1 Introduction to MATLAB

1.1 General considerations

The aim of this laboratory is to review some useful MATLAB commands in digital signal processing. MATLAB is one of the fastest and most enjoyable ways to solve problems numerically. The computational problems arising in many applications can be solved much more quickly with MATLAB, than with the standard programming languages. It is easy to generate some results, draw graphs to look at the interesting features, and then explore the problem further [1].

The starting procedure takes the user to the command window where the command line is indicated with `>>`. All commands are typed in after the prompt `>>`, i.e. the command: `>> var = 0 : 9` creates the `var` variable, displaying on the screen the tenth elements from `var(1) = 0` to `var(10) = 9`:

```
var =
    0    1    2    3    4    5    6    7    8    9
```

Previous commands can be corrected and reused; just press the keypad up arrow and the previous command appears; make the necessary corrections and press `[Enter]` to run the corrected command.

Besides the command window working, MATLAB also works with programs included in files, called M-files (`*.m`). A program can be written as a script file or as a function file. A script is an external file that contains a MATLAB commands sequence. After completely execution of a script file, the created variables remain in the application zone memory.

If the first line of the file contains the word `function`, it is a function file, that involves working with arguments. After finalizing the execution of a function, in the PC memory only the output variables will be available.

MATLAB works with two types of windows: a command window and a graphical representation one. At a certain moment only one command window can be opened, but more graphical representation windows. MATLAB programming environment is case sensitive. The name of a function must be always lowercase. The comments lines from a script/function file are preceded by the `%` character.

1.1.1 Defining variables

Numerical values are assigned to variables, and the numerical expression is directly displayed. If on the command line is typed: `>> var1 = 1+5` the result will be:

```
var1 =
    6
```

One often does not want to see the result of intermediate calculations; in this case the assignment statement or expression must be terminated with semicolons `;`, i.e. `>> var1 = 1+5;`.

A formula using arithmetic operators defined in MATLAB (see Tab. 1.1 [2]) or one or more already defined measures (even if in the current command) can be also assigned to a variable. Considering that `var1` was previously defined, `>> var2 = var1^2` will return:

```
var2 =
        36
```

Symbol	Description
+	Addition
-	Subtraction
*	Multiplication
.*	Array multiplication
/	Division
./	Array right division
^	Power
.^	Array power
'	Transpose or complex-conjugate transpose
.'	Array transpose
( )	Precedence in arithmetic expressions

Table 1.1: Arithmetic operators

In MATLAB there are some predefined variables – these cannot be declared and are global accessible in every M-file. These special variables are usually introduced in MATLAB functions code, returning useful scalar values [3]. Some of special variables and constants are illustrated in Tab. 1.2.

All computations in MATLAB are done in double precision. The format – how MATLAB prints numbers – is controlled by the `format` command. Type `help format` for full list. The function form of the syntax is `format option`. Some output examples are presented in Tab. 1.3 [4].

Function	Return value
ans	Most recent answer (variable); if you do not assign an output variable to an expression, MATLAB automatically stores the result in ans
eps	Floating-point relative accuracy; this is the tolerance MATLAB uses in its calculations (implicit value is $2.2204e-016$)
pi	Permanent variable that has assigned the value $\pi = 3.141592653589\ldots$
i, j	Imaginary unit ($\sqrt{-1}$)
inf	Infinity; calculations like $n/0$, where n is any nonzero real value, result in inf
NaN	Not a Number, an invalid numeric value. Expressions like $0/0$ and inf/inf result in a NaN, as do arithmetic operations involving a NaN; also, if $n \in \mathbb{C}$ with a zero real part, then $n/0$ returns a value with a NaN real part

Table 1.2: Special variables/constants

Option	Result	Example
+	+, −, blank	+
bank	Fixed dollars and cents	3.14
hex	Hexadecimal representation of a binary double-precision number	$400921fb54442d18$
long	Scaled fixed point format, with 15 digits – double; 8 digits – single	3.14159265358979
long e	Floating point format, with 15 digits – double; 8 digits – single	$3.141592653589793e+00$
long g	Best of fixed or floating point, with 15 digits – double; 8 digits – single	3.14159265358979
rat	Ratio of small integers	355/113
short	Scaled fixed point format, with 5 digits	3.1416
short e	Floating point format, with 5 digits	$3.1416e+00$
short g	Best of fixed or floating point, with 5 digits	3.1416

Table 1.3: Some allowable values for `option` and an example for π

1.1.2 Built-in functions

Some of the MATLAB built-in functions are illustrated in Tab. 1.4. Regarding the way to use these functions use the `help` command accompanied by the desired function name.

