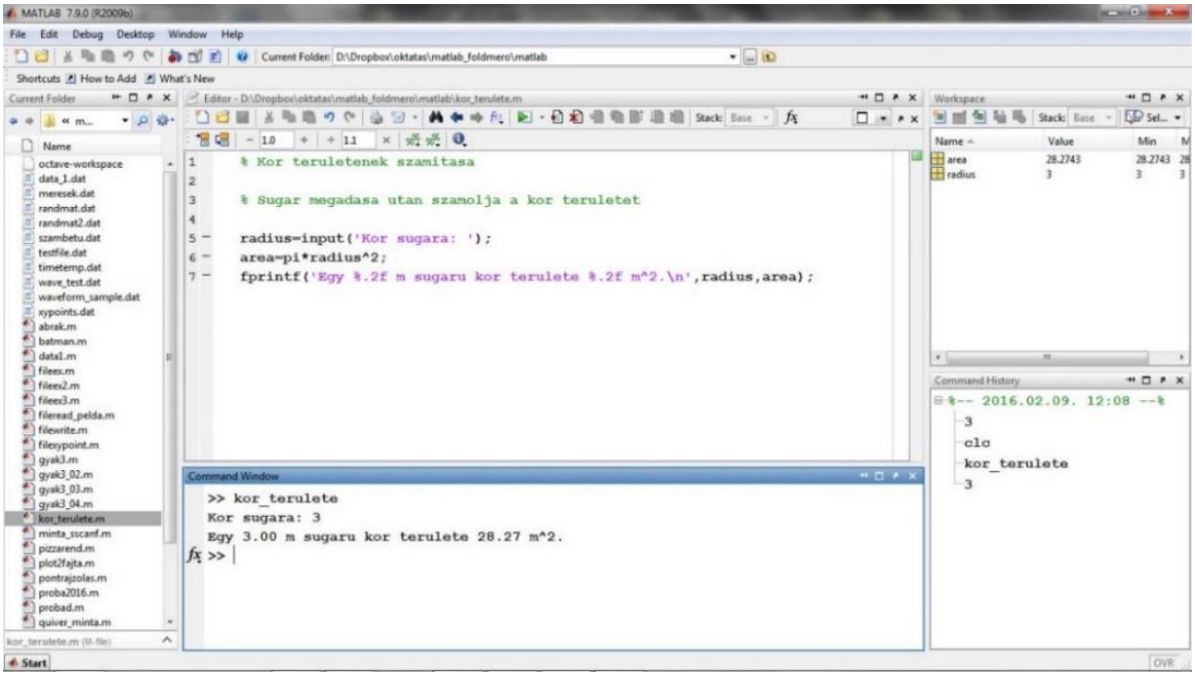
1. MATLAB/OCTAVE BASICS [1]

