# Matlab for PhD students – Basics 1 – WS 2010/11

**Matlab windows**

The matlab desktop can be edited and designed according to your personal needs. Sizes can be changed, individual windows can be deleted and brought back with the "Desktop" menu, the windows can be docked and undocked from the main Matlab window (click on the little arrow in the top right corner).
In case you have changed the appearance of your desktop in an unwanted way and you want to get back the default: click *Desktop -> Desktop Layout -> Default*

- Command window
  - Issues commands to Matlab for processing
  - >> means that Matlab is ready to take new commands.
  - While Matlab is processing a program, the >> cursor is missing and you can read "busy" below the command window.
  - If you want to interrupt a running program, press *Ctrl – C*
  - The tab-key completes beginnings of commands or file names you type. If there are several possibilities you will get a list of possible completions.
  - The "arrow up" key in the command window brings up the last command from the command history. Typing the beginning of a command and "arrow up" jumps to the last command in the history that matches the beginning.
- Command history
  - Lists the previous commands issued in the command window
  - Double clicking of a command runs this command.
  - You can drag and drop commands into the command window to edit them.
  - In case you prefer to store all steps of your matlab session and their outputs, you can use the diary command in the command window: `diary filename` starts to copy all subsequent commands and their outputs into a text file `filename` in the current directory. `diary off` suspends it. `diary on` turns it back on.
- Workspace
  - Lists all current variables in workspace with data type, name, value, min, max (depending on the size of the data)
  - Left mouse button to mark one variable to use tool bar
  - Double click on a variable opens the array editor for this variable
  - Right mouse button: contextual menus
- Current directory window
  - Graphical user interface for directory and file manipulation in Matlab
  - Double clicking on a matlab program file (ending .m) opens the file in the editor
  - Double clicking on a matlab data file (ending .mat) loads the content of the file into the workspace
- Current directory bar
  - Shows the name of the current directory
- Array editor
  - Is opened when a variable is double clicked in the workspace window
  - Table-based editor for all kinds of variables.

    - Values can be seen and changed by typing into the editor. (CAREFUL! In contrast to data stored in files or generated with a matlab program, data that is typed in cannot be reproduced, unless you save it in a file.)
- Figures
    - Figures are usually not docked to the Matlab-window
    - If no figure is open, every plot command will open a figure window (see below)
- Help
    - Very convenient html-based help tool (see below)
    - Standard: not docked to matlab window
- Editor
    - Tool to write and edit matlab programs
    - If the file edited ends with .m the editor automatically uses highlighting shifting to the appropriate level
    - The editor is at the same time the debugger (will be discussed in detail in the second part of the course)
- Profiler
    - Tool to check the performance of your program and to get suggestions for speeding up (will be discussed in detail in the second part of the course)

**Ways to get help**

Unfortunately, I do not know a good book about Matlab that covers the newest Matlab version. But the Matlab help is really good (maybe the best reason to buy Matlab instead of using a free Software) and always up-to-date.

- Help browser

  Html based browser to search and view documentation and demonstration for Matlab. If you install toolboxes the toolbox help is automatically installed as well. The material available in the help browser can be accessed via
    - Contents: Sorted like a book to learn about Matlab
    - Index: Alphabetic list of all Matlab functions
    - Search results: Look for keywords or function names and get a list of pages on which the keywords are present.
    - Demos: Videos, m-files and GUIs for several examples of different topics (all program code is visible)

  The available material e.g. in the contents menu is grouped into different categories:
    - Getting started (green dot): Explanation of several programming concepts (at least as good as a Matlab book)
    - User guides (blue pages): More advanced text explanations for several programming concepts
    - Examples (yellow light bulb): Very helpful examples how to use Matlab syntax and several programming concepts
    - Reference pages for each function (orange pages): Syntax, description, examples, and other information for that function. Each reference page includes links to related functions and additional information.
    - Release notes (white paper): Mentions changes between Matlab versions, known and fixed bugs etc.
    - Printable documentation (pdf symbol): Most content of the Matlab help is also available in pdf format for printing

