# 2. MATLAB PROGRAMMING ELEMENTS

**Objectives of the paper:**

- familiarity with Matlab file types,
- review of some vectorial computing elements,
- fixing knowledge regarding vectorial computing problem solving using the Matlab programming environment, by studying some examples and solving some problems.

It is recommended to go through Annex M2 before studying paragraphs 2.1 and 2.2.

## 2.1. Matlab programing elements

### *Matlab file types*

Matlab files (M-files) are of two types:

- *function files;*
- *script files.*

Function files differ from script files by the fact that the first can work with arguments (can get parameters and return values), and the others only operate on the *workspace* variables. Script files are typically used to solve problems that require the execution of a large set of commands, for which use of command line mode would be cumbersome.

Obligatory, the first line of a function file has the form:

```
function [output_parameters]=function_name(input_parameters)
```

where:

| | |
|---|---|
| `function` | is the keyword that declares the file as a function file; |
| `function_name` | represents the name of the function, is the same as the name under which the file is saved; |
| `output_parameters` | represents the list of output parameters, separated by comma, enclosed in square brackets; if the function does not have output parameters, then the brackets and the equal sign are omitted; |
| `input_parameters` | represents the list of input parameters, separated by comma, enclosed in parantheses; if the function does not have input parameters, then the parentheses are omitted. |

***Running a script program*** (script file) is done either by typing the file name (without extension) in the command line, or by using the command *Run M- File* from the *File* menu; or by pressing the F5 key when the file is opened in the editor and the editor is the active window.

***Running a function type program*** (function file) is done in the command line using the syntax:

```
[returned_values_variables]=function_name(input_parameter_values)
```

where:

| | |
|---|---|
| `function_name` | the name of the function (of the file where the function is defined); |
| `returned_values _variables` | represents the list of variables for storing output parameter values, separated by comma, enclosed in square brackets; if the function does not have output parameters, then the brackets and the equal sign are omitted; if the function has only one output parameter, then the brackets are omitted; |
| `input_parameter _values` | represents the list of input parameter , separated by comma, enclosed in parantheses; if the function does not have input parameters, then the parentheses are omitted. |

## *Comments in Matlab*

A comment in a Matlab program has the form:

```
%comment
```

If the % character appears on a program line, then the part of the line following this character will be omitted by the interpreter. If a comment spans multiple lines, each of these lines must begin with the character %.

If a comment appears immediately below the function file declaration line, it will constitute the *help* of the function file, that is, the text that will be displayed on the command line following the command:

```
>> help function_name
```

where `function_name` is the name (without extension) of the function file.

There is also a special form of a *comment on multiple lines*, called *block comment*, the comment being delimited by two special lines. The comment itself is written on the lines between the two delimiters:

```
%{
comment
%}
```

## *Control instructions*

Tables 2.1 and 2.2 contain the **relational operators** and, respectively, the **logical operators**, used in Matlab:

***Table 2.1.*** *Ralational operators used in Matlab.*

| **Relational operators** | **Meaning** |
|---|---|
| < | less than |
| > | greater than |
| <= | is less than or equal to |
| >= | is greater than or equal to |
| == | equal to |
| ~= | not equal to |

***Table 2.2.*** *Logical operators used in Matlab.*

| **Logical operators** | **Meaning** | **Priority** |
|---|---|---|
| ~ | NOT | 1 |
| & | AND | 2 |
| \| | OR | 3 |

*Comments:* 1. When using relational operators <, >, <=, >= on the set of complex number matrices, the imaginary part is omitted. Thus, 0>=i will result in response 1 (i.e. TRUE) (0>=0, where the second 0 is the real part of i).

2. Logical operators are always used to compare matrices with elements 0 and 1 (with false, respectively true meanings), calculated with relational operators.

***The "if" statement*** can be used in one of the following three forms, with the syntaxes:

- simple *if* statement:

```
if logical_expresion
     group_of_instructions
end
```

- *else* clause:

```
if logical_expresion
     group_of_instructions_1
else
     group_of_instructions_2
end
```

- *elseif* clause:

```
if logical_expresion_1
     group_of_instructions_1
elseif  logical_expresion_2
     group_of_instructions_2
...
elseif  logical_expresion_n-1
     group_of_instructions_n-1
else
     group_of_instructions_n
end
```

***The syntax of the "for-loop" statement*** in Matlab is as follows:

```
for index=expresion
     group_of_instructions
end
```

where:

- *expresion* is a matrix, a vector or a scalar; most often in the form:

```
      k=initial:step:final
```

- if *expresion* is a matrix, than the iterations are done column by column.

