A Short Review of Matlab

MATLAB = "**Mat**rix **Lab**oratory"

https://www.mathworks.com/help/matlab/language-fundamentals.html
http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf

Working in the Command Window

● You can enter simple commands and call functions at the command line, indicated by the prompt >>
>> 1/3
ans = 0.3333

● The default format style for the decimal numbers in the output is **short**. In order to display more values in the decimal places you can set the output format **long**.
>>pi
ans = 3.1416
>>format long
>>pi
ans = 3.141592653589793

● In the following output
>>format short
>>1/2024
ans = 4.9407e-04

the letter "e" appears (from the word "exponent"), represents the number: $4.9407*10^{-4}$.

● Using **fprintf** you can display decimal numbers with a desired precision, text messages, etc.
>> fprintf('1/2024 is approximately %7.10f \n',1/2024)
1/2024 is approximately 0.0004940711

- Formats are specified with **%**:
  - %f (or more specific) for floating-point (real) numbers;
  - %d (or more specific) for integers;
  - %e for real numbers in the form mantissa/exponent.
- Formats and any messages are given in between **single** quotation marks **'**.
- The quotation marks (**'**) are followed by comma (**,**) and then variables; the number of formats must **match** the number of variables to be displayed
- \n stands for "new line", see also \t for "tab", etc.
- Everything inside the quotation marks, except the formats, will be displayed.

● The commands **help** and **doc** display the documentations for the specified functions/operators/etc.
>> help fprintf – displays information in the Command window
>> doc fprintf – opens the Help menu

Working with M-files (scripts and functions)

Matlab files have the extension **.m**, which is automatically added to the name.

● In a *script* file, you simply write all the commands to be executed. After being saved (e.g. under the name *my_first_script*), you run it in the Command Window by simply typing its name.

```
x=1;
y=2;
z=x+y
t=x*y
```

- **Caution!** The name **cannot** be:
  - usual mathematical or logical operations, *sin, cos, log, abs, exp, input, max, min, if, else, for, while*, etc.
  - a numeral;
  - **cannot** contain arithmetic operations, +, -, *, /, ^ (however, it can contain _)
- The semicolon symbol **;** suppresses the display of the result of any assignment (omitting the semicolon can be viewed as the **disp** command).

● You can also write an .m-file as a *function*.

```
function output = my_first_function(input)
output = NaN; %NaN="not a number"
if input ~= 1 %~= the "not equal to" logical operator
    return
else
    disp('Hello World!'); output = 1;
end
end
```

- **Caution!** The name under which the function file is saved **must match** the name of the function itself (for the above example: "my_first_function.m"). In order to call the function, first, you have to set the current folder to be the one where the file is saved.

```
>> x = my_first_function(1)
Hello World!
x = 1
>> x=my_first_function(0)
x = NaN
```

- In the above function, the semicolon **;** also allows multiple commands in the same line.
- The texts after **%** are comments.

● Examples of other logical operators: **==**, **<=**, **>=**, **<**, **>**, **&&**, ||.

Working with matrices and arrays

● Examples of operations/functions with matrices:

```
>> zeros(2,3)
ans =  0    0    0
       0    0    0

>> ones(2)
ans = 1    1
      1    1

>> eye(2,3)
ans =
   1    0    0
   0    1    0
```

```
>> eye(3)
ans =
    1    0    0
    0    1    0
    0    0    1

>> A = [1 2; 3 4]
A =  1    2
     3    4

>> B = [5; 6]
B = 5
    6

>> C = [7, 8, 9]
C =  7    8    9

>> D = [A B; C]
D = 1    2    5
    3    4    6
    7    8    9

>> D'
ans = 1    3    7
      2    4    8
      5    6    9

>> D(1, [3 1])
ans =  5    1

>> D(:, 2)
ans =  2
       4
       8

>> diag(D)
ans =
1
4
9

>> triu(D)
ans =
    1    2    5
    0    4    6
    0    0    9

>> tril(D)
ans =

    1    0    0
    3    4    0
    7    8    9
```

```
>> det(D)
ans = -2.0000

>> inv(D)
ans =
   6.0000  -11.0000    4.0000
  -7.5000   13.0000   -4.5000
   2.0000   -3.0000    1.0000
```

