Democratic and Popular Algerian Republic
Ministry of Higher Education and Scientific Research

MUSTAPHA STAMBOULI UNIVERSITY OF MASCARA
FACULTY OF SCIENCE AND TECHNOLOGY

Course Handout

Programming with Matlab

Presented by:

Dr. BESSAIM Mohammed Mustapha

This course is for undergraduate students specialty Civil Engineering/Public Works

Algeria
2023

Foreword

In the academia’s realm, where learning and innovation converge, the pursuit for knowledge knows no bounds. It's within this dynamic landscape that we present this modest course on Matlab, tailored precisely for undergraduate students, enrolled in the field of Civil Engineering, speciality: Civil Engineering / Public Works.

Matlab, an acronym for “Matrix Laboratory” assists as a trusted tool during your formation, allowing you to transform abstract concepts towards tangible solutions. Not a limited to a specified discipline, Matlab offers the keys to unlock any analytical thinking, problem-solving and creativity.

This course of five chapters was written in a way to allow to the students how to easily get familiar with Matlab R2019a environment. Furthermore, this course includes set of examples and illustrations to make students interact directly with the Matlab world.

Overall, our hope is that, armed with the knowledge and insights found in these chapters, you will approach your studies and research with newfound confidence and enthusiasm.

This course corresponds to the Computer Practical Work 3 (Matlab), which is taught at the University of Mustapha Stambouli, Mascara, for the undergraduated students, enrolled in Civil Engineering, specialty: Civil Engineering and Public Works.

Mascara, 12th November, 2023.

Table of contents

Foreward i

Abstract iv

Notations and Abreviations v

Introduction 9

Chaptre 1 Introduction to Matlab 11

1.1 Programming Environment 11

1.2 Running your code 13

1.3 Getting help 13

1.4 Scalar quantities and variables 13

1.5 Rules when naming variables in Matlab 16

1.6 Mathematical functions 17

1.7 The way to display numbers in Matlab: Format 20

1.8 Variables and their size 21

Chaptre 2 Vectors and Matrices 23

2.1 Vectors 23

2.1.1 Vectors creation 23

2.1.2 Calculation of the row vector transpose 24

2.1.3 Matlab's syntax when creating row vectors 25

2.1.4 Acces to any element from a given vector 26

2.1.5 Element by element operations for vectors 28

2.1.6 The linespace function 31

2.2 Matrices 32

2.2.1 Matrices creation 32

2.2.2 Referencing and access to matrix elements 34

2.2.3 Size of a matrix 37

2.2.4 Matlab's predefined matrices functions 38

2.2.5 Element-by-element matrix operations 39

2.2.6 Additional useful matrices functions 42

Chaptre 3 Matlab Plotting 46

3.1 Matlab plot function 46

3.2 Curves customizing 51

3.3 Multiple curves in a same graph 53

3.3.1 The command ***hold*** 53

3.3.2 Plot with several arguments 54

3.4 Histogram and bar graphs in Matlab 55

3.5 Plotting 3D curves in Matlab 58

Chaptre 4 Matlab’s loops and conditional statements 62

4.1 The “if, else, eleseif” condition 62

4.2 The “switch case otherwise end” condition 64

4.3 Loop control statements 66

4.3.1 The “for” loop 66

4.3.2 The “while” loop 67

Chaptre 5 Matlab Toolbox 69

5.1 Add additional Toolbox 69

5.2 Using of the “Curve Fitting Toolbox” 71

5.3 Creating your own Toolbox 72

References 78

Abstract

This course destinated for undergraduate students, enrolled in Civil Engineering and Public Works, aims to enter you into the world of computational exploration and problem-solving via the Matlab (R2019a) programming language. In this course, our main mission is to offer you the fundamental elements of Matlab, through a comprehensive foundation and extending into advanced applications. The set of the given examples ensure that your understanding is tangible and practical. Moreover, you will discover Matlab's true power shines in its mastery of vectors and matrices. With intuitive syntax and a wealth of built-in functions given by a set of illustrations, it will show you how it will be easy to handle complex linear algebra.

Not limited to that, you will explore the Matlab richness in graph plotting, with its extensive array of functions and libraries, as well as discovering the impressive diversity of chart types offered by Matlab, passing from simple line graphs, bar charts, 3D plotting and so on.

Beyond plotting, Matlab's Toolbox extends to specialized fields. you can explore the “Curve Fitting Toolbox”, and learn how to manage/add further Toolbox. At the end, you will be able to creat your own costumized Toolbox.

In summary, this course provides a versatile and accessible platform for learning and applying computational mathematics, data analysis, as well as enhancing your problem-solving skills.

Keywords: Matlab, Programming Language, Problem Solving, Graph Plotting, Toolbox

Notations and Abreviations

+	Addition
-	Minus
*	Multiplication
/	Division (From left to right)
\	Division (From right to left)
sin ()	Sinus in radiant
cos ()	Cosines in radiant
tan ()	Tangent in radiant
sind ()	Sinus in degree
cosd ()	Cosines in degree
tand ()	Tangent in degree
asin ()	Inverse of sinus
acos ()	Inverse of cosines
atan ().	Inverse of tangent
exp ()	Exponential
log ()	Natural logarithm
log10 ()	Common logarithm
sqrt ()	The square root
abs(x)	The absolute value $x \rightarrow |x|$
round(x)	Rounds a number to the nearest integer
ceil(x)	Rounds a number up to the nearest integer
floor(x)	Rounds a number down to the nearest integer

fix(x)	Rounds a number to the nearest integer towards zero
rem(x)	The remainder left after division
mod(x)	The signed remainder left after division
abs(x)	The absolute value of x
sign(x)	The sign of x
factor(x)	The prime factors of x
format long	To display 14 digits after the decimal point
format short	To display 4 digits after the decimal point
format bank	To display only 2 digits after the decimal point
format rat	To display numbers as a ratio
clear a	Delete variable a
clear, clear all	Delete all variables
clc	clears the Command Window (clear screen)
exit, quit	Close the Matlab environment
transpose []	The transpose of a vector / matrix
.*	Element-by-element multiplication
./	Division element by element
.^	Element-by-element power
ones(n)	Generates an n × n matrix with all elements = 1
ones(m,n)	Generates an m × n matrix with all elements = 1
zeros(n)	Generates an n × n matrix with all elements = 0
zeros(m,n)	Generates an m × n matrix with all elements = 0
eye(n)	Generates an n × n identity matrix (ones on the main diagonal and zeros elsewhere)
rand(m,n)	Generates an m × n matrix of random values
magic(n)	Generates a magic matrix of dimension n × n

plot (x , y)	Draw a 2D line plot of the data, in ‘y’ versus the corresponding values ‘x’
if, elseif, else	Execute statements if condition is true
switch, case, otherwise	Execute one of several groups of statements
for	for loop to repeat specified number of times
while	while loop to repeat when condition is true

Introduction

In the field of computational tools, Matlab stands as an incredible titan, revered by engineers, scientists, and problem solvers of several disciplines. Short for "Matrix Laboratory," Matlab is more than just a software; it is a mathematical-engineering package, and a powerful programing language, that turning complex concepts into practical solutions [1].

Matlab's origins can be traced back to the late 1970s, when it was developed by “Cleve Moler” as a computational tool for his students at the University of New Mexico. Over the years, it has evolved into a comprehensive software platform, encompassing an enormous array of Functions and Toolboxes, that cater to nearly every field of study and research. whether you are a student exploring the fundamentals of a subject or a seasoned expert pushing the boundaries of your discipline, Matlab has always something to offer [2].

At its core, Matlab excels in numerical computing, making it an impressive instrument for tackling mathematical problems as well as performing data analysis. It operates with matrices, which allows for an effective and an intuitive manipulation of data. However, its value extends far beyond mathematics. Matlab facilitates programming, modeling and also simulation, while its diverse Toolboxes are opening doors to different fields. From control systems and optimization to signal analysis, Matlab provides a wide arsenal of functions for a wide range of applications [3].

One of Matlab's uniqueness features is its emphasis on visualization. The capability to create captivating, informative graphs and charts is integral to scientific and engineering research. Matlab's plotting ability are not only versatile but also highly customizable. It offers the creation of 2D graphs, 3D plots as well as contour maps, all of them enable users to communicate their outcomes effectively [4].

This course represents a selection of information and data that undergraduate students may be required to use during their academic background. Furthermore, this course is intended to be a solid basis for restoring the technical and practical knowledge essential for an efficient usage of the Matlab.

This course comprises five main chapters :

- Presentation of the Matlab environment, Scripts and Scalar Quantities and Variables;
- Vectors and Matrices creation;
- Matlab’s Graphs Plotting;
- Loops and Conditional Statements;
- MATLAB’s Toolboxes.

Each chapter encloses sections, that cover specific topics in Matlab.

The first chapter wil explain the Matlab’s environment and start with basic scalar operations, showing how Matlab can acts as a very powerful calculator.

The second one will illustrate the most useful commands for Vectors and Matrices, allowing to students how to deal easily with any encountered mathematical operation.

The third chapter, will present a huge variety of graphs, namely 2D line graphs, pie chart as well as bar graphs. Moreover, a step-by-step explanation is given on how to draw 3D graphs, contour plot, filled contour plot and a surface plot.

The fourth one will present how to properly use the Matlab’s loops and conditional statement. Therefore, a set of examples will be given with their flowcharts to make these laters comprehensible for the students.

The last chapter will show to the students the easiest way to deal with the Matlab’s Toolboxes. The first part will illustrate how to add and manage Matlab’s Toolboxes. The second part is dedicated to the use of the “Curve fitting Toolbox”, a relevant example for any future Engineer. The last point will explain the path to create a customized Toolbox.

At the end, the students will be able to use Matlab effectively in their coursework and research. Note that each part begins with a listing of Matlab’s commands, followed by a set of examples showing how to write accurately your programming language.

Chapter 1 Introduction to Matlab

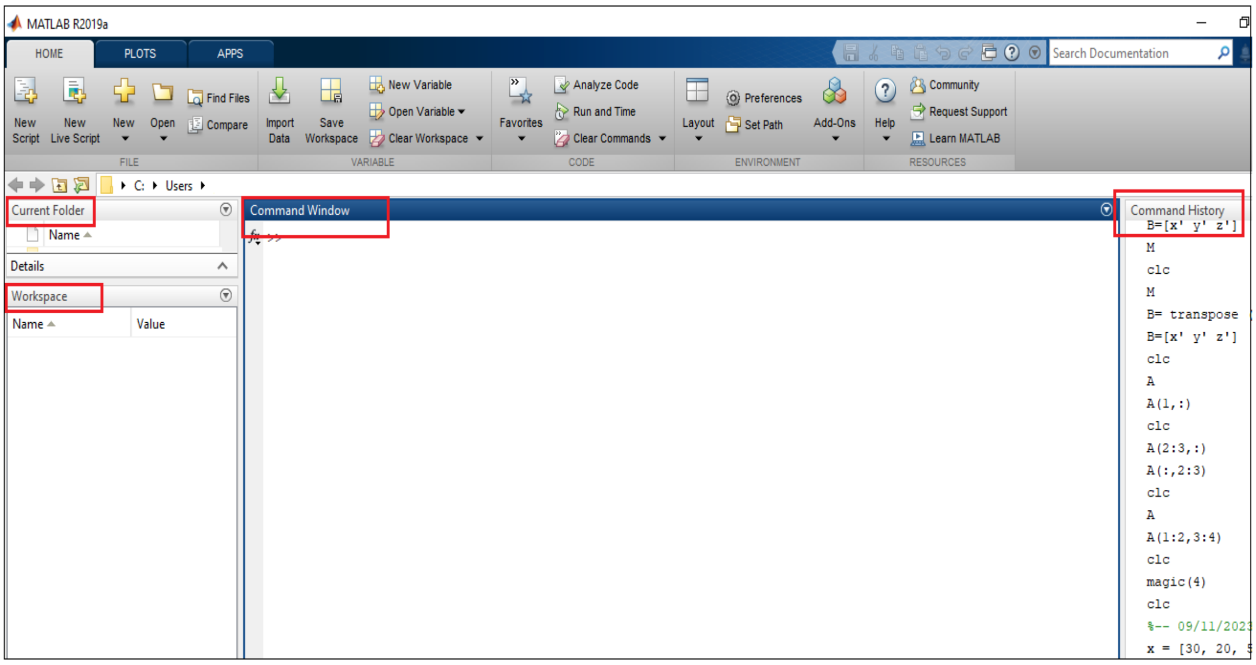
The acronym Matlab stands for "matrix laboratory." Other programming languages typically operate on single integers, whereas Matlab is intended to operate largely on complete matrices and arrays.

It is a language for scientific computing, data analysis, visualization, and algorithm development. Its interface offers, on the one hand, an interactive console-type window for executing commands, and on the other hand, an integrated development environment (IDE) for programming applications [5].

1.1 Programming Environment

The Matlab's environment is shown in Figure 1.1. As seen, it consisted from four sections:

- The Current folder box showing the folder's content,
- The Matlab Command Window started with sign »,
- The Workspace (memory of Matlab) displaying the used variables;
- The Command history box illustrating the recent commands.



```
>>
```

```
B=[x' y' z']
M
clc
M
B= transpose
B=[x' y' z']
clc
A
A(1,:)
clc
A(2:3,:)
A(:,2:3)
clc
A
A(1:2,3:4)
clc
magic(4)
clc
%-- 09/11/2023
x = [30, 20,
```

Figure 1.1 *The Matlab Programming Environment.*

The Matlab can execute commands typed directly in the Command Window, however, it is better to store the code in a bespoke ‘m. file’ or Matlab script. The creation of a new script is shown in Figure1.2.

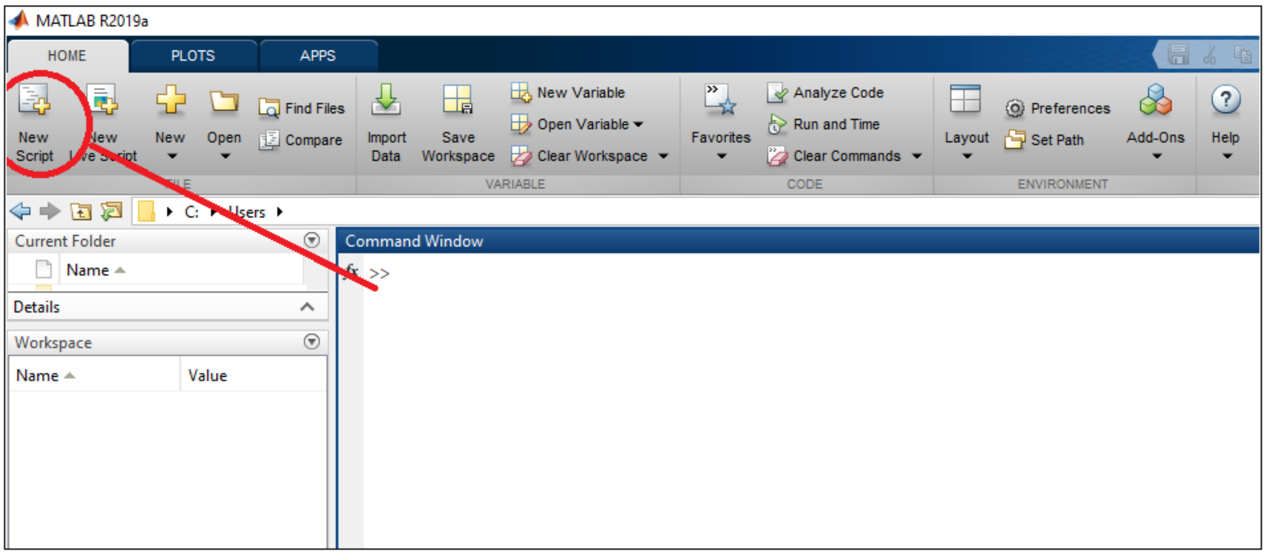


Figure 1.2 *The creation of a new Script*

Thereafter, the following editor window will appear:

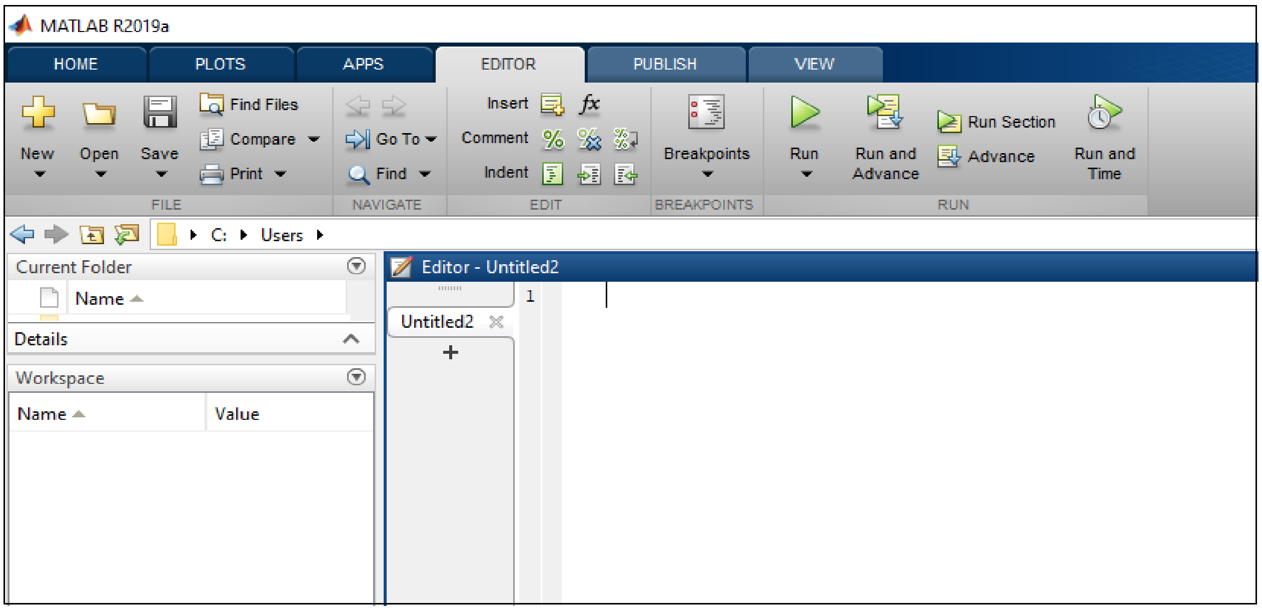


Figure 1.3 *The editor Window*

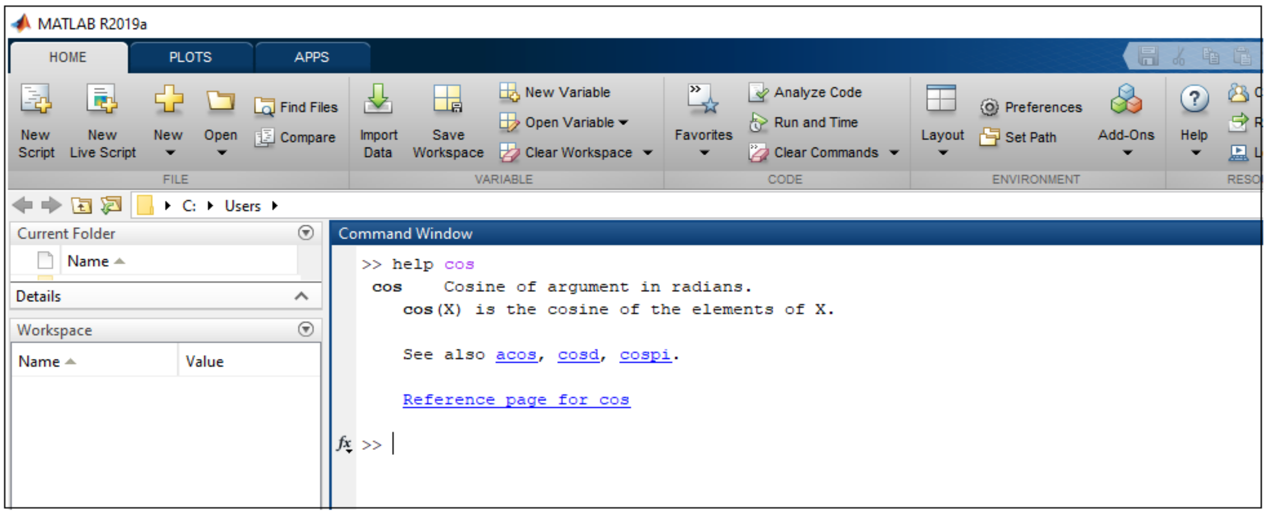
1.2 Running Your Code

Matlab commands can be executed by one of the following ways:

- Type the commands directly in the Matlab Command Window.
- Or if you have created a new script, you should type the commands within it and then click on the run icon, shown in Figure1.3. Matlab will ask you to save the file if you have not done this already.

