-function discussed above.
- It can be difficult for the optimization to find a good solution if you input the real time and mass as these can be quite high values. It is better to scale them first so that the

first data point for a curve fit always is zero time and zero mass. Note that you have to do a reverse scaling afterwards.

- The resulting program should be called sorpnn2.m.
- The output from the program should be the same as for the sorpnn-program, but with a third added row with the water gains calculated using the calculated final mass at each step (m_f in the above equation).

EXERCISE 9b (for those not using the DVS sorption instrument)

The Gompertz function is used to relate microbiological growth to time. Within a sample, e.g. a package of foodstuffs, one often finds a lag-phase with no growth, then an exponential growth phase, and finally a plateau where the number of microorganisms are quite constant. The Gompertz function can model all these three phases. Find the best parameters A, B, C and D to fit the Gompertz function N=A*exp(B*exp(-exp(C-D*T))) to the following somewhat scattered data (also found in Gompdata.m):

```
t= [0.10  %time (days)
   0.91
   2.56
   3.65
   4.66
   5.12
   6.30
   7.59
   9.18
   9.82
   12.12
   15.31];
```

```
N=   [1.2e+000  % number of microorganisms per mm^3
  1.0e+000
  1.1e+000
  1.1e+000
  6.3e+000
  1.0e+003
  8.0e+005
  5.8e+007
  1.3e+008
  1.4e+008
  1.3e+008
  1.5e+008];
```

EXERCISE 9c (optional):

The Stanton friction factor ϕ for a smooth pipe is a function of the Reynolds number Re according to:

$\phi = 8/Re$ $\quad$ $Re \leq 2500$

$\phi = 0.0396\sqrt[4]{\mathrm{Re}}$ $2500 < \mathrm{Re} \leq 10^5$

$\phi = 2.5\ln(\mathrm{Re}\sqrt{\phi})$ $10^5 < \mathrm{Re} < 10^7$

Write a function fi=stantonxx(Re), where xx are your initials, that calculates the Stanton friction factor as a function of the Reynolds number. Note that the last equation must be solved iteratively. When Re is outside the range of the equations, NaN should be returned.

---

10. Simple forward differences.

In many types of physical problems it is possible to divide an object into a number of discrete parts whose connections are known. Typical examples are: heat capacities connected by thermal conductances, vapor capacities connected by diffusive conductances, and electrical capacities connected by electrical conductances. Writing relatively small programs in MATLAB can solve some such problems by describing the physical model and stepping through it with small time steps. One advantage with writing such small programs is that it is possible to control everything the way you want. If you use other commercial software to solve your problems you are usually more limited (it is for example difficult to change the geometry during a run or have strange boundary conditions).

Make a model in the following steps:

1. Divide the physical problem into a number of different parts.
2. Calculate the conductances between neighboring parts.
3. Calculate the capacities of each part.

This is implemented into a MATLAB program with the following general structure:

```
%----give physical parameters----

%----give starting values----

%----simulation; stepping forward in time with a time step dt----

%-----output---------
```

Note that the time step cannot be too high. If it exceeds a certain critical time step the solution will become unstable (oscillate). Therefore, test different time steps. If you reduce the time step to half its value (from an non-oscillating value) and still get the same result the time-steps are good. The reason there is a maximal possible time step is that above that time step you start moving too much heat, mass etc. around in the system. If you have two bodies at temperatures 20 and 10^{o}C there will be a heat flow from 20 to 10^{o}C. It is good in a simulation to move heat in one time step so that the temperatures changes slightly, e.g. to 19.9 and 10.2^{o}C, but it is not possible to have such a large time step that the temperatures change to 14 and 16^{o}C because than the flow direction will be reversed during the next step.

Test your simulation programs by:

- Change parameters to see that the model behaves as it should, e.g. higher conductances should lead to faster evening-out of temperature differences.
- Run a simple case that you can solve analytically. If you have a complex model this may be difficult, but you could try to reduce the simulation model, e.g. by setting some conductances to zero during testing.
- Let the program store different parameters like flow rates and plot these afterwards to see that they behave as expected.

There are many examples of physical problem types that may be solved by this method. If you can give the rules, you can make a program that simulates the problem (at least if it is a small problem). If it is a large problem or a two- or three-dimensional problem, requiring that the system be divided into many parts, than other softwares like COMSOL Multiphysics are better to use.

EXAMPLE: The file emission.m, tanks.m and tanks2.m are examples of short finite difference programs.

EXERCISE 10:

Write a finite difference program heatnn.m (where nn are your initials) that simulates the heat transfer between two bodies A and B with the heat capacities CA and CB. The thermal conductance between A and B is kAB. B is also connected to a heat sink with the constant temperature zero through a conductance kB0. At start (t=0) the temperature of both A and B are zero. There is a heat input P to A for the first half of every hour. Simulate the system for 10 hours and plot the temperature of A as a function of time.

