9.1 Exercise 1

Since in MATLAB variables are always interpreted as matrices we will start with defining row and column vectors. Type in the command window

```
x = [1 2 3]
```

Pressing the RETURN-key MATLAB writes the result below the input line in the command window (here a row vector).

```
x = x'
```

results in the transposed vector (now `x` is a column vector).

```
y = x * x
```

leads to an error message because the multiplication sign '*' is interpreted as a matrix multiplication for which the first and second parameter must have corresponding numbers of elements in row and column.

```
y = x' * x
```

is allowed and calculates the scalar product (i.e. a number) of the two vectors. Changing the order is allowed as well and calculates the dyadic product (i.e. a matrix)

```
y = x * x'
```

Often the operation should be applied to each element which needs for the dot-operation command (i.e. for multiplication '.*')

```
y = x .* x
```

calculates the square of each element of the vector x.
You want to suppress the output of the result! Just add a semicolon at the end of a command line!

```
y = x .* x;
```

Easily we now can plot a graph

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

Obviously this is parabolic curve, just more data points are necessary to get a nice graph

```
x = 0:0.1:10;
y = x .* x;
plot(x, y)
```

Here the first command line is important for our lecture; it generates an array with floating point numbers between 0 and 10 with distance 0.1. How many points are in this array?

```
l = length(x);
```

gives the answer.
NOTE: `length(x)` is not returning the largest value (here 10.0), or the distance between the largest and smallest values in this array, but an integer counting the number of elements. Contrary to e.g. *C* in MATLAB arrays typically start with index 1 (so `x(1)` is 0) and end with the index `length(x)` (so `x(length(x))` is 10.0).
Most mathematical functions are defined for real numbers. In most computer languages such numbers are represented by floating numbers which consist of a sign bit, a mantissa and an exponent. The accuracy is defined by the number of (significant) digits of the mantissa. On example will show some consequences of the limited accuracy of floating number representation:

```
x = log(7);
y = exp(-x);
z = 7.*y-1;
```

The analytical result for `z` is `0`. But you will find a slight deviation which is a consequence of the truncation of real numbers to a limited number of digits and of errors occurring from the numerical calculation of functions like `log` and `exp`.
We will now program our first function. A function is nothing but a set of commands placed between `function - end`. The commands are stored in a text-file which for MATLAB programs has the extension '.m'. To change a text-file an editor is used. Only the pure text (no color, no highlighting, no text style) has a meaning for the program. The correct syntax for a MATLAB function you ensure by calling 'NEW/function'. The editor opens with a screen showing

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

end
```

The meaning of `input_args` and `output_args` variable we will discuss soon. For the moment we just erase both parameters, rename our function to `my_first_plot`.
For our computer lab course always store functions in the standard directory and never use capital characters in the string for function name; this may lead to problems with the WINDOWS operating system because no strict distinction between small and capital characters exists.
Copy the corresponding lines from the command window into your program

```
function my_first_plot
    x = 0:0.1:10;
    y = x .* x;
    plot(x, y);
 end
```

Save the function. From the command window you can now plot a parabolic curve just by calling in the command window

```
my_first_plot;
```

Now we will for the first time take benefit of the extremely well organized help-files of MATLAB. Use it to check for parameter options for the `plot` command. Draw a dotted line with red color. You will find several solutions; most easily it is done using `plot(x,y,'r.')`.
In order to be more than just a list of commands which are executed by calling a function name we now take benefit of the variables as input and output of functions.
NOTE: Variables are containers!!!!
You can store a lot of things in a container, e.g. containers, numbers, characters,...
The standard container any function provides to store data needed for operating a function is `input`. In our first example we will use this variable to provide values for the interval limits for plotting the graph

```
function my_first_plot(x_min, x_max)
    x = x_min:0.1:x_max;
    y = x .* x;
    plot(x, y);
 end
```

To call this function we now have to add two parameters

```
my_first_plot(-10,10);
```

This will plot the the parabolic curve between -10.0 and +10.0.
Much more flexibility we gain by using the function to be drawn as a variable. This needs for a quick introduction. The `@` - command in nearly all computer languages means 'storage position of'; so `@sin` means storage position of the sin function (basically where the list of commands is stored). `sin(x)` is no function!!! It is the sin-function value of `x` (provided the variable `x` has been defined). Thus `@sin(x)` is the position where the result of `sin(x)` is stored! Don't mix this up!
The change in the function is trivial

```
function my_first_plot(fu, x_min, x_max)
    x = x_min:0.1:x_max;
    y = fu(x);
    plot(x, y);
 end
```

To call this function we now have to write e.g.

