Introduction to MATLAB for signal processing

1.1 Setup and Documentation

MATLAB is a high-level programming language utilised in the different areas of numerical computing, it includes both a command line interface and a script interpreter. The language is most efficiently used when solutions to various problems are implemented in the form of matrix computations. Moreover, it is predominantly used for numeric analysis, signal processing and graphical representations in the field of engineering disciplines.

MATLAB is a proprietary software solution. To install it, access the link at: https://www.mathworks.com/help/install/install-products.html and select the MATLAB product that you intend to install. Because the language falls under proprietary software, installing it will require a license which is either bought on an individual basis or is provided to you by a university/institution.

Documentation for programming with MATLAB can be found at the following link: https://www.mathworks.com/help/MATLAB/.

1.2 Short Introduction to Mathematical Processing

There are two types of MATLAB programs:

- Scripts, which consist of sequences of commands that use predefined MATLAB functions;
- User-defined functions, which have a specific number of input parameters and include the operations associated to processing these parameters, the final results being returned as output parameters.

When opening MATLAB, the path to the current directory (the working directory) is displayed, together with the following 4 windows (Figure 1):

- The Command Window, where instructions are typed and executed in real time;
- The Workspace, where the currently defined variables can be observed;
- The Current Folder, where files can eventually be saved;
- The Script Editor, with which scripts can be added to the working directory. All of the commands written in the editor can be executed successively (by pressing F5). Alternatively, only the selected commands in the editor can be executed (by pressing F9).

If the description above does not match what is displayed when opening MATLAB, you can proceed as indicated in Figure 2, by changing the layout to its default configuration.