***The "while-loop" statement*** is used to repeat a set of instructions, as long as a specified condition is true. The syntax of this instruction has the form:

```
while expresion
       group_of_instructions
end
```

with the following comments:

- the *group_of_instructions* are executed as long as all the elements in the *expresion* are nonzero;
- *expresion* most often has the form:

```
      expresion_1 relational_operator expresion_2
```

- forced exit from an infinite loop is done by pressing the `[Ctrl]` and `[C]` keys simultaneously.

## *Testing functions*

In table 2.3 some of the functions used to test values of variables, variable types, objects, etc. are briefly presented:

***Table 2.3.*** *Matlab testing functions*

| Function | Effect |
|---|---|
| *ischar* | tests whether the given argument is a string |
| *isempty* | tests whether the argument given is the null matrix (matrix with no element) |
| *isequal* | tests whether two arrays are identical |
| *isfinite* | finds the finite elements of an array |
| *isfloat* | tests whether the argument given is a floating point array |
| *isinf* | finds the infinite value elements of an array (Inf) |
| *isinteger* | tests whether the argument given is an integer array |
| *iskeyword* | tests whether the argument given is a Matlab keyword |

***Table 2.3.*** *Matlab testing functions-continuation*

| Function | Effect |
|---|---|
| *isnan* | finds the elements of an array that are not numbers (NaN) |
| *isnumeric* | tests whether the argument given is an array of numbers |
| *isprime* | finds the elements of an array that are prime numbers |
| *isreal* | tests whether all the elements of an array are real numbers (the coeficient of the imaginary part is zero) |
| *isscalar* | tests whether the argument given is a scalar |
| *issorted* | tests whether a set of elements is in ascending order |
| *isvector* | tests whether the argument given is a vector |

## *Vectorial computing*

In table 2.4 the main functions used for ***vectorial computing*** are presented.

***Table 2.4.*** *Matlab vectorial computing functions*

| Function | Sintax | Effect |
|---|---|---|
| *norm* | `norm(v,p)` | returns the norm p for the vector v |
| | `norm(v)` | returns the euclidean norm for the vector v (identical with the case p=2) |
| *dot* | `dot(v,w)` | returns the scalar product of the v and w vectors of same length |
| *cross* | `cross(v,w)` | returns the vectorial product of the v and w vectors of 3 elements |
| *sum* | `sum(v)` | returns the sum of the elements of the vector v |
| *length* | `length(v)` | Returns the lenght of the vector v (i.e. the number of elements) |

## 2.2. Examples

**Example 2.1:** Generate the fourth order A and B square matrices defined by the relationships below. Then display their sum, their product, the square of matrix A, the result of the left division of the matrix A through B and the rank of matrix B. Solve the problem by using a script file.

$$A_{i,j} = \frac{1}{i+j}, \qquad i,j = \overline{1,4}$$

$$B_{i,j} = \begin{cases} 1, & i = j \\ i + j, & i > j, \quad i, j = \overline{1,4} \\ i - j, & i < j \end{cases}$$

*Solution*: The steps for solving the problem are as follows:

i) Creating an M-file, named, for example, op_matr.m:

```
>> edit op_matr
```

ii) Solving the problem requirements by writing the commands in the script file:

```
% generating the matrix A
for i=1:4
    for j=1:4
        A(i,j)=1/(i+j)
        ;
    end
end
% generating the matrix B
for i=1:4
    for j=1:4
        if i==j B(i,j)=1;
        elseif i>j B(i,j)=i+j;
        else B(i,j)=i-j;
        end
    end
end
% displaying the matrix A
A
% displaying the matrix B
disp('matrix B')
disp(B)
% calculating and displaying the sum
Su=A+B
% calculating and displaying the product
Product=A*B
% calculating and displaying the square of matrix A
square_A=A^3
% calculating and displaying the result of the left division
disp('The result of the division A\B is')
Res=A\B
% rank of matrix B
rank_B=rank(B)
```

It is recommended to save the file constantly.