- Command window
  - `help plot` displays the help text (first comment in the program file) for the command `plot` in the command window. This works for build-in functions, toolboxes and self-made programs – if you know the correct name of the command. Particularly useful if you want to look up input or output arguments.
  - `type command` displays the full text of a program file. This does not work for some built-in functions, but many functions provided by Matlab and its toolboxes can be read. You also need to know the correct name. Particularly useful to get inspiration for your own programs or to check if a program really does what you think it does
  - `lookfor keyword` gives you a list of function names in which the keyword is included in the help text. Particularly useful to look up the right name of a command
- Internet resources
  - Mathworks home page: http://www.mathworks.com
  - Matlab newsgroup FAQ: http://matlab.wikia.com/wiki/FAQ
  - Mastering Matlab7 website http://www.eece.maine.edu/mm

**Variables**

Variables are used to store values in the workspace. Each variable has an unambiguous name, which can be used to refer to the value(s) stored in the variable.

- Variable definition:
  - Variable consist of a name and a value (or in a vector or a matrix several values). They are defined e.g. by `a=4` (always in the order `name=value`)
  - When no variable name is given for a calculation (e.g. when you just type in `8*7.7`), Matlab stores the result in the variable `ans`. This variable will be overwritten next time a calculation without output name is performed.
- Pre-defined constants

Matlab has stored values for the following mathematical constants. However, their values can be overwritten! It is good style not to use these names for your self-defined variables.

```
pi              % Ratio of circle's circumference to its
                % diameter
i               % Imaginary unit
j               % Imaginary unit
Inf             % Infinity
NaN             % Not-a-Number
intmax          % Largest value of specified integer type
intmin          % Smallest value of specified integer type
eps             % Floating-point relative accuracy
realmax         % Largest positive floating-point number
realmin         % Smallest positive floating-point number
```

- Variable names
  - Up to 63 characters
  - Consisting of letters, digits and underscores
  - Variable names have to start with letters
  - Variable names are case sensitive (`A` and `a` are different variables)

- Reserved words which cannot be used as variable names are `for, end, if, while, function, return, elseif, case, otherwise, switch, continue, else, try, catch, global, persistent, break`
- Special variables defined by matlab are `ans, beep, nargin, nargout, varargin, varargout, bitmax`, and the names of the pre-defined constants (see above).
- However, it is possible to use these names or the names of functions or scripts as variable names. The new variable definition will overwrite the old one in the current workspace! So if strange things happen the reason may be the unfortunate choice of a variable name. Use the `clear` command to get the old definition back.

- Workspace
  - The currently available variables can be seen in the workspace window.
  - As soon as a variable is defined, it is available in the workspace and can be addressed by its name (for example typing `a` in the command window returns its value `4` if the variable `a` has been defined before as `a=4` as above).
  - When a variable is present in the workspace and a variable with the same name is set to a new value, the old value will be overwritten.
  - The clear command removes variable from the workspace, freeing up system memory

```
clear x          % removes variable x from workspace
clear x y z      % removes variables x, y and z from workspace
clear            % removes all variables from workspace
```

**Some syntax basics**

- Decimal numbers
  - Since Matlab is an American program, decimal numbers are separated by . not by , (e.g. `u=8.786`)
- Separator symbols
  - Within command lines, blanks are usually ignored (you can write `a=9` or `a = 9`)
  - In vectors and matrices, rows are separated by ; columns are separated by blank or by , (e.g. `M=[1 0; 7 8]` is the same as `M=[1,0;7,8]`)
  - You can combine several commands in one line if they are separated by ; or by ,
- If a line is inconveniently long, you can continue in the next line after using … to indicate the continuation (only possible between variable names and operators, but not within names).
- Command window output
  - By default, results of all commands are written in the command window. To suppress them, a command line has to end with *;*

**Error messages**

- Errors
  - Errors stop the execution of a program at the line in which they occur.
  - Error message are written in red and produce a sound.

