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
```

```
max     Largest component.
        For vectors, max(X) is the largest element in X. For matrices, 
        max(X) is a row vector containing the maximum element from each 
        column. For N-D arrays, max(X) operates along the first 
        non-singleton dimension.

[Y,I] = max(X) returns the indices of the maximum values in vector I.
       If the values along the first non-singleton dimension contain more
       than one maximal element, the index of the first one is returned.

max(X,Y) returns an array with the largest elements taken from X or Y.
       X and Y must have compatible sizes. In the simplest cases, they can be
       the same size or one can be a scalar. Two inputs have compatible sizes
       if, for every dimension, the dimension sizes of the inputs are either
       the same or one of them is 1.

[Y,I] = max(X,[],DIM) operates along the dimension DIM.

When X is complex, the maximum is computed using the magnitude
max(abs(X)). In the case of equal magnitude elements, then the phase
angle max(ANGLE(X)) is used.

max(...,NANFLAG) specifies how NaN (Not-A-Number) values are treated.
NANFLAG can be:
   'omitnan' - Ignores all NaN values and returns the maximum of the
                non-NaN elements. If all elements are NaN, then the
                first one is returned.
   'includenan' - Returns NaN if there is any NaN value. The index points
                  to the first NaN element.

Default is 'omitnan'.

Example:
   X = [2 8 4; 7 3 9]
   max(X,[],1)
   max(X,[],2)
   max(X,5)

See also min, cummax, median, mean, sort.
```

Reference page for max
Other functions named max

```
lookfor rand  % it creates a list from every function in connection, with "rand"
```