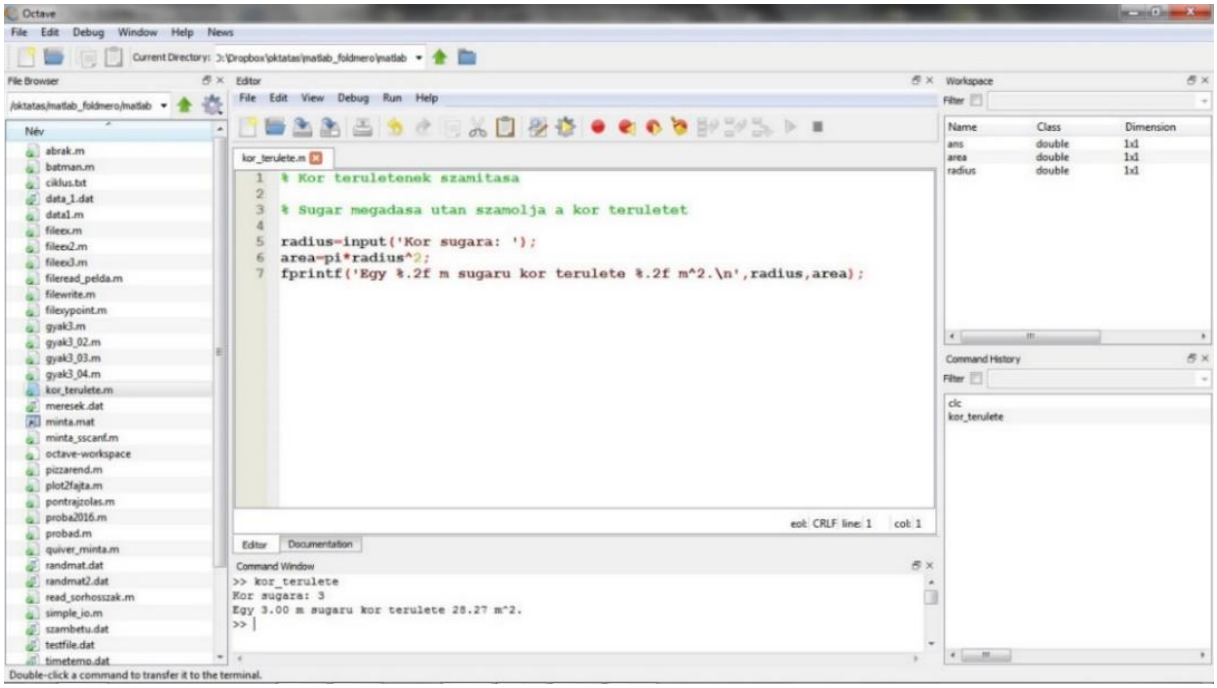
During the next practices, various engineering problems will be solved with numerical methods algorithms using the Matlab mathematical environment. Matlab software is available for the students of Budapest University of Technology and Economics (BUTE) for educational purposes from March 2017. A good alternative might be the Octave mathematical environment, which is a free, open-source program where you can use essentially the same commands as in Matlab. Even the graphical interface is very similar in the two programs, with optional rearrangeable windows (see Figures 1, 2). **Octave** can be downloaded from https://www.gnu.org/software/octave/, the current version is 5.1.0, which came out on Mar 1, 2019.

The students can create a **MathWorks account** on the Matlab site using BME email address, using this account they can access Online Matlab also. With the MathWorks account, students can practice the basics of Matlab solving the tasks of the **Matlab Onramp** online study site, which is recommended for everyone (it takes about 2-3 hours). There are other good online practice options at **Matlab Cody** site.



1MATLAB GRAPHICAL ENVIRONMENT

[1] Refernce for Matlab/Octave Basics: Todd Young and Martin J. Mohlenkamp: Introduction to Numerical Methods and Matlab Programming for Engineers, Department of Mathematics, Ohio University, July 24, 2018, http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/book.pdf



1 OCTAVE GRAPHICAL ENVIRONMENT

STARTING MATLAB/OCTAVE

The most important parts of the graphical interface are the **current folder** where the current directory is stored, the **command window**, the **workspace** with the used variables, the **command history** with all the used commands and the **editor** (editor). Left-click and hold on the name of a panel to drag and drop them into the blue-colored area. You may want to dock the editor by clicking on the arrow at the top right of the Editor window.

Keep in mind that file names used in Matlab should not begin with a number or contain special characters or spaces! In the program, only those functions and files can be used or run which are in the current directory.

HELP, DOCUMENTATION

We can learn a lot from the proper use of documentation. In older Matlab versions (before 2019), typing **help** to the Command Window listed the categories of various built-in functions. Here you could either double-click on the category name or type in the name of the category after the help command (in this handbook, the > sign indicates the Matlab commands). From Matlab R2019a, typing only the '**help**' command will display help of the last used command. Try the next:

```
> help elfun
```

This will list the 'Elementary math functions', e.g. trigonometric, exponential, complex and rounding functions. If we know the name of a function, we can type it after help:

```
> help 'function name'
```

This gives you a description of the command and how to use it. E.g. try:

```
> help rand
```

This command describes that the **rand** function generates an evenly distributed random number between 0 and 1, specifies how to use it to call with one or more inputs, and lists some associated commands (e.g. **randn**, which generates standard normal random numbers with 0 expected values and 1 standard deviation).