- Error messages often occur because of syntax errors (e.g. unbalanced brackets) and typographic errors (e.g. misspelled variable names)
- Errors often occur when the usage of a command does not match command's specification (e.g. not enough input arguments).
- The text of the error message gives you information about the source of the error. There are several different specific error messages.

- Warnings
  - Warnings inform the user about potentially unwanted effects of a command.
  - Warnings are written in black.
  - Warnings do not stop the execution of a program.

**Vectors and Matrices**

In Matlab, data is generally stored in matrices. Matlab is optimized to perform calculations on matrices in a fast and efficient way.
The following commands are also demonstrated in the script `vectors_and_matrices.m` (download from the course homepage).

- General nomenclature
  - A single number is also called a scalar.
  - A vector consists of elements (numbers), which are arranged linearly.
  - The vector can either be a row vector (numbers are written horizontally) or a column vector (numbers are written below each other).
  - The position of an element in a vector is given by its index, e.g. in vector `v=[8 0 77 5]` the element with value 0 has the index 2.
  - A two-dimensional matrix with n rows and m columns is called a n-by-m matrix
  - In a matrix, two indices specify the position of an element by giving first the row and then the column index.
  - A matrix can have more than two dimensions.
  - In a square matrix (dimensions n-by-n) the diagonal consists of all elements, in which row and column index are equal.

- Generation of vectors and matrices in Matlab
  - Matrices and vectors are generated in Matlab either by typing in the values of their elements, or by using specific functions.
  - Square brackets [ ] generally generate vectors and matrices.
  - Columns in a vector or matrix can be separated either by a space or by a comma
  - Rows in a vector or matrix are separated by a semicolon.
  - The colon operator generates vectors by specifying startvalue:step:endvalue. If no step is given, startvalue:endvalue counts from startvalue to endvalue in steps of 1.
  - Specific matrices filled with zeros, ones or random values can be generated with specific functions. These functions need two numbers as input arguments to specify the dimensions of the matrix, e.g. `z=zeros(rownumber,columnnumber)`

```
s=7                 % generates a variable s with scalar value
                    % Dimensions of s are 1-by-1
vr=[7 -9 8.8]       % generates a row-vector vr
                    % (dimensions 1-by-3).
```

```
vr2=[8,-55,9.7]     % also generates a 1-by 3 row-vector vr2
vc=[0.5; -4; 0]     % generates a 3-by-1 column vector.
M=[0 5; 0.1 7; -9 8.2]  % generates a 3-by-2 matrix
L=[]                % generates an empty variable L that can
                    % later be filled with values
C=1:7               % generates a vector with consecutive
                    % numbers from 1 to 7 (C=[1 2 3 4 5 6 7])
C2=0:3:10           % generates a vector with numbers from 0 to
                    % 10 in steps of 3 (C2=[0 3 6 9])
C3=1:-0.1:0.6       % generates a vector with numbers from 1 to
                    % 0.5 in steps of -0.1 (C3=[1 0.9 0.8 0.7 0.6])
y=linspace(37,95,17)% generates a row vector y of 17 points linearly
                    % spaced between and including 37 and 95.
                    % (Can be used alternatively to : if you know the
                    % number of elements rather than the difference
                    % between the values.)
Z=zeros(7,3)        % generates a 7-by-3 matrix with all elements 0
O=ones(1,5)         % generates a 1-by-5 vector with all elements 1
R1=rand(4,2)        % generates a 4-by-2 matrix with random
                    % elements, values are  uniformly distributed
                    % between 0 and 1
R2=randn(4,2)       % generates a 4-by-2 matrix with random
                    % elements, values are normally distributed
                    % with mean 0 and standard deviation 1
p=randperm(7)       % generates a vector with all numbers from 1 to
                    % 7 in random order.
```