If `v` is a vector and `M` a matrix, for the data analysis functions, the syntax is presented in Tab. 1.5.

Function	Description
Operations regarding complex numbers	
abs	Absolute value and complex magnitude
angle	Phase angle, in radians
conj	Complex-conjugate
real	Real part of a complex number
imag	Imaginary part of a complex number
Trigonometric functions	
sin	Sine of an argument in radians
cos	Cosine of an argument in radians
tan	Tangent of an argument in radians
cot	Cotangent of an argument in radians
Square root, exponential and logarithm	
sqrt	Square root
exp	Natural exponential (base e)
log	Natural logarithm (log base e)
log2	Base 2 logarithm
log10	Base 10 logarithm
pow2	Base 2 power and scale floating-point numbers
Basic operations	
round	Round to nearest integer
floor	Round towards $-\infty$
fix	Round towards zero
ceil	Round towards $+\infty$
frac	Symbolic matrix element-wise fractional parts

Table 1.4: Some elementary built-in MATLAB functions

1.1.3 Scalars, vectors and matrices

In MATLAB, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or one column, which are vectors. MATLAB has other ways of storing both numeric and nonnumeric data, but in the beginning, it is usually best to think of everything as a matrix; the operations are designed to be as natural as possible. MATLAB allows you to work with the entire matrices quickly and easily [4].

You can enter matrices in several different ways: enter an explicit list of elements, load matrices from external data files, generate matrices using built-in functions, create matrices with your own functions in M-files. The easiest way consists in entering an explicit list of elements, following a few conventions:

- separate the elements of a row with blanks or commas `,`;

Syntax	Description
sum(v)	Returns the sum of the elements of the vector v
prod(v)	Returns the product of the elements of the vector v
sum(M)	Treats the columns of M as vectors, returning a row vector of the sums of each column
prod(M)	Treats the columns of M as vectors, returning a row vector of the products of each column
max(v)/ min(v)	Returns the largest/smallest element in v
[m, p] = max(v)/ [m, p] = min(v)	Finds the index of the largest/smallest element in v and returns it in output variable p; if there are several identical maximum/minimum values, the index of the first one found is returned
max(M)/ min(M)	Treats the columns of M as vectors, returning a row vector containing the maximum/minimum element from each column
[m, p] = max(M)/ [m, p] = min(M)	Finds the indices of the maximum/minimum values of M, and returns them in vector p; if there are several identical max/min values, the index of the first one found is returned
mean(v)	Returns the mean value of v
mean(M)	Treats the columns of M as vectors, returning a row vector of mean values

Table 1.5: MATLAB functions for data analysis

- use a semicolon `;` to indicate the end of each row;
- surround the entire list of elements with square brackets `[]`.

The matrices elements can be any real or complex numbers, or any other variable. The elements of the matrix `A` could be identify using $\mathbf{A}(i,j)$ (the element from the `ith` row and `jth` column) – two indices are needed; to refer to an element of a vector only one index is needed.

The matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and the vectors $\mathbf{B} = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$ and $\mathbf{C} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$ can be enter in MATLAB as follows:

```
>> A=[1,2,3;4,5,6]
```

```
>> A=[1 2 3;4 5 6]
```

```
>> A=[1 2 3 Enter
      4 5 6]
```

all syntax returning:

```
A =
    1    2    3
    4    5    6
```

```
>> B = [7 8 9]
```

```
B =
    7    8    9
```

```
>> C = [-1; -2]
```

```
C =
    -1
    -2
```

One can interchange the matrix elements, or add new ones, without retyping the whole matrix:

```
>> A(1, 3) = 0
A =
    1   2   0
    4   5   6
```

```
>> A(3, 3) = -2
A =
    1   2   0
    4   5   6
    0   0  -2
```

```
>> C(3) = 1
C =
    -1
    -2
     1
```

Higher dimension matrices can be obtained from lower dimension ones. We will use for exemplification the previous matrix **A** and the vectors **B** and **C** in their last configuration:

```
>> D = [A; B]
D =                 % matrix D (4x3) was obtained by adding vector B to matrix A
    1    2    0     %  (as last row);
    4    5    6     % A and B must have the same number of columns
    0    0   -2
    7    8    9
>> E = [A, C]
E =                       % matrix E (3x4) was obtained by adding vector C to matrix A
    1    2    0   -1      % (as last column);
    4    5    6   -2      % A and C must have the same number of rows
    0    0   -2    1
```

If **v** is a vector and **M** a matrix we have the syntax presented in Tab. 1.6.