1.3 Getting Help

Matlab has a help icon at the top right of its interface. Furthermore, to the help item in the menu, Matlab proposes the help command. For instance, help cos prints the help article regarding the command cos shown directly in the Command Window as shown:

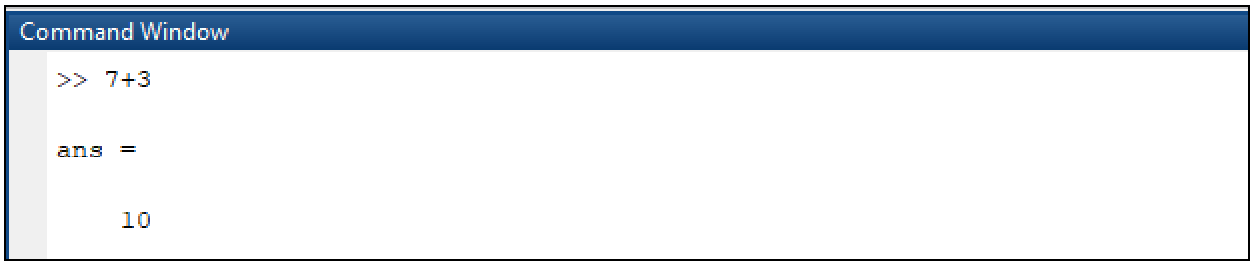


```
>> help cos
 cos    Cosine of argument in radians.
    cos(X) is the cosine of the elements of X.

    See also acos, cosd, cospi.

    Reference page for cos

>> |
```

Figure 1.4 *The help icon*

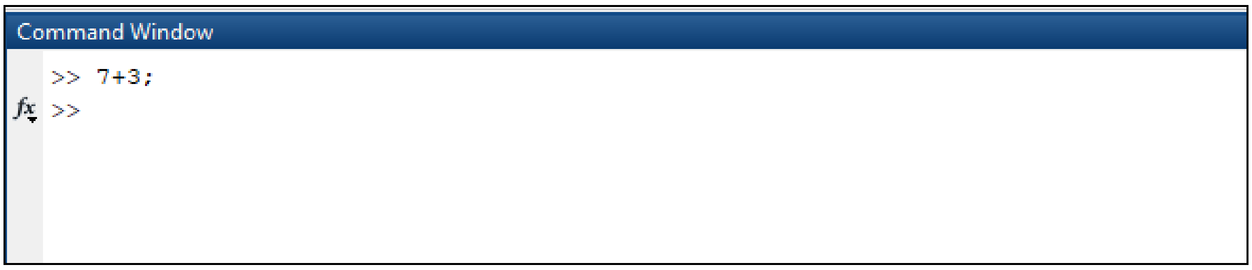
1.4 Scalar Quantities and Variables

The first interactions with Matlab are basic equations and variables. Try to write the following commands as they are shown :



Command Window

```
>> 7+3

ans =

    10
```



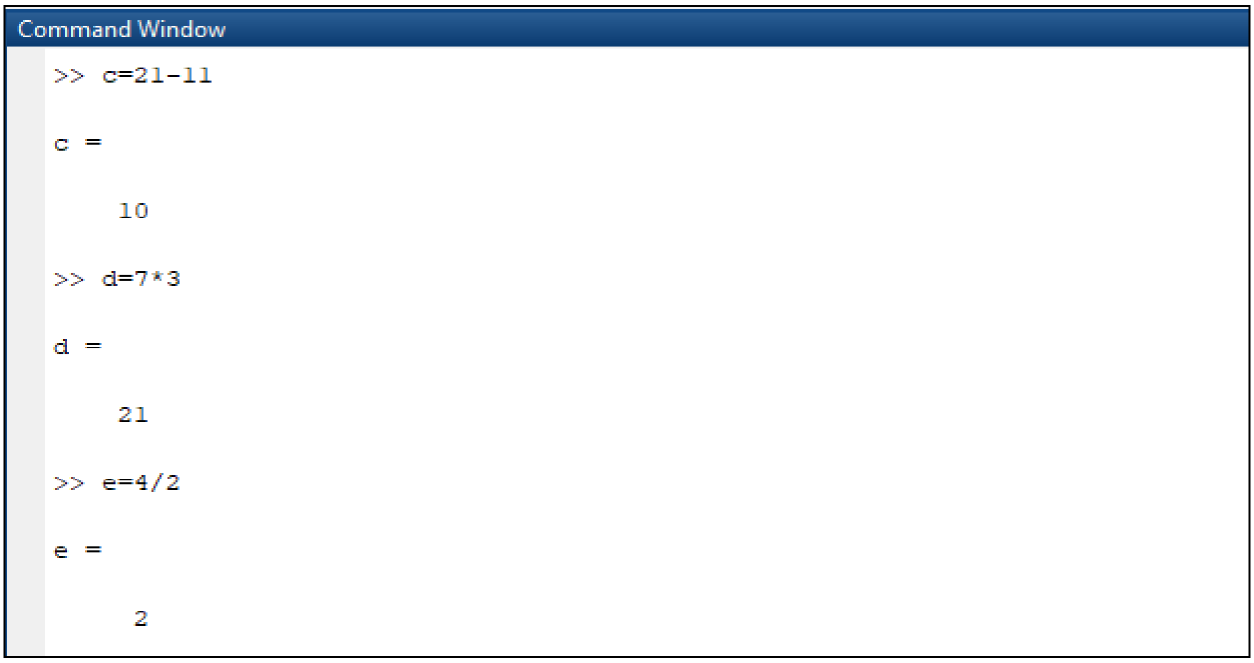
Command Window

```
>> 7+3;
>>
```

You may notice that the second command differ from the first one. The second one uses the semicolon " ; ". When using it, Matlab executes the command without showing any output. By contrast, we can see in the first one that the results have been set 10. Matlab may be used as a powerful calculator. We can start by considering simple arithmetic operations.

For instance, try to find the outcomes of these operations:
c= 21-11 ; d=7x3 ; e= 4/2

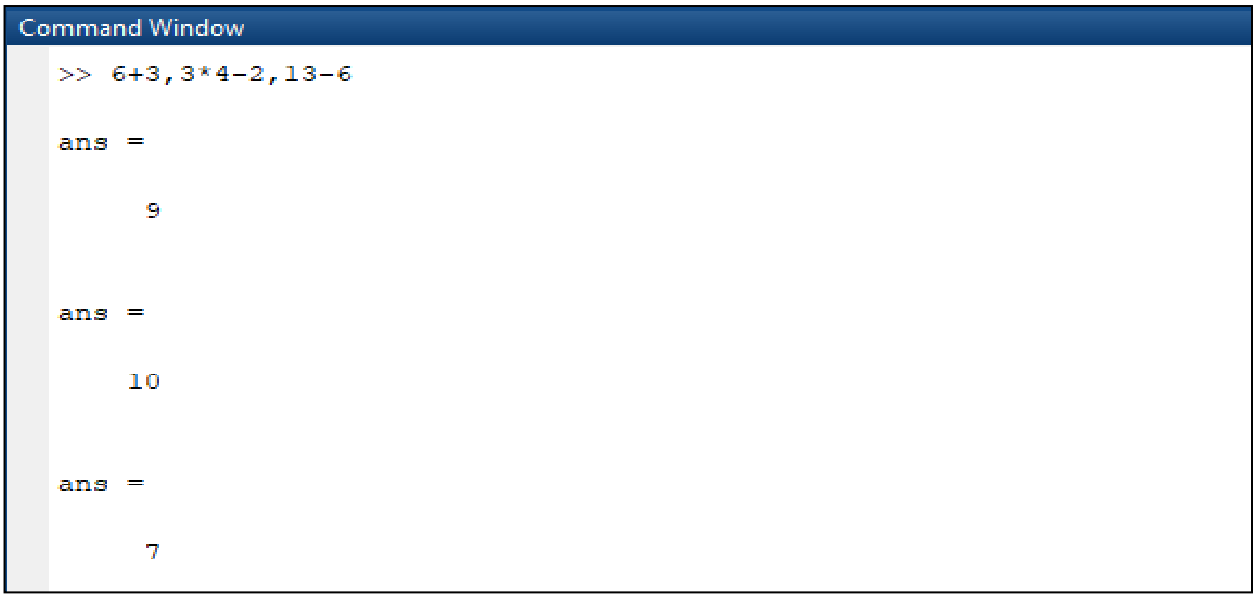
These calculations can be written in Matlab as follows:



```
>> c=21-11

c =

    10

>> d=7*3

d =

    21

>> e=4/2

e =

     2
```

In aim to approach to idea better, try writing the following instructions into Matlab:
x = 32;
y = 16;
z = -7;
x / y ;
x – z ;
y + z – x.
the answers are 2; 39; -23.

Besides, Matlab offers the possibility to write several operations in the same line as shown:


Command Window

```
>> 6+3,3*4-2,13-6

ans =

     9


ans =

    10


ans =

     7
```

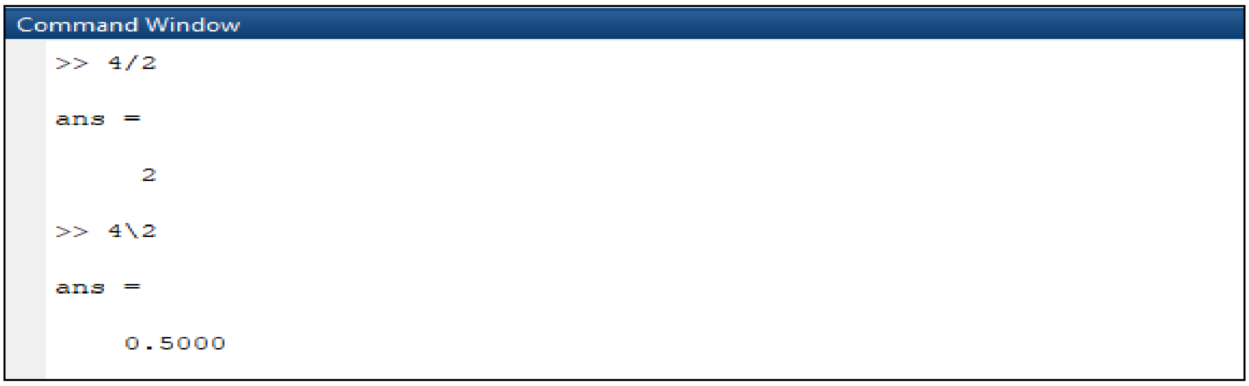
The basic arithmetic operations are summarized in the Table 1:

Table 1. *Basic arithmetic operations in Matlab*

Operation	**Signification**
+	Addition
-	Minus
*	Multiplication
/	Division (From left to right)
\	Division (From right to left)

Remarque

Matlab is case sensitive when writing the operation. In the case of division, you may notice that the inversion of the sign of division / to \ will impact the output, as you can see in the following example:


Command Window

```
>> 4/2

ans =

     2

>> 4\2

ans =

    0.5000
```

1.5 Rules when naming Variables in Matlab

In all previous examples we have simply used variable names which appeared to suit with tasks at hand with no mention of restrictions on allowable variable names in Matlab. The rules for naming variables in MATLAB can be summarized like so:

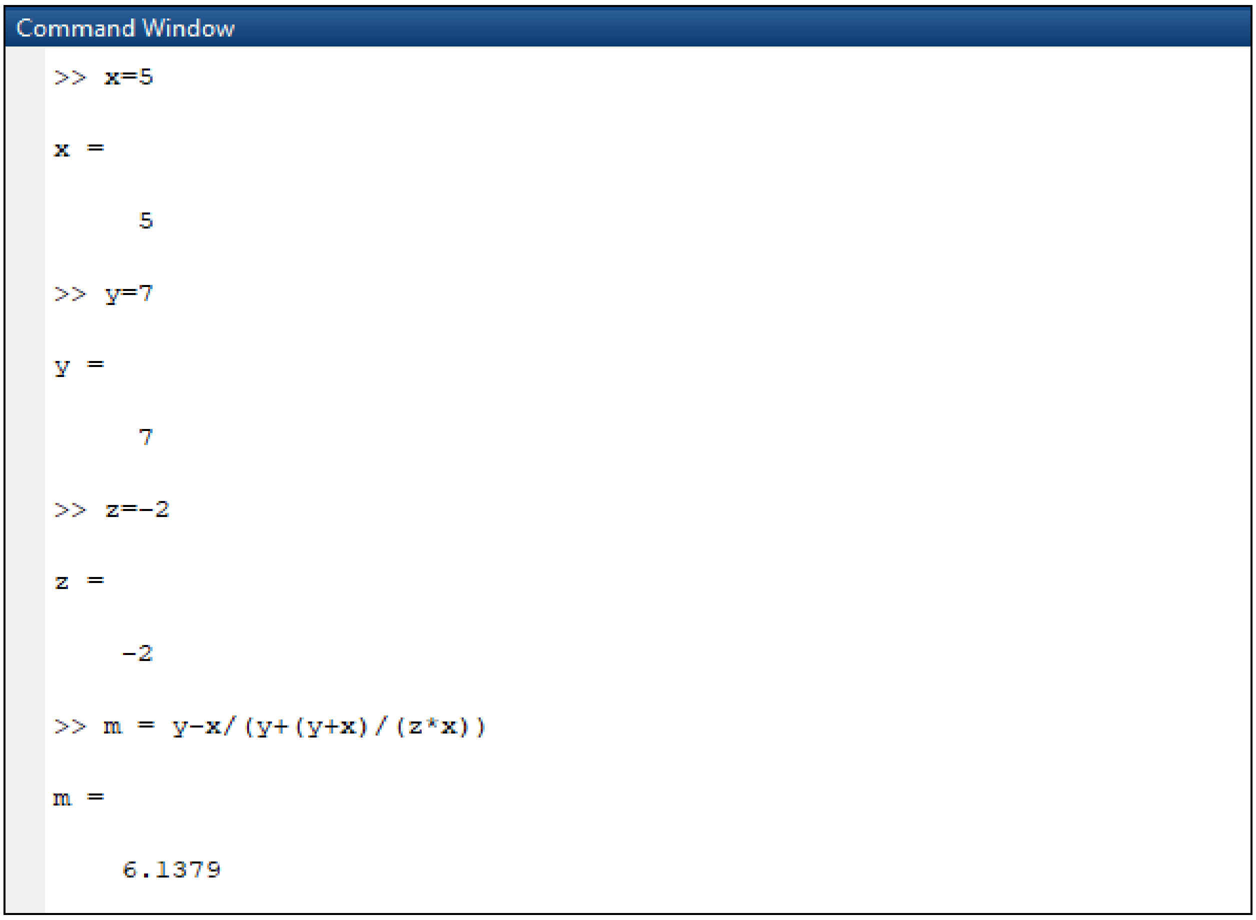
- Variable names in Matlab must be only alphanumeric characters and can be more to 31 characters long;
- Matlab is very case sensitive, so that " **a** " and " **A** " are two different variables;
- Variables names must not match with a predefined Matlab command, or with any user-defined subroutines.

Another important parameter besides on how to name variables is the usage of brackets. We must make into mind that in Matlab the syntax of x/y*z is not equal to (x/yz) but ((x/y)z). In aim to ensure that the denominator is calculated first, we would need to use x/(y*z).

To assimilate accurately the use of these brackets, try to determine the value of the following example:

$$y - \frac{x}{y + \frac{y+x}{zx}}$$

Where, x=5; y=7 and z=-2.
The solution is:
m = y-x/(y+(y+x)/(z*x))


Command Window

```
>> x=5

x =

     5

>> y=7

y =

     7

>> z=-2

z =

    -2

>> m = y-x/(y+(y+x)/(z*x))

m =

    6.1379
```

1.6 Mathematical Functions

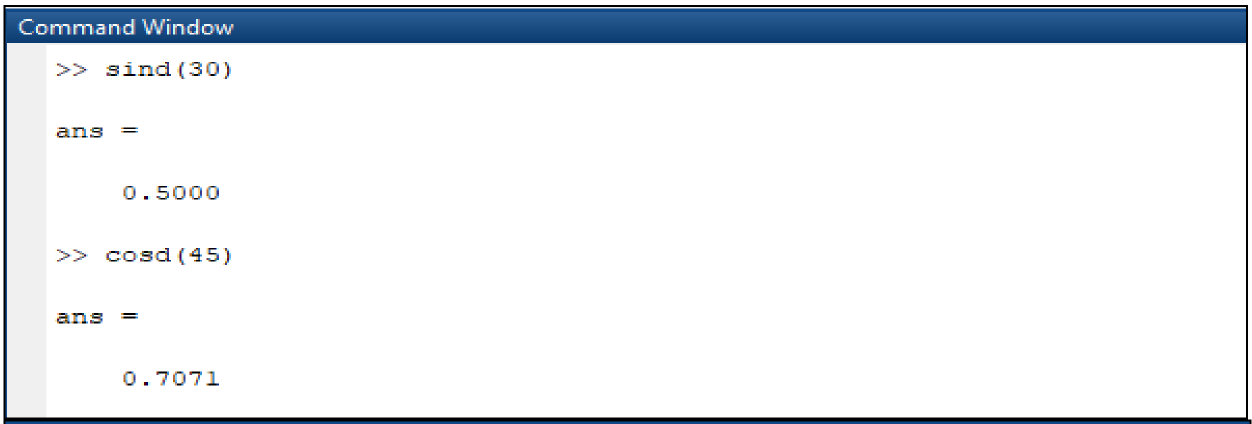
As said before, Matlab is a powerful calculator. The table 2 summarize the additional mathematical operation.