The **doc** command works much like help, but is a much more detailed description of the command, with many examples.

```
> doc randn
```

Let's try the **pwd** function! Use the **help** command to figure out what the command does!

Another useful function is the **lookfor** command. This can be used to search for a word in the name of the command or in its description. Let's try the following:

```
> lookfor rand
```

This will list all the commands whose names or their short description contains the word **rand**. If the search takes too long, it can be aborted by pressing CTRL + C.

ONLINE DOCUMENTATION

Help for Octave: https://www.gnu.org/software/octave/doc/interpreter/

Help for Matlab: http://www.mathworks.com/help/matlab/

Useful tips, functions in Matlabcentral: http://www.mathworks.com/matlabcentral/

Practice material: https://matlabacademy.mathworks.com/

Other practicing examples to solve: https://www.mathworks.com/matlabcentral/cody/

SOME USEFUL COMMANDS

`clc` – clears the content of the command window

`clear` – deletes variables (see workspace)

`close` – closes current figure or all (close all)

`CTRL+C` – interrupts the given command (e.g. exit an infinite loop)

`%` – comment (the program ignores everything in the line after this sign)

`%%` – you can open a new section in your script

`;` – using this sign at the end of the command the result will not be displayed in the Command window (cancels echo)

USING THE 'TAB' AND ARROW BUTTONS IN THE COMMAND WINDOW

The '**Tab**' button is very useful. If we do not know exactly the name of a particular command, just the beginning, let's start typing the beginning to the command line e.g. **pref**, then press '**Tab**'. If there is only one command with **pref** beginning, the '**Tab**' button will complete the command, if there are more, the '**Tab**' will list them. In this case, there are several commands with the beginning of **pref**, e.g. **prefdir** (the name

of the directory where all settings, history, etc. are located) or **preferences**, which opens the settings window.

The **up and down arrow buttons** are also very useful in the command line. With them, the previous commands can be retrieved, executed and modified. Previous commands can also be executed using the **command history** by double-clicking on the command name.

TAB	- use it to finish the started function/variable etc. name
ARROW KEYS	- you can scroll around the previously processed commands (you can access those from the command history too)
CTRL+ENTER	- run the whole section
F9	- run the selected part
F5	- run the whole script

ENTERING NUMBERS/VECTORS, VARIABLE TYPES, FUNCTIONS

```
> %% Entering numbers
> a = 0.01
> b = 1e-2
> c = 1d-2;
> a+c
> clear a % deletes the variable 'a'
> clear % (or 'clear all') clears all variables
> pi % built-in value (3.14)
> e = exp(1) % e^1 = e = 2.71
> b = e^-10 % exponentiation
```

Basic formatting:

```
> format long % 14 decimal digits
> e, b
> format short % 4 decimal digits
> e, b
```

The basic variable type in Matlab is the matrix. Vectors are special matrices, 1xn row vectors or mx1 column vectors. Square brackets are used in Matlab to define a matrix or vector. If you work with matrices/vectors, be careful with the different parenthesis, because each type is doing different operations: ( ), [ ], { } ! The elements are separated by a comma or space within the row, and the rows are separated by semicolon.

```
> z = [1 3 45 33 78] % rowvector
> z = [1,3,45,33,78] % rowvector
> t = [2; 4; 22; 66; 21] % column vectors
> M = [1,2,3; 4,5,6] % matix, with 2x3 size
```

When displaying a vector, it can be problematic to have very small and very large numbers at the same time. Let's look at the following vector, for example

```
> x = [25 56.31156 255.52675 9876899999];
> format short
> x
```

The result is difficult to read and not very informative:

```
x = 1.0e+09 *
```

```
    0.0000    0.0000    0.0000    9.8769
```