Running the script file is done from the command line:

```
>> op_matr
```

Running an M file from the command line has the effect of running the last saved version of the file.

iv) View the results displayed in the command window:

```
 A =
      0.5000    0.3333    0.2500    0.2000
      0.3333    0.2500    0.2000    0.1667
      0.2500    0.2000    0.1667    0.1429
      0.2000    0.1667    0.1429    0.1250
 matricea B
```

```
      1    -1    -2    -3
      3     1    -1    -2
      4     5     1    -1
      5     6     7     1
   Su =
     1.5000   -0.6667   -1.7500  -2.8000
     3.3333    1.2500   -0.8000  -1.8333
     4.2500    5.2000    1.1667  -0.8571
     5.2000    6.1667    7.1429   1.1250
Product =
     3.5000    2.2833    0.3167  -2.2167
     2.7167    1.9167    0.4500  -1.5333
     2.2310    1.6405    0.4667  -1.1738
     1.8964    1.4310    0.4512  -0.9512
square_A =
     0.4516    0.3263    0.2571   0.2127
     0.3263    0.2358    0.1859   0.1538
     0.2571    0.1859    0.1465   0.1213
     0.2127    0.1538    0.1213   0.1004
The result of the division A\B is
Res =
   1.0e+004 *
    -0.0600    0.2380   -0.4940  -0.1420
     0.4620   -1.5900    3.7980   1.0920
    -0.9660    2.8980   -7.7280  -2.2260
     0.5880   -1.5680    4.5640   1.3160
rank_B =      4
```

**Example 2.2**: Write a Matlab function that receives as arguments three values a,b,p and generates the linearly spaced vector v=a:p:b. The function returns the generated vector, its length, and the sum of its elements. Test the created function for the following triplets of values: (0,25,5), (2,19,3), (5,-3,-2), (5,-2,0), (2,19,-1).

*Solution*: The steps for solving the problem are as follows:

i) Creating an M-file, named, for example, `vect_lin.m`:

```
>> edit vect_lin
```

ii) Solving the problem requirements by writing the commands in the function file:

```
function [v,leng,s]=vect_lin(a,b,p)
v=a:p:b; leng=length(v);
if leng>0
    s=sum(v);
else
    s=[];
end
```

iii) Running the function file for the required triplets of values and displaying the results:

```
>> [v,leng,s]=vect_lin(0,25,5)

v =
     0     5    10    15    20    25
```

```
leng  =     6
su    =    75
>> amin=2; amax=19; step=3;
>> [w,l,s]=vect_lin(amin,amax,step)
w =
      2      5      8     11     14    17
l  =     6
s  =    57
>> a=5; b=-3; p=-2;
>> [v,leng,s]=vect_lin(a,b,p)
v =
      5      3      1     -1    -3
leng  =     5
su    =     5
>> [v1,l1,s1]=vect_lin(5,-2,0)
v1 =
   Empty matrix: 1-by-0
l1 =     0
s1 =     []
>> [v2,l2,s2]=vect_lin(2,19,-1)
v2 =
   Empty matrix: 1-by-0
l2  =    0
s2  =    []
```

**Example 2.3**: Write a Matlab function that receives as arguments two vectors, v and w, of length 3, and returns the euclidean norms of the vectors, their scalar product, their vectorial product and the angle between the vectors expressed in radians.

*Solution*: The steps for solving the problem are as follows:

i) Creating the M-file:

```
>> edit vectors
```

ii) Writing the commands in the created M-file:

```
function [n_v,n_w,ps,pv,angle]=vectors(v,w)
if length(v)~=3 | length(w)~=3
    disp('Vectors do not have a length of 3!')
    n_v=[]; n_w=[]; ps=[]; pv=[]; angle=[];
    return;
end
n_v=norm(v); n_w=norm(w);
ps=dot(v,w); pv=cross(v,w);
if n_v==0 | n_w==0
    disp('Angle cannot be calculated, one of the vectors is zero.')
    angle=[];
else
    angle=acos(ps/(n_v*n_w));
end
```

iii) Testing the program diferent pairs of vectors:

```
>> v=[1 -1 3]; w=[0 3 -2];
>> [n_v,n_w,ps,pv,angle]=vectors(v,w)
```

```
n_v =    3.3166
n_w =    3.6056
ps =    -9
pv =    -7     2     3
angle =    2.4228
>> a=[1 2 3]; b=[0 0 0]; c=[-1 -2];
>> [n_v,n_w,ps,pv,angle]=vectors(a,b)
Angle cannot be calculated, one of the vectors is zero.
n_v =    3.7417
n_w =      0
ps  =    0
pv  =    0       0     0
angle =      []
>> [n_v,n_w,ps,pv,angle]=vectors(a,c)
Vectors do not have a length of 3!
n_v =      []
n_w =      []
ps  =     []
pv  =     []
angle =     []
```

*Comments:* 1. It is necessary to verify compliance with the conditions imposed on the arguments submitted to the function (In this case, the two vectors transmitted as parameters each must have a length equal to 3).

