Numerical methods - Introduction to MATLAB

IMPORTANT: In MATLAB there are some restrictions for the filenames/variable names:
- cannot start with numbers
- avoid spacebar/special characters and mathematical operators

For info about any MATLAB specific questions:

```
help max  % help function_name, or just search in the help guide
lookfor rand % it creates a list from every function in connection, with "rand"
```

If you want to stop the process, hit: CTRL + C

On any function, you can get info from the documentation by hitting F1

```
clear all % clears the command window
% clears x - deletes variable x from the workspace
% you can add comments to your script
% you can open new section to your script
cancels echo
TAB % use it to finish the started functionname/variable name etc.
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
```

MATLAB basics

Defining variables:

```
a = 0.01
b = 1e-2
c = 1d-2
clear a % deletes variable 'a'
clear all % deletes all the variables
pi % built in variable for pi
exp(1) % e^1 = e = 2.71
```

The results of your calculations are presented till 4 decimals by default, BUT: this is JUST A REPRESENTATION!

For calculations: use the variable, where you stored the calculated value

```
format long % to increase the number of decimals presented in the command window
pi
```

If you work with matrices/vectors, be careful with the different parenthesis, because each type is doing different operations: (), [], {}!

```
M = [1 2 3; 4 5 6] % 2-by-3 matrix
size(M) % size of the matrix
v = [1 3 45 33 78] % rowvector
length(v) % length of a vector
t = [2; 4; 22; 66; 21] % column vector
v2 = v' % via transpose operator ('') you can switch the rows and columns
```

You can access the elements of a vector/matrix with index numbers:

```
t(2) % second element of vector t, result: 4
v(end) % the last element of vector v: 78
```

You can rewrite the value of each element with the same way:

```
t
  t(2)=47;
t
p = [] % empty vector/matrix
v(3)=[] % deletes the third element of vector v, therefor from this point: v = 1 3 33 78
```

You can access multiple elements of a vector/matrix too:

```
t(2:4) % result: 47 22 66
```
You can create sequences easily via semicolon:

\[
\begin{align*}
\text{x1} &= 1:10 \quad \% \text{ creates a list from 1 to 10 - startnumber:endnumber} \\
\text{x2} &= 1:0.3:10 \quad \% \text{ creates a list from 1 to 10 with a stepsize of 0.3 - startnumber:stepsize:endnumber} \\
\text{x3} &= \text{linspace}(1,10,10) \quad \% \text{ creates a list from 1 to 10, that has 10 elements - linspace(startnumber,endnumber,\text{number of elements})}
\end{align*}
\]

The vectors and matrices could be combined together horizontally/vertically easily - if they have the same number of rows/columns:

\[
\begin{align*}
\text{X} &= \text{rand}(2,3) \quad \% \text{ generates a 2-by-3 matrix filled with random numbers in the interval [0,1]} \\
\text{Y} &= \text{ones}(2,4) \quad \% \text{ generates a 2-by-4 matrix filled with ones} \\
\text{Z} &= \text{eye}(3) \quad \% \text{ generates a 3-by-3 identity matrix} \\
\text{XY} &= [\text{X} \quad \text{Y}] \quad \% \text{ a 2-by-7 matrix, bye combining X and Y horizontally} \\
\text{XZ} &= [\text{X} ; \text{Z}] \quad \% \text{ a 5-by-3 matrix, bye combining X and Z horizontally}
\end{align*}
\]

Accessing one row/column in a matrix:

\[
\begin{align*}
\text{XY}(:,1) \quad &\% \text{ the first row of the matrix XY - via semicolon, you can access ALL elements} \\
\text{XY}(:,1:end) \quad &\% \text{ the first column of the matrix XY - via semicolon, you can access ALL elements} \\
\text{XY}(1,:,:) \quad &\% \text{ the last column of the matrix XY - via semicolon, you can access ALL elements}
\end{align*}
\]

Strings as vectors created from characters:

\[
\begin{align*}
\text{str} &= 'p' \quad \% \text{ string type variable, size 1-by-1} \\
\text{me} &= ['\text{University of Technology}'] \quad \% \text{ string type variable, size 1-by-1} \\
\text{bme} &= ['\text{Budapest}' 'me'] \quad \% \text{ you can combine them together with brackets '[]' - Budapest University of Technology} \\
\text{bme(13:19)} \quad &\% \text{ The strings could be handled and taken apart as well}
\end{align*}
\]

**Plotting - the basics**

```
xy = load('stressstrain.txt');
\% \text{ xy = [0, 0.2, 2, 20, 25; 0, 300, 285, 450, 350];}
x = xy(:,1); 
y = xy(:,2);
plot(x,y);
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:**

```
plot (x,y, 'o', 'LineWidth', 2, 'MarkerSize', 10, 'MarkerEdgeColor', 'b', 'MarkerFaceColor', [0.5, 0.5, 0.5])
```

**Additional useful tips for plotting**

Each figure, and the elements plotted on them could 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.