In this case, it is better to use a different display format, such as **shortG** or **longG**, which displays numbers of each different magnitude in their compact format (**shortG** – 5 significant digits, **longG** 15 significant digits).

```
> format shortG
> x
> format longG
> x
```

The results are:

```
x = 25       56.312       255.53          9.8769e+09 % format shortG
x = 25       56.31156     255.52675       9876899999 % format longG
```

You can query any element of a vector by putting the element index number in round brackets.

```
> t(2) % result: 4
> M(2,3) % result: 6
> z(end) % last element of z: 78
```

Or you can override the value of any element in the same way:

```
> t(2)=47
> p = [] % empty vector
> z(3)=[]; % deletes the 3rd element, after: z = 1 3 33 78
```

You can query a part of the vector or the matrix

```
> t(2:4) % result after the previous override command: 47 22 66
> t(1:29 % error message after mistyping
  t(39
     ?
 Error: Expression or statement is incorrect--possibly unbalanced (, {, or
[.
> t(1:29)
Index exceeds matrix dimensions.

> M(1:2,2:3) % result: [2,3; 5,6]
```

Transposed vector/matrix (switched rows and columns):

```
> tt = t' % t transposed, line vector
> Mt = M' % result: [1,4; 2,5; 3,6]
```

There are some useful commands that you can use to generate vectors:

```
> x1 = 1:10 % line vector: integers from 1 to 10
> x2 = 1:0.3:10 % line vector from 1 to 10, with 0.3 spacing
> x3= (1:0.3:10)' % column vector from 1 to 10, with 0.3 spacing
> x4 = linspace(1,10,4) % 4 numbers between 1 and 10, equaly spaced
```

It is easy to concatenate vectors/matrices horizontally/vertically when they have the same number of rows or columns.

```
> X = rand(2,3) % A matrix of random numbers between [0,1], size: 2x3
> Y = ones(2,4) % matrix of ones, size: 2x4
> Z = eye(3) % 3x3 identity (unit) matrix
> W = zeros(2,4) % Matrix of zeros, size: 2x4
> XY = [X, Y] % 2x7 sized matrix, X and Y horizontally concatenated
> XZ = [X; Z] % 5x3 sized matrix, X and Z vertically concatenated
> XY2 = [X; Y] % error message
```

```
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
   > XZ2 = [X,Z] % error message
Error using horzcat
Dimensions of matrices being concatenated are not consistent.
```

Accessing one row/column in a matrix:

```
> XY(1,:) % first row of XY (the : sign means all items in that row)
> XY(end,:) % last row of XY
> XY(:,1) % first column of XY (all items in that column)
> XY(:,end-1) % penultimate column of XY
```

Texts as vectors of characters

```
> s = 'p' % a text/character (string) variable of size 1x1
> u = 'University of Technology' % string type variable, size 1-by-24
> b = 'Budapest '
> bu = [b,u] % you can combine them using square brackets '[]'
> % Budapest University of Technology
> bu(24:29) % The strings could be handled as vectors
> % Techno
```

MOST COMMON VARIABLE TYPES

- Double: represents a double-precision floating-point number, mostly used to represent rational numbers. This is the default data type for numbers.
- Integer: represent an integer number (no fractional part). Be careful when calculating with both integers and doubles.
- Array (vector/matrix): a (multi-dimensional) collection of numbers. Can only contain data of similar type.
- Character array: textual data enclosed in single quotes, e.g. 'vehicle'.
- Cell array: similar to the regular, multi-dimensional array, but the types of the contained data can be different.
- Structure: an array with named fields, can contain varying data types.
- Table: array in a tabular form whose columns can be named and can be of varying data types.

WRITING AND RUNNING SCRIPT FILES

So far, we have been working from the command line, but solving a more complex calculation would be very difficult using only the command window. Therefore it may be better to collect the necessary commands/functions into a script file. The default file type for Matlab is the * .m file, a simple text file that can be edited with any text editor. In addition, in recent versions of Matlab, you can use the LiveScript file type (* .mlx), in which you can mix Matlab instructions and formatted text, images, and see the results right away. The disadvantage of this format is that this is Matlab's own format, which can only be opened within Matlab.