- Indexing in Matlab

You need indexing to address specific elements of a vector or a matrix to either read their values (e.g. to copy them to a new variable) or to change their values in the matrix or vector.

```
el1=vr2(2)          % assigns the second element of vector vr2 to
                    % the variable el1 (for vr2=[8,-55,9.7] you get
                    % el1=-55)
vr2(1)=0            % assigns the value 0 to the first element of
                    % vector vr2 (vr2=[8,-55,9.7] changes to
                    % vr2=[0,-55,9.7])
el2=M(2,1)          % assigns the element in the second row and
                    % first column of matrix M to the variable el2
                    % (for M=[0 5; 0.1 7; -9 8.2] you get el2=0.1)
M(3,2)=66           % assigns the value 66 to the element in row 3
                    % and column 2 of Matrix M.
                    % (M=[0 5; 0.1 7; -9 8.2] changes to
                    % M=[0 5; 0.1 7; -9 66])
el3=M(4)            % Matrices can also be indexed linearly with
                    % only one index! Elements are counted
                    % column-wise, continuing after all elements of
                    % the first column with the first element of the
                    % second column
                    % (for M=[0 5; 0.1 7; -9 66] you get el3=5)
l=C(end)            % end is the position of the last element in a
                    % vector or matrix. (for C=[1 2 3 4 5 6 7] l=7)
Cpart=C3([2,5])     % Vectors can be used for indexing of several
                    % elements of a vector or matrix. (For
                    % C3=[1 0.9 0.8 0.7 0.6] you get Cpart=[0.9 0.6])
C3([2,5])=[7,9]     % Vectors used for indexing and replacing
                    % several elements. (For C3=[1 0.9 0.8 0.7 0.6]
```

```
                    % you get C3=[1 7 0.8 0.7 9])
Cshort=C(4:6)       % You can also use the : to generate a vector of
                    % indices.
C(1:3)=0            % a scalar value after the = replaces all indexed
                    % elements with this value (C=[1 2 3 4 5 6 7]
                    % becomes C=[0 0 0 4 5 6 7])
last4=M(end-3:end)  % end can also be used to generate vectors
                    % of indices. (For M=[0 5; 0.1 7; -9 66] you get
                    % last4=[-9 5 7 66])
column1=M(:,1)      % the : alone means that the entire dimension is
                    % taken. Here, all elements of the first column
                    % of matrix M are copied into the column vector
                    % column1
long_vec=M(:)       % a single : applied as index to a matrix takes
                    % all elements of M in linear order (down the
                    % columns).
                    % For the 3-by-2 matrix M the result long_vec is
                    % a column vector with 6 elements.
```

- Determine the size of vectors and matrices

It is often important to determine the size of a data set, e.g. to make sure to apply a calculation to all elements.

```
l=length(vc)        % the function length gives back the number of
                    % elements in vector vc to its output variable l
l2=length(M)        % If length is applied to a matrix, it gives back
                    % the length of the longest dimension.
r=size(M,1)         % size(M,1) determines the length of the first
                    % dimension of M, the rows, and gives it back to
                    % its output variable r
c=size(M,2)         % length of the second dimension, the columns, is
                    % written to variable c
[r2,c2]=size(M)     % If size is used without specifying the
                    % dimension, it gives back the length of both
                    % dimensions. The length of the rows is written
                    % to the first output argument r2, the length of
                    % the columns to the second c2.
```

- Concatenation of vectors and matrices

In Matlab, vectors and matrices can be combined with square brackets [ ]. However, since all rows and all columns of a matrix must have the same length, concatenation is only possible if it does not break this rule.

```
vec_long=[[0 1 2], Cshort]      % square brackets concatenate
                                % vectors to a long vector (Cshort
                                % is a row vector)
vec_long=[vec_long, 1:5, last4] % the vector vec_long is made longer
                                % by concatenating itself and two
                                % other vectors
mat1=[[0 1 2]; Cshort]          % vectors of equal length can be
                                % concatenated to a matrix.
mat2=[vec_long(1:3);vec_long(4:6)]   % you can re-combine parts of
                                % vectors and matrices.
                                %(mat2 is equal to mat1)
square=[mat2;[0.5 7 -0.5]]      % vectors and matrices can be mixed,
                                % as long as their dimensions match
square=[square(2,:);square(3,:);square(1,:)]  % re-sorts the rows
```