Table 2. *Trigonometric and exponential functions*

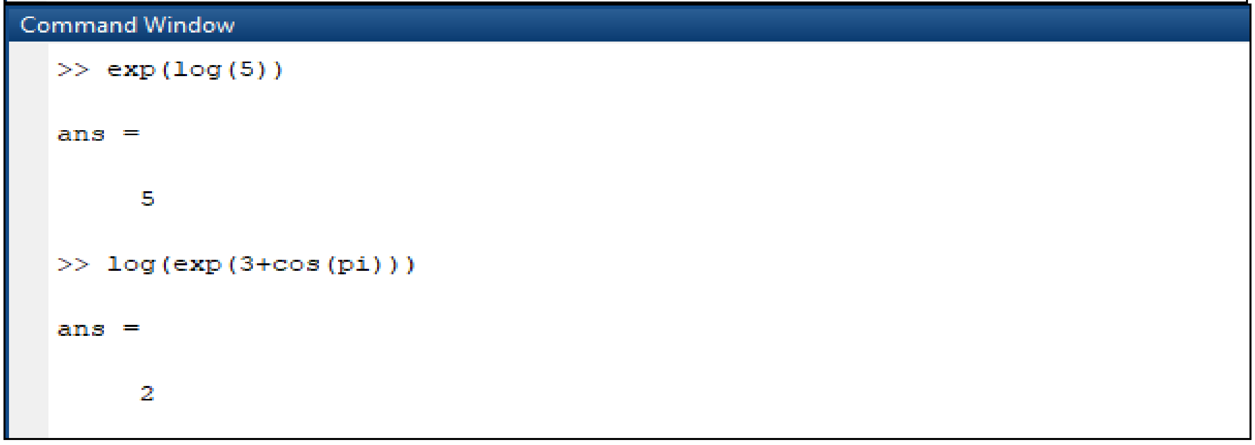
Symbol	**Signification**
sin ()	Sinus in radiant
cos ()	Cosines in radiant
tan ()	Tangent in radiant
sind ()	Sinus in degree
cosd ()	Cosines in degree
tand ()	Tangent in degree
asin ()	Inverse of sinus
acos ()	Inverse of cosines
atan ().	Inverse of tangent
exp ()	Exponential
log ()	Natural logarithm
log10 ()	Common logarithm
sqrt ()	The square root
abs(x)	The absolute value $x \rightarrow |x|$

To understand better, try to calculate the following expressions:
sin 30°; cos 45°; exp (ln (5)); ln exp(3+cosπ)

The answer will be like so:



```
>> sind(30)

ans =

    0.5000

>> cosd(45)

ans =

    0.7071
```



```
>> exp(log(5))

ans =

     5

>> log(exp(3+cos(pi)))

ans =

     2
```

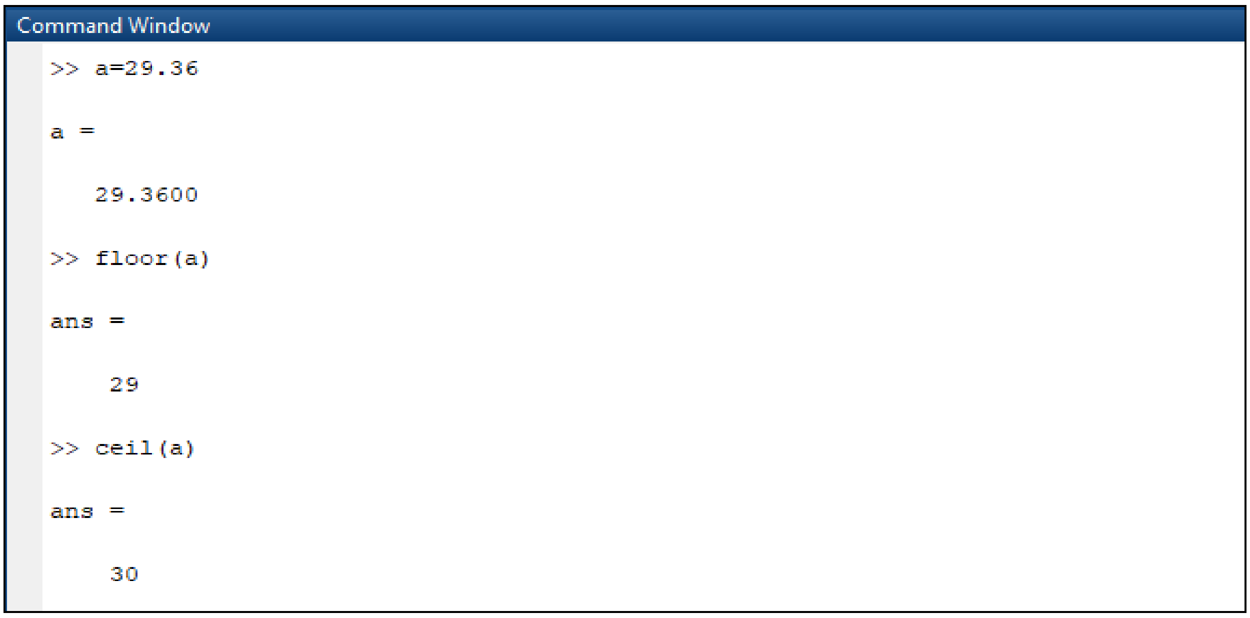
As known, Matlab encloses a huge variety of useful functions. These laters are given in the table bellow:

Table 3. *Useful functions in Matlab*

Expression	**Signification**
round(x)	Rounds a number to the nearest integer
ceil(x)	Rounds a number up to the nearest integer
floor(x)	Rounds a number down to the nearest integer
fix(x)	Rounds a number to the nearest integer towards zero
rem(x)	The remainder left after division
mod(x)	The signed remainder left after division
abs(x)	The absolute value of x
sign(x)	The sign of x
factor(x)	The prime factors of x

The command **round (), ceil (), floor ()** aims to rounds a number to the nearest, up to the nearest and down to the nearest integer, respectively. While the command **fix ()** tends to rounds a number to the nearest integer towards zero.

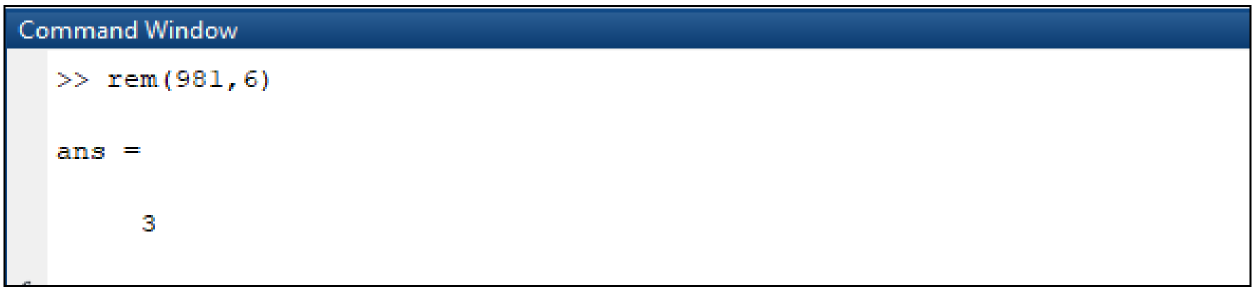
For instance, a value a equal to 29.36, we would like to use the **floor** and **ceil** functions. The outcome in Matlab will be in this way:



```
>> a=29.36

a =

   29.3600

>> floor(a)

ans =

    29

>> ceil(a)

ans =

    30
```

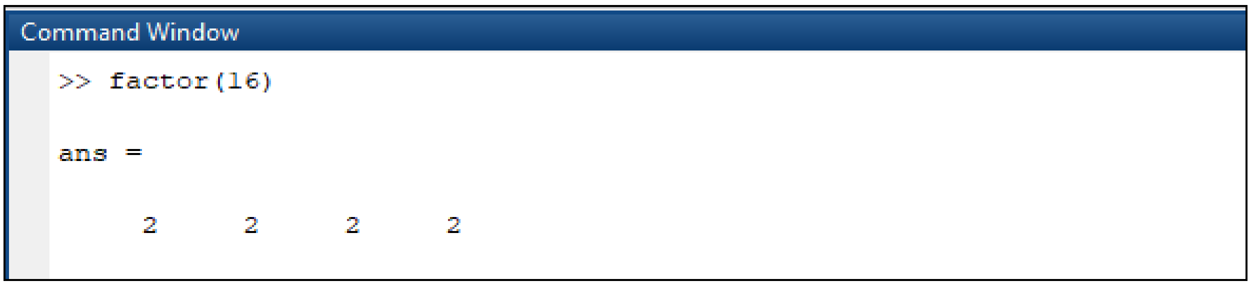
The command rem(a,b) calculates the remainder when a is divided by b.

For instance, 981 = 6 ×163 + 3, so the remainder when 981 is divided by 6 is equal to 3. We can determine this with Matlab by simply using rem (981,6).



```
>> rem(981,6)

ans =

     3
```

The command factor provides the prime decomposition of an integer. For instance, factor of the number 16 in Matlab is as follows:



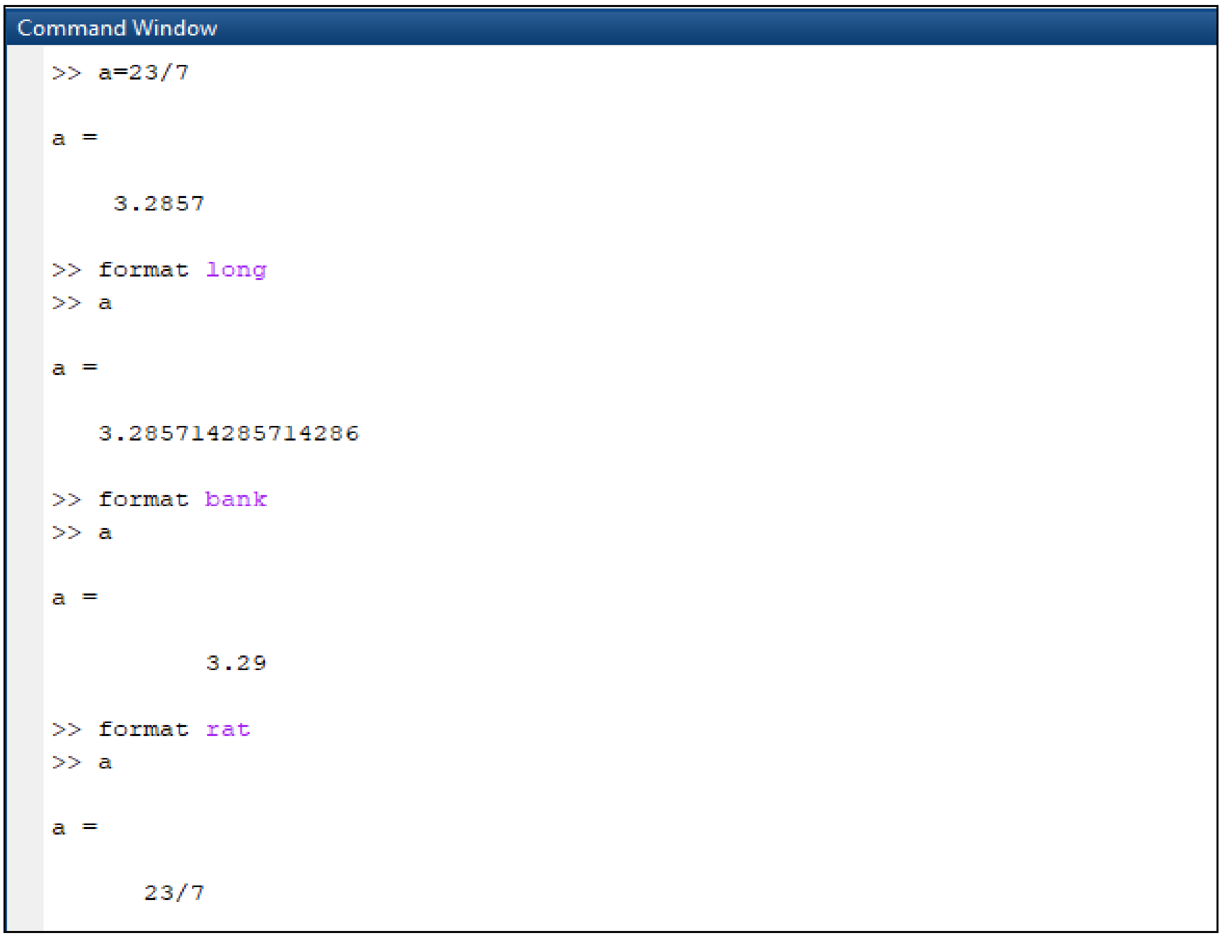
```
>> factor(16)

ans =

     2     2     2     2
```

1.7 *The way to display numbers in Matlab: Format*

Matlab uses real numbers to perform the calculations, which allows precision in the calculation of up to 16 significant digits. Accordingly, the following points should be noted:

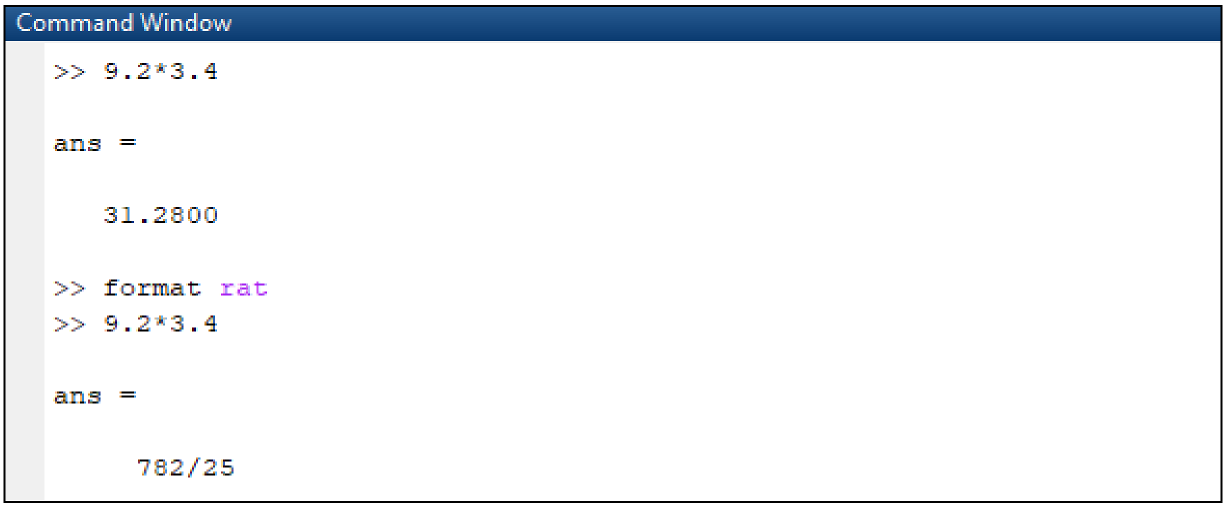
The result of a calculation operation is by default displayed with four digits after the decimal point:

- To display more numbers, use the format long command (14 digits after the decimal point);
- To display only 2 digits after the decimal point, use format bank command;
- To display numbers as a ratio, use format rat command;
- To return to the default display, use the format short command.
- The following example illustrate how to use these commands correctly.



```
>> a=23/7

a =

    3.2857

>> format long
>> a

a =

   3.285714285714286

>> format bank
>> a

a =

          3.29

>> format rat
>> a

a =

      23/7
```

Another example is how to write the multiplication result in a ration form.

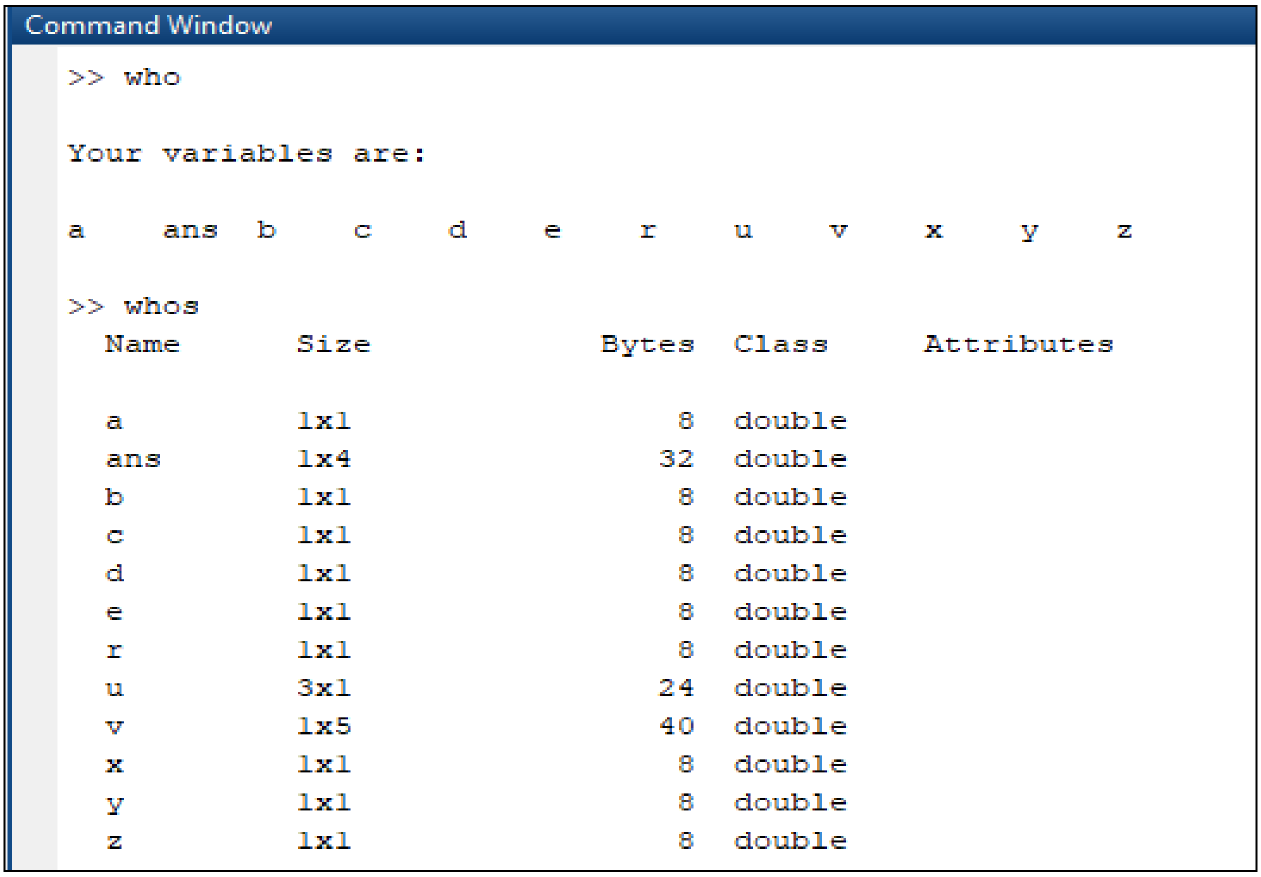


```
>> 9.2*3.4

ans =

   31.2800

>> format rat
>> 9.2*3.4

ans =

     782/25
```

1.8 Variables and their size

In Matlab, to see the list of the used variables, either look in the 'Workspace' window or use the 'who' or 'whos' commands.