You can run the script files by typing their names or just pressing **F5**, this will save and run the program right away. Make sure that the filename does not start with a number or contain spaces or special characters! Filenames must start with a letter and contain only the English alphabet, numbers, and underscores (_).

If you don't want to run the whole program, you can write a **return** command somewhere, and after pressing **F5** the program will only run until that point. Alternatively, pressing **F9** will run only the selected part. You can divide the file into sections using double **%** characters (**%%**) followed by a section title. You can execute commands in a particular section by pressing **CTRL+Enter**. Let's start a new script file by clicking on the plus sign (new) in the upper left corner and this will open a blank page in the Editor. Save it to your current directory as practice1.m file!

PLOTTING - THE BASICS

The next table contains the data of a stress-strain diagram (σ-ε) of a steel bar for reinforced concrete:

ε [%]	0	0.2	2	20	25
σ [N/mm^2=Mpa]	0	300	285	450	350

1. TABLE, STRESS-STRAIN DIAGRAM OF A STEEL BAR FOR REINFORCED CONCRETE

Type the next commands to **practice1.m** script file!

```
> %% STRESS-STRAIN DIAGRAM OF STEEL
> x = [0,0.2,2,20,25] % prints its contents to the command window
> y = [0,300,285,450,350]; % does not display its contents in the
  command window
```

Run either the entire file with **F5** or a selected section with **F9** or the actual section with **CTRL + Enter**. You can check the content of any variable by typing its name into the command window or to the script file.

```
> x, y
```

To plot data stored in vectors, use the **plot** command:

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

This will connect the points with a line. If you want to mark points with symbols, try the following:

```
> plot(x,y,'x')
> plot(x,y,'o-')
> plot(x,y,'r*-')
```

You can add many arguments to define the specifics of a plot, like the shape and size of the markers, specifics of the lines, etc.

```
> plot(x,y,'--gs','LineWidth',2,'MarkerSize',10,...
>       'MarkerEdgeColor','b','MarkerFaceColor',[0.5,0.5,0.5])
```

Useful specifiers:

Marker	Description
o	Circle
+	Plus sign
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
p	Pentagram
h	Hexagram

Long Name	Short Name	RGB Triplet
'yellow'	'y'	[1 1 0]
'magenta'	'm'	[1 0 1]
'cyan'	'c'	[0 1 1]
'red'	'r'	[1 0 0]
'green'	'g'	[0 1 0]
'blue'	'b'	[0 0 1]
'white'	'w'	[1 1 1]
'black'	'k'	[0 0 0]

LineWidth	Line width
MarkerEdgeColor	Color of the outline of the symbol
MarkerFaceColor	Color of the symbol fill
MarkerSize	Size of the symbol

You can name the axes, or add a legend or a title:

```
> xlabel('Strain')
> ylabel('Stress')
> title('STRESS-STRAIN DIAGRAM OF STEEL')
```

ADDITIONAL USEFUL TIPS FOR PLOTTING

Each figure and also the plotted elements can be named with a handle (an identifier). You can use this to access any figure/element later if you want to set a property or clear them. If you don't do further settings, by plotting a new element, the previous one will be deleted.

```
> clf         % clear figure
> f1 = figure; % the figure function creates a new figure
> p1 = plot(x,y,'r*');
> hold on  % you can fix the diagram, to add new elements to it without
  cleaning it first
> p2 = plot(x,y);
> p3 = plot(x,y,'bo');
> delete(p1)  % you can delete elements by using its handle
> figure(1)   % by referring to the number of the figure, you can work
  on any previous figure
> close % close figure
```