```
qmult   - Pre-multiply matrix by random orthogonal matrix.
randcolu   - Random matrix with normalized columns and specified singular values.
randcorr  - Random correlation matrix with specified eigenvalues.
randhess   - Random, orthogonal upper Hessenberg matrix.
randjorth  - Random J-orthogonal matrix.
rando      - Random matrix with elements -1, 0 or 1.
randsvd    - Random matrix with pre-assigned singular values.
rand     - Uniformly distributed pseudorandom numbers.
randn    - Pseudorandom integers from a uniform discrete distribution.
randp    - Normally distributed pseudorandom numbers.
randperm  - Random permutation.
rng      - Control the random number generator used by RAND, RANDI, and RANDN.
get      - Get a random stream property.
rand      - Pseudorandom numbers from a uniform distribution.
randi     - Pseudorandom integers from a uniform discrete distribution.
randn     - Pseudorandom numbers from a standard normal distribution.
randperm  - Random permutation.
RandStream - Random number stream.
reset     - Reset a random stream to its initial internal state.
set       - Set a random stream property value.
rfr       - Random Jacobi rotation.
sprand    - Sparse uniformly distributed random matrix.
```
Example_CopulaRNG - Generate random numbers with copulas
Example_StratifiedRNG - Random number generation with terminal stratification
simardata - Simulate values missing uniformly at random in iid data
RandomStartPointSet - A random start point set.
selectiontournament - Each parent is the best of a random set.
selectionunifom - Choose parents at random.
resetDriftRNG - Reset the random number generator to a saved state.
privateselectionheaderandterminator - Extracts header and terminator for serial blocks.
dividerand - Partition indices into three sets using random indices.
randnc - Normalized column weight initialization function.
randnr - Normalized row weight initialization function.
rands - Symmetric random weight/bias initialization function.
randsmall - Small random weight/bias initialization function.
randtop - Random layer topology function.
trainr - Random order weight/bias training.
trainru - Unsupervised random order weight/bias training.
setdemorandomstream - SETDEMORANDOMSTREAM Set default stream for reliable example results.
getoptimizerstates - Get states of random number generator for solvers
setoptimizerstates - Set states of random number generator for solvers
rssnb - Generates random continuous-time state-space model D with NX states.
dataSample - Randomly sample from data.
weightedDataSample - Draws weighted samples without replacement at random for tall arrays
betarnd - Random arrays from beta distribution.
binornd - Random arrays from the binomial distribution.
chisqrand - Random arrays from chi-square distribution.
copularnd - Random vectors from a copula.
dataSample - Randomly sample from data, with or without replacement.
evrand - Random arrays from the extreme value distribution.
exprnd - Random arrays from exponential distribution.
frnd - Random arrays from the F distribution.
gamrnd - Random arrays from gamma distribution.
georn - Random arrays from the geometric distribution.
gevrnd - Random arrays from the generalized extreme value distribution.
gprnd - Random arrays from the generalized Pareto distribution.
hygev - Random arrays from the hypergeometric distribution.
lwirnd - Generate inverse Wishart random matrix.
johnsrnd - Random arrays from the Johnson system of distributions.
lognornd - Random arrays from the lognormal distribution.
mvnrand - Random vectors from the multivariate distribution.
mvrand - Random vectors from the multivariate normal distribution.
mvtrand - Random matrices from the multivariate t distribution.
nbinrnd - Random arrays from the negative binomial distribution.
nfirdn - Random arrays from the noncentral F distribution.
nctrand - Random arrays from the noncentral t distribution.
nx2rand - Random arrays from the non-central chi-square distribution.
normrnd - Random arrays from the normal distribution.
parsrnd - Random arrays from the Pearson system of distributions.
polssrnd - Random arrays from the Poisson distribution.
randg - Gamma random numbers (unit scale).
random - Generate random arrays from a specified distribution.
randsample - Random sample, with or without replacement.
randtool - Demonstration of many random number generators.
rnstest - Runs test for randomness.
trnd - Random arrays from Student's t distribution.
unidrnd - Random arrays from the discrete uniform distribution.
unifrnd - Random arrays from the continuous uniform distribution.
wbirnd - Random arrays from the Weibull distribution.
wishrnd - Generate Wishart random matrix.
random - Random vector generation.
gmrnd - Random vectors from a multivariate Gaussian mixture model.
rand - Random number generation for piecewise distribution.
grandset - Quasi-random point set class.
grandstate - This undocumented class may be removed in a future release.
grandstream - Create a quasi-random stream.
digitalShifts - Create random digital shifts.
statguidists - Distribution data for RAND TOOL and DIST TOOL.

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

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

clc 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
MATLAB basics

Defining variables:

\[ a = 0.01 \]
\[ a = 0.0100 \]
\[ b = 1e-2 \]
\[ b = 0.0100 \]
\[ c = 1d-2 \]
\[ c = 0.0100 \]

\% deletes variable 'a'
\% deletes all the variables
\% built in variable for pi

\[ \text{clear a} \]
\[ \text{clear all} \]
\[ \text{pi} \]
\[ \text{ans} = 3.1416 \]
\[ \text{exp(1)} \]
\[ \text{ans} = 2.7183 \]

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!

\% to increase the number of decimals presented in the command window

\[ \text{format long} \]
\[ \text{pi} \]
\[ \text{ans} = 3.141592653589793 \]

A problem with format long is that it sometimes tends to be pretty heavy on the eyes. For example:

\[ \text{vec} = [0.0002, 0.02, 200, 2000, 200000] \]

\[ \text{vec} = 1\times5 \]
\[ 1e5 \times \]
\[ 0.0000000000000002 \]
\[ 0.0000000000000002 \]
\[ 0.0020000000000002 \]
\[ 0.0200000000000002 \]
\[ 2.0000000000000002 \]

In these cases, when our data contains numbers with smaller and bigger exponents, it is better to change to format longG or format shortG:

\% to increase the number of decimals presented in the command window

\[ \text{format longG} \]
\[ \text{vec} \]
\[ \text{vec} = 1\times5 \]
\[ 0.0002 \]
\[ 0.02 \]
\[ 200 \]
\[ 2000 \]
\[ 200000 \]

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

ans =
   5

t = [2; 4; 22; 66; 21] % column vector

t = 5x1
  2
  4
 22
 66
21

v2 = v' % via transpose operator ('') you can switch the rows and columns

v2 = 5x1
   1
   3
 45
 33
 78

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

t(2) % second element of vector t, result: 4

ans =
   4

v(end) % the last element of vector v: 78

ans =
   78

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