The who command gives only the names of the variables, whereas whos command illustrates a detailed description (the name of the variable, its type and its size). The figure bellow illustrates the used of variables.



```
>> who

Your variables are:

a     ans   b     c     d     e     r     u     v     x     y     z

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  ans       1x4                32  double
  b         1x1                 8  double
  c         1x1                 8  double
  d         1x1                 8  double
  e         1x1                 8  double
  r         1x1                 8  double
  u         3x1                24  double
  v         1x5                40  double
  x         1x1                 8  double
  y         1x1                 8  double
  z         1x1                 8  double
```

If you desire to delete the variable “x”, you can use the command clear x. Matlab offers a set of command that can erase all the variable or just to clear the screen (Command Window). The signification of these commands is shown in the table below:

Table1.5. *Matlab’s command to delete variables and exit the Matlab environment*

Command	**Signification**
clear a	Delete variable a
clear, clear all	Delete all variables
clc	clears the Command Window (clear screen)
exit, quit	Close the Matlab environment

Chapter 2 Vectors and Matrices

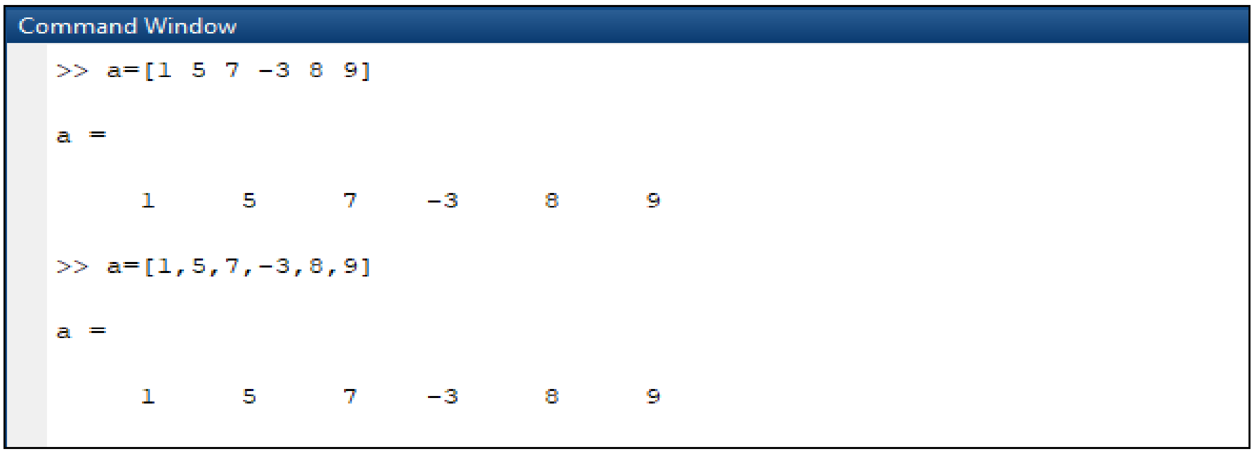
Matlab was basically designed to allow mathematicians, scientists and engineers to easily use the mechanics of linear algebra [6]. In this chapter we will illustrate the idea of initiating vectors and matrices and how to manipulate them as "MATLAB objects".

2.1 Vectors

A vector is an ordered list of elements. If the elements are arranged horizontally, we say that the vector is a row vector, on the other hand if the elements are arranged vertically, we say that it is a column vector.

2.1.1 Vectors creation

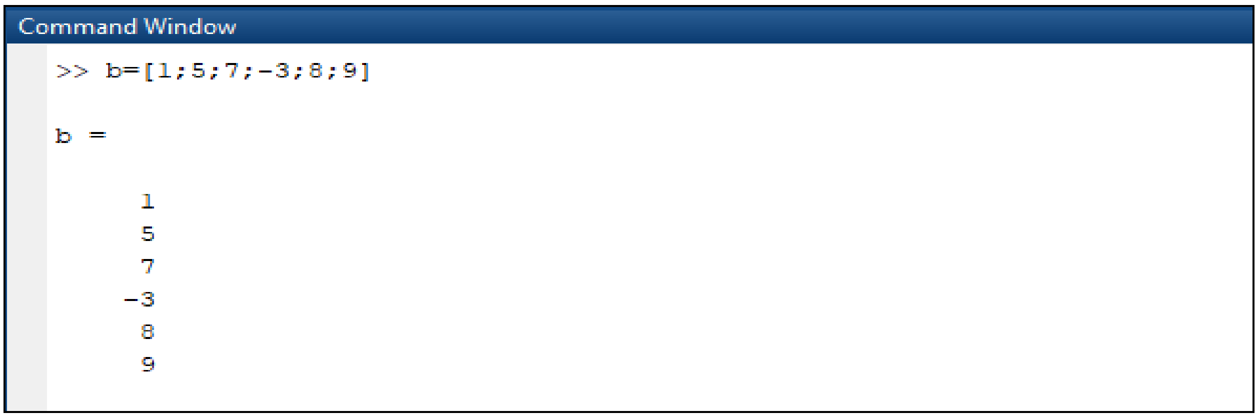
To create a row vector, simply write the list of its components in square brackets **[ ]** and separate them either by spaces or commas like so:



```
>> a=[1 5 7 -3 8 9]

a =

     1     5     7    -3     8     9

>> a=[1,5,7,-3,8,9]

a =

     1     5     7    -3     8     9
```

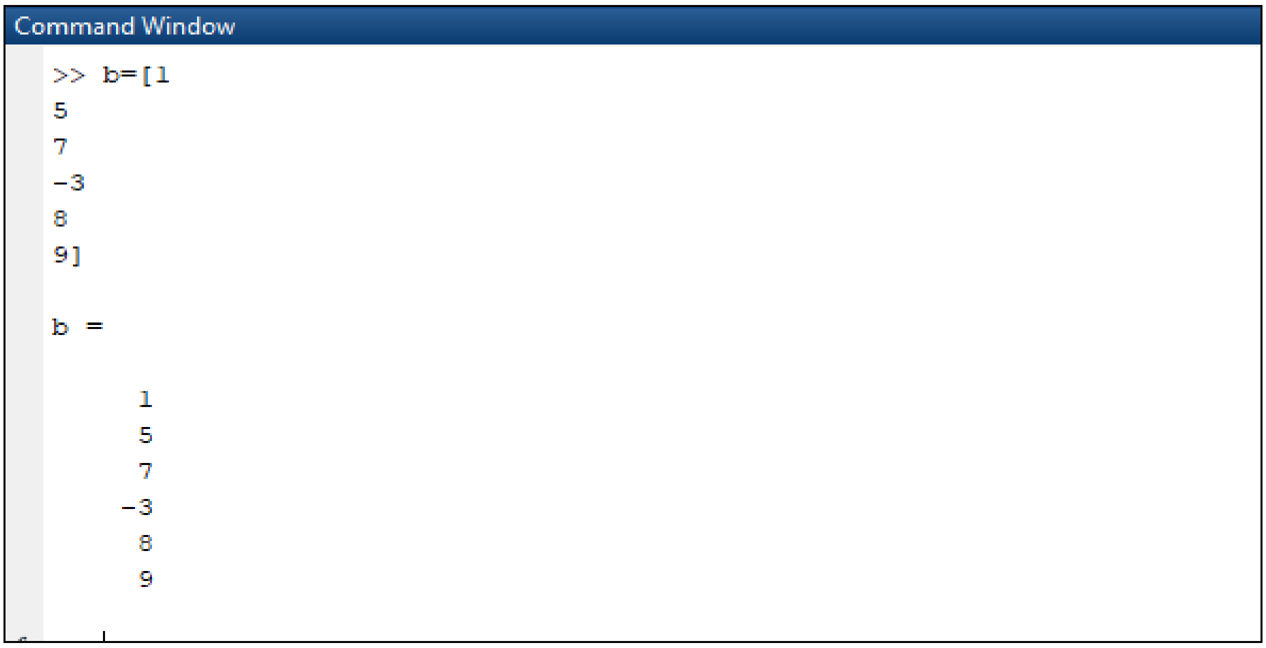
To create a column vector, it is possible to use one of the following methods:

- Write the components of the vector in square brackets [ ] and separate them with semicolons (;) like so:



```
>> b=[1;5;7;-3;8;9]

b =

     1
     5
     7
    -3
     8
     9
```

- Or, we can write the vector vertically:

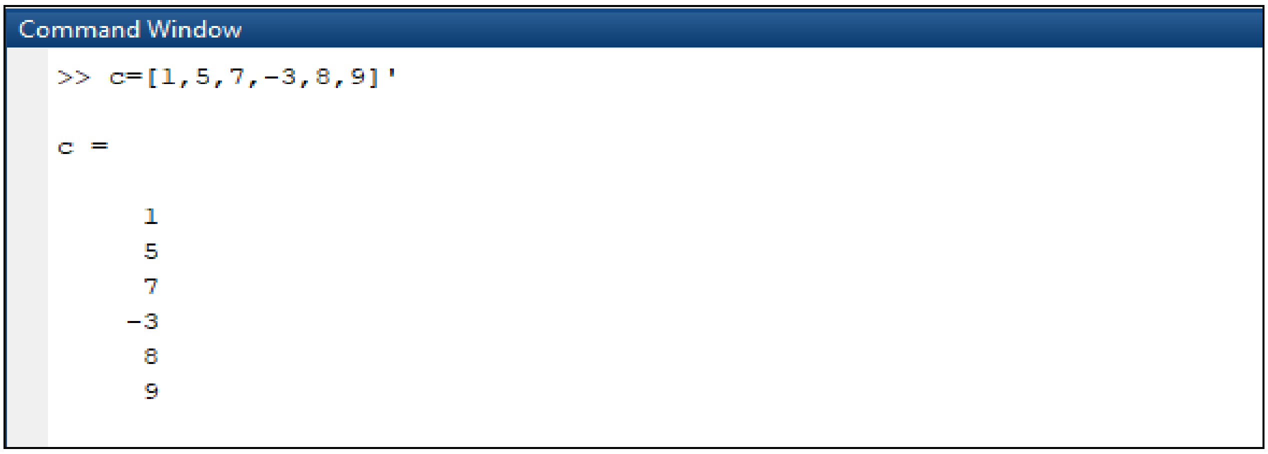


Command Window

```
>> b=[1
5
7
-3
8
9]

b =

     1
     5
     7
    -3
     8
     9
```

2.1.2 Calculation of the row vector transpose

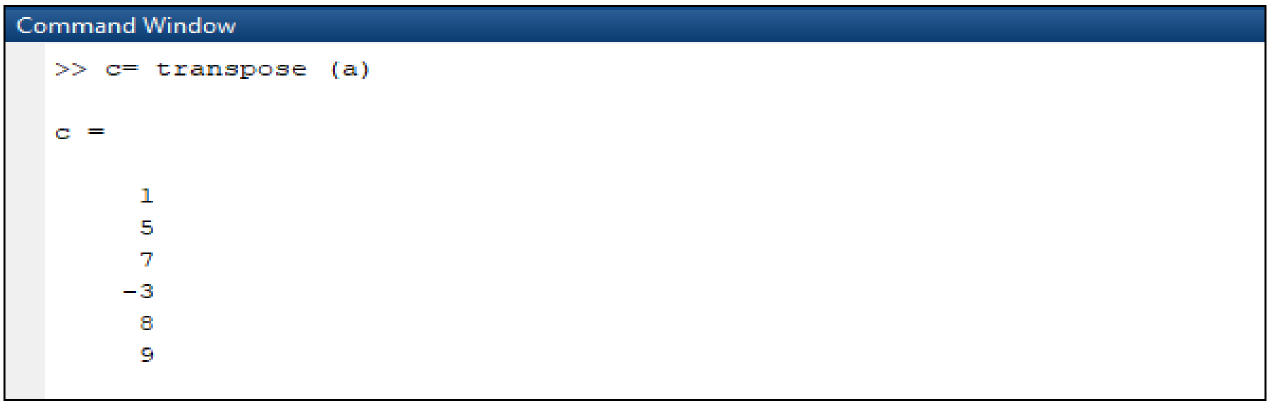
To calculate the transpose of a row vector, two methods exist, either by writing the vector as usual between square brackets **[ ]** and to add the apostrophe **[ ]'**;



Command Window

```
>> c=[1,5,7,-3,8,9]'

c =

     1
     5
     7
    -3
     8
     9
```

or by simply writing **transpose []** as shown in the following example:

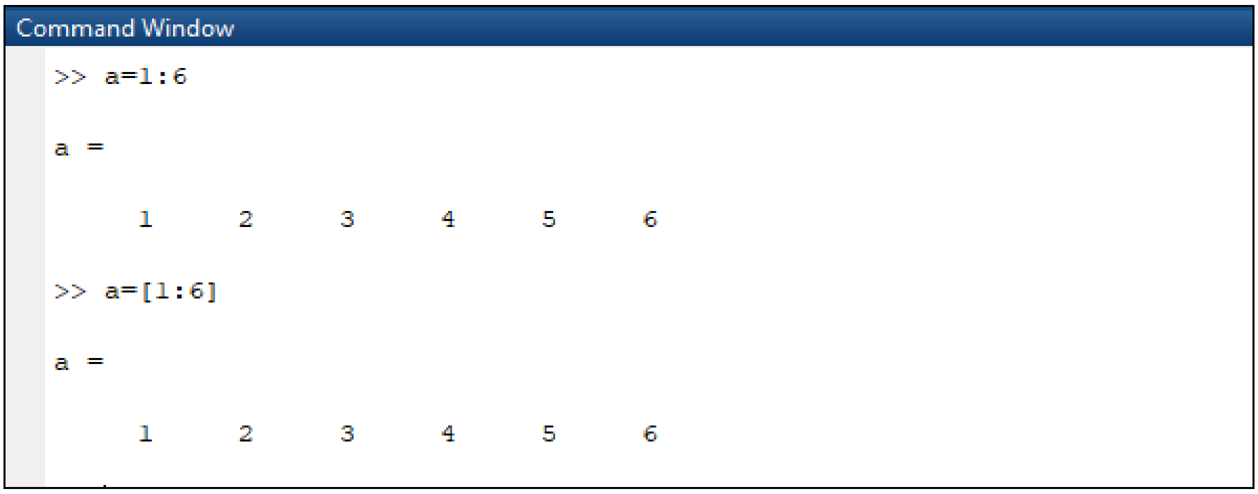


Command Window

```
>> c= transpose (a)

c =

     1
     5
     7
    -3
     8
     9
```

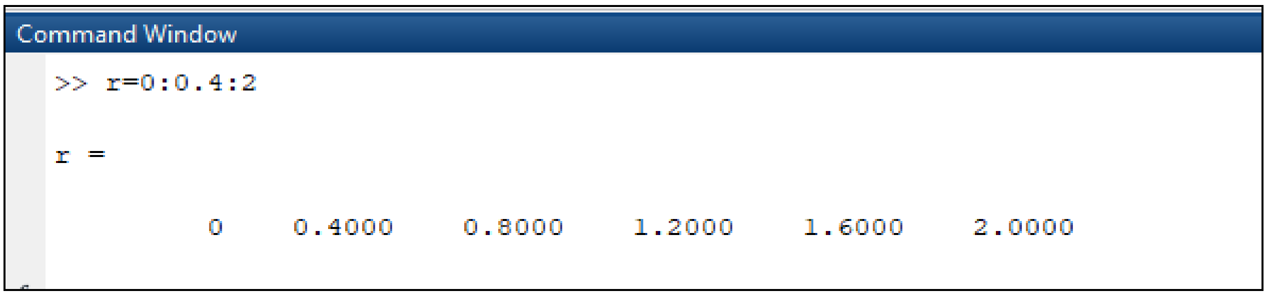
2.1.3 Matltb's syntax when creating row vectors

Matlab makes the creation of row vectors a very easy tasks, i.e., if I desire to write a row vector that runs from **'a'** to **'b'** in steps of '**one = 1'**, the code will be in this manner:



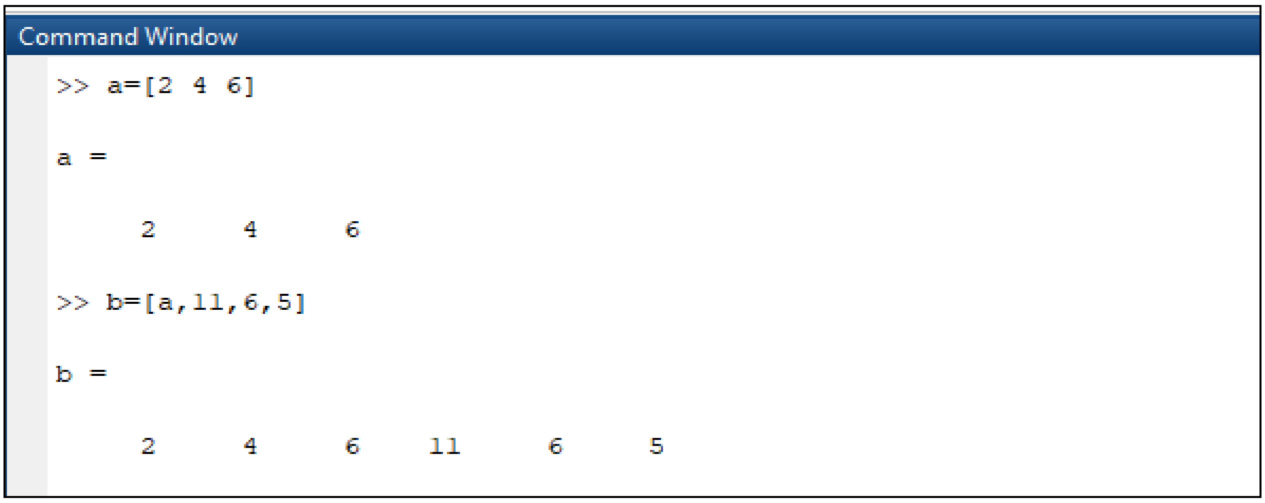
```
>> a=1:6

a =

     1     2     3     4     5     6

>> a=[1:6]

a =

     1     2     3     4     5     6
```

The step can be changed by using the slightly more involved syntax: **v = a:s:b**, which creates the vector r running from 0 to 2 in steps of 0.4, as seen accordingly:



```
>> r=0:0.4:2

r =

         0    0.4000    0.8000    1.2000    1.6000    2.0000
```

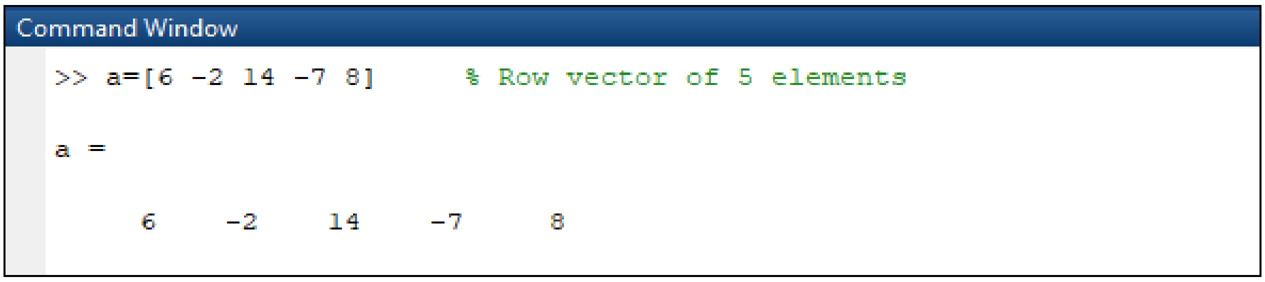
In Matlab, we can write a vector from a previous written vector, as shown in the example:



```
>> a=[2 4 6]

a =

     2     4     6

>> b=[a,11,6,5]

b =

     2     4     6    11     6     5
```

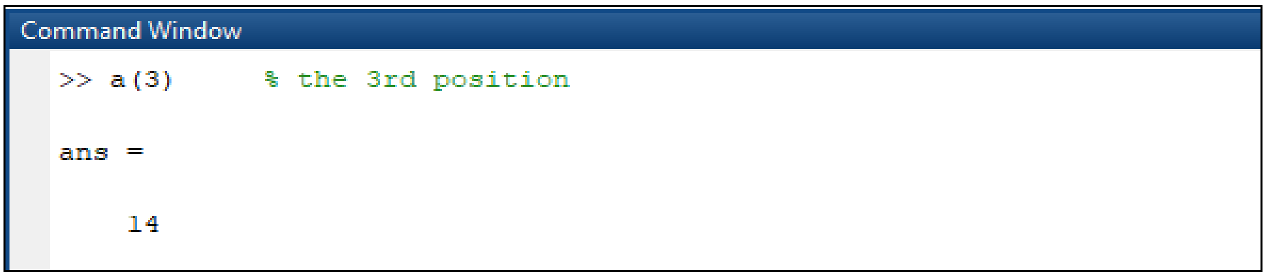
2.1.4 *Access to any element from a given vector*