CA	100	J/K
CB	170	J/K
kAB	1	W/K
kB0	0.03	W/K
P	21	W

---

11. Noise reduction (filtering)

It is common that measured signals are corrupted by disturbances, noise, of different kinds. Quite often part of the noise is periodic, e.g. if it originates in another instrument in a laboratory (a freezer that regularly switches on and off, daily variations in the ventilation system etc). Fourier analysis is a method to find which frequency components that are present in a signal. The MATLAB function `fft` does Fast Fourier Transform (an efficient method for doing Fourier analysis). To test `fft` we make a linearly increasing signal with random noise and two periodic components (sampling rate 1 Hz (one sample per second), frequencies of 0.06 and 0.2 Hz for the periodic components):

```
t=0:10000; %10000 data sampled at 1 Hz
y=t/1000+sin(2*pi*0.06*t)+sin(2*pi*0.2*t)+ 2*randn(size(t));
plot(t(1:100),y(1:100))
title('Noisy signal')
xlabel('time / s')
```

It is not possible from the figure to sort out the different frequency components, but with `fft` it is easy:

```
Y=fft(y,512); %Make fft with 512 points
Pyy=Y.*conj(Y)/512; %The power spectrum, a measurement of the
                    %power at various frequencies
f=(0:256)/512; %Plot the first 257 points (the other 255
%points are redundant) on a meaningful frequency axis:
plot(f,Pyy(1:257))
title('Frequency content of y')
xlabel('frequency  / Hz')
```

The two added periodic components are clearly seen as sharp peaks in the graph. Note that filtering is usually discussed in terms of frequency (measured in Hz, kHz, MHz etc.), but frequencies can easily be converted to periods (in units of seconds) as the period is the inverse of the frequency.

Some MATLAB functions, including `fft`, do not require the x-axis-vector as input. This may cause some confusion, but it is just assumed that the x-values are equidistant and the actual sampling frequency you have to take care of afterwards.

Often one wants to decrease the noise level, either by removing periodic components of by reducing the general noise level. This is usually called filtering and can be made with the filter function or (more convenient) with functions in the Signal Processing Toolbox of MATLAB.

Simpler filters can be made with only MATLAB itself by using the MATLAB-function filter:

FILTER One-dimensional digital filter.
   Y = FILTER(B,A,X) filters the data in vector X with the
   filter described by vectors A and B to create the filtered

```
    data Y.  The filter is a "Direct Form II Transposed"
    implementation of the standard difference equation:

    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

    If a(1) is not equal to 1, FILTER normalizes the filter
    coefficients by a(1).
```

The above equation may look strange, but look at the following examples:

```
b=[1 1 1 1 1]; a=[5];
```

gives a moving average filter that takes the mean of five values. An example:

```
t=1:3600;
T=randn(size(t));
plot(t,T,'k-')
b=[1 2 3 2 1];
a=sum(b);
TT=filter(b,a,T);
hold on
plot(t,TT,'r-')
hold off
```

One problem with the MATLAB function filter is that it moves the data towards the end of the vector. To take away this effect one can run the filter twice, the second time with the input vector the other way (zoom in to see the difference between the green and the red curves in this example):

```
t=1:3600;
T=randn(size(t));
plot(t,T,'k-')
b=[1 2 3 2 1];
a=sum(b);
TT=filter(b,a,T);
TTT=filter(b,a,TT(end:-1:1));
TTT=TTT(end:-1:1);
hold on
plot(t,TT,'r-')
plot(t,TTT,'g-')
hold off
```

The above filters just reduced random noise, but there are also more advanced types of filters that also use the *a*-parameter in the filter equation above (it is much more difficult to intuitively understand what parameter *a* does, than to understand what *b* does!):

low-pass filter: reduces high frequencies.
high-pass filter: reduces low frequencies
band-pass filter: reduces all frequencies except those in a certain frequency interval.
band-stop filter: reduces frequencies in a certain frequency interval.