● Pay extra attention to the *matrix operations* * , / , ^, (without the dot .) and the *dot operations* (with the dot .) .*, ./ , .^, (they perform term-by-term operations)! Also, pay attention to the dimensions of the arrays/matrices!

```
>> D^2
ans = 42   50   62
      57   70   93
      94  118  164

>> D.^2
ans =  1    4   25
       9   16   36
      49   64   81

>> B^2
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.

>> [nrr, nrc]=size(D)
nrr = 3
nrc = 3

>> d = D(:)'
d = 1    3    7    2    4    8    5    6    9

>> d1 = d(3: end)
d1 =  7    2    4    8    5    6    9

>> d2 = d(1: end - 1)
d2 =  1    3    7    2    4    8    5    6
```

● Examples of numerical vectors with equally spaced elements:

```
>> v = 1 : 10
v =  1    2    3    4    5    6    7    8    9   10

>> v = 10 : -1 : 1
v =  10    9    8    7    6    5    4    3    2    1

>> v = 0 : 2 : 10
v =  0    2    4    6    8   10

>> v = 3 : -1.5 : -3
v = 3.0000    1.5000         0   -1.5000   -3.0000
```

```
>> v = linspace(1, 2, 11)
v = 1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000
```

● Examples of functions with vectors:

```
>> v = repmat([1 :3 ], 1, 3)
v = 1     2     3     1     2     3     1     2     3

>> length(v)
ans =  9

>> sum(v)
ans = 18

>> cumsum(v)
ans =  1     3     6     7     9    12    13    15    18

>> prod(v)
ans = 216

>> cumprod(v)
ans = 1     2     6     6    12    36    36    72   216

>> diff(v)
ans =  1     1    -2     1     1    -2     1     1

>> find(v==1)
ans =  1     4     7
```

Polynomials

```
>> my_poly = [1 -3 2]
my_poly = 1    -3     2

>> polyval(my_poly, 4)
ans = 6

>> roots(my_poly)
ans =
     2
     1
```

Working with symbolic variables, expressions, Symbolic Math Toolbox

```
>> syms x  e
>> e = exp(x)
e = exp(x)

>> subs(e,x,1)
ans = exp(1)

>> vpa(ans,10)
ans = 2.718281828

>> x = sym('1/2024')
```

x = 1/2024

>> vpa(x,10)
ans = 0.0004940711462

Working with logical statements

- Two implementations of the *double factorial* (examples for *conditional* and *loop control* statements: **if**, **else**, **elseif**, **for** and **while**):

```
function out = double_factorial_v1(n)
% n is a strictly positive integer
out = 1;
if mod(n, 2) == 0
    first = 2;
else
    first = 1;
end
for step = first : 2 : n
    out = out * step;
end
end

function out = double_factorial_v2(n)
% n is a strictly positive integer
out = n;
while n >= 3
    out = out *(n - 2);
    n = n - 2;
end
end
```

Working with function handles

- Example for "*function handle*":

>> f = @(x) cos(x).^2
f = function_handle with value:
    @(x)cos(x).^2

>> x = linspace(0, pi, 4)
x = 0 1.0472 2.0944 3.1416

>> f(x)
ans = 1.0000 0.2500 0.2500 1.0000

Graphics

- Commands **plot**, **plot3**, **mesh**, **title**, **legend**, **subplot**, **axis**, **xlabel**, with all the options.

>> help plot

Clearing commands

- **clear *var*** clears the value of the variable *var*;
- **clear all** removes items (all variables) from *Workspace*, freeing up the memory;
- **clc** clears all the text from the *Command Window*;
- **clf** clears the figure