Matlab allows a rapid and easy access to any elements of a vector. This example shows how to access selected elements of a vector. We have a vector “a” composed from 5 elements like so:



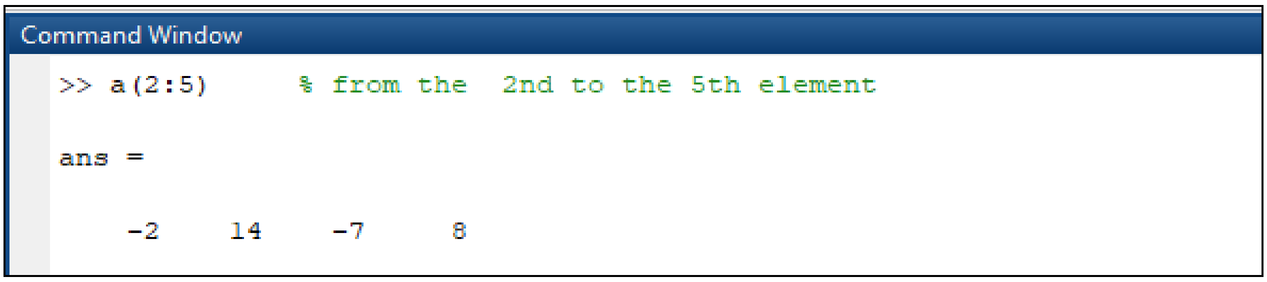
```
>> a=[6 -2 14 -7 8]      % Row vector of 5 elements

a =

     6    -2    14    -7     8
```

We seek for the 3rd element of the vector, we write:



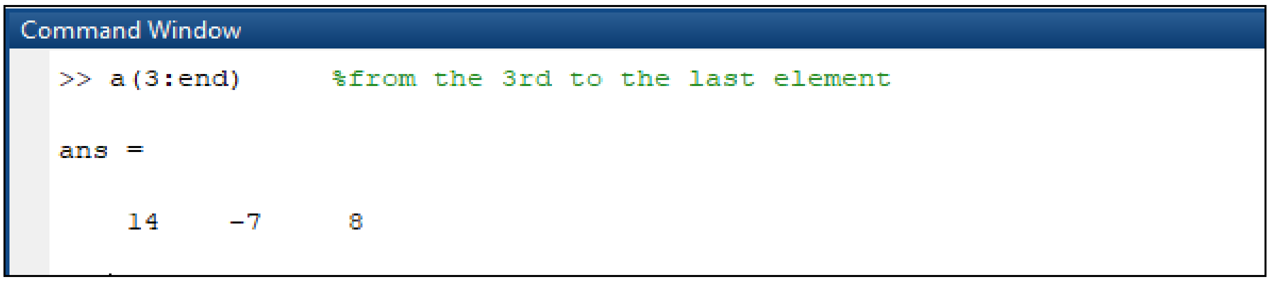
```
>> a(3)      % the 3rd position

ans =

    14
```

If we desire to seek elements from the second to the fifth position, it will be as follows:



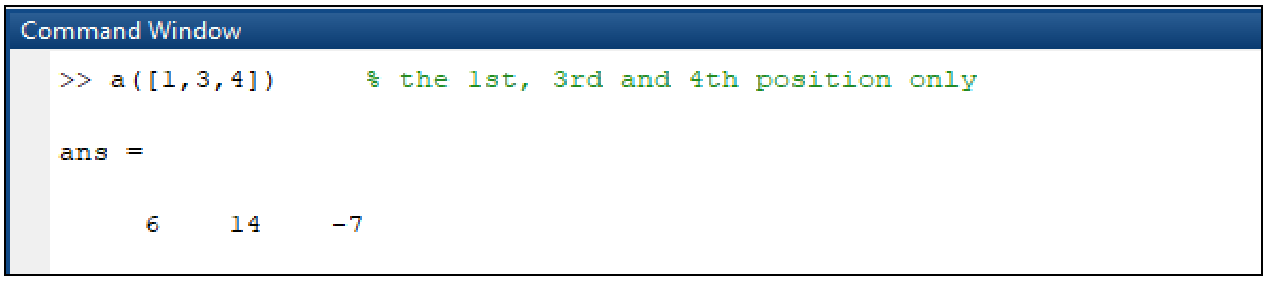
```
>> a(2:5)      % from the  2nd to the 5th element

ans =

    -2    14    -7     8
```

To display elements from the 3rd to the end position, we write accordingly:



```
>> a(3:end)      %from the 3rd to the last element

ans =

    14    -7     8
```

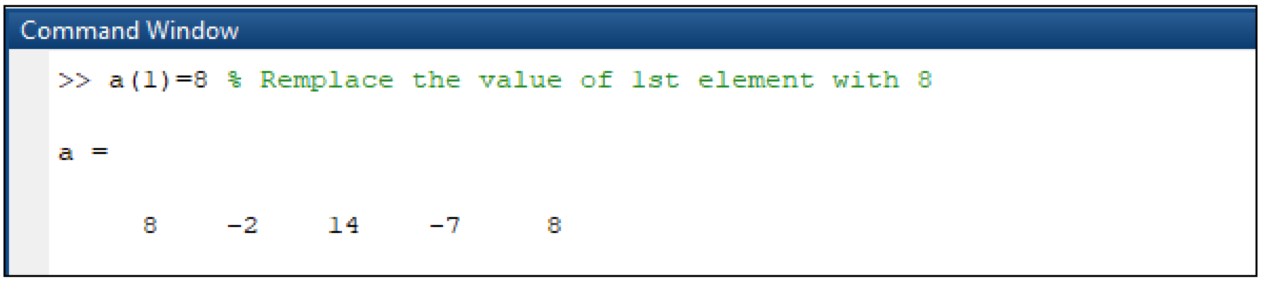
To illustrate the 1st, 3rd and 4th position only, we write the following syntax:



Command Window

```
>> a([1,3,4])      % the 1st, 3rd and 4th position only

ans =

     6    14    -7
```

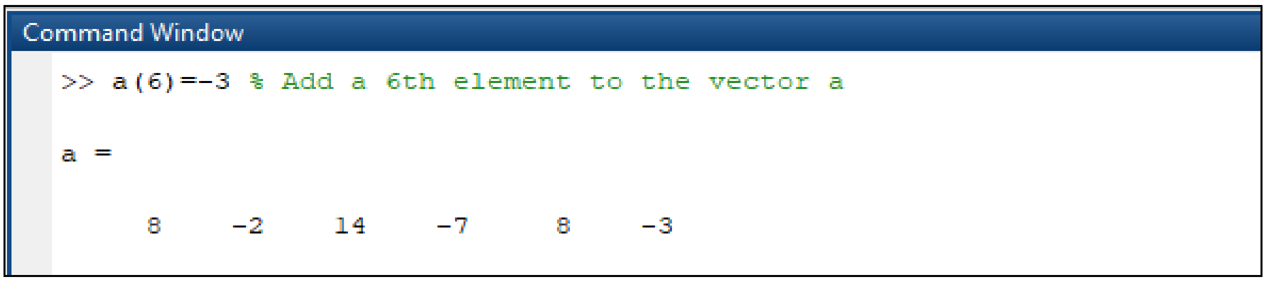
Matlab offers a posibility to replace an element from an already written vector. For instance, if we would like to replace the 1st element with a value of 8, we write in such way:



Command Window

```
>> a(1)=8 % Remplace the value of 1st element with 8

a =

     8    -2    14    -7     8
```

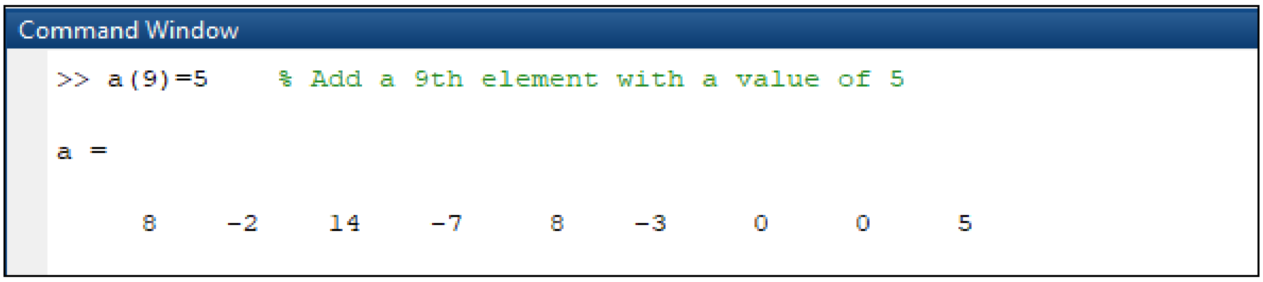
To add a sixth element with value -3



Command Window

```
>> a(6)=-3 % Add a 6th element to the vector a

a =

     8    -2    14    -7     8    -3
```

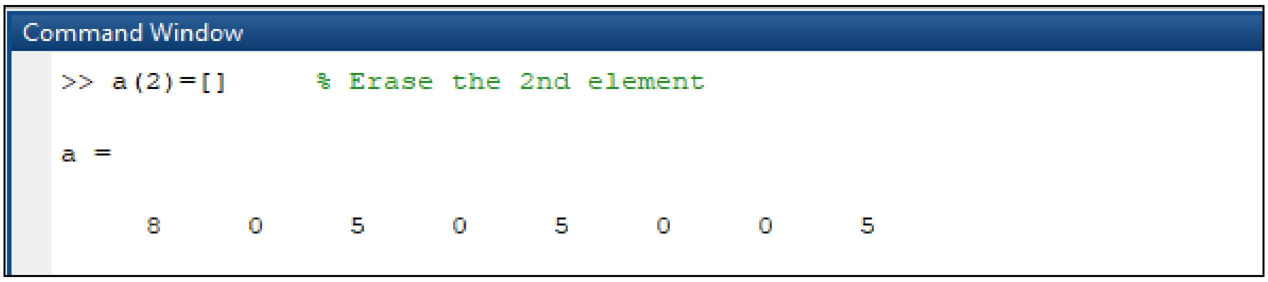
To add a ninth element with value 5, where the 7th and 8th element are equal to zero, we write accordingly:



Command Window

```
>> a(9)=5    % Add a 9th element with a value of 5

a =

     8    -2    14    -7     8    -3     0     0     5
```

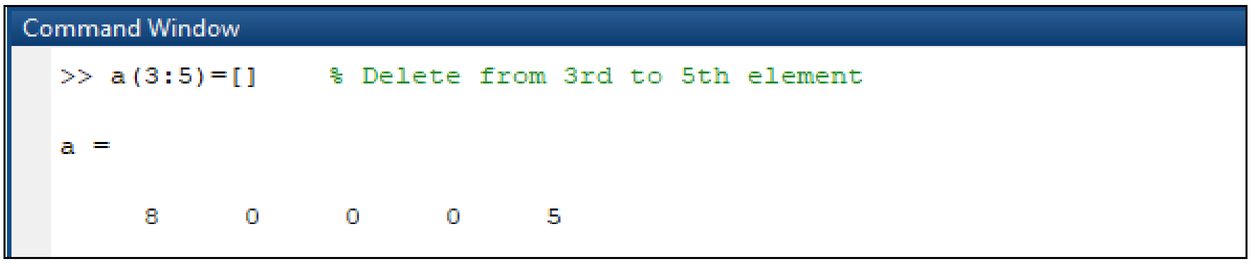
To erase the second element from the vector “a”, we can rapidly write as shown:



Command Window

```
>> a(2)=[]      % Erase the 2nd element

a =

     8     0     5     0     5     0     0     5
```

To delete Elements from 3rd to 5th element, we write the following syntax:



Command Window

```
>> a(3:5)=[]     % Delete from 3rd to 5th element

a =

     8     0     0     0     5
```

2.1.5 *Element-by-element operations for vectors*

The table below illustrates how to perform element-by-element calculations In Matlab.

Table2.1 *Basics operations for vectors*

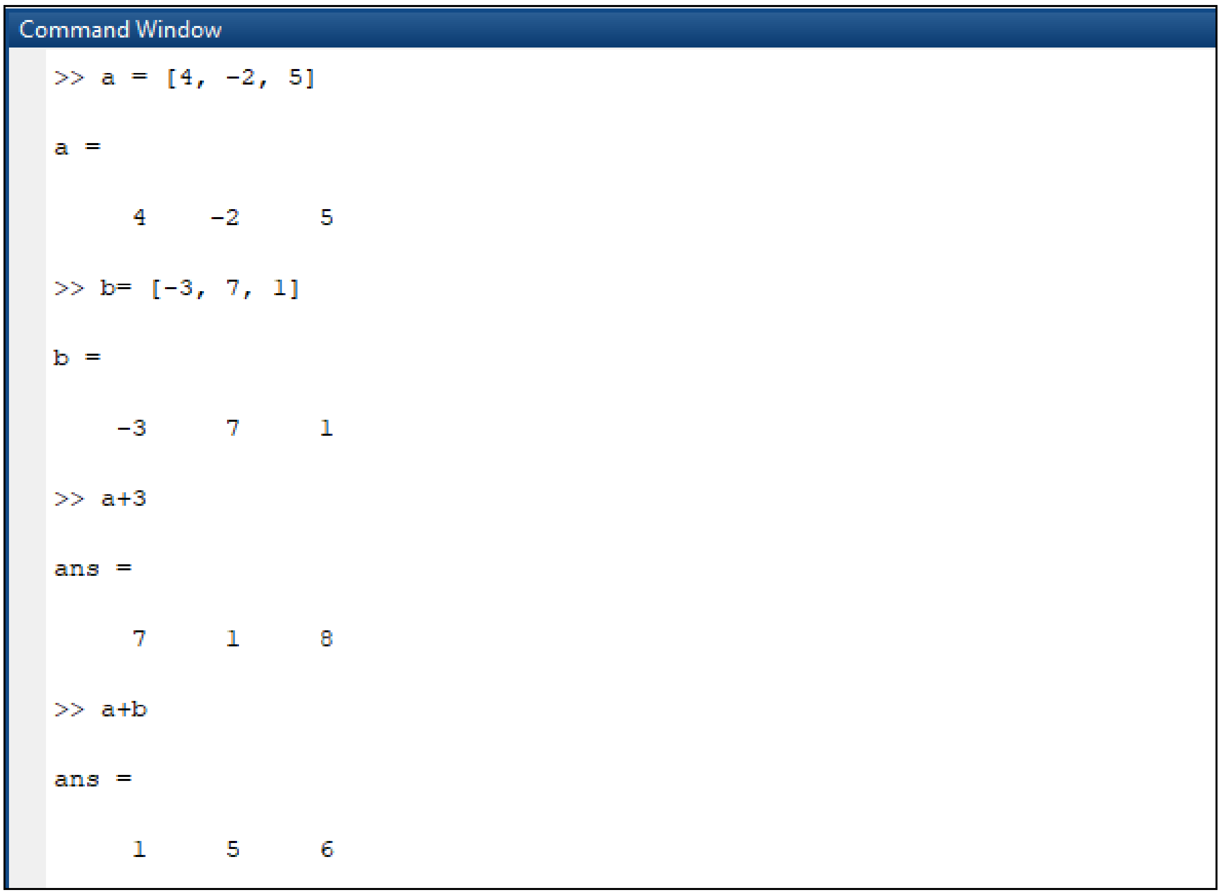
Operation	**Signification**
+	Addition of vectors
-	Subtraction of vectors
.*	Element-by-element multiplication
./	Division element by element
.^	Element-by-element power

For instance, we have two vectors a vector a = [4, -2, 5] and b= [-3, 7, 1]. Try to do the following operation:

a+3; a+b; b-2; b-a; a*3; a.*3; a.*b; a/3; a./3; a./b; a.^3 ; a.^b

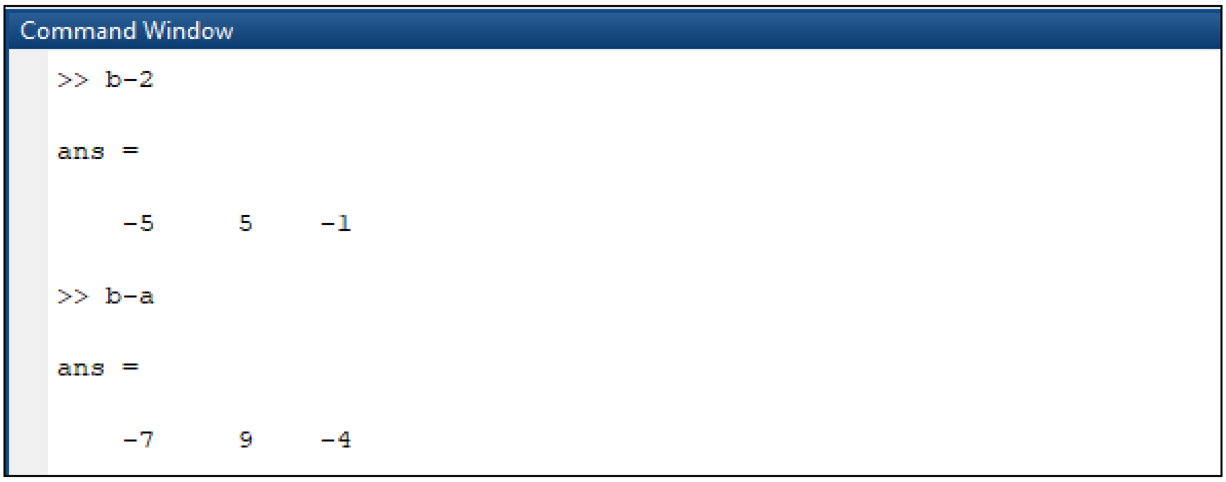
The answers will be in this manner:

- The addition:



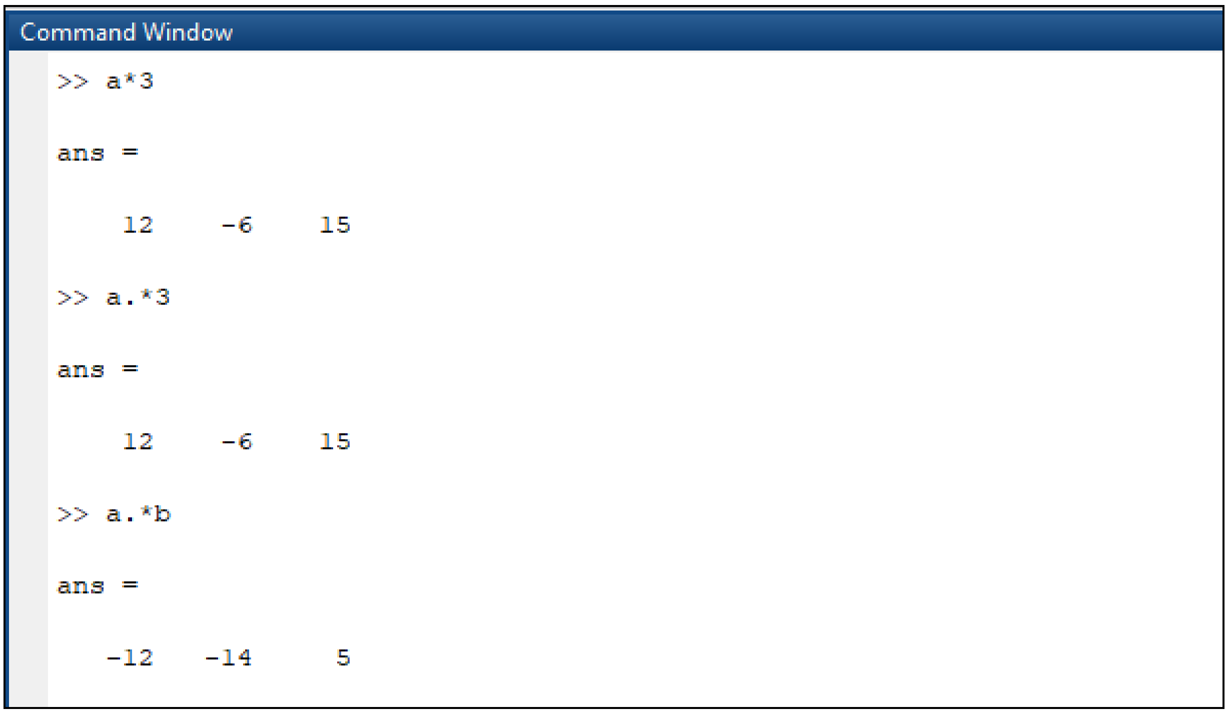
```
>> a = [4, -2, 5]

a =

     4    -2     5

>> b= [-3, 7, 1]

b =

    -3     7     1

>> a+3

ans =

     7     1     8

>> a+b

ans =

     1     5     6
```