The question is now: How do I choose a and b vectors to get other types of filters? Several functions in the MATLAB Signal Processing Toolbox can help with this. Here are three examples of simple filters made with the Yule-Walker method:

A low-pass filter of order 2 that takes away frequency components higher than 100 times the sampling rate:
```
a=[1.0000   -1.2779    0.4499];
b=[ 0.0189    0.0159    0.0189];
```

A high-pass filter of order 2 that takes away frequency components lower than 100 times the sampling rate:
```
a=[1.0000   -1.2748    0.4485];
b=[0.9860   -1.2836    0.4535];
```

A high-pass filter of order 2 that takes away frequency components lower than 50 times the sampling rate:
```
a=[1.0000   -1.2650    0.4439];
b=[0.9741   -1.2811    0.4533];
```

The order of a filter is the length of the longest vector of a and b minus 1. Here is a last example of a low-pass filter of order 10:
```
a=[1.0000
   -6.2109
   15.0346
  -15.5620
    0.0000
   15.4861
  -12.6033
   -0.0000
    5.1841
   -2.8303
    0.5017];
b=1e-4 *[0.0323
   -0.0927
    0.0481
    0.1055
   -0.1313
   -0.0042
    0.0740
   -0.0285
   -0.0098
    0.0079
   -0.0012];
```

In the exercise another good but slow method of reducing noise is described. This method can be tailored so that it performs different types of noise reduction on different parts of the curve (by letting n and ord (see below) be functions of the vector index).

EXERCISE 11:

Write a program reducenoisenn=filtxx(x,y,n,ord) that reduces the noise in *y(x)* using a quite different principle from what was described above: At every data point (*x(i),y(i)*) you make a polynomial curve-fit of order *ord* using 2*n*+1 data-points (the data point *i* plus *n* data points on each side of the data point *i*). Then calculate a new value in the data point *i* using the polynomial curve fit. All curve fits shall be made on the original data and you have to take care not to use points outside the range of the vector for the curve fits. Test the function on any noisy data, e.g. on

```
t = 0:10000; %10000 data sampled at 1 Hz
y = t/1000+sin(2*pi*0.06*t)+sin(2*pi*0.2*t)+ 2*randn(size(t));
```

from the first example.

---

12. Graphical User Interfaces (GUI:s)

Graphical User Interfaces (GUI:s) are windows in which the user can view, control etc. MATLAB programs. GUI:s are built with a number of different building blocks:

Push Buttons
Sliders
Toggle Buttons
Frames
Radio Buttons
Listboxes
Checkboxes
Popup Menus
Edit Text
Axes
Static Text
Figures

It is possible to directly program your GUI:s, but it is much simpler (still not trivial) to use MATLAB's GUIDE that helps you control the layout etc. Use GUIDE like this:

1. Start GUIDE with the command `guide` from the workspace.
2. Make the layout of buttons etc. that you want. You can change FontSize etc. by right-clicking over an object.
3. Activate the GUI with the green triangle (arrow). This will open the editor with a machine written functions that controls the GUI.
4. Make changes to the functions (this is the difficult part).

Data handling within a GUI is made with the help of a data structure called 'handles'. If you for example have a variable x that you want to save you can do so by:

```
handles.x=x;                %placing x in the handles-structure
guidata(hObject, handles); %saving handles
```

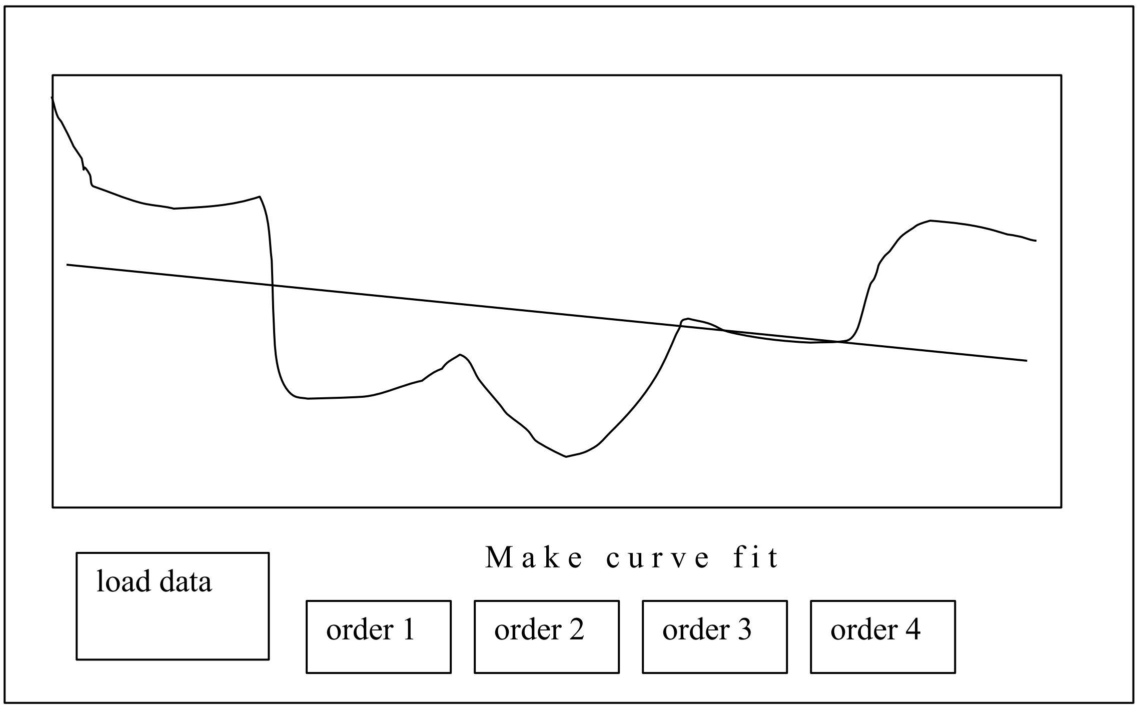
The second line is a standard line that saves the handles-strucure (should always look like this). If you have saved the data with the above commands, you can later use it, for example like:

```
plot(handles.x)
```