The dimension of a matrix, its determinant, and its inverse can be determined using the syntax from Tab. 1.7.

The syntax for commonly used vectors and matrices is illustrated in Tab. 1.8.

1.1.4 Function M-files

Much more powerful than scripts are functions, which allow the user to create new MATLAB commands [5]. A function is defined in an M-file that begins with a line of the following form:

```
function [output1, output2, ...] = name(input1, input2, ...)
```

- function – key word that declare the script as function (mandatory);

Syntax	Description
v(i:k)	Selects the elements from positions i, i+1, i+2, ..., k of the vector v; if i>k, the resulting vector is empty
v(i:j:k)	Selects the elements from positions i, i+j, i+2j, ..., k (using step j); if j>0 and if i>k or if j>0 and if i<k, the resulting vector is empty
v([i, j, k])	Selects the elements from positions i, j, k
v(:)	If the vector is row vector it becomes column vector; if it is column vector it remains as it is
M(:, j)	Selects the jth column of the matrix M
M(i, :)	Selects the ith row of the matrix M
M(:, i:j)	Selects the columns from ith to jth of the matrix M
M(i:j, :)	Selects the rows from ith to jth of the matrix M
M(:, i:j:k)	Selects the columns i, i+j, i+2j, ..., k of the matrix M
M(i:j:k, :)	Selects the rows i, i+j, i+2j, ..., k of the matrix M
M(i:j, k:l)	Extract the submatrix consists in the elements from ith to jth rows and from kth to lth columns of M
M(:, [i, j, k])	Selects the columns i, j, k of the matrix M
M([i, j, k], :)	Selects the rows i, j, k of the matrix M
M([i, j, k], [l, m, n])	Extract the submatrix consists in the elements from ith, jth, kth rows and from lth, mth, nth columns
M(:, :)	Selects the entire matrix M
M(:)	Selects all the elements of the matrix M and return them in a column vector

Table 1.6: MATLAB syntax to select vector/matrix's elements

Syntax	Description
length(v)	Returns the length (the number of the elements) of the vector v
[l, c] = size(v)	One of the dimensions will be equal by 1; if v is a row vector l=1; if v is a column vector c=1
length(M)	Returns the size of the longest dimension of M
[l, c] = size(M)	Returns the size of matrix M in separate variables l and c
det(M)	Returns the determinant of the square matrix M; if M contains only integer entries, the result is also an integer
inv(M)	Returns the inverse of the square matrix M; a warning message is printed if M is badly scaled/nearly singular

Table 1.7: Dimension of a matrix. Determinant and inverse

- `name` – function's name (the name of the M-file); this name cannot be the same as one of an already declared M-file;
- `output1, output2, ...` – are the output parameters of the function; they must be separated by commas, and must be within square brackets; if