- Transposing vectors and matrices

Transposing turns a row-vector into a column vector and vice versa. In matrices, the indices of rows and columns are interchanged. In square matrices this means that the matrix is mirrored across the diagonal.

```
C_trans=C'              % transposing with ' makes a column vector
                        % out of a row vector
C_again=C_trans'        % ... and the other way round
square_trans=square'    % transpose a square matrix
mat1=mat1'              % replace e.g. a 2-by-3 matrix with a 3-by-2
                        % matrix
```

- Reshaping

  Reshaping – changing the dimensions of a matrix and thereby the order of the matrix elements – can be done either “by hand” by using indexing and concatenating the matrix elements in the right order (sometimes also transposing is needed) or by using the reshape command.

```
vec2=mat2(:)                % reshapes the 2-by-3 matrix mat2 to a
                            % column vector
vec2a=reshape(mat2,6,1)     % produces the same column vector with
                            % reshape (the two numbers as second and
                            % third input arguments of reshape
                            % specify the number of rows and columns
                            % of the new matrix that the elements of
                            % mat2 shall be rearranged in. Their
                            % product must equal the number of
                            % elements of mat2)
M_new=reshape(mat2,3,2)     % reshapes the 2-by-3 matrix mat2 into a
                            % 3-by-2 matrix M_new using linear
                            % indexing (picking the elements down the
                            % columns)
M_newa=[mat2(1:2);mat2(3:4);mat2(5:6)] % also generates a 3-by-2
                                       % matrix, but elements are in
                                       % a different order
```

**Mathematical operations**

- General rules
  - Mathematical operations +, -, *, / are performed in Matlab according to the usual mathematical rules.
  - The order of operations is given by the mathematical rules (* and / before - and +). If another order is desired, parentheses ( ) are used to control the order of operations.
  - Operations can be performed to
    - Two scalar values
    - A scalar and a matrix (or a vector)
    - Two matrices (or vectors) with matching dimensions
  - The use of mathematical operations in Matlab is demonstrated in the script `matrix_mathematics.m`

- Operations combining scalars and vectors or matrices
  - In Matlab, scalars can be multiplied, divided, added and subtracted to and from scalars, vectors and matrices. The operations are performed for each of the vector or matrix elements.
  - Like usual, a*b is equal to b*a and a+b equal to b+a. (But for – and / the order is important.)
  - The notation for power is x^y

```
S1=4-7
V1=[1 3 0.5]+S1
Sevens=7*ones(7,7)
P1=sevens^7
P2=2^V1
```

- Element-by-element vector and matrix operations
  - Mathematical operations can be applied to vectors and matrices if their dimensions match.
  - The resulting matrix (or vector) has the same dimensions.
  - The operations are performed for each pair of corresponding elements.
  - For multiplication, division and power the element-by-element calculation is obtained by specific notation: a.*b; a./b; a.^b
  - For element-by-element operations the same rules of commutativity apply as above.

```
V2=[7 9 0]+[10:12]
M1=sevens.*randn(7,7)
P3=[5 7].^[2 -1]
```

- Matrix multiplication
  - The mathematical definition of matrix multiplication is not the element-by-element multiplication!
  - C=A*B is defined by $C(i,j)=\Sigma^{n}_{k=1} A(i,k)*B(k,j)$
  - Therefore, the number of rows in B and of columns in A must be the same.
  - For a l-by-n matrix A and a n-by-m matrix B the resulting matrix C has the dimensions l-by-m.
  - Matrix multiplication is not commutative! A*B and B*A are different!
  - The same rules apply to division.
  - All of the following results are different:

```
A=[5 1]*[0; 2]
B=[0; 2]*[5 1]
C=[5; 1]*[0 2]
```