The different objects (axes, push-buttons etc.) have unique names, for example `axes1`, `pushbutton2`. These can be used to set and get the states of the objects, similarly to what is done with handles graphics. If you want to plot in axes1 you must first make that axes the present axes:

```
axes(handles.axes1);     %similar to the figure command
```

`curvef` is an example of a GUI that makes curve fits. Note that it is made up of two files curvef.m and curvef.fig. To use it you load data (the data must in this example be in two columns in a text file; curvef_testdata contains one set of test data) and press the curve fit buttons:



The GUI code in curvef.m is quite long with many machine generated commands and comments. Here is an abbreviated version that shows the parts where I as a user have made changes to make the GUI do curve fitting. Note the use of the MATLAB function uigetfile which lets you click a file from a menu.

```
function varargout = curvef(varargin)
.
.
.
%LOAD DATA (AND DRAW CURVE)
[filename,pathname]=uigetfile('*.*','Open two-column data file');
%DATA IS TWO-COLUMN. uigetfile is a MATLAB function
X=load([pathname,filename]);
handles.x=X(:,1)
handles.y=X(:,2)
guidata(hObject, handles)
axes(handles.axes1);hold off
plot(handles.x,handles.y,'k*');

% --- Executes on button press in pushbutton2.
```

```
function pushbutton2_Callback(hObject, eventdata, handles)
%CURVE FIT ORDER 1
k=polyfit(handles.x,handles.y,1);
hold on
xx=linspace(min(handles.x),max(handles.x),100);
plot(xx,polyval(k,xx),'r');
hold off

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
%CURVE FIT ORDER 2
k=polyfit(handles.x,handles.y,2);
hold on
xx=linspace(min(handles.x),max(handles.x),100);
plot(xx,polyval(k,xx),'b');
hold off

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
%CURVE FIT ORDER 3
k=polyfit(handles.x,handles.y,3);
hold on
xx=linspace(min(handles.x),max(handles.x),100);
plot(xx,polyval(k,xx),'g');
hold off

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
%CURVE FIT ORDER 4
k=polyfit(handles.x,handles.y,4);
hold on
xx=linspace(min(handles.x),max(handles.x),100);
plot(xx,polyval(k,xx),'m');
hold off
```

EXAMPLES: Many MATLAB Toolboxes used GUIs. Also look at MATLAB demos and Basic Fitting and Data Statistics (found under Tools in figure windows).