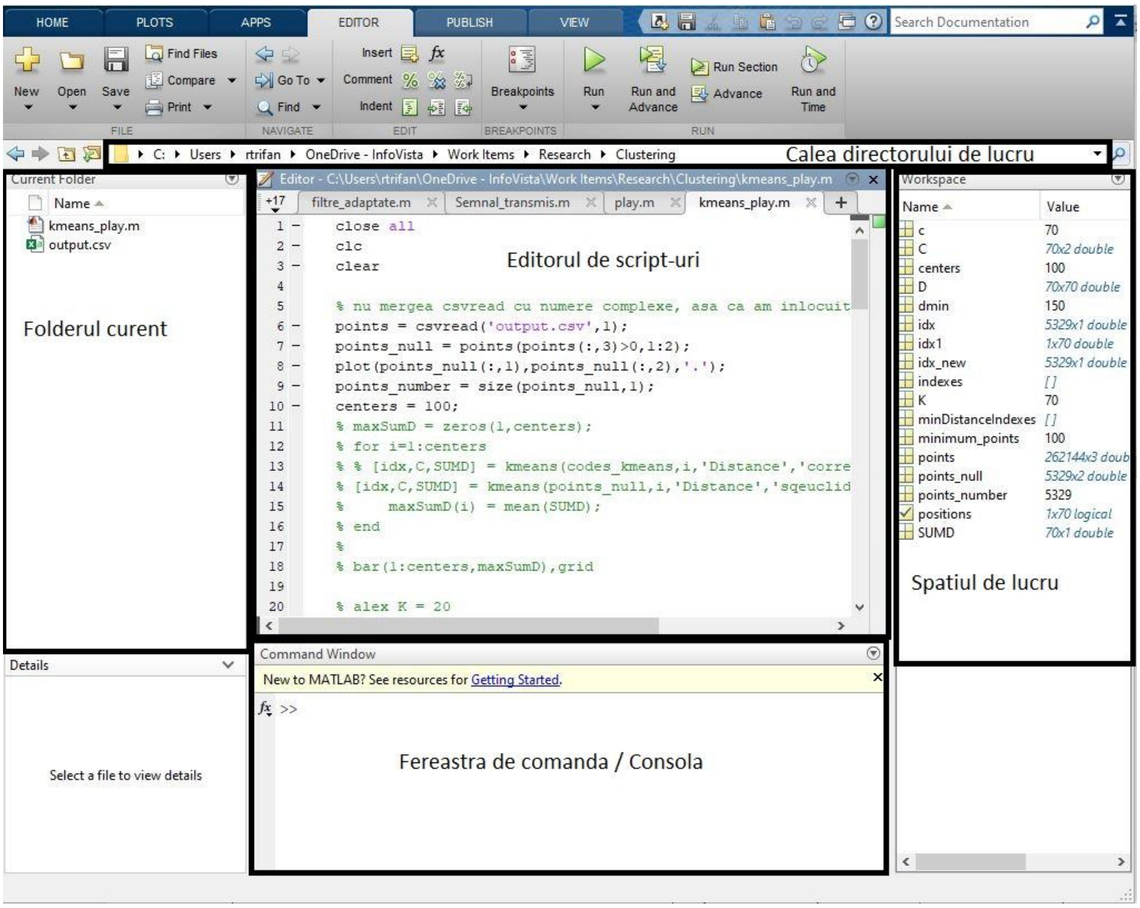


Figure 1 MATLAB Interface

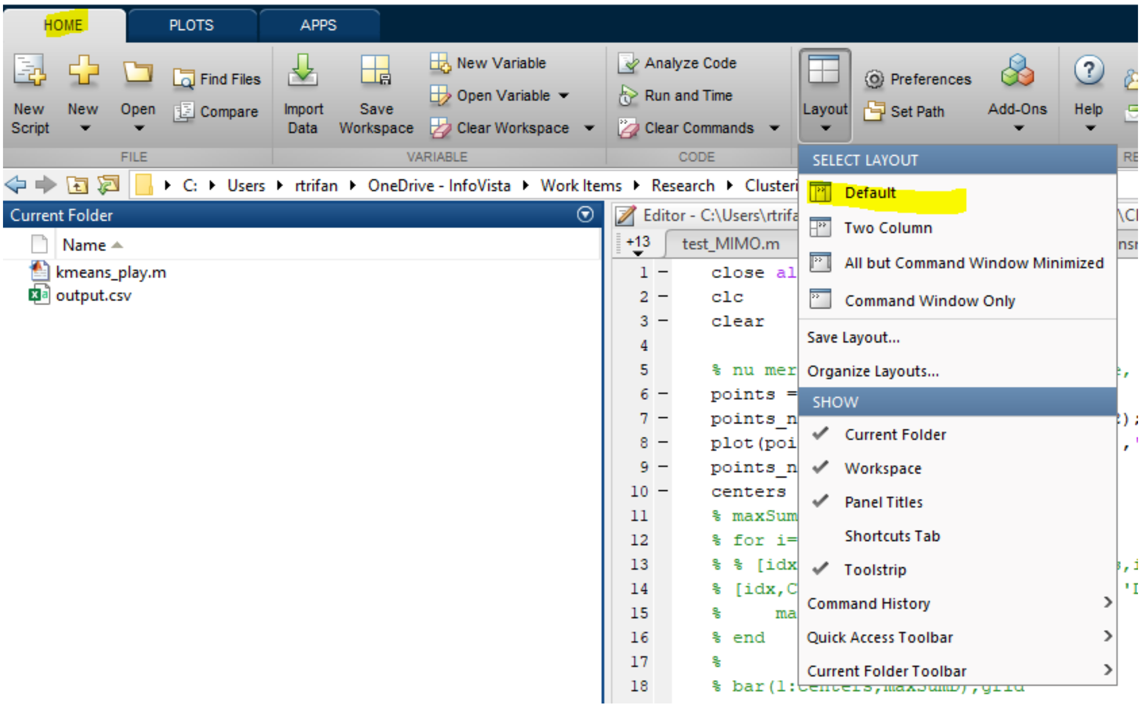


Figure 2 Default layout

1.3 Basic MATLAB Functions

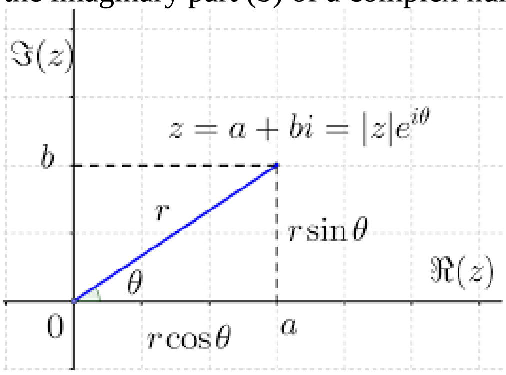
Table 1.1 lists the main functions and parameters that are available in MATLAB, together with their descriptions and certain examples.