**Some build-in mathematical functions**

- Functions with one input argument and one output argument

All of these functions are applied element-wise to their input arguments, which can be scalar values, vectors or matrices. The values of the other elements do not influence the results of the calculation for a given input element.

```
d=sqrt(9)            % square root
s=sin(2*pi);         % sine
c=cos(v);            % cosine
```

```
l=log(m)              % natural logarithm
l2=log2(m)            % base 2 logarithm
l10=log10(m)          % base 10 logarithm
ex=exp(m)             % exponential
r=round(7.98)         % round to nearest integer
c=ceil([6.9 9.7])     % round to next larger integer
f=floor(M)            % round to next smaller integer
```

- Functions with two or more input arguments and one output argument

These functions can be applied to scalar values, vectors or matrices. The operations are applied to corresponding values of both inputs. Therefore, the input arguments must have the same size. The mathematical operations listed here specifically require integer inputs.

```
r=rem(104,10)         % remainder after integer division
                      % ("Rest") of corresponding elements of
                      % both input arguments
g=gcd(v,w)            % greatest common divisor of
                      % corresponding elements of v and w.
                      % v and w must contain integer values
```

- Functions which can have one or more input arguments

All of the following functions can be applied either with one or more input arguments. The first input argument is always the data set. If this data set is a vector, it is save to use the version with this only one input argument. The mathematical operation will be applied to the entire vector. If your data set is stored in a matrix it is safer to give the dimension to which the operation should be applied as second input argument. `dim=1` means calculation along the columns, the resulting vector is a row-vector. `dim=2` calculates along the rows, resulting in a column vector.

```
s1=sum(v)             % for vector v: sum all elements of v.
                      % for matrix v: sum elements along the
                      % columns. For a m-by-n matrix v, s1 will
                      % be a row vector with n elements.
s2=sum(M,dim)         % for dim=1: s2 is a row-vector with
                      % summed elements of each column.
sall=sum(M(:))        % sum all elements of M
min1=min(v)           % minimum value, usage like for sum
min2=min(M,dim)
max1=max(v)           % maximum value, usage like for sum
max2=max(M,dim)
m1=mean(v)            % mean, usage like for sum
m2=mean(M,dim)
med1=median(v)        % median, usage like for sum
med2=median(M,dim)
sd1=std(v)            % standard deviation (sum(v)/length(v))
sd1a=std(v,1)         % the second input argument indicates two
                      % options to calculate the std:
                      % 0: sum(v)/length(v) (default)
                      % 1: sum(v)/(length(v)-1)
sd2=std(M,0,dim)      % Since the second input argument is used
                      % for the two options, the dimension is
                      % given by the third argument.
v1=var(v)             % variance, usage like for std
v2=var(M,0,dim)
```

**Saving and loading Matlab data**

- Saving data
  - In Matlab, data is saved as variables with name and content in a file
  - Matlab data files have the ending .mat
  - Commands:

```
save myfile.mat       % saves all variables of the current
                      % workspace to a file called myfile.mat
save myfile           % saves all variables of the current
                      % workspace to a file called myfile.mat
                      % Matlab automatically generates the
                      % ending .mat for a data file
save afile a          % saves only variable a to file afile.mat
save bfile a b c      % saves variables a, b, c to bfile.mat
                      % all other variables contained in the
                      % workspace will not be saved.
```

- Loading data
  - When you load a Matlab data file with ending .mat the variables saved to this file will be included into the current workspace.
  - If a saved variable has the same name as a variable that is already contained in the workspace, the value of this variable will be overwritten with the saved value without warning!
  - Commands:

```
load myfile.mat       % loads all variables saved in the file
                      % called myfile.mat to the current
                      % workspace
load myfile           % loads all variables saved in the file
                      % called myfile.mat to the current
                      % workspace. Matlab automatically adds
                      % the ending .mat
load afile a          % loads only variable a from file
                      % afile.mat into the workspace
save bfile a b c      % loads variables a, b, c from bfile.mat
                      % into the workspace.
```

- Changing directories
  - To change the current directory, you can interactively use the current directory window or bar
  - Or you can use the following commands:

```
cd subdirname         % makes the sub-directory subdirname the
                      % current directory
cd ..                 % makes the next higher directory the
                      % current directory
load ../dir2/file27.mat % loads file27.mat from sub-directory
                        % dir2 of the next higher directory
```

**Basic graphics:**

Plotting data is very simple in Matlab, you just have to click on the variable name in the workspace window and choose “plot” from the context menu. However, for automation and reproducibility of figures you should use command-based plotting. Matlab automatically scales the axes to minimum and maximum values.
Please refer to the Matlab help for graphic options.
We will discuss more graphics options in January in the course.