```
cf - clear figure

f1 = figure; 
pl1 = plot(x,y,'r*');
hold on
pl2 = plot(x,y);
pl3 = plot(x,y,'bo');
delete(pl1)
figure(1)
```

**Functions in MATLAB**
The arguments of functions are always inside the regular parenthesis, and while you are typing MATLAB is constantly giving you hints what kind of additional inputs can you add. Use the help guide anytime by pressing F1 on the function!

\[
\sin(p) \quad \text{result: 0 (or at least within the bounds on floating point representation error)} \\
3^4 \quad \text{result: 81} \\
\exp(0) \quad \text{result: 1} \\
\text{this is the } e^0, \text{ which is 1. } e \text{ means the euler number 2.7183}
\]

The built-in functions are also working on vectors:

\begin{verbatim}
x = linspace(0, 2*pi, 40)
y = sin(x)
figure
plot(x, y, 'bx-')
\end{verbatim}

You can also write your very own functions, and you have more options. For the more simple functions you can create anonymous functions, that are not saved as a separate program: they are just assigned to a variable! For example, if you want to create a function for the following:

\[
f(x) = 2x^2 - 3x + 1
\]

The MATLAB code will look like this:

\begin{verbatim}
f = @(x) 2*x.^2 - 3*x + 1
\end{verbatim}

This means the function is assigned to the variable \( f \), and it has a single argument. You define the input parameters with the @ sign, and you can put multiple arguments inside the parenthesis if you wish.

If you want to call the function, you can refer to its variable, as follows:

\begin{verbatim}
f(1.2345)
\end{verbatim}

This function works also on vectors, not only on single values! Its because we added a dot operator before we squared \( x \). If you want to make your own function to work similarly, you should add a ".\(^{\text{th}}\) before multiplication, division and in exponential expressions as follows:

\[
\cdot \quad \cdot \quad \cdot
\]

If you add/subtract, or multiply your inputs by a scalar value, this is not necessary.

If you want to write a separate function, you should apply the following structure in a new script file:

\begin{verbatim}
function [output1, output2] = functionname(input1, input2, ...)
    output1 = formula1
    output2 = formula2
end
\end{verbatim}

\begin{verbatim}
x1 = -2:0.1:2;
y1 = probafv(x1);
clf
plot(x1, y1)
\end{verbatim}

About the functions:

- It should start with the keyword \texttt{function}
- It should have at least one input and one output
- In the first row you should have the output, the functions name, and the input; the functions name should be the same as, the \texttt{.m} file name
- Inside the function you should assign a value to the output
- The variables defined inside the functions are local variables, those won’t appear in the workspace, also the running function has no access to the variables inside the workspace, only if you added them, as an input

Your own function could have multiple inputs, just modify the previous function as follows:

\begin{verbatim}
function y = probafv2(x, p)
    y = 2*x.^p - 3*x + 1;
end
\end{verbatim}

Save it as a separate function, with its name, and recall it:

\begin{verbatim}
y1 = probafv2(x1, 3);
plot(x1, y1)
\end{verbatim}

A function could have multiple outputs as well, defined as a vector:

\begin{verbatim}
function [x2, x3, x4] = hatvany(x)
    x2 = x.^2;
    x3 = x.^3;
    x4 = x.^4;
end
\end{verbatim}
Save this as a separate file and let's present the results on a figure:

```matlab
x = -1:.1:1;
[x2, x3, x4] = hatvany(x);
plot(x,x,x2,x,x3,x,x4)
```

On the figure you will see each element with separate colors, but you can customize them on your own too:

```matlab
plot(x,x,'black',x,x2,'blue',x,x3,'green',x,x4,'red')
```

You can add additional information too e.g. legend, title, axis label as follows:

```matlab
legend('x','x^2','x^3','x^4','Location','SouthEast') \ % adding legend
title('Power functions') \ % adding title for the figure
xlabel('x'); \ % adding labels for the horizontal axis
ylabel('Functions value'); \ % adding labels for the vertical axis
```

```matlab
function y = probafv(x)
    y = 2*x.^2 - 3*x + 1;
end
```

```matlab
function y = probafv2(x,p)
    y = 2*x.*p - 3*x + 1;
end
```

```matlab
function [x2, x3, x4] = hatvany(x)
    x2 = x.^2;
    x3 = x.^3;
    x4 = x.^4;
end
```