Syntax	Description
v = initial:step:final	Produce a row vector v with the elements starting from initial to final, with step equal by step (the value of step could be positive or negative)
v = initial:final	Produce a row vector v with the elements starting from initial to final, with step equal by 1
v = linspace(a, b, n)	Generates a row vector v of n points linearly spaced between and including a and b
v = logspace(a, b)	Generates a row vector v of 50 logarithmically spaced points between decades 10^a and 10^b
v = logspace(a b, n)	Generates n points between decades 10^a and 10^b
x = [ ]	Generates an empty matrix
ones(n)	Returns an n-by-n matrix of 1s; an error message appears if n is not a scalar
ones(m, n)	Returns an m-by-n matrix of 1s
ones(size(M))	Returns an array of 1s that is the same size as M
zeros(n)	Returns an n-by-n matrix of zeros; an error message appears if n is not a scalar
zeros(m, n)	Returns an m-by-n matrix of zeros
zeros(size(M))	Returns an array of zeros that is the same size as M
eye(n)	Returns an n-by-n identity matrix
eye(m, n)	Returns an m-by-n matrix with 1s on the diagonal and 0s elsewhere
eye(size(M))	Returns an identity matrix same size as M
rand(n)	Returns an n-by-n matrix of random entries (uniformly distributed in the interval $(0, 1)$)
rand(m, n)	Returns an m-by-n matrix of random entries
rand(size(M))	Returns an array of random entries that is the same size as M
randn(n)	Returns an n-by-n matrix of random entries (uniformly distributed: mean 0, variance $\sigma^2 = 1$, standard deviation $\sigma = 1$)
randn(m, n)	Returns an m-by-n matrix of random entries
randn(size(M))	Returns an array of random entries that is the same size as M
diag(v)	Returns n-by-n diagonal matrix having v as its main diagonal
diag(v, k)	Returns a square symbolic matrix of order n+abs(k) (n - number of elements of v), with the elements of v on the kth diagonal; k=0 signifies the main diagonal
diag(M)	If M is a square symbolic matrix, returns its main diagonal
diag(M, k)	If M is a square symbolic matrix, returns a column vector formed from elements of the kth diagonal of M

Table 1.8: Commonly used vectors and matrices

the function has no output arguments the square brackets and the equal sign have no meaning;

- `input1, input2, ...` – are the input parameters of the function; they must be separated by commas, and must be within round parentheses; if the function has no input arguments the round parentheses and the equal sign have no meaning.

1.1.5 Conditional and loops

The capabilities of MATLAB can be extended through programs written in its own programming language. It provides the standard constructs, such as loops and conditionals; these can be used interactively to reduce the tedium of repetitive tasks, or collected in programs stored in M-files.

1. `If` conditional statement, `else` clause, `elseif` clause

```
if logicExpr      % if the logical expression logicExpr is true (it evaluates to logical 1),
   statements     % MATLAB executes all the statements between if and end,
end               % denoted by statements; if the condition is false (evaluates to
                  % logical 0), MATLAB skips all the statements between the if and
                  % end lines, and resumes execution at the line following the end
                  % statement
```

```
if logicExpr1     % if the logical expression logicExpr1 is true, MATLAB executes all
   statementsA    % the statements between the if and else clause denoted by
else              % statementsA; if the condition is false, MATLAB skips all the
   statementsB    % statements between the if and else lines, and resumes execution
end               % at the line following the else statement, denoted by statementsB
```

If the function to be evaluated has more `if-else` instructions levels, it is difficult to determine the true logical expression, that selects the statements to be executed (`elseif` clause is to be used).

```
if logicExpr1     % if logicExpr1 is true, MATLAB executes only statementsA;
   statementsA    % if logicExpr1 is false and if logicExpr2 is true, MATLAB
elseif logicExpr2 % executes only statementsB;
   statementsB    % if logicExpr1 is false and if logicExpr2 is also false, and if
elseif logicExpr3 % logicExpr3 is true, MATLAB executes only statementsC;
   statementsC    % if more logical expressions are true, the first true determines
end               % which statement group to be executed first; if all the logical
                  % expressions are false, MATLAB skips all the statements
                  % between the if and end lines
```

2. `For` loop – repeats statements a specific number of times

```
for index = expression
   statements
end
```

- `index` – counter name;
- `expression` – a matrix, a vector or a scalar;
- most of the time `expression` is `initial:step:final` (`initial` – first value of `index` ; `step` – increment for `index` (if it is not specified its default value is 1); `final` – maximum value for `index`);
- for `index` going from `initial` to `final` with increment `step` , `statements` are executed for a number of times equal by $\left[\frac{\text{final} - \text{initial}}{\text{step}}\right]_*$ ($[\,]_*$ – the integer part); `statements` can be any MATLAB expression.

In order to use `for` loop, some requirements must be fulfilled [3]:

- the index of `for` loop must be a variable;
- if the expression is an empty matrix, the loop is skipped, and execution is resumed at the line following the `end` statement;
- if the expression is a scalar, the loop is executed only once, with the index given by the scalar value;
- if the expression is a row vector, the loop is executed a number of times equal by the number of elements in the vector, each step the index being equal by the next element in vector;
- if the expression is a matrix, the index will have at each iteration the values contained in the next column of the matrix;
- at the end of `for` loops, the index has the last used value.