- Graphical display of data

```
plot(v)               % line-graph, plotting the values in
```

```
                        % vector v against their index.
plot(M)                 % When plot is applied to a Matrix, each
                        % column is plotted against its index.
                        % All lines are plotted in one graph.
plot(x,y)               % plots vector y against vector x. x and
                        % y must have the same length.
plot(x,M)               % plots each column of M against vector
                        % x in one graph. The length of x must be
                        % equal to the number of rows in M.
plot(x1,y1,x2,y2)       % plots y1 against x1 and y2 against x2 in
                        % one graph. Pairs x1 / y1 and x2 / y2
                        % must have the same length, x1 and x2
                        % can have different lengths.
plot(x,y,'ro:')         % plotting style can be determined by
                        % combinations of symbols. In this case a
                        % red dotted line with data points marked
                        % by circles will be used.
imagesc(M)              % shows the values contained in a two-
                        % dimensional matrix color coded in a
                        % two-dimensional figure.
```

- Figure Windows

```
figure                  % opens a new figure window. The windows
                        % are labeled with consecutive numbers.
figure(3)               % opens a figure window with number 3.
close(2)                % closes the figure window number 2.
close all               % closes all figure windows.
clf                     % clears the current figure window. (The
                        % figure window that was opened or
                        % clicked on last). The window stays
                        % open, only the graph is erased.
subplot(m,n,p)          % opens a figure window with m*n sub-
                        % figures, arranged in m rows and n
                        % columns, and makes sub-figure p active.
                        % The next plot command will plot into
                        % the active sub-figure.
                        % sub-figures are counted along the rows,
                        % in contrast to linear matrix
                        % indexing(!) E.g. subplot(3,2,3) will
                        open a figure window with 6 sub-figures
                        arranged in 3 rows and 2 columns and make
                        the left figure in the second row active.
```

- Text in Graphics

```
title('my first plot')      % title of figure window
legend('data1', 'data 2')   % legend for each of two lines
                            % (or the first two plotted lines if
                            % the current figure contains more),
                            % displayed in a box
xlabel('time')              % labels the x-axis
ylabel('mV')                % labels the y-axis
```

- Interactive editing of graphics

  Graphics can be edited interactively. Try to double-click on the axes or the graph and you will get a menu to optimize the appearance of your graphics.

Try to zoom into the figure and to get back to the original figure by double click on the curve. Later in the course I will show you to achieve the same effects reproducibly with commands.

- Saving and loading graphics
  - You can save Matlab graphics by using >File > Save as in the menu.
  - Matlab saves figure files with the ending .fig in a format that can only be read and edited by Matlab. (Even though figures from other programs are often also called .fig)
  - Matlab figures can be opened by clicking >File >Open in the menu or by double clicking a .fig file name in the current directory window.
  - Saved Matlab figures contain all the data used to generate the original figure, not only the visible part of it. Try to create a plot, zoom into this plot and save the figure. Then close the figure window, clear the variable from which you created the plot and load the Matlab figure. Try to zoom out.
  - If you want to use Matlab graphics with other programs, you need to save them in a different format. This is also done with >File > Save, but you need to choose a different format and change the ending of your save file accordingly. Try out to save a Matlab figure and import it into word.