- The minus:



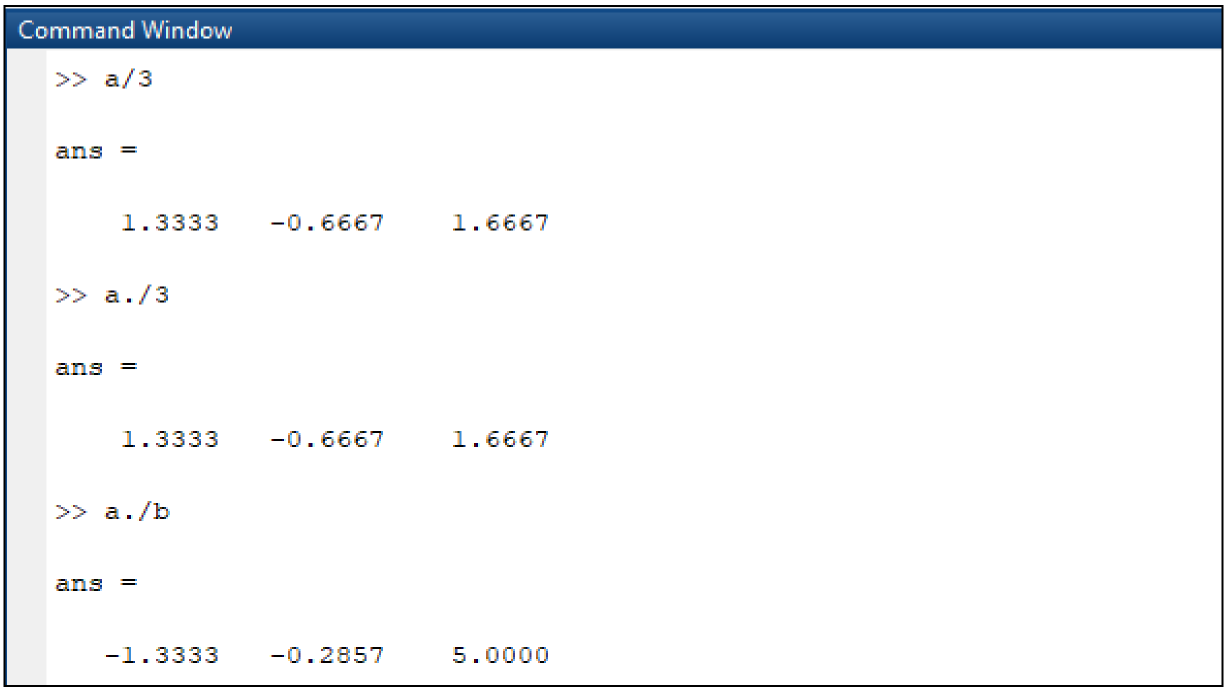
```
>> b-2

ans =

    -5     5    -1

>> b-a

ans =

    -7     9    -4
```

- The multiplication:



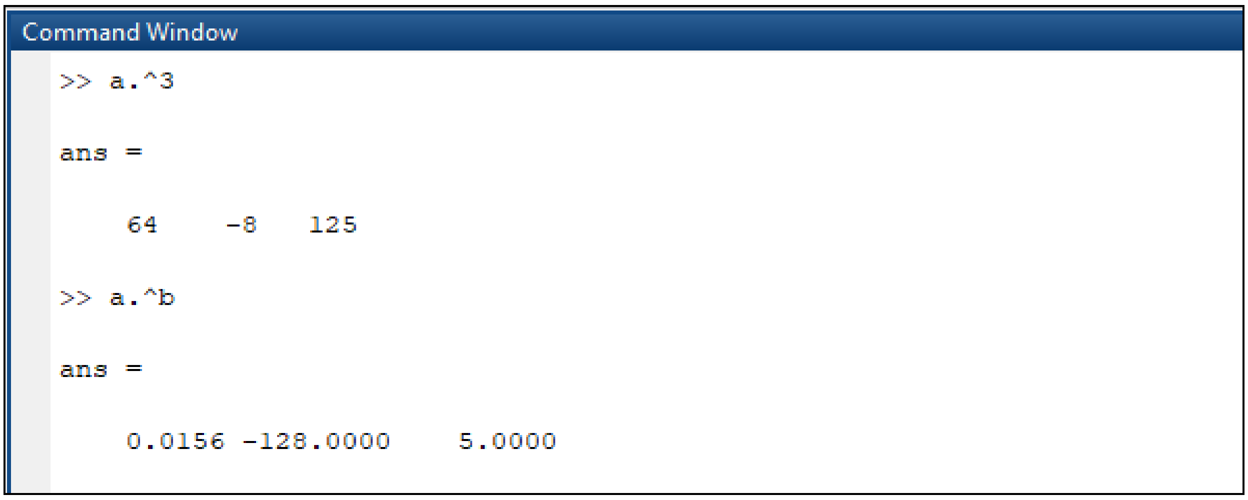
```
>> a*3

ans =

    12    -6    15

>> a.*3

ans =

    12    -6    15

>> a.*b

ans =

   -12   -14     5
```

- The division:



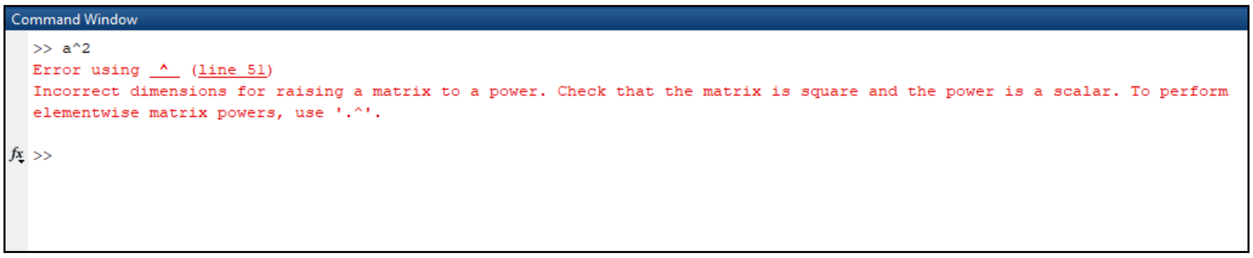
```
>> a/3

ans =

    1.3333   -0.6667    1.6667

>> a./3

ans =

    1.3333   -0.6667    1.6667

>> a./b

ans =

   -1.3333   -0.2857    5.0000
```

- Element-by-element power:



```
Command Window
>> a.^3

ans =

    64    -8   125

>> a.^b

ans =

    0.0156 -128.0000    5.0000
```

Remarque
Writing an expression such as: a^2 generates an error message.



```
Command Window
>> a^2
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power is a scalar. To perform
elementwise matrix powers, use '.^'.

>>
```

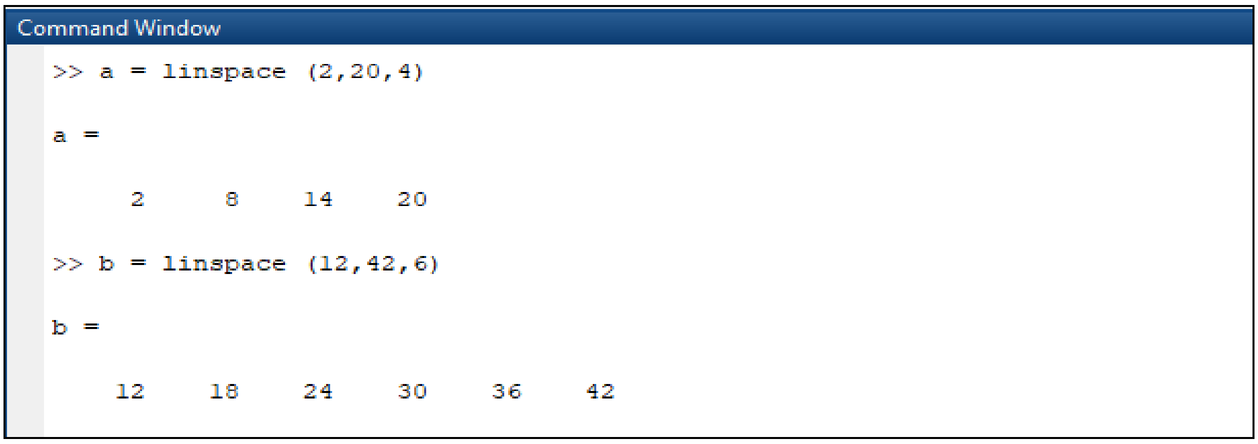
The reason is that expression refers to a multiplication of matrices (a*a must be rewritten a*a' or a'*a to be valid).

2.1.6 *The linspace function*

The linspace function can be used to create a vector with elements that are arranged in a predetermined order and a known number. The function can be written as follows:

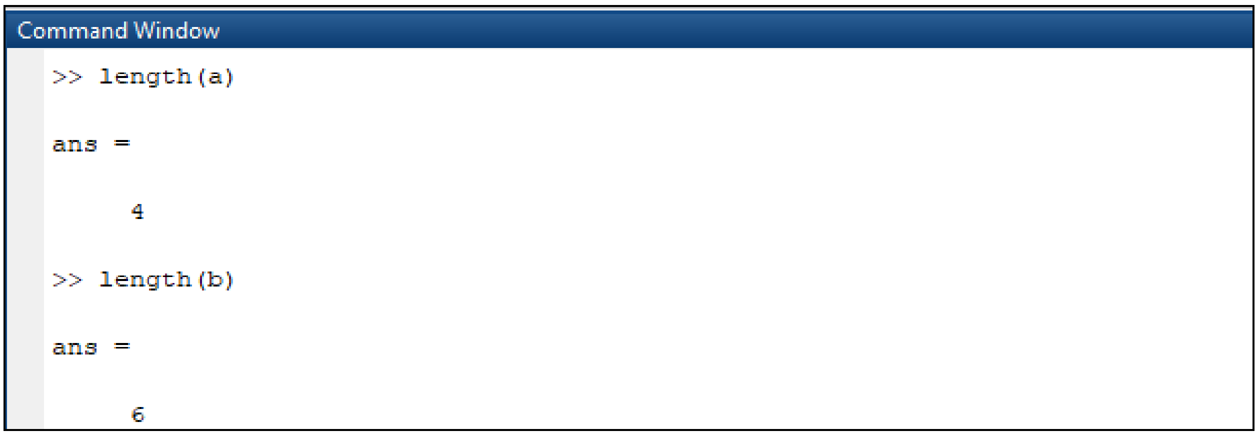
“ linspace (start, end, number of elements)”.

Replicate the following examples to understand accuratly.



```
Command Window
>> a = linspace (2,20,4)

a =

     2     8    14    20

>> b = linspace (12,42,6)

b =

    12    18    24    30    36    42
```

The length of a vector (the number of its components) can be obtained by suing the ‘**length’** function thusly:


Command Window

```
>> length(a)

ans =

     4

>> length(b)

ans =

     6
```

2.2 *Matrices*

2.2.1 *Matrices creation*

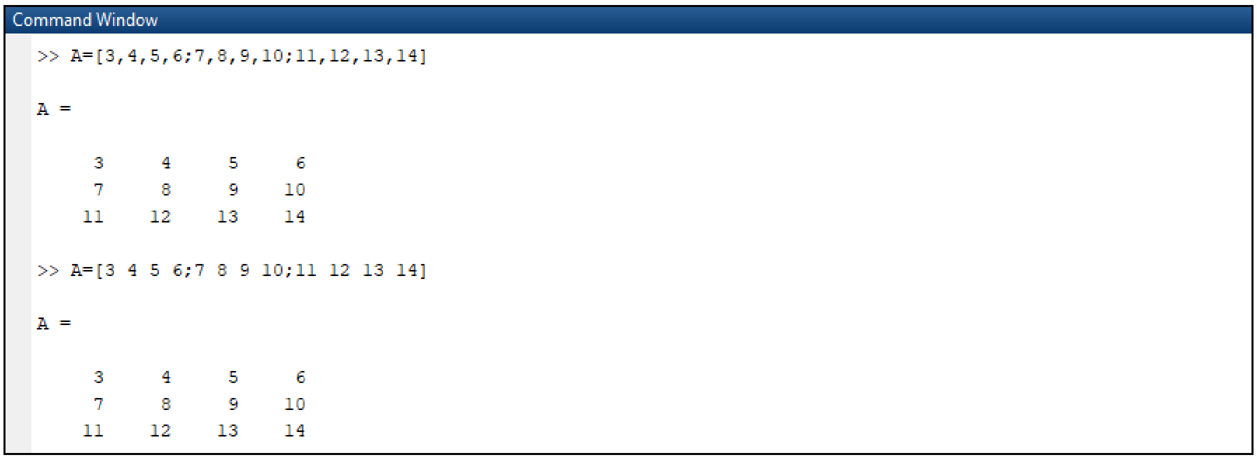
A matrix is a rectangular array of (two-dimensional) elements. Vectors are matrices with a single row or column (one-dimensional). To insert a matrix, you must respect the following rules:

- Elements must be enclosed in square brackets **[ ]**;
- Spaces or commas are used to separate elements in the same line;
- A semicolon (or the enter key) is used to separate lines.

To illustrate this, considering the following matrix:

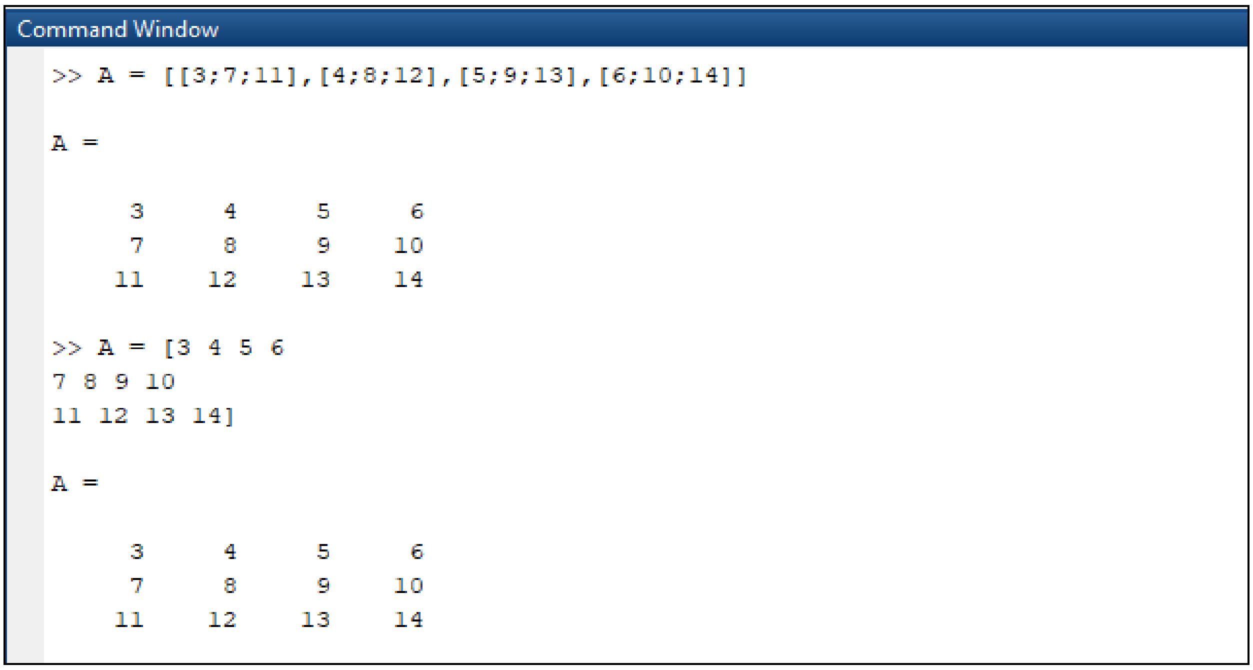
$$A = \begin{bmatrix} 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 \end{bmatrix}$$

To write this matrix different syntax ways exists, such as:


Command Window

```
>> A=[3,4,5,6;7,8,9,10;11,12,13,14]

A =

     3     4     5     6
     7     8     9    10
    11    12    13    14

>> A=[3 4 5 6;7 8 9 10;11 12 13 14]

A =

     3     4     5     6
     7     8     9    10
    11    12    13    14
```

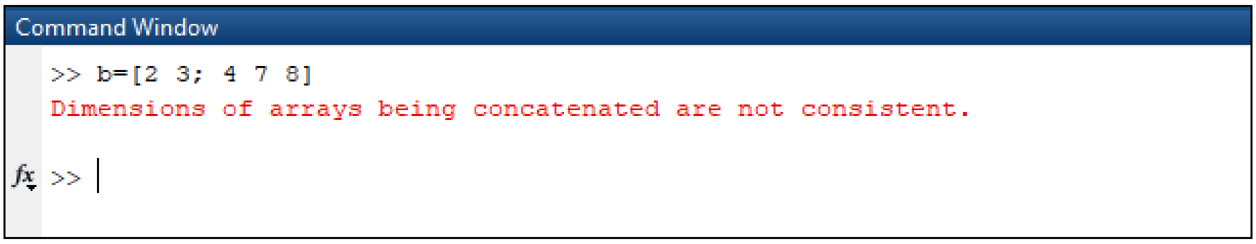
Or, we can write like so:



```
>> A = [[3;7;11],[4;8;12],[5;9;13],[6;10;14]]

A =

     3     4     5     6
     7     8     9    10
    11    12    13    14

>> A = [3 4 5 6
7 8 9 10
11 12 13 14]

A =

     3     4     5     6
     7     8     9    10
    11    12    13    14
```