3. `While` loop – executes a statement or group of statements repeatedly as long as the controlling expression is true

```
while expression
   statements
end
```

4. `Break` statement – terminates the execution of a `for` loop or `while` loop; when a `break` statement is encountered, execution continues with the next statement outside of the loop; in nested loops, `break` exits from the innermost loop only.

5. Relational and logical operators – inside an algorithm, most of the time a selection of the next group of statements that follows to be executed is needed, conditioned by the true value of an expression. The conditional

instructions use relational and logical operators. In MATLAB there are six relational operators used to compare two matrices of the same dimensions (see Tab. 1.9). A relational operator compares two matrices of matrix expressions, element by element. The result is a matrix with the same dimension as the matrices to be compared, with elements: **1** – if the relationship is true; **0** – if the relationship is false. To combine one or more logical expressions, the logical operators from Tab. 1.10 should be used.

Operator	
<	Lower
<=	Lower or equal
>	Greater
>=	Greater or equal
==	Equal
∽=	Not equal

Table 1.9: Relational operators

Operator		Priority
∽	Not	1
&&	And	2
||	Or	3

Table 1.10: Logical operators

1.1.6 Graphics

If `v`, `x` and `y` are vectors (`x` and `y` have the same length) and `M` and `N` are two matrices with the same dimension, syntax for graphical representation is illustrated in Tab. 1.11 [4].

MATLAB defines string specifiers for line styles, marker types, and colors. Tab. 1.12 list these specifiers. A `stem` plot displays data as lines (stems) terminated with a marker symbol at each data value. In a 2-D graph, stems extend from the x-axis.

For graphical representations in logarithmic coordinates `loglog`, `semilogx` and `semilogy` functions are used. The syntax presented for linear coordinates remains unchanged, the only difference being the way of scaling the axes. The `loglog` function scales both axes (abscissa and ordinate) using the base 10 logarithm, so on the axes we will have powers of 10. The `semilogx` function realizes same type of scaling but only for abscissa, and `semilogy` proceeds the same way, scaling only the ordinate.

More than one plot can be put on the same figure using the `subplot` command. It divides the current figure into rectangular panes that are numbered rowwise. Each pane contains an axes. Subsequent plots are output to the current pane. I.e. if we want to partition the figure window in 2×3 matrix of small subplots, we will have the next order: $\begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline \end{array}$. The function `subplot(m, n, p)` creates

Syntax	Description
plot(v)	If v is a vector with real elements produces a linear graph of the elements of v vs. the index of elements of v; if v is a complex-valued vector, the representation will be done function of its real part (on the abscissa) and of its imaginary part (on the ordinate)
plot(M)	Plots the columns of M vs. their index if M is real-valued; if M is complex, the syntax is equivalent to plot(real(M), imag(M))
plot(x, y)	Produces a graph of y vs. x; the length of vector x must be the same as the length of y
plot(x, M)	If the length of x is the same as the number of rows for M, results a graph of the columns of the matrix M vs. x; if the length of x is the same as the number of columns for M, results a graph of the rows of the matrix M vs. x; if the length of x is equal neither by the number of rows and neither by the number of columns of M, representation cannot be done
plot(M, N)	Plots the columns of N vs. the columns of M (the kth column of N will be represented vs. the kth column of M); the dimension of the two matrices must be the same
plot(x1, y1, x2, y2, ..., xn, yn)	Plots on the same graphic y1 vs. x1, y2 vs. x2, ..., yn vs. xn (can be vectors or matrices); the considerations already presented are still valid

Table 1.11: Graphical representation in linear coordinates

Line style		Marker type		Color	
–	Solid line (default)	+	Plus sign	r	Red
		∘	Circle	g	Green
––	Dashed line	∗	Asterisk	b	Blue
:	Dotted line	·	Point	c	Cyan
–.	Dash-dot line	×	Cross	m	Magenta
		'square' or s	Square	y	Yellow
		'diamond' or d	Diamond	k	Black
		^ / v	Upward/Downward-pointing triangle	w	White
		> / <	Right/Left-pointing triangle		
		'pentagram' or p	Five-pointed star		
		'hexagram' or h	Six-pointed star		

Table 1.12: Line styles, marker types, and colors

an axis in the pth pane of a figure divided into an m-by-n matrix of rectangular panes [6]. The new axes becomes the current axes; if p is a vector, it specifies an axis having a position that covers all the subplot positions listed in p.

