# SIGNAL PROCESSING I

## Introduction to Matlab

**MATLAB** is an interactive, matrix-based system for scientific and engineering numeric computation and visualization. Its strength lies in the fact that complex numerical problems can be solved easily and in a fraction of the time required by a programming language such as Fortran or C. Matlab It has its own script language. This language it is very easy to use. Another advantages, is connected with the fact simple programming capability, MATLAB can be easily expanded to create new commands and functions. You can easily and quickly build complex algorithms, using elements such as filters (eg. Low-pass filter, high-pass filter, band (reject) filter, adaptive filers etc), Transform (eg. Wavelet, Fourier, Hough and many others).

# 1. Introduction to Matlab script language.

## 1.1 Basics of language: variables, arrays, matrix, mathematical operations...

Variables:

variable_name = value;

Examples:

- *integer_number = 2;*
- *integer_number = -25;*
- *integer_number = 10e5;*
- *real_number = 3.14;*
- *real_number = -0.75;*
- *real_number = 0.1e5;*
- *complex_number = 1+2i;*
- *complex_number = 10-i;*
- *String_vaeiable = 'some text';*

To display value of variable just type name of this variable:

```
>>integer_number

>> integer_number  =

ans =
        2
```

Arrays and Matrix:

*array_name = [value1, value2, … ,valueN];*

*array_name = zeros(rows,cols);* % create matrix with rows and cols and fill zeros

*array_name = ones(rows,cols);* % create matrix with rows and cols and fill ones.

How to use it ?

To view the entire array just type name of this array

*A = [1 2 3 4 5];*

*>>A*

*A =*
*1 2 3 4 5*

To display the specified item (s) of the matrix, enter the coordinates of this item:
Warning! Matlab arrays and matrices are numbered from 1 (not 0)!

*matrix = [1 2 3 ; 4 5 6 ; 7 8 9];* %create 3x3 matrix (as below)

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

matrix(row,cols) %display element on position row, cols

*>>matrix(1,1) =*

*ans = 1;*

*>>matrix(1,3) =*

*ans = 3;*

*>>matrix(3,1) =*

*ans = 7;*

To display the specified **row** just type:

*>>matrix(1,:) =*

*ans = 1 2 3;*

*>>matrix(1,1:2) =*

*ans = 1  2 ;*

To display the specified **col** just type:

*>>matrix(2:3,1) =*

*ans =*
*4*
*7*

One way to enter a n-dimensional array (n>2) is to concatenate two or more (n-1)-dimensional arrays using the `cat` command. For example, the following command concatenates two 3x2 arrays to create a 3x3x2 array:

```
>> C = cat(3,[1,2;3,4;5,6],[7,8;9,10;11,12])
C(:,:,1) =
     1     2
     3     4
     5     6
C(:,:,2) =
     7     8
     9    10
    11    12
```

Transpose of Array / Matrix:

*>>transpose_matix = matrix'*

*ans =*

| | | |
|---|---|---|
| *1* | *4* | *7* |
| *2* | *5* | *8* |
| *3* | *6* | *9* |

<u>Basic mathematical operation (small part of Matlab operatos...):</u>

Let assume that:
*a = 2*
*b = 3*
*d = 10+6i;*

Then:

*>>a+b*

*ans =*
*5;*

*Similarly behave other operators:*
*- minus*
** multiply*

*/ divide*
*^ - power (ex. 5^3)*
*<> - relational operation*
*| - logical OR*
*& -logical AND*
*! -negation (logical NOT)*
*== -equality*

*mod(a,b) – modulo*
*abs(a) - absolute value*
*roud(a) - r*ound to nearest decimal or integer
ceil(a) - round toward positive infinity
*sin() - s*ine of argument in **radians**

and...
*asin(), cos(), ascos(), tan(), atan()..........*

*exp() - Exponential*
*log() - Natural logarithm*
*log10() - Common logarithm (base 10)*
*log2() -Base 2 logarithm and dissect floating-point numbers into exponent and mantissa*
*sqrt() - Square root*

compex numbers:

```
 >>complex(d)
ans =
        10.0000 + 6.0000i


 >>real(d)              %real part
ans =
        10.0000

 >>imag(d)              %imaginary par
ans =
        6.0000i
```

Arrays and Matrix Basic Operations:

Let assume that:
*A = [1 2 3];*
*B = [4 5 6];*

```
>>A+B
ans =
        4       7       9


>>A-B
ans =
        -2      -3      -3
```

*>>A+B*
ans =
4 7 9

*>>A.*B* *%multiply value by value*
ans =
3 10 18

*>>A*B*
Error ! Matrix dimensions must agree!
Buy, we can transpose own array...

>>A.*B'
ans =
31.

*>>sum(A)* %sum of all elements of A
*ans =*
*6*

*>>sum(A(1:2))* %sum of all elements on position (1,1) and (1,2 ) of A
*ans =*
*3*

*TableA = [1:100000]* *%Create table from 1 do 100000.*

*length_table = 1000;*
*TableA = [1:length_table]* *%Create table from 1 do 1000.*

<u>Control Flow</u>

Example of *if-elseif-else* structure. *Command1* is executed only if *condition1* is satisfied. Otherwise, if *condition2* is satisfied, the *command2* is executed.
If any *condition* are satisfied, the *command3* is executed.
(The number of *if-else* is unlimited)

```
if condition1

      command1

elseif condition2

      command2

else

      command3

end
```

Example of for loop:

*for variable=begin : ratio : endCondition*
…
*end*

*begin – start loop value*
*ratio – tell how “variable” will be change on each iteration (*increase *by 1,2,.... or* decrease by 1,2,...*)*
*endCondition – loop stop condition1*

*examples:*

*for x=1: 1 : 5*
*x*
*end*

result:

*1*
*2*
*3*
*4*
*5*

*for x=1: 2 : 5*
*x*
*end*

result:

*1*
*3*
*5*

*for x=1: -1 : 5*
*x*
*end*

result:

*1*
*0*
*-1*
*-2*
*-3*
*-4*
*-5*

Example of while loop

```
x = 5;
while(x>0)          %stop loop condition
    x
    x = x – 1;
end
```

result:

```
    5
    4
    3
    2
    1
```

Until stop loop condition isn't  satisfied loop is executed

Warning!
We cannot use Incrementation and decrementation operator !