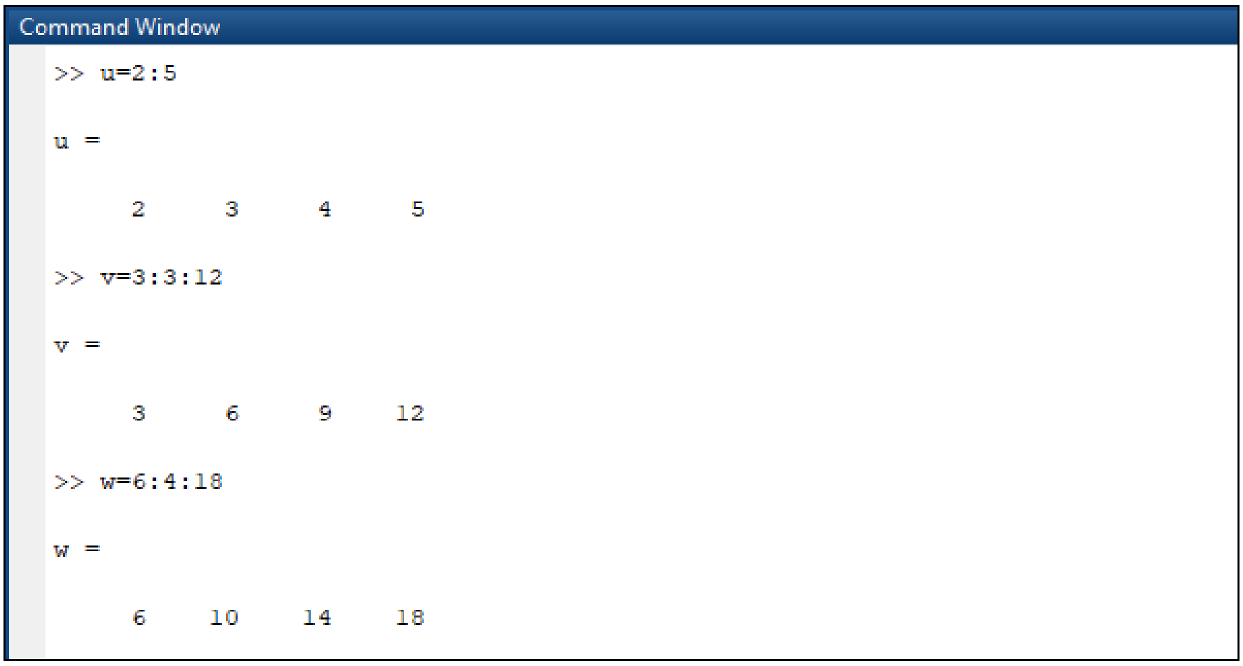
Remark

The number of elements in each row (number of columns) must be the same in all rows of the matrix, otherwise an error will be reported by Matlab, as seen in the following example:



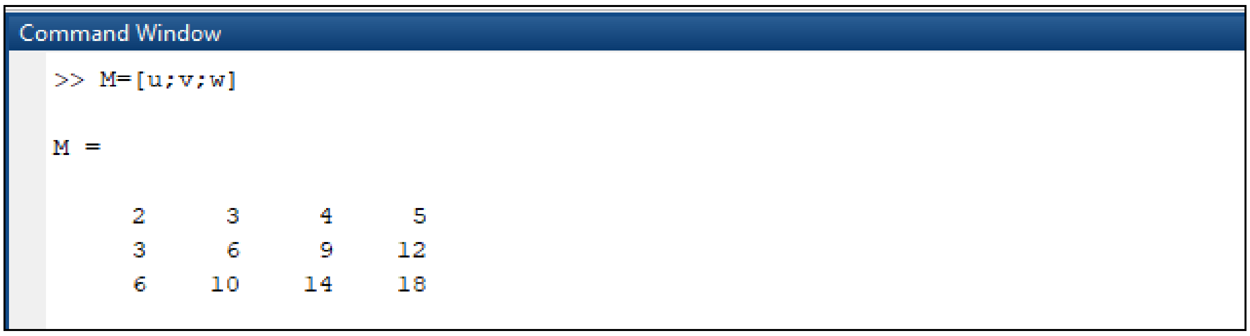
```
>> b=[2 3; 4 7 8]
Dimensions of arrays being concatenated are not consistent.

>> 
```

A matrix **"M"** can be generated from a set of vectors (u,v,w) as shown in the following example:



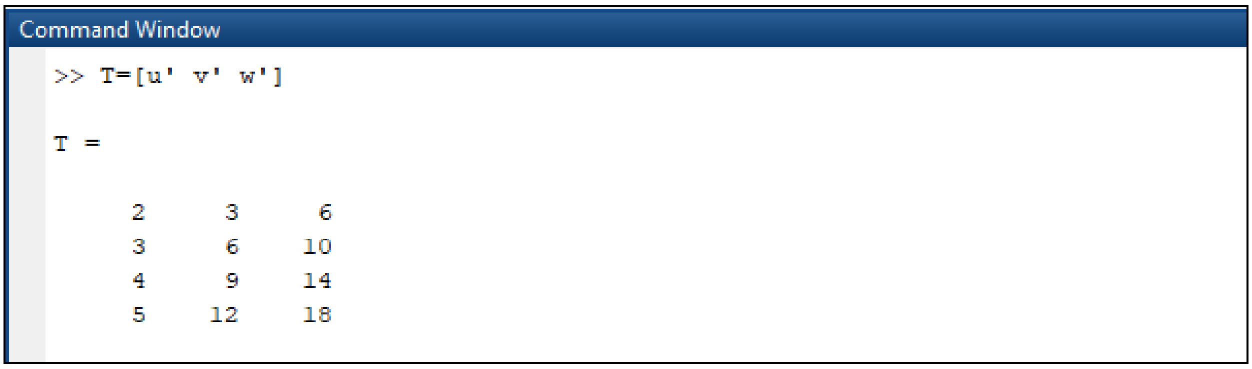
```
>> u=2:5

u =

     2     3     4     5

>> v=3:3:12

v =

     3     6     9    12

>> w=6:4:18

w =

     6    10    14    18
```

The syntax of the matrix ‘**M’** will be as follows:


Command Window

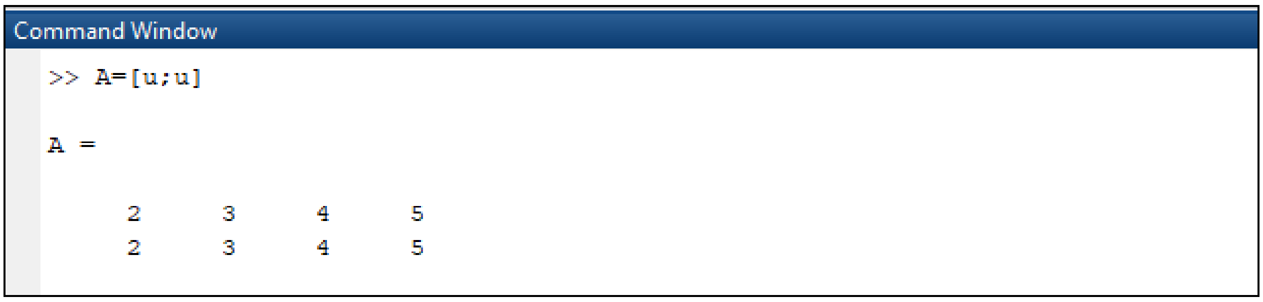
```
>> M=[u;v;w]

M =

     2     3     4     5
     3     6     9    12
     6    10    14    18
```

The transpose of the matrix M can be written from the given vectors (u,v,w) as seen:


Command Window

```
>> T=[u' v' w']

T =

     2     3     6
     3     6    10
     4     9    14
     5    12    18
```

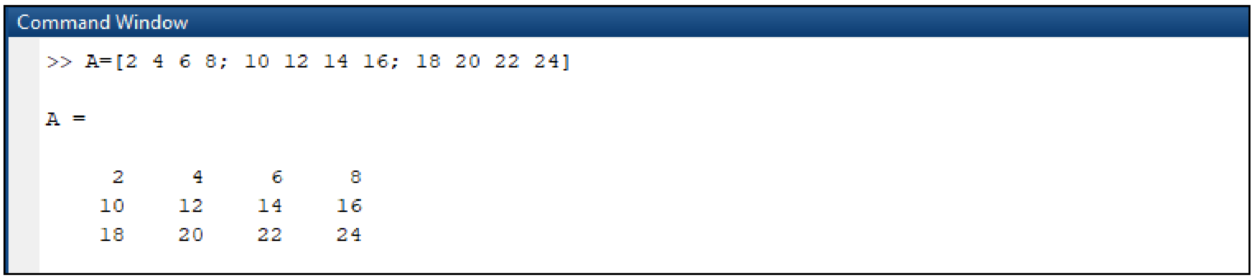
Another point, is the possibility to create a matrix C from the previously ‘u’ vector. Thus, the syntax will be in this manner:


Command Window

```
>> A=[u;u]

A =

     2     3     4     5
     2     3     4     5
```

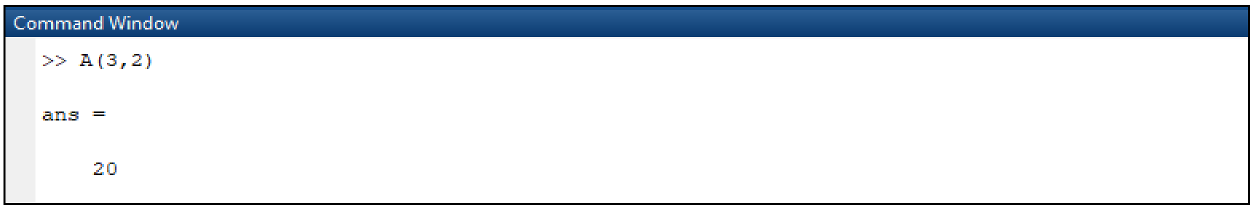
2.2.2 Referencing and access to matrix elements

Matlab allows you to acces to any element from the matrix. For example, a matrix “A” composed from 3 rows and 4 columns as seen:


Command Window

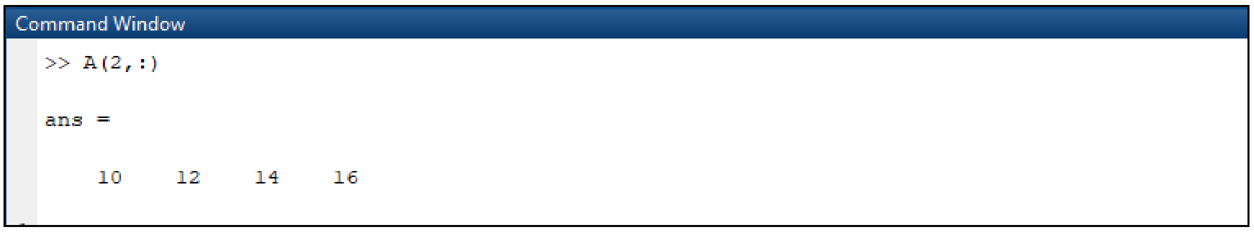
```
>> A=[2 4 6 8; 10 12 14 16; 18 20 22 24]

A =

     2     4     6     8
    10    12    14    16
    18    20    22    24
```

For instance, to acces to the element situated in the 3rd row and 2nd column we write the following syntax:


Command Window

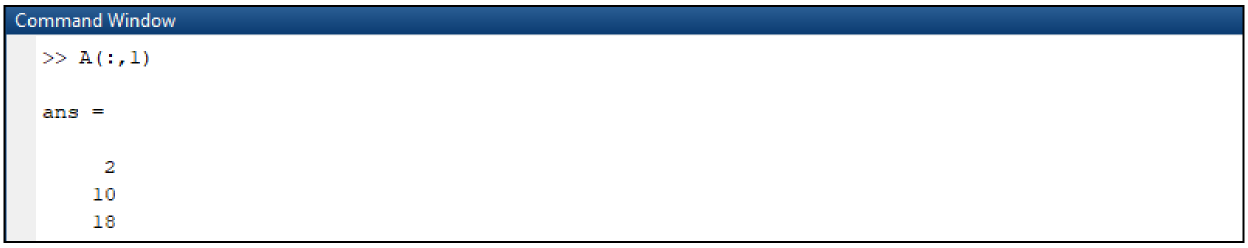
```
>> A(3,2)

ans =

    20
```

All elements of the second row:


Command Window

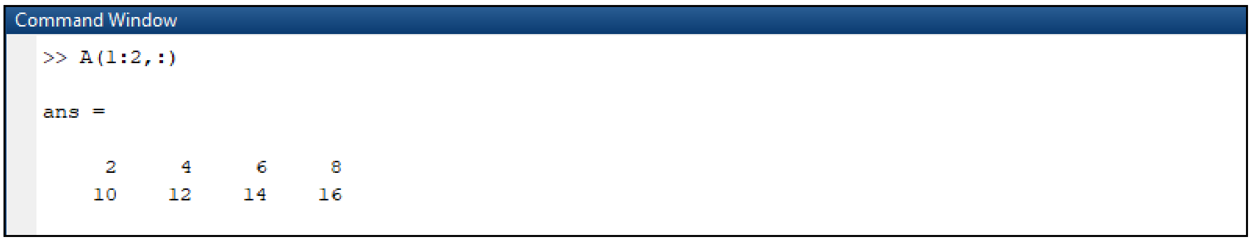
```
>> A(2,:)

ans =

    10    12    14    16
```

All elements of the first column:


Command Window

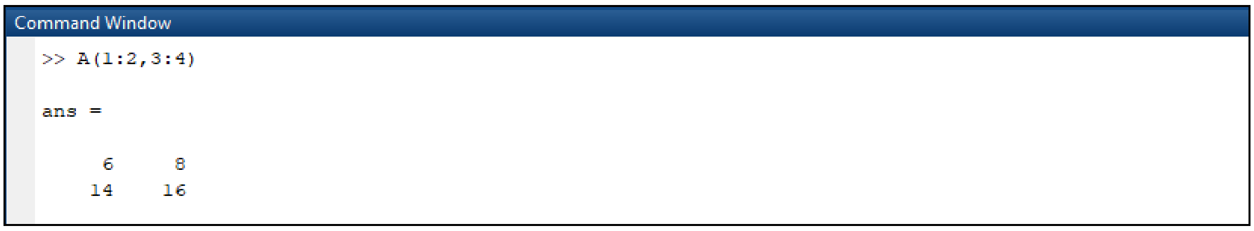
```
>> A(:,1)

ans =

     2
    10
    18
```

All elements of the 1st and 2nd line:


Command Window

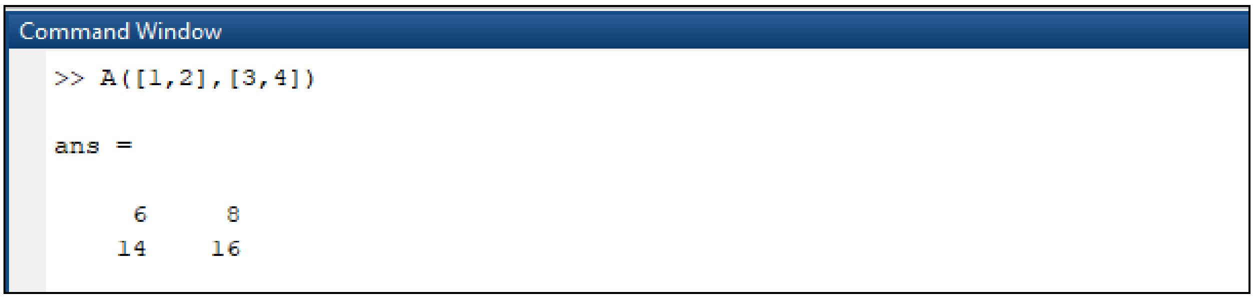
```
>> A(1:2,:)

ans =

     2     4     6     8
    10    12    14    16
```

The superior right submatrix, with a size of 2*2:


Command Window

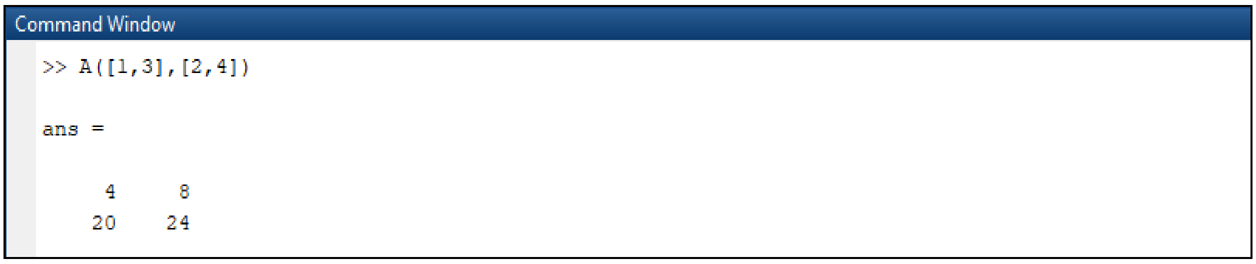
```
>> A(1:2,3:4)

ans =

     6     8
    14    16
```

The submatrix, rows (1,2) and (3,4) columns:



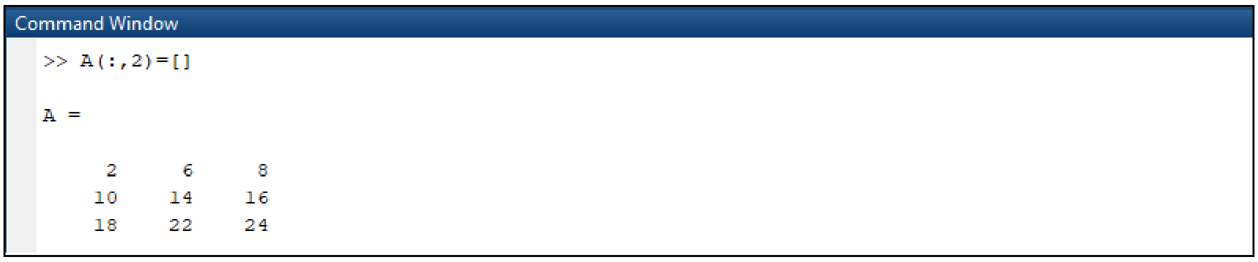
```
>> A([1,2],[3,4])

ans =

     6     8
    14    16
```

The submatrix, rows (1,3) and columns (2,4):



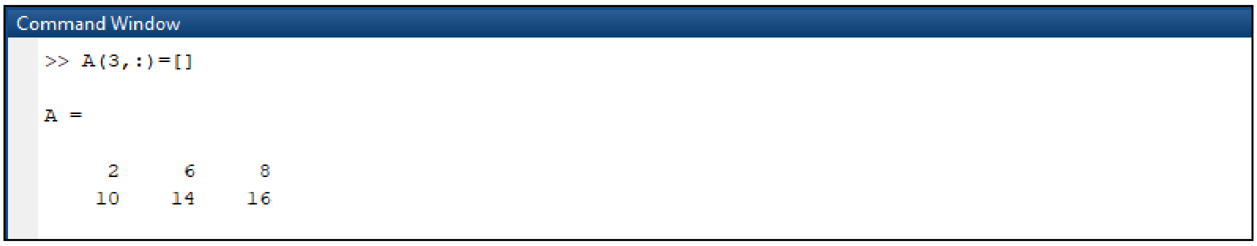
```
>> A([1,3],[2,4])

ans =

     4     8
    20    24
```

Delete 2nd column:



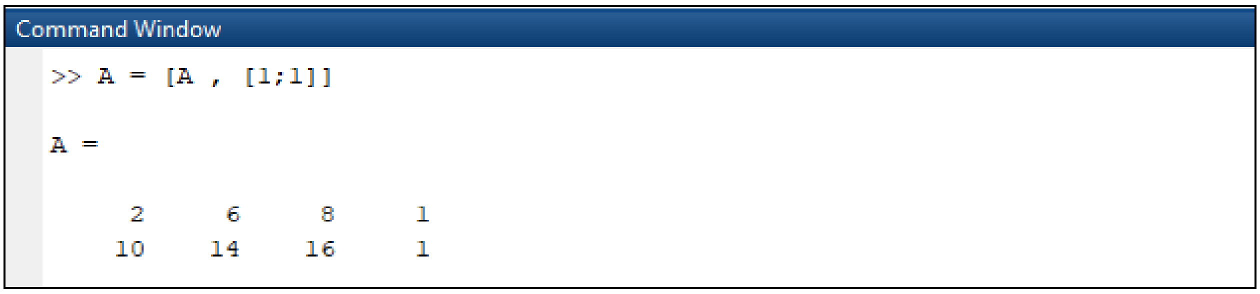
```
>> A(:,2)=[]

A =

     2     6     8
    10    14    16
    18    22    24
```