t

t = 5x1
  2
  4
 22
 66
21

t(2)=47;

t

t = 5x1
  2
47
 22
 66
21

p = [] % empty vector/matrix

p =
[]

v(3)=[] % deletes the third element of vector v, therefore from this point: v = 1 3 33 78

v = 1x4
   1    3   33  78

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

t(2:4) % result: 47 22 66

ans = 3x1
   47
   22
   66

You can create sequences easily via semicolon:
x1 = 1:10  % creates a list from 1 to 10 - startnumber:endnumber
x1 = 1:10
   1  2  3  4  5  6  7  8  9 10

x2 = 1:0.3:10  % creates a list from 1 to 10 with a stepsize of 0.3 - startnumber:stepsize:endnumber
x2 = 1:31
   1   1.3   1.6   1.9   2.2

x3 = linspace(1,10,10)  % creates a list from 1 to 10, that has 10 elements - linspace(startnumber, endnumber, number of elements)
x3 = 1:10
   1  2  3  4  5  6  7  8  9 10

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

X = rand(2,3)  % generates a 2-by-3 matrix filled with random numbers in the interval [0,1]
X = 2x3
   0.814723686393179  0.126986816293506  0.63235924622541
   0.905791937075619  0.91375856130919  0.897540489994095

Y = ones(2,4)  % generates a 2-by-4 matrix filled with ones
Y = 2x4
   1  1  1  1
   1  1  1  1

Z = eye(3)  % generates a 3-by-3 identity matrix
Z = 3x3
   1  0  0
   0  1  0
   0  0  1

XY = [X Y]  % a 2-by-7 matrix, by combining X and Y horizontally
XY = 2x7
   0.814723686393179  0.126986816293506  0.63235924622541  1  1
   0.905791937075619  0.91375856130919  0.897540489994095  1  1

XZ = [X; Z]  % a 5-by-3 matrix, by combining X and Z horizontally
XZ = 5x3
   0.814723686393179  0.126986816293506  0.63235924622541
   0.905791937075619  0.91375856130919  0.897540489994095
   1  0  0
   0  1  0
   0  0  1

Accessing one row/column in a matrix:

XY(1,:)  % the first row of the matrix XY - via semicolon, you can access ALL elements
ans = 3x7
   0.814723686393179  0.126986816293506  0.63235924622541  1  1

XY(:,1)  % the first column of the matrix XY - via semicolon, you can access ALL elements
ans = 2x1
   0.814723686393179
   0.905791937075619

XZ(:,end)  % the last column of the matrix XY - via semicolon, you can access ALL elements
ans = 5x1
   0.63235924622541
   0.897540489994095
   0
   0
   1

Strings as vectors created from characters:

str = 'p'  % string type variable, size 1-by-1
str = 'p'
me = 'University of Technology'  % string type variable, size 1-by-24
bme = ['Budapest ' me]  % you can combine them together with brackets '[]' - Budapest University of Technology
bme = 'Budapest University of Technology'
bme(13:19)  % The strings could be handled and taken apart as well
ans = 'versity'

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, eg. '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.

Errors and warnings

Whenever we make some kind of error in writing our script or code, be it a speling mistake in the name of a function or some kind of syntactical error, MATLAB will give us an error. Encountering errors and interpreting the error message is an essential part of learning to code correctly and thankfully, MATLAB is pretty good at giving us valuable information about the mistakes we made.

For example, if we try to add a 2x2 matrix to a 2x3 matrix (which is not really mathematically sound), the software will not be able to carry out the operation and instead will return an error message. Errors are always represented by red text and will always say what the error is. In this case, ‘Matrix dimensions must agree.’ will tell us, that we have to check if the dimensions of the matrices to be summed correspond to each other. Many error messages come from accidentally omitting or changing up opening/closing parentheses, typos in function names, mistaking the ‘;’ for a ‘,’ and so on, so watch out for these. If an error is encountered in the code, MATLAB will stop running the script until the error is fixed, or some other measures are taken to ignore or handle the error.

Warnings are a bit different, as in most cases they do not stop the running of the code. Their purpose is to inform us about possible inefficiencies and arguments or options which may have an effect on the running of the code or the results. Sometimes, they also warn us about using outdated or deprecated functions/scripts. The warning message should be read carefully as well, as in some cases we can just ignore them and in other cases the reason for the warning is something we might want to take a closer look at.

Plotting - the basics

xy = [0, 0.2, 2, 20, 25, 0, 300, 285, 450, 350];
x = xy(1,:) ;
y = xy(2,:) ;
plot(x,y)
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.

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

<table>
<thead>
<tr>
<th>Marker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>Circle</td>
</tr>
<tr>
<td>+</td>
<td>Plus sign</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>1</td>
<td>Point</td>
</tr>
<tr>
<td>x</td>
<td>Cross</td>
</tr>
<tr>
<td>s</td>
<td>Square</td>
</tr>
<tr>
<td>d</td>
<td>Diamond</td>
</tr>
<tr>
<td>&gt;</td>
<td>Upward-pointing triangle</td>
</tr>
<tr>
<td>&lt;</td>
<td>Downward-pointing triangle</td>
</tr>
<tr>
<td>v</td>
<td>Right-pointing triangle</td>
</tr>
<tr>
<td>c</td>
<td>Left-pointing triangle</td>
</tr>
<tr>
<td>p</td>
<td>Pentagon</td>
</tr>
<tr>
<td>h</td>
<td>Hexagram</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Solid line (default)</td>
</tr>
<tr>
<td>--</td>
<td>Dashed line</td>
</tr>
<tr>
<td>:</td>
<td>Dot dash line</td>
</tr>
<tr>
<td>-</td>
<td>Dash-dot line</td>
</tr>
</tbody>
</table>

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.

```matlab
cif - clear figure

f1 = figure;  % the figure function creates a new figure
p1 = plot(x,y,'r*');
hold on
p2 = plot(x,y);
p3 = plot(x,y,'bo');
delete(p1)  % you can delete elements by using its handle
```
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!

The built-in functions are also working on vectors:

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:

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:

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 "." before multiplication, division and in exponential expressions as follows:

\[ * \quad . \quad .^ \]

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:

```matlab
function [output1, output2] = functionname(input1, input2, ...)
    output1 = formula1
    output2 = formula2
end
```

About the functions:

- It should start with the keyword `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 ".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:

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

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

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

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

Save this as a separate file and lets present the results on a figure:

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

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