FUNCTIONS

There are many mathematical and other built-in functions in Matlab, it is useful to browse the documentation to get to know them. Let's look at some of the basic math functions, from 'Elementary math functions' in the documentation, using the '**help elfun**' command! The input variables of the functions are always enclosed in round brackets. The default angular unit for trigonometric functions is radian.

```
> sin(pi) % its value is 0 within the accuracy of the numerical
  representation
> cos(pi) % -1
> tan(pi) % a large number instead of infinite
> log(100) % natural logarithm
> log10(100) % 10 based logarithm
> 3^4 % értéke: 81
> sqrt(81) % 9
> abs(-6) % 6
> exp(0)
```

exp(0) is e^0, its value is of course 1. The built-in functions work not only on numbers but also on vectors.

```
> x = linspace(0,2*pi,40)
> y = sin(x)
> figure(1); plot(x,y)
```

For more settings see **help plot**!

USER DEFINED ANONYMOUS ('SINGLE-LINE') FUNCTIONS

There are many ways to specify your own functions in Matlab, in simple cases we use the so-called anonymous function, which is not saved as a separate program, only assigned to a variable. Let's define cos(2x) function in this way!

```
> f = @(x) cos(2*x)
```

Here, after the @ symbol, you must specify the independent variables of the function, and after a space you can write the formula. Let's plot this function also next to the sine function you have just drawn. When drawing the sine function, we first calculated some points of the function and plotted them using the **plot** command. We can plot functions without calculating their points beforehand, using symbolic **fplot** or **ezplot** commands. (Note: In Octave and older Matlab versions you can use only the **ezplot** command, in newer Matlab **fplot** is recommended).

```
> hold on
> fplot(f,[0,2*pi]) % or ezplot(f,[0,2*pi])
```

Without the **hold on** command, the previously drawn elements will be deleted from the figure, the **hold off** command restores this default mode. For **fplot**, you can specify an interval where you want to display the function (it is optional, there is a default interval also).

Calculate the squares of the integers between 1-10 using your own function! First, let's define a function to calculate the square of any number!

```
> f1 = @(x) x^2
```

Check for a given value of x:

```
> f1(3) % value: 9
```

And now let's calculate the squares of the integers between 1-10!

```
> x = 1:10;
> y = f1(x)
```

We got an error message:

```
Error using  ^
One argument must be a square matrix and the other must be a scalar. Use
POWER (.^) for elementwise power.
Error in gyak1>@(x)x^2
Error in gyak1 (line 100)
y = f(x)
```

Why? Because our variable x is a line vector, and when it is squared we try to multiply two line vectors, which is mathematically incorrect (mathematically the line vector could be multiplied by a column vector). If we do not want to perform a vector/matrix operation, but element by element operation, we must put a dot before the * operation symbol (elementwise operation).

```
> f1 = @(x) x.^2
> y = f1(x) % 1  4   9  16  25  36  49  64  81  100
```

Since addition/subtraction, division/multiplication with a scalar for vectors/matrices is done element by element, there is no need to put a dot before the operation symbol in these cases, only in case of multiplication, division and power of vectors:. *,. /,. ^.

One of Matlab's strengths is its ability to perform operations with vectors and matrices, so in many cases, we can avoid the use of the much slower loops with vectorized operations.

FUNCTIONS IN SEPARATE FILES

Matlab's default file type is a text file with the *.m extension. There are two main types, script file and function. We have used the former in our work so far, now let's see the differences between functions and script files!

One of the advantages of writing the functions into a separate file compared with the anonymous functions is that it can be invoked from any other script file also. Another advantage is that it can be used to perform more complex calculations, it is easy to parameterize the input and output variables and a description can be added for help.

Let's rewrite the previous square function into a separate function file! Click the plus sign (new) in the upper left corner and a blank page will open in the Editor. Type in the following, and save it as **squarefun.m** to the current directory. Important: The name of the function (written in bold in the following code) must be the same as the file name, otherwise the function cannot be called!

```
> function y = squarefun(x)
> % Calculate the square of a number
>     y = x.^2;
> end
```

Some characteristics of the functions are:

- begins with the word function
- There are at least one output and one input
- The output, function name and input arguments are on the first line and the function name must match the *.m filename
- We need to assign value to the output somewhere inside the function
- Internal variables in a function are local variables, they will not appear in the workspace, and the function will not have access to the variables in the workspace except to the defined input arguments.
- A function cannot be executed, it can only be called from another file or command line! To be called, the function must be in the current directory (or in a directory that is in the path).
- The comments written after the first line of the function are listed when using the help command with this function name.