Table 1.1 Basic MATLAB Functions

<table>
<tr><th>Function/Parameter</th><th>Description and Examples</th></tr>
<tr><td>help function_name</td><td>Function used for providing documentation on the usage of other MATLAB functions</td></tr>
<tr><td>ans</td><td>The default name that is given to the result of an operation executed in the workspace or the command window</td></tr>
<tr><td>pi</td><td>The value of the “pi” constant</td></tr>
<tr><td>Inf,
NaN</td><td>Infinity
Undefined value error — Not a Number</td></tr>
<tr><td>+, -, *, /, ^,</td><td>The addition, subtraction, multiplication, division and exponentiation of numbers, vectors or matrices
Example:
A = [1 2 3], B = [4 5 6]'
A*B = 32</td></tr>
<tr><td><, <=, >, >=, ==, ~=</td><td>The comparison operators:
“lower than”, “lower or equal to”, “greater than”, “greater or equal to”, “equal” and “not equal”</td></tr>
<tr><td>&, | , ~</td><td>AND, OR, NOT logical operators</td></tr>
<tr><td>i, j, 1i, 1j</td><td>$i = j = \sqrt{-1}$</td></tr>
<tr><td>abs(z)

angle(z)

real(z)

imag(z)</td><td>The magnitude (r), phase (θ, in radians), real part (a) and the imaginary part (b) of a complex number z:


Example:
z1 = 7;
z2 = -7;
z3 = 1i * 7;
z4 = -1i * 7;

real(z1), real(z2), real(z3), real(z4)
imag(z1), imag(z2), imag(z3), imag(z4)
abs(z1), abs(z2), abs(z3), abs(z4)
angle(z1),angle(z2), angle(z3), angle(z4)
% radians</td></tr>
</table>

`cos(x), sin(x), tan(x)`	Cosine, sine and tangent functions of the angle *x*
`acos(x), asin(x),` `atan(x)`	Inverses of the cosine, sine and tangent functions, in radians
`deg2rad(x), rad2deg(x)`	Conversion of an angle *x*, from degrees to radians and from radians to degrees, respectively
`exp(x), log(x), log10(x)`	Exponential function, natural logarithm and base-10 logarithm

`ceil(x)`
 	
`floor(x)`
 	
`round(x)`
	Rounding *x* to the next whole number, rounding to the previous whole number and rounding to the nearest whole number
	Example:
	
	`floor(2.6) = 2`
	`ceil(2.6) = 3`
	`round(2.6) = 3`

`A.*B`	Element-wise multiplication of 2 matrices or vectors *A* and *B*
	Example:
	`A = [1 2 3], B = [4 5 6]`
	`A.*B = [4 10 18]`
	
	`A = [7 8 9]; % line vector`
	`B = [10; 20; 30]; % column vector`
	`z2 = A .* A`
	`z3 = A .* B`
	`z4 = B .* B`
	
	`A = [1 2 3; 4 5 6] % 2 x 3 matrix`
	`B = [10, 20, 30; 40, 50, 60] % 2x3 matrix`
	`A.*B`

`A./B`	Element-wise division of 2 matrices or vectors *A* and *B*
`A.^B`	Element-wise exponentiation of 2 matrices or vectors *A* and *B*

`A’`
 	
`A.’`
	`A =`
	`7.0000 + 9.0000i   10.0000 + 5.0000i`
	`7.0000 + 7.0000i    2.0000 +10.0000i`
	`2.0000 + 2.0000i    1.0000 + 2.0000i`
	`2.0000 + 4.0000i    6.0000 + 9.0000i`
	
	Transpose and complex-conjugate of a matrix *A*
	`>> A'`
	`ans =`
	`7.0000 - 9.0000i   7.0000 - 7.0000i   2.0000 - 2.0000i   2.0000 - 4.0000i`
	`10.0000 - 5.0000i   2.0000 -10.0000i   1.0000 - 2.0000i   6.0000 - 9.0000i`
	
	Transpose of a matrix *A*
	`>> A.'`
	`ans =`
	`7.0000 + 9.0000i   7.0000 + 7.0000i   2.0000 + 2.0000i   2.0000 + 4.0000i`
	`10.0000 + 5.0000i   2.0000 +10.0000i   1.0000 + 2.0000i   6.0000 + 9.0000i`

`conj(A)`	Complex conjugate of a matrix A
	`>> conj(A)`
	`ans =`
	`7.0000 - 9.0000i   10.0000 - 5.0000i`
	`7.0000 - 7.0000i    2.0000 -10.0000i`
	`2.0000 - 2.0000i    1.0000 - 2.0000i`
	`2.0000 - 4.0000i    6.0000 - 9.0000i`

`x=start:step:stop`	Generation of a vector x with its first element *`start`*, its last element *`stop`*, having its elements evenly spaced by *`step`*
	Example:
	`>> i = 1:2:5`
	`i =`
	`1     3     5`