<u>Own function</u>

```
function [outPar1,outPar2,...,outParN] = functionName(inputPar1,inputPar2,...,inputParN)
…
end
```

*inputPar1,inputPar2,...,inputParN - inputParameters*
*outPar1,outPar2,...,outParN – Output parameters*
*functionName – our function name*

Function code should be write on separate file (and file name should be identical as function name) eg. add.m (".m" is Matlab script file extension, sometimes it's called as m-files).

example:

```
function [out] = add(num1, num2)
out = num1 + num2;
end
```

```
>> add(2,5)   % calling our function

ans =
    7
```

Ofcorse this is only a smart part of functionality of Matlab.
See also [https://www.mathworks.com/help/index.html](https://www.mathworks.com/help/index.html)

# 1.2 Signal Processing  - Basic examples and exercises

<u>Generating a simple pdic signal.</u>

```
%% signal parameters
N=1000;                    % number of samples
A=5;                       % amplitude
fx=10;                     % frequency (in Hz)
fp=1000;                   % sampling frequency (in Hz)
dt=1/fp;                   % sampling period
t=dt*(0:N-1);              % vector of sampling moments
x=A*sin(2*pi*fx*t);         % signal
plot(t,x); grid; title('Signal x(t)'); xlabel('Time [s]');
pause
% calculation of statistical signal parameters:
x_sred1=mean(x), x_sred2=sum(x)/N  % average
x_max=max(x)                                          % max value
x_min=min(x)                                           % min value
x_std1=std(x), x_std2=sqrt(sum((x-mean(x)).^2) / (N-1))   % standard deviation
x_eng=dt*sum(x.^2)                                    % signal energy
x_moc=(1/N)*sum(x.^2)                                 % average power
x_skut=sqrt(x_moc)   % effective value
```

<u>Exercises:</u>

1. Create function “sinusSignalGenerate” with parameters:
   - samples number
   - samples frequency
   - signal amplitude
   - signal frequency

The output of function should be a array with generated signal.

1.1 Create function for displaying signal (from array).

1.2 Amplitude = 1, frequency = 10KHz, sample frequency = 2000Hz, number of samples = 4000.

- Show result. Whether it is a correct sinusoidal signal ? Why is incorrect ? Do you can correct this ?

1.3 Generate and show signals:

- Amplitude = 2, frequency = 1000Hz, signal Time = 2s
- Amplitude = 5, frequency = 1500Hz, signal Time = 1s
- Amplitude = 1, frequency = 25000Hz, signal Time = 0.5s

What should be the parameters (number of samples and samples frequency) for each signal?

2. Generate and show signals (adjust number of samples and samples frequency for second signal):
- Amplitude = 1, frequency = 50Hz, signal Time = 2s
- Amplitude = 1, frequency = 100Hz, signal Time = 2s

2.1 Add both signal. Show results.
2.3 Generate and show signals (adjust number of samples and samples frequency for second signal):
- Amplitude = 1, frequency = 50Hz, signal Time = 2s
- Amplitude = 1, frequency = 10000Hz, signal Time = 2s

2.1 Multiply both signal. Show results.

3. Create function for displaying Fourier spectrum of the signal:

```
% y – signal, fp – sample rate (frequency sampling)

function FFTspectrum( y,fp )
dt=1/fp;
N = size(y,2);
X = fft(y);                     % Fast Fourier Transform
        df = 1/(N*dt);          % basic frequency f0 = df = 1/T = 1/(N*dt)
        f = df * (0 : N-1);     % more frequencies in the Fourier series

        subplot(311); plot(f,real(X)); grid; title('Real(X)'); xlabel('Hz');
        subplot(312); plot(f,imag(X)); grid; title('Imag(X)'); xlabel('Hz');
        subplot(313); plot(f,abs(X)); grid; title('Abs(X)'); xlabel('Hz');
end
```

3.1 Show signal from point 2 and 2.3

4. Designing a simple low-pass filter.

```
% signal – pinput signal
%  filterOrder – filter Order
%  cutOffFrequency – cut-off frequecny (all signals with a frequency higher than cut-off frequecny will be suppressed)
% sampligFrequency -  samplig Frequency
%  filterResponseDisp [true/false] - whether to display the impulse response filter?

function [y] = lowPassFIlter(signal,filterOrder, cutOffFrequency, sampligFrequency, filterResponseDisp)

% 'N,Fc' – get filter order and  samplig frequency to desing filter
d = fdesign,lowpass('N,Fc',filterOrder, cutOffFrequency, sampligFrequency);
designmethods(d);
Hd = design(d);

        % if  'filterResponseDisp' is true then show the  impulse response filter
        if (filterResponseDisp)
                fvtool(Hd)
        end
end
```

4.1Using signal from point 2 try to remove (suppressed) all signals with a frequency Higher than 50Hz.  Results of removing signal you can check by using signal spectrum (FFT).