Let's call the written function on our vector x! To do this, switch back to the **practice1.m** script file!

```
> squarefun(11) % 121
> squarefun(x) % 1  4  9  16  25  36  49  64  81  100
> help squarefun %  Calculate the square of a number
```

A function can have multiple inputs, listed after the name of the function in round brackets. Let's modify our previous function as follows and save it as **powerp.m**!

```
> function y = powerp(x,p)
>     y = x.^p
> end
```

A function can have multiple outputs collected in a vector in square brackets (powers.m):

```
> function [x2 x3 x4] = powers(x)
>     x2 = x.^2;
>     x3 = x.^3;
>     x4 = x.^4;
> end
```

Call the above functions also from our script file!

```
> powerp(x,3) % 1  8  27  64  125  216  343  512  729  1000
> [a b c] = powers(x)
> % a = 1 4  9 16 25 36 49 64 81 100
> % b = 1 8 27 64 125 216 343 512 729 1000
> % c = 1 16 81 256 625 1296 2401 4096 6561 10000
```

Plot the results in a new Figure (number 3) using the figure command. You can list several graphs to plot in the same figure in one **plot** command!

```
> figure(3)
> plot(x,a,x,b,x,c)
```

We can add custom colors and legend. The text of the legend should be in the same order as we plotted the graphs.

```
> plot(x,a,'black',x,b,'blue',x,c,'green')
```

```
> plot(x,a,'k',x,b,'b',x,c,'g')
> legend('square','x^3','x^4','Location','North')
> legend('square','x^3','x^4','Location','Best')
```

COMMENTS AND HELP TO USER DEFINED FUNCTIONS

Comments are an important part of multi-line programs. On the one hand, others also can understand our code, and on the other hand, we will also remember it if we want to use it again or modify later. It is advisable to write comments not only at the beginning of the program but also for each new section. In Matlab, you can write comments after the % sign. For a function, it is useful to specify in the comments what the purpose of the function is, what input and output values are used. In the case of a function, the comments written after the first line will be displayed when calling the help command with this function name.

MATLAB ERROR MESSAGES

When writing a script we encounter many error messages, as we have seen some earlier. It is important that we could interpret these to correct our mistakes!

Let's look a mistyping error in '**clear all**'!

```
> cler all
Undefined function or variable 'cler'.
Did you mean:
>> clear all
```

Matlab is case sensitive:

```
> X = 3/4; x
Undefined function or variable 'x'.
```

Let's look an example of a syntactically incorrect Matlab statement:

```
> 1 X
1 X
  ↑
Error: Unexpected MATLAB expression.
```

Wrong number of input parameters:

```
> sin(pi,3)
Error using sin
Too many input arguments.
```

The number of rows/columns in the matrices does not match:

```
> M = [1 2;3]
Dimensions of matrices being concatenated are not consistent.
> [3, 4, 5] * [1; 2; 3; 4]
Error using  *
Inner matrix dimensions must agree.
> a = 1:5, b = 1:3
> [a;b]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

An easy typing error would be to type 8 or 9 instead of round parentheses:

```
   >  sin(pi9
sin(pi9
        ↑
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
```

Missing parenthesis:

```
   >  abs(sin(rand(2))
abs(sin(rand(2))
                ↑
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
```

We want to perform element-by-element operation on a vector, but the point is missed:

```
   >  v =1:4;
   >  1/v
Error using  /
Matrix dimensions must agree.
```

Worst of all, when there is no error message, but the result is still incorrect. Example: Calculate $\frac{1}{2\pi}$ with the following statement. Why is the result wrong?