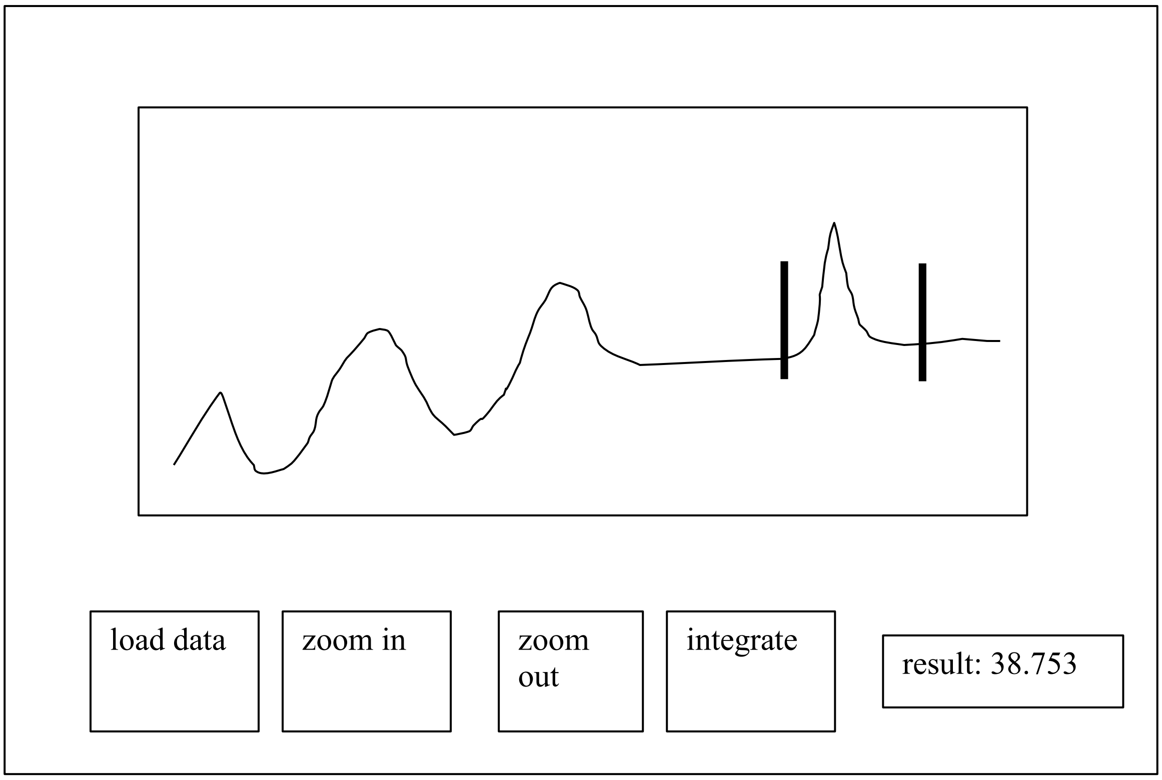
EXERCISE 12

Make a graphical user interface (GUI) that lets you integrate peaks on a curve. In the GUI you should be able to zoom in on a peak and mark the start and the end of the integration. The start and the end-points of the integration should also be shown in the GUI together with the result (the integral). The result shall be displayed in the GUI. The data that you load into the GUI shall be two-column ASCII (text), for example:

```
1   2.345
2   3.456
3   3.678
.   .
.   .
.   .
```

You shall test your GUI with the data in gcpeaks.

The GUI can, e.g., look like this:


load data
zoom in
zoom out
integrate
result: 38.753

13. Monte Carlo simulation

The following description of the Monte Carlo method is an abbreviated version of an article from Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/Monte_Carlo_method).

Monte Carlo methods are algorithms for solving various kinds of computational problems by using random numbers, as opposed to deterministic algorithms. Monte Carlo methods are extremely important in computational physics and related applied fields, and have diverse applications from esoteric quantum chromodynamics calculations to designing heat shields and aerodynamic forms. These methods have proven efficient in solving the integro-differential equations defining the radiance field, and thus these methods have been used in global illumination computations which produce photo-realistic images of virtual 3D models, with applications in video games, architecture, design, computer generated films and special effects in cinema, and other fields.

Interestingly, the Monte Carlo method does not require truly random numbers to be useful. Much of the most useful techniques use deterministic, pseudo-random sequences, making it easy to test and re-run simulations. The only quality usually necessary to make good simulations is for the pseudo-random sequence to appear "random enough" in a certain sense. That is that they must either be uniformly distributed or follow another desired distribution when a large enough number of elements of the sequence are considered.

A Monte Carlo algorithm is a numerical Monte Carlo method used to find solutions to mathematical problems (which may have many variables) that cannot easily be solved, for example, by integral calculus, or other numerical methods. Its efficiency relative to other numerical methods increases when the dimension of the problem increases.

Deterministic methods of *numerical integration* operate by taking a number of evenly spaced samples from a function. In general, this works very well for functions of one variable. However, for functions of vectors, deterministic quadrature methods can be very inefficient. To numerically integrate a function of a two-dimensional vector, equally spaced grid points over a two-dimensional surface are required. For instance a 10x10 grid requires 100 points. If the vector has 100 dimensions, the same spacing on the grid would require 10^{100} points – that's far too many to be computed. 100 dimensions is by no means unreasonable, since in many physical problems, a "dimension" is equivalent to a degree of freedom.

Monte Carlo methods provide a way out of this exponential time-increase. As long as the function in question is reasonably well-behaved, it can be estimated by randomly selecting points in 100-dimensional space, and taking some kind of average of the function values at these points. By the central limit theorem, this method will display $1/\sqrt{N}$ convergence – i.e. quadrupling the number of sampled points will halve the error, regardless of the number of dimensions.