**Scripts:**

- Definition
  - Matlab scripts are programs. They usually consist of several commands that are processed sequentially by Matlab.
  - When Matlab processes a script, the processing and all results are the same as if the commands were typed into the command window.
  - The only difference is that you need to type only the name of the program instead all of the commands.
  - And a second advantage is that you can read your script later and recall how you generated your results
- Name
  - The program is saved under a name that ends with .m
  - For the choice of program names the same rules apply as for the choice of variable names.
- Comments
  - When a % is included into a matlab program, the rest of the line will be considered as a comment and not evaluated by Matlab (and appear green in the Matlab editor)
  - To write a block of comments you can use `%{ comment text %}` instead of starting each separate line with `%`.
  - Comments in the beginning of a Matlab file (before the first command) are used as help text. Please note that the block of commented lines used as help text ends if a line is not marked as comment.
  - It is good style to write a help text for each Matlab program, explaining its function, inputs and outputs!

- You can structure your Matlab programs in so called “cells” by using comment lines starting with `%% name of the next block` to define a new block of commands.

**Homework**

**1. Getting familiar with the Matlab environment**

a. Locate all Matlab windows described above on your computer. Open help window, editor etc.
b. Define a variable `z=7` and look at the reactions in the different Matlab windows. Define a second variable with `x=10;` Which of the reactions are different?
c. Calculate `y=x+z` and look at the windows. Repeat the calculation with just typing `x+z` What happens?
d. Define two matrices: `M1=[1 1 1; 5 6 7]` and `M2=[0 -1; -2 0.5; 0.1 1]` Copy the value of the element in the second row and first column of `M1` into a new variable. Which index pair belong to the value `0.1` in `M2`? Which linear index belongs to this value?
e. Generate a vector `s` that runs from -10 to 10 in steps of 0.1. Look at it by plotting it.
f. Determine the size of `s`.
g. Take a look at the context menu of the variables listed in the workspace window.
h. Generate a vector that applies the square root to each element of `s`
i. Separate the first 10 elements of `s` into a new variable `f` and the last 5 elements into a variable `l`.
j. Define some vectors of your choice, e.g. `v1=[1 2 7 0.5]`, `v2=randn(1,6)` and some matrices, e.g. `m1=[0 1; 1 0]`, `m2= zeros(4,6)`. Look at these variables with the array editor.
k. Apply some mathematical operations and some of the functions listed above to your variables. Which of them work for vectors and which for matrices? For which of them is the order of application important (e.g. is `M*N` equal to `N*M` if `N` and `M` are scalar values / vectors / Matrices?)
l. What happens if you transpose your variables and apply the operations again?
m. Save all variables of your workspace to a Matlab data file. Clear all variables and load the file. Make sure that all variables are back again. Also try to save only one variable, clear and load it back in.
n. Find some operations that produce `Inf` or `NaN` as output.
o. Try to make some errors. Which error messages do you get? When do you get warnings?

**2. Scripts**

a. Generate a directory for your Matlab programs and change into this directory for working.
b. Write a script that simulates tossing 5 dices. (Toss the dices only once without anything like putting dices back or having unfair dices – but if you are eager to program Kniffel, you will have learned everything you need to know by day 3.)
c. Write a script that generates a 2-by-3 matrix and reshape this matrix into a 3-by-2 matrix first by using the reshape command and than by using a

series of indexing and concatenation commands to generate the same matrix.

d. Write a script that generates a matrix representing a 8-by-8 chessboard with alternating values of 0 and 1. Display this chessboard graphically.
e. Save the matrix “picturematrix.mat” from the course homepage to your computer. Write a script to load the matrix into the workspace and display it with `imagesc` graphically. Copy only the part of the matrix containing the house to a new variable and display it graphically in a new graphics window. Create a third matrix containing only one of the stars and display it in a new window. (Why does the star change its color? Do you have an idea how to fix that?)
f. Save the data set “demo_data.mat” from the course homepage to your computer. Write a script that loads the data, and generates a new plot for each row of the matrix.
g. Include into your script of 2f the calculation of mean and standard deviation of the demo_data in each row of the matrix. (For which rows does this make sense?)
h. Write a script that generates and plots a sine wave running in 500 steps from 0 to 5 π.
i. Modify your script by generating an additional data vector, in which you add white noise to the sine wave. Plot the noisy sine wave together with the perfect sine wave into one plot. (If you use much smaller noise amplitudes than the amplitude of the sine wave, your plot will resemble an intracellular recording ;-)