`x=linspace(start,stop,n)`	Generation of a vector x consisting of n evenly spaced out elements, starting from `start` and ending with `stop`
	Example:
	`>> i = linspace(1,5,3)`
	`i =`
	`1     3     5`

`A=[]`	An empty matrix

`A=[x1;x2]`	A matrix whose lines are the x_1 and x_2 vectors
	Example:
	`>> x1 = [1 3 5];`
	`>> x2 = [5 6 7];`
	`>> X = [x1;x2]`
	`X =`
	`1     3     5`
	`5     6     7`

`A=[x1c, x2c]`	A matrix whose columns are the x_{1c} and x_{2c} vectors (by concatenation)
	Example:
	`x1c =`
	`1`
	`3`
	`5`
	`x2c =`
	`5`
	`6`
	`7`
	`>> X = [x1c,x2c]`
	`X =`
	`1     5`
	`3     6`
	`5     7`

`ones(N,M)`
 	
`zeros(N,M)`
 	
`eye (N,M)`
	N rows, M columns matrix, full of ones;
	
	N rows, M columns matrix, full of zeroes;
	
	N rows, M columns matrix, all elements null except for

	the main diagonal which consists of ones
	Example:
	`eye(4) = eye(4,4)`
	`>> eye(4)`
	`ans =`
	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`

`rand(N,M)`
 	
`randn(N,M)`
 	
`randi([min,max],N,M)`
	N rows, *M* columns matrix, containing random values in the interval (0,1), according to the uniform distribution (no value is more likely to occur than other values)
	
	N rows, *M* columns matrix, containing random values corresponding to the Gaussian distribution with a null mean and a standard deviation of 1
	
	N rows, *M* columns matrix, containing random integer values, according to the uniform distribution on the interval (`min`, `max`)

`A(i,j)`	The element found in the matrix *A* at line *i* and column *j*
	`>> A = randi(5,5)`
	`A =`
	`1 5 1 3 3`
	`5 4 2 1 2`
	`3 2 1 5 5`
	`3 3 1 5 2`
	`1 3 2 3 1`
	`>> A(1,3)`
	`ans =`
	`1`

`A(i,:)`	Line *i* of matrix *A*
	`>> A(2,:)`
	`ans =`
	`5 4 2 1 2`

`A(i:j,:)`	Lines *i* through *j* of matrix *A*
	`>> A(2:3,:)`
	`ans =`
	`5 4 2 1 2`
	`3 2 1 5 5`

`A(i:k:j,:)`	Lines *i, i+k, i+2k,…, j* of matrix *A*
	`>> A(1:2:5,:)`
	`ans =`
	`1 5 1 3 3`
	`3 2 1 5 5`
	`1 3 2 3 1`

<table>
<tr><td><code>A([i,j,k],:)</code></td><td>Only the lines i,j,k of matrix A
<code>>> A([1,3,5],:)</code>
<code>ans =</code>
<code>1 5 1 3 3</code>
<code>3 2 1 5 5</code>
<code>1 3 2 3 1</code></td></tr>
<tr><td><code>A(:,j)</code></td><td>Column j of matrix A
<code>>> A(:,2)</code>
<code>ans =</code>
<code>5</code>
<code>4</code>
<code>2</code>
<code>3</code>
<code>3</code></td></tr>
<tr><td><code>A(:,i:j)</code></td><td>Columns i through j of matrix A
<code>>> A(:,2:3)</code>
<code>ans =</code>
<code>5 1</code>
<code>4 2</code>
<code>2 1</code>
<code>3 1</code>
<code>3 2</code></td></tr>
<tr><td><code>A(:,i:k:j)</code></td><td>Columns i, i+k, i+2k,…, j of matrix A
<code>>> A(:,1:2:5)</code>
<code>ans =</code>
<code>1 1 3</code>
<code>5 2 2</code>
<code>3 1 5</code>
<code>3 1 2</code>
<code>1 2 1</code></td></tr>
<tr><td><code>A(:,[i,j,k])</code></td><td>Only the columns i,j,k of matrix A
<code>>> A(:,[1,3,5])</code>
<code>ans =</code>
<code>1 1 3</code>
<code>5 2 2</code>
<code>3 1 5</code>
<code>3 1 2</code>
<code>1 2 1</code></td></tr>
<tr><td><code>size(A)</code></td><td>Number of lines and respectively columns of matrix A
<code>A = []</code>
<code>size(A)</code>
<code>ans =</code>
<code>0 0</code>
<code>A = 72</code>
<code>size(A)</code>
<code>ans =</code>
<code>1 1</code>
<code>A = randn(3,6)</code>
<code>size(A)</code>
<code>ans =</code>
<code>3 6</code></td></tr>
</table>