Another powerful and very popular application for random numbers in numerical simulation is in *numerical optimisation*. These problems use functions of some often large-dimensional vector that are to be minimized. Many problems can be phrased in this way: for example a computer chess program could be seen as trying to find the optimal set of, say, 10 moves which produces the best evaluation function at the end. The traveling salesman problem is another optimisation problem: given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city once and then returns to the starting city? There are also applications to engineering design, such as multidisciplinary design optimization. Most Monte Carlo optimisation methods are based on random walks. Essentially, the program will move around a marker in multi-dimensional space, tending to move in directions which lead to a lower function, but sometimes moving against the gradient.

EXERCISE 13a: A travelling salesman problem. In a country there are 20 cities. A salesman living in city 1 needs to travel to all other 19 cities and return home to city 1 again. He wants to do this as cheaply as possible. In what order shall he then travel? Generally, there are (n-1)! possible routes between n cities; for 20 cities there are about 121645100408832000 possible ways of travelling, so not all routes can be tested... .

The function tspcost gives the cost of travelling between two cities (the costs are stored in a matrix in tspcost and are quite random. There is no relation between, e.g., distance and cost, so jit can be cheap to travel between 1 and 2, and between 2 and 3, but terribly expensive to travel from 1 to 3. Write a program that randomly selects travelling routes and look for the cheapest one. I found one trip that only costs 1144. Can you find a cheaper one? Write a program that looks for cheap trips when you are not at work. I found the 1144 route after more than 50 miljon loops (made during two nights):

```
Found route with cost=1144
Route: 1   6  17  16   2  15   9   4  11  13  18  19   3  12  20  14   5  10   7   8   1
```

Note that there are also other smarter ways of finding good solutions to the travelling salesman problem. One can for example sort all possible routes between all the cities and then only look for solutions among a cheapest fraction of the solutions. With such a smarter program one student found the following route with cost=887!

```
1  8  20  5  14  15  6  17  11  7 10  2  16  13  18  19  3  12  9  4  1
```

But for exercise 13 you only need to write a program that looks for random solutions!

EXERCISE 13b

One can calculate at least approximate probabilities of things happening in complex problems by making a large number of runs with random input data (Monto Carlo method). Calculate with MATLAB the probability that when you throw three dices the result can be sorted as three consecutive numbers, e.g. 2-3-4 or 4-5-6. Call your program dicesnn.m.

14. Examples of clever ways of using MATLAB

1. Never make files larger than about 100 lines. Divide files up into smaller files (functions or script files). This makes it easier to work with your programs. Note that functions that are used by only one function can be placed at the end of the function, e.g.

```
function d=mainfunc(a,b,c)
a=1;
b=2;
c=round(10*rand);
d=scrf(a,b,c);

function out=scrf(x,y,z)
out=x+y^2+z^3;
```

2. Try making separate small programs (functions) that do things that you may want to do again. You can put such programs in your personal toolbox. An example of a small functions that calculates the vapor pressure of water as a function of temperature:

```
function psat=T2psat(T,plotflag)%T2Dv.m
%temperature (oC) to saturation vapor pressure (Pa) of water vapor
%Reference:CRC Handbook of Chemistry and Physics, 74th edition
% Dv=T2psat(T,plotflag)
%     T is temperature (vector) (oC) RANGE: 0-50oC
%     plotflag is optional flag which set to 1 plots the ref. data
%     Lars Wadsö 20 Aug 2002

if nargin==1;plotflag=0;end
Tvect=[0:5:50];
Dvect=[611.29 872.60 1228.1 1705.6 2338.8 3169.0 4245.5 5626.7 ...
 7381.4 9589.8 12344];
k=polyfit(Tvect,Dvect,4);
if plotflag
   plot(Tvect,Dvect,'+')
   hold on
   Dfit=polyval(k,0:1:50);
   plot(0:1:50,Dfit,'-r')
   hold off
   xlabel('temperature / oC')
   ylabel('water saturation vapor pressure / Pa')
end
psat=polyval(k,T);
```

Note that I made this program with an optional plotting function, and that it accepts a scalar or a vector as input.

