Matlab lecture 4

Table of Contents

Version 0.1 by Markus Schellenberg, Winter Term 16/17

Clean up

```
clc, clear, close all
```

Functions (recap)

All variables that are defined or used in a function are local variables. They exist during the function runtime but are deleted from the workspace, when the function is done. This is different while using a skript. Variables that are used in the function calling script do not interfere with the variables in a function, even if they have the same name. The transfer of arguments (input and output) is done by 'call by value', that means, that only a copy is transfered. The variables inside a Matlab script are invisible for the function. The whole communication between the calling code/script/function and the called function is done via the input and output arguments.

Function with no input and no output arguments

Create function plot_sin which is a function with no input and no output arguments. Create follwing function. Note, that you do not any brackets or equal lsigns.

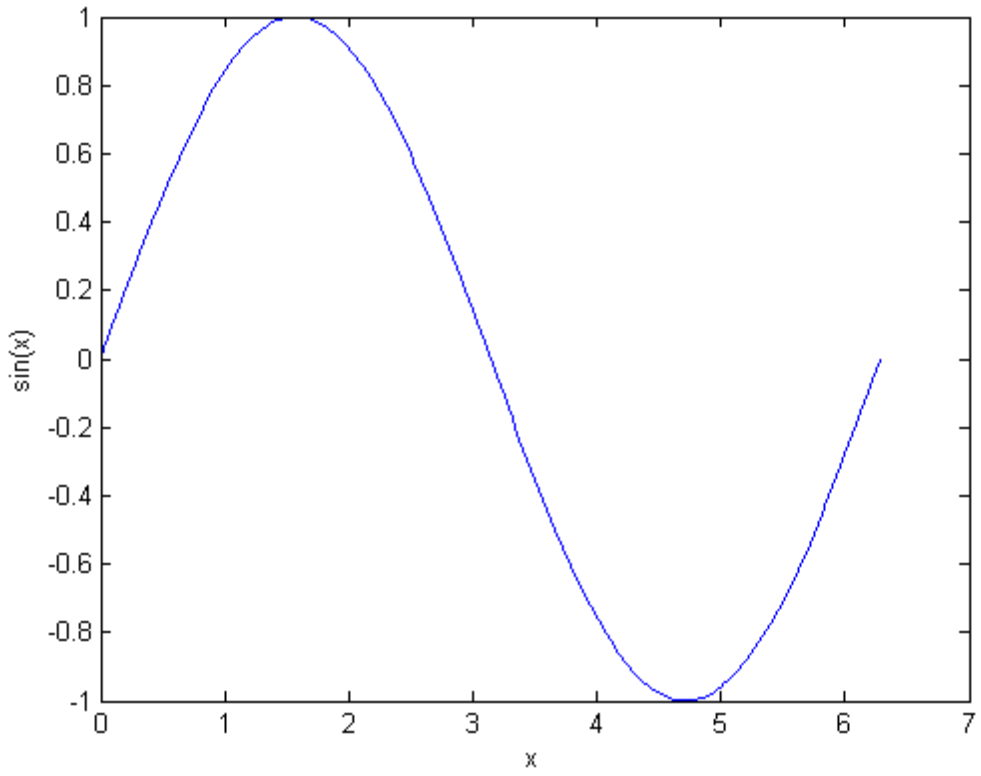
```
function plot_sin
%PLOT_SIN Summary of this function goes here
% Detailed explanation goes here
% It will be displayed when you use the help command on this function

x = linspace(0,2*pi,200);   % define the input
y = sin(x);                 % calculates the sine of the input

figure                      % creates an empty figure
```

```
plot(x,y)                    % Plots the curve inside the created figure

% Set Labels
xlabel('x')
ylabel('sin(x)')
```

Use the help command to look at the header of your function and execute the function by typing in the function name:

```
help plot_sin
plot_sin
```

```
    PLOT_SIN2 Summary of this function goes here

       Detailed explanation goes here
       It will be displayed when you use the help command on this function
```



Function with one input and one output argument

Create function f_my_function which is a function with one input and one output arguments. Create follwing function. Note, that you only need a pair of round backets on the input side.

```
function y = f(x)
%F_MY_FUNCTION does a simple calculation and gives back the result
```

```
% NOTE: even if the filename is called f_my_function the internal name
% could be something completely different (e.g. f(x)).
%
% This function does the follwing:
%
% It calculates:
%%
%   $$x^2 sin(x)$$

y = x.^2.*sin(x);

end
```

Test your function by typing e.g.:

```
result = f_my_function(5)
```

```
result =

  -23.9731
```

Function with one input and several output arguments

Create function f_sin_cos which is a function with one input and two output arguments. Create follwing function. Note that you put the ouput variables in rectangular brackets and the input variables in round brackets

```
function [ out_sin, out_cos ] = f_sin_cos(input)
%F_SIN_COS Calculates the sin and the cos of a number
% my name, date

out_sin = sin(input);
out_cos = cos(input);

end
```

And now test the function by typing in:

```
[result_sin, result_cos] = f_sin_cos(pi)
```

```
result_sin =

   1.2246e-16


result_cos =

    -1
```