`length(x)`	Number of lines/columns of the *x* vector
	`A = []`
	`length(A)`
	`ans =`
	`     0`
	`A = 72`
	`length(A)`
	`ans =`
	`     1`
	`A = randn(3,6)`
	`length(A) % largest dimension`
	`ans =`
	`     6`

`mean(x)`, `sum(x)`, `min(x)`, `max(x)`	Mean, sum, minimum value and maximum value of the *x* vector

`find(x)` or `find(x~=0)`
 	
`find(condition)`
	Returns the positions inside the *x* vector that contain values that are non-null or that satisfy a given condition:
	`find(x)` or `find(x~=0)`
	`>> find(x)`
	`ans =`
	`     2     3     4     5     6`
	`find(x>5)`
	`>> find(x>5)`
	`ans =`
	`     4     5     6`

`x(find(x))` or `x(x~=0)`
 	
`find(condition)`
	Returns the values inside the *x* vector that are non-null or that satisfy a given condition:
	`x(find(x))` or `x(x~=0)`
	`>> x(find(x))`
	`ans =`
	`     2     4     6     8    10`
	`>> x(x~=0)`
	`ans =`
	`     2     4     6     8    10`
	`>> x(x>5)`
	`ans =`
	`     6     8    10`

`for i=start:step:stop`
`    commands`
`end`
	"For" loop
	`x = [7 8 9]; % line`
	`y = [10; 20; 30]; % column`
	`r = 0; % run a multiply-accumulate operation`
	`for index=1:length(y)`
	`    r = r + x(index)*y(index);`
	`end`
	`r % 500`

`if condition`
`      commands`
`else/elseif`
`      commands`
`end`
	"If/else/elseif" conditional statements

`while (condition)`
`    commands`
`end`
	"While" loop

`plot(x,y)`
 	
`stem(x,y)`
	Continuous graphical representation of the points given by the *x* and *y* vectors (using linear interpolation)
	
	Discrete graphical representation (segments) of the points given by the *x* and *y* vectors

`Figure`	Initialising a figure before plotting a graph
`grid`	Display a grid on the plot
`xlabel('text'), ylabel('text')` `title('text')`	Set the Ox and Oy axis titles, respectively Set the title of a figure
`clear, clc, clf, close all`	Clearing all variables stored in the memory, clearing the command line history, closing the currently selected figure, closing all figures

Exercises

Create a working directory (on the Desktop). Copy the path of this directory in MATLAB. Then create a new script file sand save it in the current directory.

Generate a square matrix A with 10 rows and 10 columns, which contains random integer values according to a uniform distribution on the interval 1:20.

1. Create new square matrices, each with 10 rows and 10 columns, starting from matrix A so that they contain:

   a. Only the even elements of A;
   b. Only the elements on A's main diagonal.

2. Generate a vector of complex numbers whose real part consists of the 2nd row of matrix A, and its imaginary part consists of the 4th column of matrix A. Determine:

   a. The vector that contains the magnitudes of the elements of the complex number vector and the vector containing the phases of the elements of the complex number vector;
   b. The sum of the elements of the element-wise product of the complex numbers vector with its conjugate (try to achieve the desired sum without using a FOR loop).

3. The `min`, `max`, `sum` and `mean` functions can be used on both matrices and vectors. When used on matrices, they can act on specific rows or columns. Additionally, the `min` and `max` functions can return the index of the minimum/maximum value that was found.

   a. Determine the maximum value of each column of matrix A;
   b. Norm each column of matrix A with respect to the maximum values calculated in point a.;
   c. Determine the maximum global value and fins its position in the matrix A.

Examples

1. Given two complex numbers, z_1 and z_2:
   a. Display the real and imaginary part of z_1 and z_2;
   b. Calculate the real number a, represented by the real part of the sum between z_1 and the conjugated z_2;
   c. Calculate the angle represented by the sum of angles corresponding to z_1 and z_2 in the complex plane and convert the value to degrees;
   d. Calculate the natural logarithm of number a

Example of command usage (exercise solution):

```
clear all; clf
z1 = 3 + j*5;
z2 = -9 + 3*j;
z1real = real(z1);
z1imaginary = imag(z1);
a = real(z1+conj(z2));
SumAngRadians = angle(z1) + angle(z2);
SumAngDeg = SumAngRadians*180/pi;
NatLog = log(a);
```

1.4 Generating Signals in MATLAB

In MATLAB, a signal is represented by a vector (for one-dimensional signals), a matrix (for two-dimensional signals), or a sequence of matrices (in case 3 or more dimensions are needed). These data structures contain the values that are obtained by sampling continuous signals, the sampling process may depend on either time or two spatial variables (i.e.: an image).