3. If you make programs that someone else is going to use it is a good idea to test input parameters to check that they are reasonable. Here is one example (note the use of the switch-case-construction):

```
function calco=evalcal(n,U,t,R,Rext)
% CALCO=EVALCAL(N,U,T,R,Rext) evaluates calibration coefficients (CALCO)
% from electrical calibrations. N is the measurement numbers that you want
% to evaluate, U (V) are the voltages measured over an external resistor,
% (normally 100 Ohm), T (s) are the duration of each peak, and R (Ohm)are
% the resistances of the heaters (if no resistances are given they are all
% assumed to be 100.0 Ohm). REXT is the resistance of the external resistor
% (over which the voltage U is measured), default is 100.0 Ohm.
%
%   Lars Wadsö 30 August 2002
dispresult=1;
switch nargin  %checking input data
case 0
    error('You must give at least N and U as input arguments to
EVALCAL');
    return
case 1
    error('You must also give voltage(s) U as input to EVALCAL');
    return
case 2
    error('You must give calibration times');
    return
case 3%make 100.0 Ohm vector
    if size(U)~=size(t)
        error('Vectors U and T must have the same size');
        return
    end
    R=ones(size(n))*100.0;
    Rext=100.0;
case 4
    if size(n)~=size(R)
        error('Vectors N and R must have the same size');
        return
    end
    Rext=100.0;
case 5
    %OK
otherwise
    error('Too many input arguments');
    return
end
npeaks=max(size(U));
nsamples=max(size(n));
calco=zeros([npeaks,nsamples]);
disp([int2str(npeaks),' calibration peaks to be evaluated for ' ...
,int2str(nsamples),' calorimeters']);
for k=1:npeaks
    [a,b]=blcorr(n(1),['Make baseline for peak ',int2str(k)]);
    if nsamples>1
        blcorr(n(2:end),'',a,b);
    end
    [I,c]=heats(n(1),['Integrate peak ',int2str(k)]);
    I=zeros([1 nsamples]);
    for kk=1:nsamples
        I(kk)=heats(n(kk),'',c);
    end
    calco(k,:)=(U(k)/Rext)^2*t(k)*R./I;
end
if dispresult
    for kk=1:nsamples
        disp(['-------calorimeter ',int2str(kk)])
        for k=1:npeaks
            disp(['   peak ',int2str(k),' ---> cal.coeff. = ' ...
,num2str(calco(k,kk)*1000),' mW/mV']);
        end
    end
end
disp('End of program')
```

4. Plot data to spot problems and errors. MATLAB has excellent plotting capabilities and it is often difficult to work "in the dark" without seeing the figures on a plot. If you find a strange peak, locate the peak by plotting just that data vector (e.g. `plot(y)`) and zooming in to find the strange data. On the x-axis you will then find the index of that data.

5. Sometimes data files contain corrupted data that can be fixed, e.g.

- data with some values equal to zero. Fix with
```
ind=find(y==0);
y(ind)=(y(ind-1)+y(ind+1))/2; %replace with mean value
   %(does not work for the first or the last datum).
```

- data with bad (noisy etc.) part between index i1 and i2. Fix with
```
ind=[i1-10:i1-1 i2+1:i2+10]; %OK data before and after
k=polyfit(x(ind),y(ind),2);
y(i1:i2)=polyfit(k,x(i1:i2)); %replace with fitted data
```

- data with a lot of noise. Fix with filtering.

- missing data etc. can be replaced by NaN (Not a Number) that is usually processed by MATLAB in a good way (e.g. not plotted at all)

The fixing of bad data should of course not be used to *produce* wanted data in the computer...!

6. If you have problems with MATLAB (also see lecture 7), use the internet resources at www.mathlab.com or call MATLAB help desk (if you have a licence...). You can also send questions to COMSOL in Stockholm (support@comsol.se). You can also search the www as there are many sites devoted to MATLAB.

7. When you have made an evaluation program that runs well, why not make a "master"-program that runs the evaluation program for all the files or cases you want? It make take some time to write such a program, but when you have it you can run the whole evaluation again just by running the master file. An example:

The program eval3 evaluates a file with measurements. The program master3 runs eval3 for all the files in a list, e.g.:

```
%master3
filelist=['cem1.txt  ';'cem2.txt  ';'anl1.txt
';'anl321.txt'];
s=size(filelist);
s=s(1); %number of files
for k=1:s
   filename=deblank(filelist(k,:));
   eval3(filename)
end
```

Note that the filenames are padded with spaces (they have to have the same length) which are removed by `deblank`. Note also that eval3 has been written as a function that accepts the filename as an input.

8. The previous example can be made even more advanced/convenient by letting MATLAB find all files of a specific type and then work on them. If you for example call all you measurement files meayymmdd.txt, where yy is the year, mm is the month and dd is the day, then you can for example find all files that were made later than a certain date. With the command `files=dir`; you will get all files in the present directory into the variable files, `length(files)-2` gives you the number of files (or sub-directories...) and `files(n).name` gives you the name of name n in the directory.

9. Learn how to handle strings so that you can manipulate your files, e.g.
   - find files to evaluate, e.g. files containing certain letters (previous example).
   - construct output files with evaluation results (background data, used parameters, etc).