2. Both the general case and the particular cases that generate errors must be considered during testing.

3. The `return` command causes the execution of the program to be stopped and returns the control to the invoking function or keyboard.

**Example 2.4**: For a square matrix transmitted as a parameter to a Matlab function it is required to write a sequence of instructions through which to: specify whether the matrix is invertible; if the matrix is invertible, display its inverse, and if the matrix is not invertible, display its rank.

*Solution*: The steps for solving the problem are as follows:

i) Creating the M-file:

```
>> edit matr_calcul
```

ii) Writing the source cod in the file:

```
function matr_calcul (M)
[lin,col]=size(M);
if lin~=col
    disp('The matrix is not a square one!')
    return
end
if det(M)~=0
    disp('Matrix is invertible.')
    inverse=inv(M)
else
    disp('Matrix is not invertible.')
    rank_M=rank(M)
end
```

iii) Testing the program:

```
>> A=[1 0; -1 1]; B=[1 1; 2 2]; C=[1 1 2; 2 2 3];
>> matr_calcul(A)
Matrix is invertible.
inverse =
     1     0
     1     1
>> matr_calcul(B)
Matrix is not invertible.
rank_M =    1
>> matr_calcul(C)
The matrix is not a square one!
```

**<u>Example 2.5</u>:** Define in Matlab the function $f: \boldsymbol{R} \to \boldsymbol{R}, \boldsymbol{f}(x) = \frac{1}{1+e^{-3x}}$

*<u>Solution</u>*: The steps for solving the problem are as follows:

i) Creating the M-file:

```
>> edit f
```

ii) Writing the source cod in the file:

```
function y=f(x)
if imag(x)==0
    y=1./(1+exp(-3*x));
else
    y='x has to be a real number or a vector of real numbers';
end
```

iii) Testing the program:

```
>> f(0.5)
ans =     0.8176
>> y=f(-2)
y =     0.0025
>> f(1+i)

x has to be a real number or a vector of real numbers
```

*Comments:* 1. The Matlab function *imag* returns the coeficient of the imaginary part of a complex number.

2. Matlab implicitly works with complex numbers. Since the mathematical function is defined on a real number domain, it is necessary to test whether the argument received by the corresponding Matlab function is a real number or a vector of real numbers.

3. The comparison `imag(x)==0` can be replaced with `isreal(x)==1`.

4. It is advisable to use array operators for functions in order to simultaneously calculate function values across multiple points of the definition domain.

5. If it is desired to simultaneously calculate the values of the function f in several points of the definition domain, then the following steps have to be taken:

- a vector is created with the points in the definition domain where it is desired to find the values of the function f, for example:

```
>> x=[-5.3 -2 -1.47 0 0.25 1];
```

- the function file that contains the function definition f is invoked:

```
>> y=f(x)
```

- as a result of this invoking, the values of the f function at the desired points are displayed:

```
y =    0.0000    0.0025    0.0120    0.5000    0.6792    0.9526
```

## 2.3. Problems to solve

**P2.1.** Generate and display:

- the square matrix of the order *n*=4, whose elements are given by the expression:

$$M_{i,j} = \frac{i*j}{i+j-1}, \qquad i,j = \overline{1,n}$$

- matrix A with 4 lines and 5 columns, whose elements are:

$$A_{i,j} = \begin{cases} 3, & if \quad i = j \\ -3, & if \quad |i-j| = 2 \\ 1, & if \quad i+j = 3 \\ 0, & \text{otherwise} \end{cases}$$

**P2.2.** Write a program, which, using a *while-loop* statement, calculates the partial products of the elements of the vector *v1=[2 3 1 9 2 -1 -3 5]* until it encounters a negative number and displays the last calculated product. What is the result displayed if the *v*1 vector is replaced with the vector *v2=[2 3 1 9 2 1 3 5]*?