1.4.1 Harmonic Signals

The sinusoidal signal (sine wave) is defined as follows:

$$x(t) = sin(\omega_0 t)$$

Where: $\omega_0 = 2\pi f_0$ is the angular frequency of the signal,
f_0 is the frequency of the signal,
t is the time variable of the sine function.

The code below can be used to generate and visualize a 10Hz sinusoidal signal:

```
close all
clc
clear

% frequency in Hz
f0 = 10;
omega0= 2*pi*f0;
step = 0.001;
% Tmax = 500 ms
T_max= 500*10^-3;
% obtaining the time vector
t = 0:step:T_max;
% obtaining the signal as a function of time
x = sin(omega0*t);

figure (1), plot (t, x), grid;
xlabel('Time [s]');
ylabel('Amplitude');
title('x = sin(omega0*t)');
```

As seen above, the moments of time for which the sine function was evaluated were defined in the first place, as follows:

```
t = 0:step:T_max;
```

It is important that the step value is small enough for the plot to be done correctly (smaller step value — higher resolution, less interpolation). Thus, we considered the step to be 0.001. After executing the code, the image in Figure 3 is obtained:

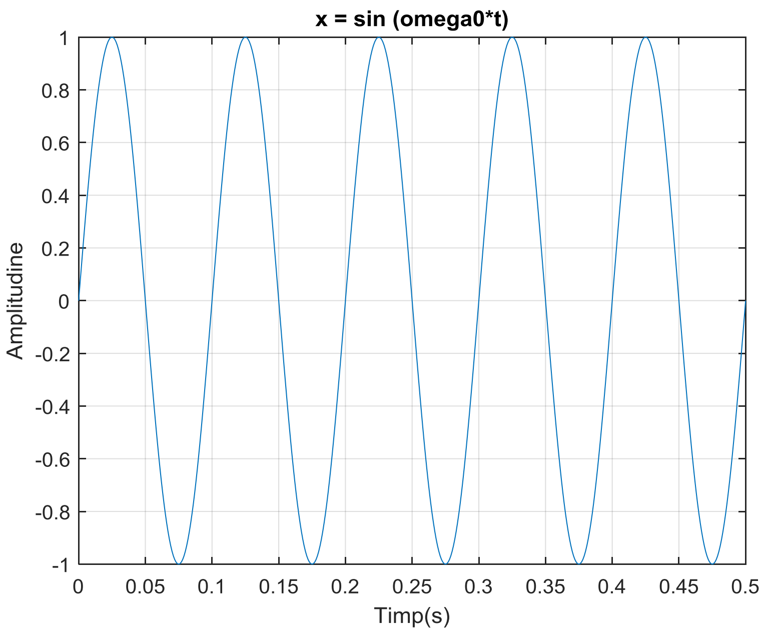


Figure 3. Sinusoidal signal

Exercises:

1. What happens to the plot's precision when the step value becomes 0.01?
2. What happens to the plot's precision when the frequency value becomes 100Hz?

   Hint: *F0< FNyquist = Fs/2*
   Ts = step = 1/Fs
3. Display the previous harmonic signal by marking intermediate points with "o". What operation does the `plot` function perform?

```
figure(1), plot(t,x,'o-'), grid
```

4. Plot the following signals:

$$x_1[n] = \left(\frac{1}{2}\right)^n - \left(-\frac{1}{2}\right)^n, \text{for } 0 \leq n \leq 10$$

$$x_2[n] = \ln\left|\cos\left(\frac{n\pi}{15}\right) - \sin\left(\frac{n\pi}{15}\right)\right|, \text{for } -20 \leq n \leq 20$$

$$x_3[n] = (-1)^n \cos\left(\frac{n\pi}{15}\right), \text{for } 0 \leq n \leq 10$$

1.4.2 Square Wave Signals

Periodic square wave signals can be generated by two methods:

- With the help of the `square` function for the direct creation of the periodic signal;
- Using the `rectpulse` function to create a single rectangle, which is then turned periodic by successive concatenation.