Relational Operators and logical Operators

Relational and logical operators in Matlab are defined as follows

Matlab Syntax ↔ mathematical Syntax

$A > B \quad \leftrightarrow \quad A > B$

$A < B \quad \leftrightarrow \quad A < B$

$A >= B \quad \leftrightarrow \quad A \geq B$

$A <= B \quad \leftrightarrow \quad A \leq B$

$A == B \quad \leftrightarrow \quad A = B$

$A \sim= B \quad \leftrightarrow \quad A \neq B$

$\&\& \quad \leftrightarrow \quad \text{and}$

$\| \quad \leftrightarrow \quad \text{or}$

$\sim \quad \leftrightarrow \quad \text{not}$

if...elseif...elseif...else...end

```
if <expression 1>
% Executes when the boolean expression 1 is true
<statement>

elseif <expression 2>
% Executes when the boolean expression 2 is true
<statement>

elseif <expression 3>
% Executes when the boolean expression 3 is true
<statement>

else
% executes when the none of the above conditions is true
<statement>
end
```

Examples for if-loops

source (https://www.tutorialspoint.com/matlab/if_elseif_else_statement.htm)

```
a = 20;

% check the boolean condition
```

```
if a == 10
   % if condition is true then print the following
   fprintf('Value of a is 10\n' );
elseif a == 20
   % if else if condition is true
   fprintf('Value of a is 20\n' );
elseif a == 30
   % if else if condition is true
   fprintf('Value of a is 30\n' );
else
   % if none of the conditions is true
   fprintf('None of the values are matching\n');
   fprintf('Exact value of a is: %f\n', a );
end
```

```
Value of a is 20
```

The logical operators always gives you back a value for TRUE (1) or FALSE (0). Try out the follwing:

Create a logical statement

```
statement = true;
```

If you type in `statement` or `~statement` you will find it the condition is TRUE (1) or FALSE (0)

```
statement
~statement
```

```
statement =

     1


ans =

     0
```

check what kind of variable `statement` is

```
whos statement
```

```
  Name           Size              Bytes  Class     Attributes

  statement      1x1                   1  logical
```

While loops

```
while <expression>
    <statement>
end
```

The wile loop will run until the logical expression is FALSE. Sometimes it is called a repeat-until loop.

```
i = 1;
value = 1;
while value < 10
    fprintf('That is loop number: %d and the value is: %d\n', i, value)
    i = i+1;
    value = value + 1 + rand;
end
```

```
That is loop number: 1 and the value is: 1
That is loop number: 2 and the value is: 2.486792e+00
That is loop number: 3 and the value is: 3.922650e+00
That is loop number: 4 and the value is: 5.369434e+00
That is loop number: 5 and the value is: 6.675783e+00
That is loop number: 6 and the value is: 8.184292e+00
That is loop number: 7 and the value is: 9.695064e+00
```

Stopping a loop by using the break command

You can abort a while- or for-loop by using the `break` command. The command `continue` just stops the current pass and let the loop continue with the next pass. Please look it up (`doc continue`).

```
i = 1;
value = 1;
while value < 10
    fprintf('That is loop number: %d and the value is: %d\n', i, value)
    i = i+1;
    value = value + 1 + rand;
    if i > 7
        fprintf('While loop stopped at loop number %d\n', i)
        break
    end
end
```

```
That is loop number: 1 and the value is: 1
That is loop number: 2 and the value is: 2.794831e+00
That is loop number: 3 and the value is: 4.439150e+00
That is loop number: 4 and the value is: 5.817759e+00
That is loop number: 5 and the value is: 7.629339e+00
That is loop number: 6 and the value is: 9.162165e+00
```

switch, case, otherwise

```
switch variable
    case value_1
        statement_1
    case value_2
        statement_2
    ...
end

switch variable
    case {value_1a, value_1b},
        statement 1
```

```
    ...
    case value_n
        statement_n
otherwise
        statement
end
```

The variable is either a scalar variable or a string variable. The switch/case scenario is a simplification of the if/else structure. Here is an example (taken from: https://www.tutorialspoint.com/matlab/switch_statement_matlab.htm):

```
grade = 'B';

   switch(grade)
   case 'A'
      fprintf('Excellent!\n' );
   case 'B'
      fprintf('Well done\n' );
   case 'C'
      fprintf('OK\n' );
   case 'D'
      fprintf('You passed\n' );
   case 'F'
      fprintf('Better try again\n' );
   otherwise
      fprintf('Invalid grade\n' );
   end
```

```
Well done
```

Logical Operators: Short-Circuit && and l l

expr1 && expr2 --> means expr1 AND expr2 must be TRUE

expr1 ll expr2 --> means expr1 OR expr2 must be TRUE

Note: you can type in the OR symbol (l l) by combining **AltGr** + <

expr1 && expr2 represents a logical AND operation that employs short-circuiting behavior. expr2 is not evaluated if expr1 is logical 0 (FALSE). Each expression must evaluate to a scalar logical result.

expr1 l l expr2 represents a logical OR operation that employs short-circuiting behavior. expr2 is not evaluated if expr1 is logical 1 (TRUE). Each expression must evaluate to a scalar logical result.

```
% Create two vectors
X = [1 0 0 1 1];
Y = [0 0 0 0 0];
```

The short-circuit operators operate only with scalar logical conditions. Use the `any` or `all` functions to reduce each vector to a single logical condition. The `any` command gives you a logical expression: 1 = TRUE and 0 = FALSE.

```
any(X)
any(Y)
any(X) || any(Y)
```

```
any(X) && any(Y)
```

```
ans =

     1


ans =

     0


ans =

     1


ans =

     0
```

More examples with logical conditions

```
% Variables: please change
b = 2;
a = 20;

% logical expression: x = TRUE if b is not zero and a/b is bigger or equal
% than 10
x = (b ~= 0) && (a/b >= 10)

switch x
    case true
        fprintf('condition fullfilled\n')
    case false
        fprintf('condition not fullfilled\n')
end
```

```
x =

     1

condition fullfilled
```

Published with MATLAB® R2013a