```
my_first_plot(@sin,-pi,+pi);
```

We have written already a powerful program which allows to plot nearly all 1D functions!
NOTE: Variables like `pi` (not the greek π because this is no allowed character in a text editor!) are of course defined in MATLAB.
The `output_args` parameter we well start to discuss next lecture but we already used it implicitly when calling the `sin` function. The result of `sin(x)` is stored in the `output_args` variable. The command

```
y = sin(x);
```

stores `output_args` in the variable `y` which we can use to access the result of `sin(x)`.
In our plotting example the variable `x` is a vector, so `y=sin(x)` results in a vector; this is the only restriction we have to bear in mind using in our example functions as input parameter: they have to allow for vectors as input parameter.
In this lecture we will do numerical analysis of data generated in a lab course all of you will participate in. The weight of a membrane is measured automatically by a micro balance as a function of time and the data is stored in a text file. Such data acquisition is standard in materials science. For the moment we will just open this text file and plot the relevant curve.

```
function tga_load_plot
    fid = fopen('m202_thin.txt');
    A = textscan(fid,'%f %f %f %f %f %f','delimiter','space','headerlines',31);
    %31 headerlines are typical
    M_infinite = (A{2}(1)-A{2}(end));
    sqrt_t = A{1}.^(0.5);
    y = (A{2}(1)-A{2})./M_infinite;
    plot(sqrt_t,y)
end
```

Here `fopen('m202_thin.txt')` opens the text file; `output_args` is a file id for later access. `textscan(fid,'%f %f %f %f %f %f','delimiter','space','headerlines',31);` reads the data of the file, ignores 31 header lines and stores the data in the matrix `A`. Column 1 contains the time data, column 2 the weight data. Details of the physics and mathematics of this lab course will be discussed in a separate exercise 9.8. Here just discuss the necessary transformation of both vectors before plotting!

- The `tga_load_plot` function file you can download here.
- The text file you can download here.

The last example we will discuss in this lecture is plotting of 2D functions. Most easily you create the following example by checking in the MATLAB-help for `ndgrid` and/or `mesh`, and copy the corresponding lines from the example.

```
function myquickplot
    [X1,X2] = ndgrid(-3:.1:3, -3:.1:3);
    Z = X1 .* exp(-X1.^2 - X2.^2);
    mesh(X1,X2,Z);
end
```

Execute the function and discuss the graph. This is an important example because in the math lecture and e.g. in thermodynamics the Gauss bell shape function will show up quite often.
Your jobs:

1. Change the above function to generate 4 extrema (2 minima and 2 maxima, across the diagonal).

2. Change the above function to generate 4 maxima.
3. Change the above function to generate 4 extrema (2 minima and 2 maxima, but now anti-symmetric to the x-axis and symmetric to the y-axis).

HOMEWORK 1

Write a function

1. With input k_1 and k_2, no output.
2. Create an x-axis starting at 0 ending at abs$(20\,\pi/(k_2 - k_1))$.
3. Plot $\sin(k_1\,x)$, $\cos(k_2\,x)$, and the sum $\sin(k_1\,x) + \cos(k_2\,x)$ in one graph.
4. Discuss your result with respect to beats (German: Schwebung).

You must be able to discuss the details of your function!

HOMEWORK 2

Write a function

1. With input a, no output.
2. Plot in one graph $\cos(x)$ and $a\,x$.
3. Discuss the intersection points between the two curves with respect to the solutions of the equation

$$\cos(x) = a\,x \tag{9.1}$$