The code below can be used to generate and visualize rectangular signals created by the two previously mentioned methods:

```
clc
clear all
close all

step = 0.0001;
tmin=-5;
tmax=5;
t1 = tmin:step:tmax;
frecv=1;
x1 = square(2*pi*frecv*t1);

t2max=0.5;
t2min=-0.5;
```

```
t2p = t2min:step:t2max-step;
x2p = rectpuls(t2p, 0.5);

no_periods=10;
x2=[];
for i=1:no_periods
    x2=[x2,x2p];
end
t2 = no_periods*t2min:step:no_periods*t2max-step;
x3 = x2 *2 - 1;
size(x2p)
size(x2)
size(t2)

figure
subplot(4,1,1), plot(t2p,x2p)
xlabel('Time [s]'), ylabel(‘Signal Level’), grid
subplot(4,1,2), plot(t1,x1)
xlabel('Time [s]'), ylabel(‘Signal Level’), grid
subplot(4,1,3), plot(t2,x2)
xlabel('Time [s]'), ylabel(‘Signal Level’), grid
subplot(4,1,4), plot(t2,x3)
xlabel('Time [s]'), ylabel(‘Signal Level’), grid
```

After executing the code, the plot in Figure 4 is obtained:

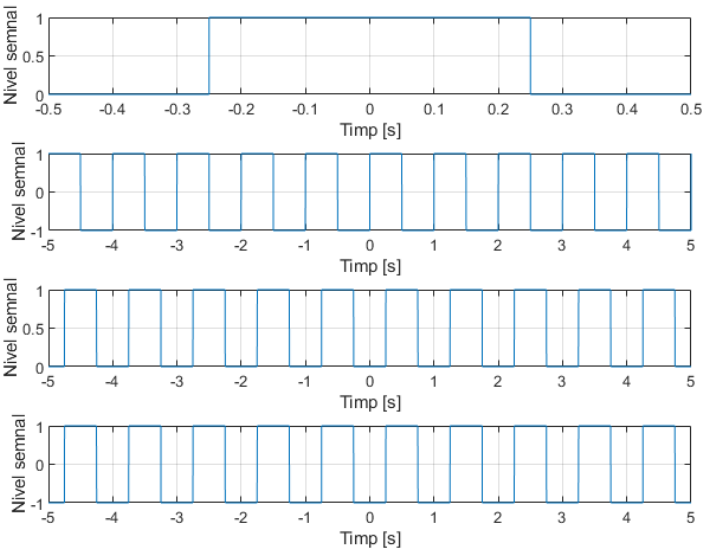


Figure 4. Square signals

1.4.3 Triangle Wave Signals

Repeat the previous exercise to generate and visualize triangular signals using the `tripulse` and `sawtooth(t,0.5)` functions.

1.5 Defining Functions in MATLAB

In MATLAB, functions are defined through the following syntax:

```
function [y1,y2,…yn]=function_name(x1,x2,…,xn)
```

Where: `x1,x2,…,xn` are the input parameters,
`[y1,y2,…,yn]` is the vector of output parameters, obtained by processing the input parameters.

For instance, the function call `[M,N] = size(X)` takes the matrix *X* as an input parameter and, through certain processes described by the `size()` function, returns a vector with 2 elements:

- First element: the number of lines *M*;
- Second element: the number of columns *N*.

Functions are essential for modularising complex programs. By using functions, you can turn a script with a large number of statements into a more organized program consisting of several functions defined in separate files. This approach to code organization allows you to reuse portions of code, thus contributing to more efficient and maintainable development.

When you create your own functions in MATLAB, you can do this by creating new files with the .m extension in the directory where you develop your main script. It is preferable that these new files have the same name as the name of the respective function.

Any .m file that defines a function begins with the syntax:

```
[y1,y2,…yn]=function_name(x1,x2,…,xn)
```

After this syntax, it is recommended to provide explanations for each output parameter (`y1, y2, …`) and for each input parameter (`x1, x2, …`), as well as a short, human-readable description of the processes carried out by the function.