```
   >  1/2*pi %  1.5708
```

ADDITIONAL ADVICE FOR USING OCTAVE

If you decide to use Octave to practice at home, **Octave** can be downloaded from https://www.gnu.org/software/octave/, the current version is 5.1.0, which came out on Mar 1, 2019. The advantage of Octave over education licensed Matlab is that it is an open source program that can be used not only for educational purposes but also for work. There are many add-on packages available, see https://octave.sourceforge.io/ and https://octave.sourceforge.io/packages.php, many of them are installed, just need to be loaded when you want to use them. You can query what is installed with the **pkg list** command).

INSTALLING THE SYMBOLIC PACKAGE

During the numerical methods practices we will perform many symbolic computations too, which requires installing an extra symbolic package for Octave (this is a separate toolbox in matlab). This package is based on python-sympy and must be installed separately. Installation under Windows (link: https://github.com/cbm755/octsympy/wiki/Notes-on-Windows-installation)

1. Download the symbolic-win-py-bundle-x.y.z.zip file at the github releases page. https://github.com/cbm755/octsympy/releases (here x.y.z is the version number, e.g.symbolic-win-py-bundle-2.8.0.tar.gz)
2. Start Octave, change folder to where your downloads are.
3. Install the package by typing into Octave:

```
>  pkg install symbolic-win-py-bundle-x.y.z.tar.gz
```

4. Load symbolic package to octave

```
>  pkg load symbolic
```

5. Check the package by typing a few symbolic commands!

```
>  syms x
>  f = sin(cos(x));
```

```
> diff (f)
```

Result ⇒ `-sin(x)·cos(cos(x))`

Functions of the symbolic package in detail:
https://octave.sourceforge.io/symbolic/overview.html

USED MATLAB BUILT-IN FUNCTIONS

help	-	matlab help categories, or help of a specific topic or function
rand	-	Random numbers between 0-1, evenly distributed
randn	-	Random numbers in standard normal distribution with 0 expected value and 1 standard deviation
doc	-	detailed documentation for a given function, in a new window
lookfor	-	search for a word, word fragment in help
clc	-	clears the contents of the command window
clear, clear all	-	deletes the specified variables or all variables
close, close all	-	closes the current graph or all
CTRL+C	-	interrupts the given command (e.g. exit an infinite loop)
%	-	comment (the program ignores what is next in the line)
;	-	at the end of the command; the result will not appear in the Command Window
Tab gomb	-	completes a command that has been started
preferences	-	opens the settings window
prefdir	-	the name of the directory where the settings, history, etc. are stored
↑↓ buttons	-	previous commands can be returned to the Command Window
pi	-	3.14… (pi number)
exp(1), exp(n)	-	e^1 = 2.71…, e^n, Euler's number
^	-	exponentiation
format long, format short	-	displaying multiple (14) or less decimal digits (4)
format shortG, format longG	-	displays numbers of different magnitudes in compact format (5 or 15 significant digits).
[1, 2, 3; 4, 5, 6]	-	vector/matrix definition
'	-	transposed vector/matrix
[A,B] vagy [A B]	-	join matrices side by side (equal number of rows)
[A;B]	-	join matrices under each other (equal number of columns)
A(1,:)	-	first row of matrix

A(:,1), A(:,end)	-	first / last column of matrix
linspace(x1,x2,n)	-	Vector between [x1, x2] with n points evenly distributed
ones	-	matrix of ones
zeros	-	matrix of zeros
eye	-	identity matrix
figure	-	Open a new graph
plot	-	drawing point vectors
xlabel, ylabel	-	x, y axis annotation
title	-	Figure title
sin, cos, tan	-	angle functions (default unit: radian)
log, log10	-	natural logarithm, 10 based logarithm
sqrt	-	square root
abs	-	absolute value
hold on, hold off	-	overwrite or do not overwrite the existing figure with the new one
fplot, ezplot	-	plotting functions
.* ./ .^	-	multiplication, division, power with vectors element-by-element
clf	-	deleting elements of the figure (does not close the window)
legend	-	legend
return	-	return point - F5 executes the program only up to this point