10. Many MATLAB functions can be used in different ways depending on what input one gives them. The number of inputs to a function can be checked with nargin:

```
function a=funcx(x,y,b);
case nargin
switch 3
   %OK
switch 2
   b=2; %default
switch 0
   error('FUNCX needs at least one data');
switch 1
   y=x;
end
a=b*sum(x)*sum(y);
```

The function `funcx` can be run with 1, 2 or 3 inputs. It is also possible to let programs do quite different things depending on what input they get. The following program removes baselines from x. If there is only one input it is assumed that the user wants to see the curve and mark which part of it that is the baseline (this is done by the function `blcorrmanual`). If a second input argument is given this is the constant baseline value that should be subtracted from x.

```
function x=blcorrx(x,bl)

if nargin==1
   x=blcorrmanual(x);
elseif nargin==2
   x=x-bl;
else % 0 or >2 input arguments
   error('blcorr can only accept one or two inputs')
end
```

In a similar way `nargout` can be used to check how many output arguments that the user wants. Here is one example in which the program displays the result if the user does not want the output in a variable. If an output variable is given the output data is not shown.

```
function a=xxfunc(b)
a=round(rand*b);
if nargout==0
  disp(['result=',int2str(a)])
end
```

If this program is called with `q=xxfunc(23)` the output will only be in q (not displayed), but if it called with `xxfunc(23)` the output will be displayed instead.

11. Make many notes with % in your programs. After a few weeks you may have forgotten how you wrote your program (this is actually the most important advice given in lecture 14).

12. Make a good directory structure. Add the most important directories to the path.

13. One should not use to many computer programs. A scientist may need one document preparation program (Word and/or LaTeX), one presentation program (PowerPoint...), one reference program (EndNote...) and one program for technical calculations (MATLAB!). If you need to do more things (control measurement instruments, make databases, make drawings, manipulate pictures...) first check what MATLAB can do before buying new software that you have to learn. MATLAB can, for example, do serial communication (RS-232) and make advanced image analysis (toolbox), FEM-simulations (FEMLAB)... . Here is a list of all MATLAB products offered by MATHWORKS/COMSOL (Oct. 2002):

Aerospace Blockset
CDMA Reference Blockset
Communications Blockset
Communications Toolbox
Control System Toolbox
Curve Fitting Toolbox
Data Acquisition Toolbox
Database Toolbox
Datafeed Toolbox
Dials & Gauges Blockset
DSP Blockset
Embedded Target for Motorola MPC555
Embedded Target for TI C6000 DSP
Excel Link
Filter Design Toolbox
Financial Toolbox
Financial Derivatives Toolbox
Financial Time Series Toolbox
Fixed-Point Blockset
Fuzzy Logic Toolbox
GARCH Toolbox
Image Processing Toolbox
Instrument Control Toolbox
LMI Control Toolbox

Mapping Toolbox
MATLAB
MATLAB COM Builder
MATLAB Compiler
MATLAB Excel Builder
MATLAB Link for Code Composer Studio Development Tools
MATLAB Report Generator
MATLAB Runtime Server
MATLAB Student Version
MATLAB Web Server
MATRIXx Product Family
Model-Based Calibration Toolbox
Model Predictive Control Toolbox
μ-Analysis and Synthesis Toolbox
Neural Network Toolbox
Nonlinear Control Design Blockset
Optimization Toolbox
Partial Differential Equation Toolbox
Real-Time Windows Target
Real-Time Workshop
Real-Time Workshop Embedded Coder
Requirements Management Interface
Robust Control Toolbox
Signal Processing Toolbox
SimMechanics
SimPowerSystems
Simulink
Simulink Performance Tools
Simulink Report Generator
Spline Toolbox
Stateflow
Stateflow Coder
Statistics Toolbox
Symbolic/Extended Symbolic Math Toolbox
System Identification Toolbox
Virtual Reality Toolbox
Wavelet Toolbox
xPC Target
xPC TargetBox
xPC Target Embedded Option

Add to this all free-ware that you can get from the www and all the programs you make yourself.

14. If you think that MATLAB is expensive there is a free MATLAB-clone called SCILAB (www.scilab.org). SCILAB has about the same functionality as the standard MATLAB and toolboxes in different areas. However, be prepared that it will take some more work (at least initially) to use SCILAB than to use MATLAB:

- SCILAB has been developed in France and a large part of the documentation is in French (the main manuals are in English).

- It is not 100% compatible with MATLAB, e.g. m-files are called sci-files, comments begin with // instead of %, true and false are %T and %F (not 1 and 0) etc. This means you cannot automatically share files with MATLAB-users.
- As SCILAB is a free software you may have difficulties in getting answers to the questions you may have... .

...but if you have time, but not money, and want to break the (almost) monopoly that MATLAB has on advanced technical calculations SCILAB is probably the best choice!

END OF COURSE!