Example

Define the step function using MATLAB, given its expression:

$$u(n) = \begin{cases} 1 & , \quad n \geq 0 \\ 0 & , \quad n < 0 \end{cases}$$

By using the time-translation property, this can be written as:

$$u(n - n_0) = \begin{cases} 1 & , \quad n \geq n_0 \\ 0 & , \quad n < n_0 \end{cases}$$

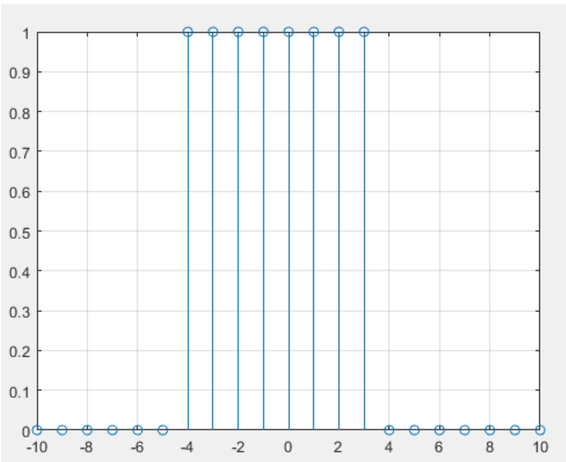
As shown below, a MATLAB function can be created for defining unit-step discrete sequences, having a finite temporal basis:

```
function [y,n] = treapta(ni,ns,n0)
    % Discrete time step function
    % Output parameters:
    % y = u(n-n0) (line vector) on the ni:ns basis
    % n = the ni:ns temporal basis
    % Input parameters:
    % ni = lower limit of the temporal basis
    % ns = upper limit of the temporal basis
    % n0 = index for u(n-n0)
    N = ns-ni+1;
    Y = zeros (1, N);
    y(n0-ni+1:N) = 1;
    n = ni:ns;
end
```

Exercises

Define and graphically represent the following sequences:

1. $y_1[n]=u[n]$
2. $y_2[n]=0.7\cdot(u[n+3]-u[n-3])$
3. $y_3[n]=u[n]+0.5u[n-4]-0.5u[n+4]$
4. $y_4[n]$:



Bonus Exercises

1. Generate and graph the signals: *sm(t)* – monoalternating rectified sine, *sd(t)* – double alternating rectified sine.
2. Generate a 10x10 square matrix which contains random integers in the interval 1:10 using the function call `M = round(10*rand(10,10))`.
   a. Compute the sum of the elements in the corners of the matrix M;
   b. Compute the sum of the elements of the matrix M;
   c. Define a function that calculates the sum of the elements on each row of the matrix M and returns a column vector of these sums.

3. Create a function that constructs the identity matrix, given the input parameters: *m* – the number of rows and *n* – the number of columns of the requested matrix.

4. Create a function that takes as input a matrix A and two values: a new value *m* and an old value *n*. The function will return the input matrix after replacing the occurrences of the old value with the new value, and the number of replaced elements.

5. Create a function that takes a matrix A as input and returns two vectors, a vector containing the even values in A and a vector containing the odd values in A.

Annex 1

Table 1.2 shows other (more advanced) functions used in MATLAB.

Table 1.2 Other MATLAB Functions

Function/Parameter	Description and Examples
`cosh(x), sinh(x), tanh(x)`	Hyperbolic cosine, hyperbolic sine and hyperbolic tangent functions
`acosh(x), asinh(x), atang(x)`	The inverses of the hyperbolic cosine, hyperbolic sine, and hyperbolic tangent functions
`Whos`	Displaying variables in the Workspace
`pause,` `pause(n)`	Pause in instruction execution Pause for *n* seconds in instruction execution
`Subplot`	Declare a subplot within a figure
`hold on, hold off`	Display next graphs over an existing graph (on) or next graphs must be independent of the current graph (off)
`axis ([x_inf x_sup y_inf y_sup])`	The axes of the figures are displayed between the limits *x_inf* and *x_sup* (on the Ox axis). The same goes for the Oy axis with *y_inf* and *y_sup*

`xlabel('OX_axis_name'),`
`ylabel('OY_axis_name'),`
`title('Figure_Title'),`
`legend(parameters)`
	Axis titles, figure title, figure legend

`save, load`	Saving or loading data between/from a file
`cd,` `pwd`	Change current directory, Display the name of the current directory
`input("Insert value using the keyboard >>")`	Reading the value of a variable from the keyboard

References

1. Valentin-Adrian Niță, Radu Alexandru Badea, Răzvan-Eusebiu Crăciunescu, "Introducere în prelucrarea semnalelor folosind Python", – Timişoara: Editura Politehnica, 2022
2. Mateescu, Adelaida, Dumitriu, N., Stanciu, L., "Semnale, circuite şi sisteme", Teora, Bucureşti, 2001
3. https://www.mathworks.com/