The plots can be customized to meet your needs. The most important way to do this is with the axis command, which changes the axis of the plot shown, so only the part of the axis that is desirable is displayed. The axis command is used by entering the following command right after the plot command (or any command that has a plot as an output):

```
axis([x0 x1 y0 y1])  % sets the limits for the x- and y-axis of the current axes
```

Another important thing for plots is labeling. You can give title to a plot (title), x-axis label (xlabel), y-axis label (ylabel), and put text on the actual plot. All of the above commands are issued after the actual plot command has been issued (see Tab. 1.13). Furthermore, text can be put on the plot itself in one of two ways: using text , or gtext command. The first command involves knowing the coordinates of where you want the text string: text(xcor, ycor, 'textstring') . To use the other command, you do not need to know the exact coordinates: gtext('textstring') , and then you just move the cross-hair to the desired location with the mouse, and click on the position you want the text to be placed.

Syntax	Description
title('text')	Outputs the string text at the top and in the center of the current axes
xlabel('text')	Labels the x-axis of the current axes with the string text
ylabel('text')	Labels the y-axis of the current axes with the string text

Table 1.13: Labeling plots

By adding more sets of parameters to plot , you can plot as many different functions on the same figure as you want. When plotting many things on the same graph it is useful to differentiate the different functions based on color and point marker. This same effect can also be achieved using the hold on and hold off commands. Always remember that if you use the hold on command, all plots from then on will be generated on one set of axes, without erasing the previous plot, until the hold off command is issued.

1.2 To be done

1. Verify all the displaying format types using the value x = pi (see Tab. 1.3).

2. Verify the next examples, typing directly in the command window:

```
>> p = pi;          % the p value will not be displayed (it is in the workspace)
>> % r = pi/4       % this line is not taken into account
>> v = r/2          % an error message will occur, because r is not recognized
>> s = 1+2+3   [Enter]   % instruction will be continued on the next line
+4+5+6
```

3. Consider the matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and the vectors $\mathbf{B} = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$ and $\mathbf{C} = \begin{bmatrix} -2 & -1 \end{bmatrix}^T$. Verify the syntax from Tab. 1.6.

4. Find the vectors elements:

```
v = 0:10:40
u = 2:8
d = 20:-2:2
```

```
linspace(4, 16, 4)
linspace(pi,-pi,3)
logspace(1, 3)
```

```
logspace(1, pi)
logspace(1, 2, 6)
logspace(0, pi, 3)
```

5. Find the values of the matrices:

```
ones(2)
ones(3, 2)
ones(size(D))
zeros(4)
```

```
zeros(5, 2)
eye(2)
eye(size(D))
rand(2)
```

```
rand(1, 4)
rand(size(D))
randn(2)
randn(1, 4)
```

6. Define two matrices `M = randn(5)` and `N = randn(3, 3)`, and find the values of their determinants.

7. Define a row vector `a = randn(1, 4)` and a matrix `A = randn(4)`. Verify the matrices:

```
diag(a)
diag(a, 1)
```

```
diag(a, -1)
diag(A)
```

```
diag(A, -2)
diag(diag(A))
```

8. Define a row vector `a = randn(1, 5)`, a column vector `b = randn(5, 1)` and a matrix `C = randn(3, 4)`. Find their length (`length`) and their size (`size`).

9. A function that evaluates the arithmetic mean of the `x` vector's values, can be written as:

```
function m = arithmeticMean(x)
n = length(x); % the length of the vector x
m = sum(x)/n; % the arithmetic mean of the values of the vector x
y = ['The arithmetic mean is:', num2str(m)]; disp(y);
```

The function must be saved as `arithmeticMean.m` [7]. The previously defined function can be called from the MATLAB command line, from another

script, or from another function

```
>> x = [1 2 3 4 5 6]; arithmeticMean(x);
The arithmetic mean is: 3.5
```

10. Display graphically the function $f(t) = \sin(2\pi 0.3t)$ with blue plus-dash line and the function $g(t) = 1 - f(t)$ with red dash-asterisk line. Label the x-axis with `Time` and the y-axis with `Amplitude`; the title of the plot should be `f(t) = sin(2π0.3t) and g(t) = 1 - f(t)`. Use `plot` in the subfigure 1, and `stem` in the subfigure 2 (Fig. 1.1). Put also the legend in each subwindow [7].