**P2.3.** Write a Matlab function that receives a matrix with at least 4 lines and 4 columns as argument and displays the results of the following matrix elements extraction operations:

- the third line;
- the last column;
- the last line;
- the submatrix determined by the lines 2-4 and columns 1-3.

**P2.4.** For the vectors:

$$\vec{v} = 2\vec{i} - \vec{j} + 3\vec{k}\,, \qquad \vec{w} = 3\vec{j} - 2\vec{k}$$

Determine, by writing a Matlab program, the euclidean norms, the scalar product, the angle cosine, the angle expressed in degrees and vectorial product for the two vectors.

**P2.5.** Write a Matlab function that receives as argument a square matrix and displays the transposed, the rank and the determinant of the matrix.

**P2.6.** Write a Matlab function that accepts as argument a nonsingular square matrix and returns the determinant and inverse of the matrix.

**P2.7.** Define in Matlab the function: $f\colon \boldsymbol{R} \to \boldsymbol{R}, \boldsymbol{f}(x) = \begin{cases} \frac{\sin(3x)}{2x}\,, & x < 0 \\ \cos(3x)\,, & x \geq 0 \end{cases}$. Calculate and display the function values for the points: -3, $-\frac{\pi}{2}$, 0, 1.25, $\frac{7\pi}{2}$.

# ANNEX M2. Vectorial algebra elements

## M2.1. Vectorial algebra elements

Considering the $n$-dimensional real space, relative to an orthogonal coordinate system having the versors: $\vec{\iota_1}$, $\vec{\iota_2}, \dots, \vec{\iota_n}$.

Let $\vec{v}$ and $\vec{w}$ be two vectors known by their projections on the axes of the coordinate system:

$$\vec{v} = v_1\vec{\iota_1} + v_2\vec{\iota_2} + \cdots + v_n\vec{\iota_n}$$

$$\vec{w} = w_1\vec{\iota_1} + w_2\vec{\iota_2} + \cdots + w_n\vec{\iota_n}$$

***The euclidian norm of the vector*** $\vec{v}$ is the real positive number determined by the relation:

$$v = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

A vector whose module is equal to 1 is called a ***unit vector*** or ***versor***.

***The p norm of the vector*** $\vec{v}$ is the real positive number determined by the relationship:

$$v_p = \sqrt[p]{|v_1|^p + |v_2|^p + \cdots + |v_n|^p},\ p \in [1,\infty)$$

***The scalar product of vectors*** $\vec{v}$ ***and*** $\vec{w}$ is a real (scalar) number determined through the relationship:

$$\vec{v}\vec{w} = v_1w_1 + v_2w_2 + \cdots + v_nw_n$$

Vectors $\vec{v}$ ***and*** $\vec{w}$ are orthogonal if and only if $\vec{v}\vec{w} = 0$.

***The angle between vectors*** $\vec{v}$ ***and*** $\vec{w}$ is the small angle determined by the positive senses of the two vectors. The angle between vectors $\vec{v}$ and $\vec{w}$ is noted with $\angle(\vec{v}, \vec{w})$. The cosine of this angle is calculated by the formula:

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v}\vec{w}}{vw}$$

In the case of the three-dimensional space (n = 3), **the vectorial product of the *vectors*** $\vec{v}$ ***and*** $\vec{w}$, if they are non-collinear and non-zero, is defined as the vector denoted $\overrightarrow{\vec{v}x\vec{w}}$ which has the direction perpendicular to the directions of the two vectors which completes a right-handed system and the module equal to the product $vw\sin(\vec{v}, \vec{w})$, respectively the null vector, if the two vectors are collinear or at least one of them is the null vector.

The vectorial product of the vectors $\vec{v}$ and $\vec{w}$ can be calculated with the formula:

$$\vec{v}\text{x}\vec{w} = \begin{vmatrix} \vec{\iota_1} & \vec{\iota_2} & \vec{\iota_3} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} = (v_2w_3\text{-}v_3w_2)\,\vec{\iota_1} + (v_3w_1\text{-}v_1w_3)\,\vec{\iota_2} + (v_1w_2\text{-}v_2w_1)\,\vec{\iota_3}$$

Two non-collinear and non-zero vectors $\vec{v}$ and $\vec{w}$ are **parallel** if and only if $\overrightarrow{\vec{v}\text{x}\vec{w}} = 0$.

1