A script file called `fgPlot.m` can be written as:

```
% plot two functions, label axes, add title
t = 0:0.1:5; f = sin(2*pi*0.3*t);  g = 1-f;
figure(1); subplot(211); plot(t, f, '-+b', 'LineWidth',1.5);
hold on                  plot(t, g, '-*r', 'LineWidth', 2);
hold off; grid; xlabel('Time'); ylabel('Amplitude');
legend('boxon'); l1 = legend('f(t)', 'g(t)', 2);
title('f(t) = sin(2\pi0.2t) and g(t) = 1-f(t)');
subplot(212); stem(t, f, 'ob', 'LineWidth',1.5);
hold on       stem(t, g, '^r', 'LineWidth', 2);
hold off; grid; xlabel('Time'); ylabel('Amplitude');
legend('boxoff'); l2 = legend('f(t)', 'g(t)', 1);
```

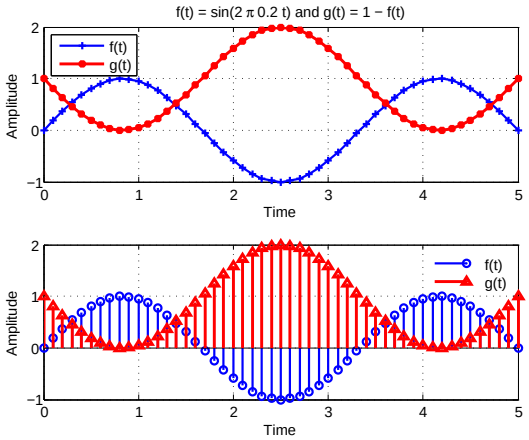

Figure 1.1: Graphics for $f(t)$ and $g(t)$

1.3 Exercises

1. Generate a linearly spaced vector between 3 and 9 with increment 2.
2. Generate a 13 element linearly spaced vector between 3 and 9.

3. Generate a 9 point logarithmically spaced vector between decades 10^{-3} and 10^{3}.

4. `y = 3:0.9:123` is the given vector. Find the length of the vector and generate another vector of the same length, with only 1s elements.

5. Consider the matrices $\mathbf{A} = \begin{bmatrix} 3 & 2 & 1 \\ 8 & 4 & 5 \\ 0 & 2 & 0 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 1 & 1 \\ 2 & 3 & 2 \end{bmatrix}$, and the scalar $m = 4$. Evaluate using MATLAB:

   - $\mathbf{C} = \mathbf{A} + \mathbf{B}$;
   - $\mathbf{F} = \mathbf{A} \cdot \mathbf{B}$;
   - $\mathbf{I} = \mathbf{B}^T$;
   - $\mathbf{L} = \mathbf{C}^m$.
   - $\mathbf{D} = \mathbf{A} - \mathbf{B}$;
   - $\mathbf{G} = \mathbf{B} \cdot m$;
   - $\mathbf{J} = \mathbf{A}/\mathbf{B}$;
   - $\mathbf{E} = \mathbf{C} + m$;
   - $\mathbf{H} = \mathbf{A}^T$;
   - $\mathbf{K} = \mathbf{A} \setminus \mathbf{B}$;

   Verify if $\mathbf{J} = \mathbf{A} \cdot \mathbf{B}^{-1}$ and if $\mathbf{K} = \mathbf{A}^{-1} \cdot \mathbf{B}$. Use the `long e` format.

6. Evaluate the scalar product for the vectors: $\mathbf{a} = \begin{bmatrix} 1 & 2 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -3 & 3 \end{bmatrix}$.

7. For the matrices $\mathbf{A} = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ evaluate the element by element product.

8. Graph $x(n) = \sin\left(2\pi\frac{1}{5}n\right)$, $n = \overline{0, 10}$, using `stem` command. The graph should be represented by red stars; label the axes and write a title.

9. In order to evaluate the sum of two variables `a` and `b`, write a MATLAB function named `bplusa.m`:

```
function sumab = bplusa(a, b);
```

10. In order to evaluate the product of two vectors `a` and `b`, generate a function named `bproducta.m`:

```
function prodab = bproducta(a, b);
```

11. Write a MATLAB function named `geomMean.m`, in order to evaluate the geometric mean of two scalars `a` and `b`:

```
function geometricmean = geomMean(a, b);
```