Delete 3rd row:



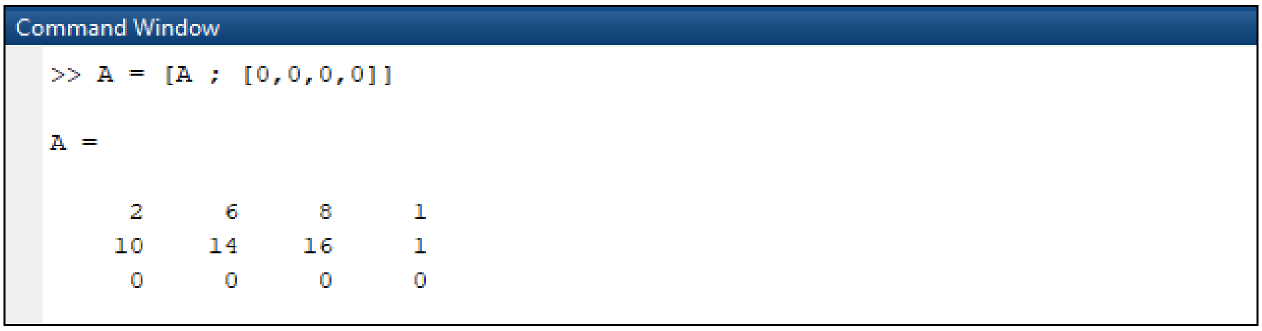
```
>> A(3,:)=[]

A =

     2     6     8
    10    14    16
```

Add a new column of one:



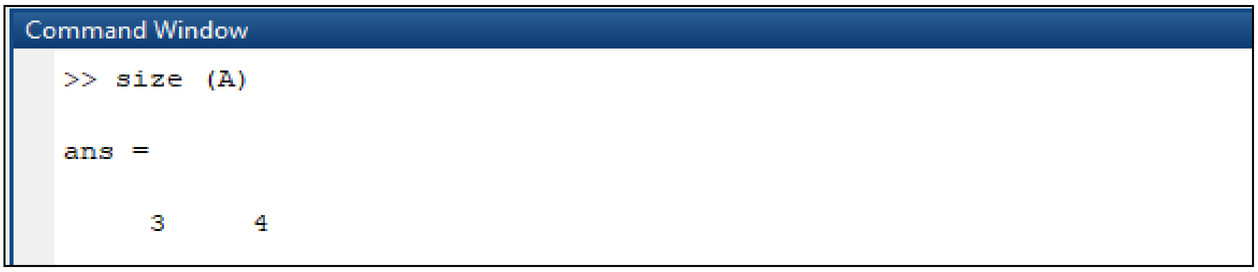
```
>> A = [A , [1;1]]

A =

     2     6     8     1
    10    14    16     1
```

Add a new row of zero:


```
Command Window
>> A = [A ; [0,0,0,0]]

A =

     2     6     8     1
    10    14    16     1
     0     0     0     0
```

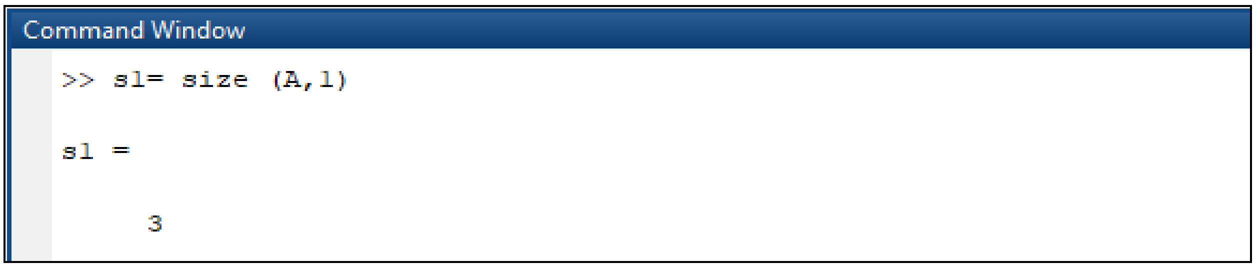
2.2.3 *Size of a matrix*

The size of a matrix can be determined by using the function "**size**". For instance, the size of the matrix '**M**' will be as follows:

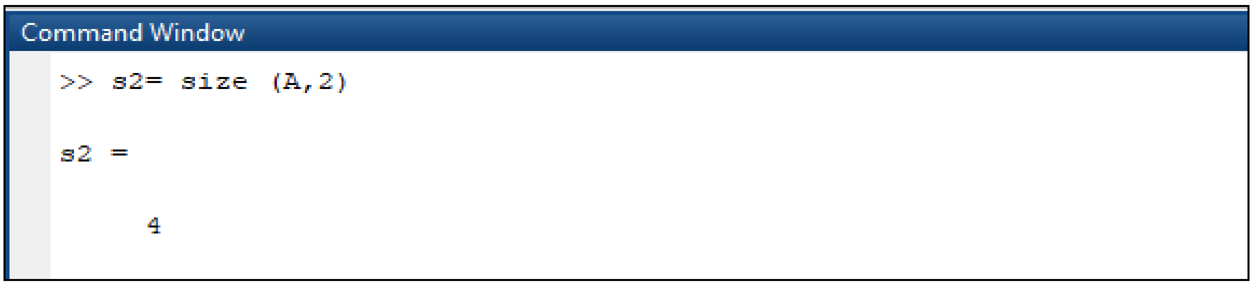

```
Command Window
>> size (A)

ans =

     3     4
```

The Matlab's outcome will be row x column. In aim to determine the number of rows or columns separately, we will use the following syntax:

- The number of rows:


```
Command Window
>> s1= size (A,1)

s1 =

     3
```

- The number of columns:


```
Command Window
>> s2= size (A,2)

s2 =

     4
```

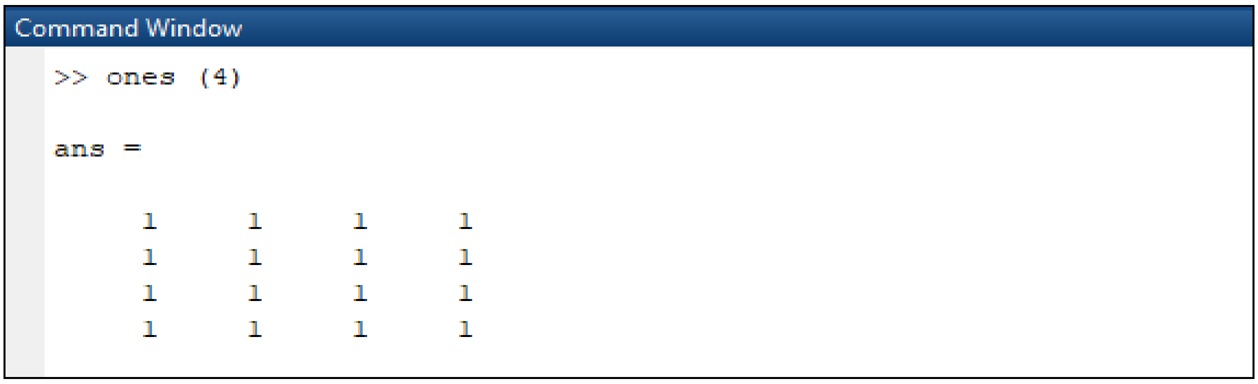
2.2.4 Matlab's predefined matrices functions

In Matlab, a set of functions that allow to users to gain time and to automatically generate a specific matrix. The following table present the most used ones:

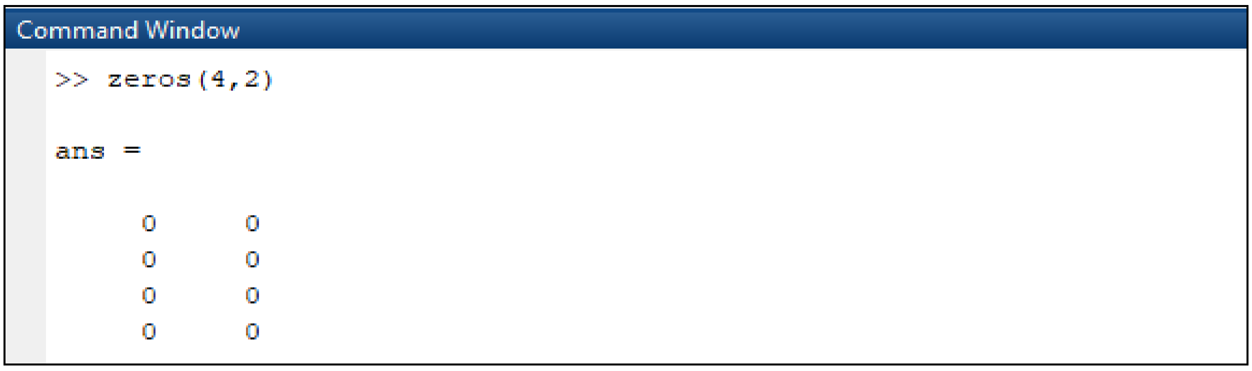
Table2.2 *Matlab predefined matrices functions*

Function	**Signification**
ones(n)	Generates an n × n matrix with all elements = 1
ones(m,n)	Generates an m × n matrix with all elements = 1
zeros(n)	Generates an n × n matrix with all elements = 0
zeros(m,n)	Generates an m × n matrix with all elements = 0
eye(n)	Generates an n × n identity matrix (ones on the main diagonal and zeros elsewhere)
rand(m,n)	Generates an m × n matrix of random values
magic(n)	Generates a magic matrix of dimension n × n

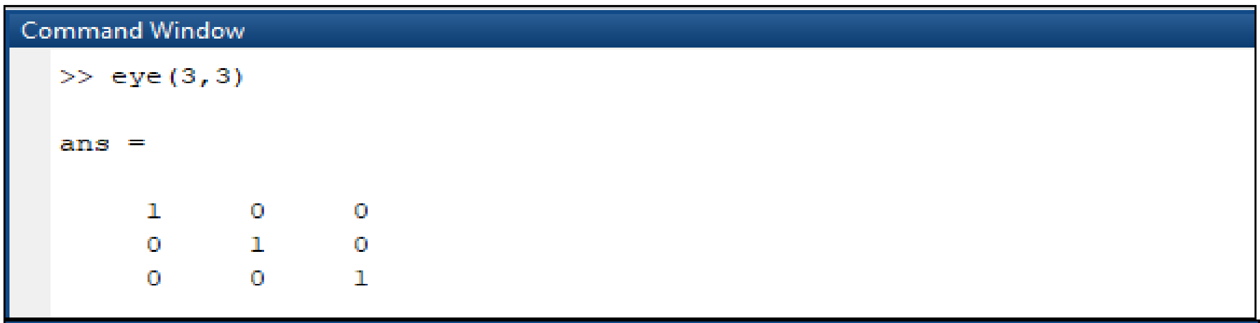
Bellow some examples to clarify the idea.



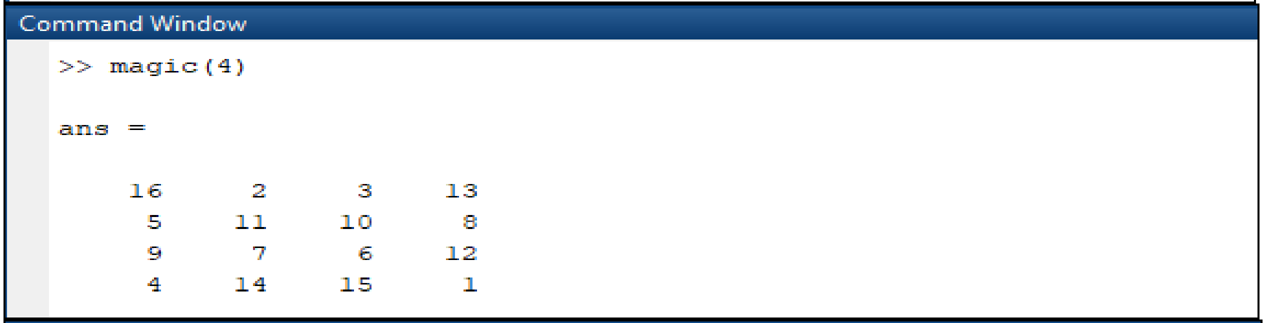
```
>> ones (4)

ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```



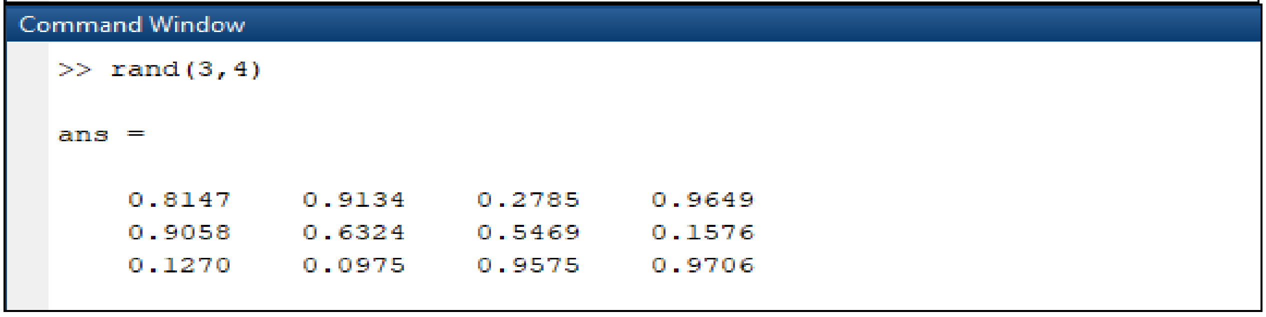
```
>> zeros(4,2)

ans =

     0     0
     0     0
     0     0
     0     0
```



```
>> eye(3,3)

ans =

     1     0     0
     0     1     0
     0     0     1
```



```
>> magic(4)

ans =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```



```
>> rand(3,4)

ans =

    0.8147    0.9134    0.2785    0.9649
    0.9058    0.6324    0.5469    0.1576
    0.1270    0.0975    0.9575    0.9706
```

2.2.5 Element-by-element matrix operations

The element-by-element matrix operations are the same as those for vectors. The only condition is that the two matrices have the same dimensions. The table below illustrates these operations.

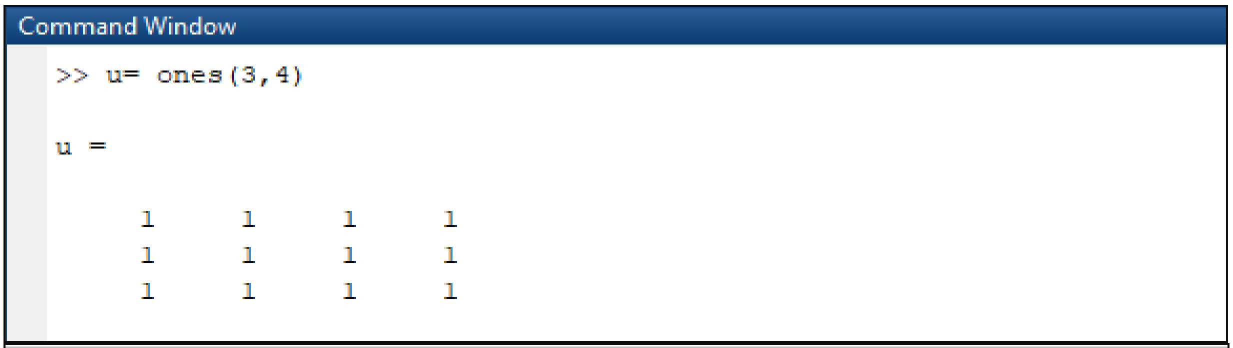
Table 2.3. *Matlab basic matrix operations*

Operation	**Signification**
+	Addition
-	Minus
. *	Multiplication Element-by-element
*	Matrix multiplication
. ^	Power Element-by-element
. /	Division Element-by-element
. \	Inverted division Element-by-element
/	Matrix division

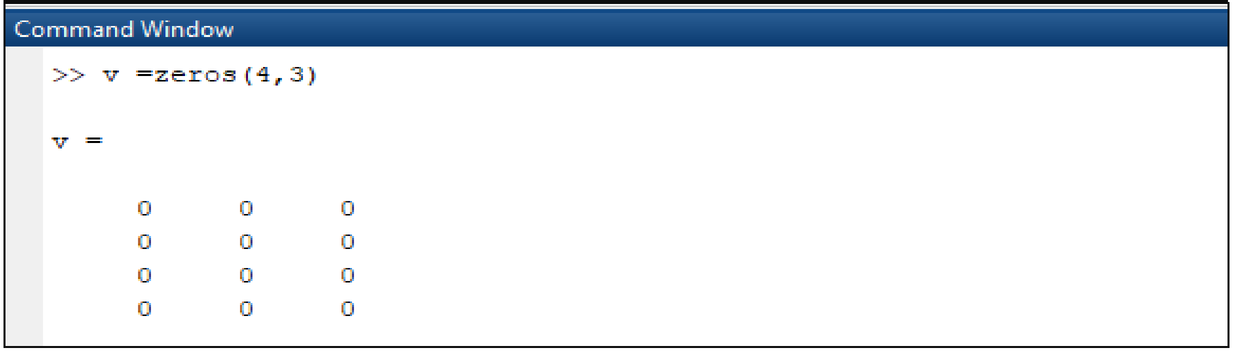
To understand better these operations, try to replicate the following example.

- Firstly, write a matrix ones named "u", composed from 3 rows and 4 columns.
- Secondly, write a matrix zeros "v", composed from 4 rows and 3 columns.

The outcomes in Matlab will be accordinagly:



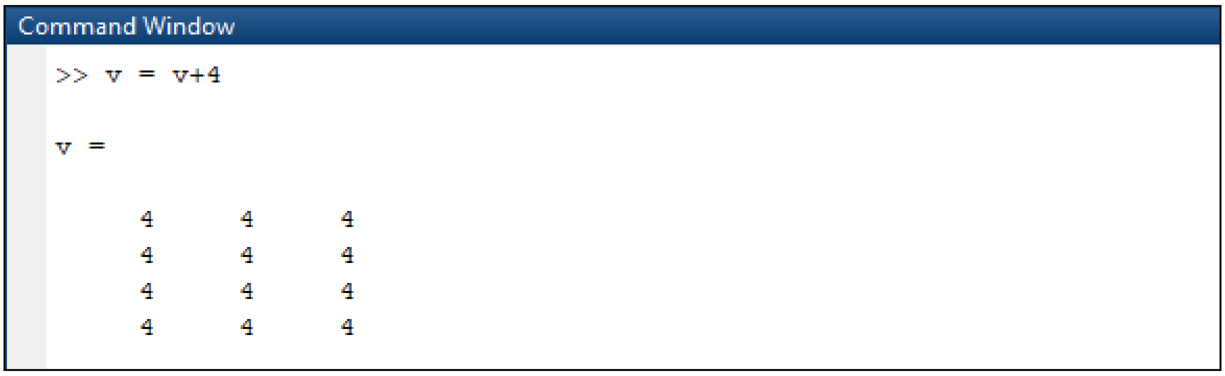
```
>> u= ones(3,4)

u =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```



```
>> v =zeros(4,3)

v =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

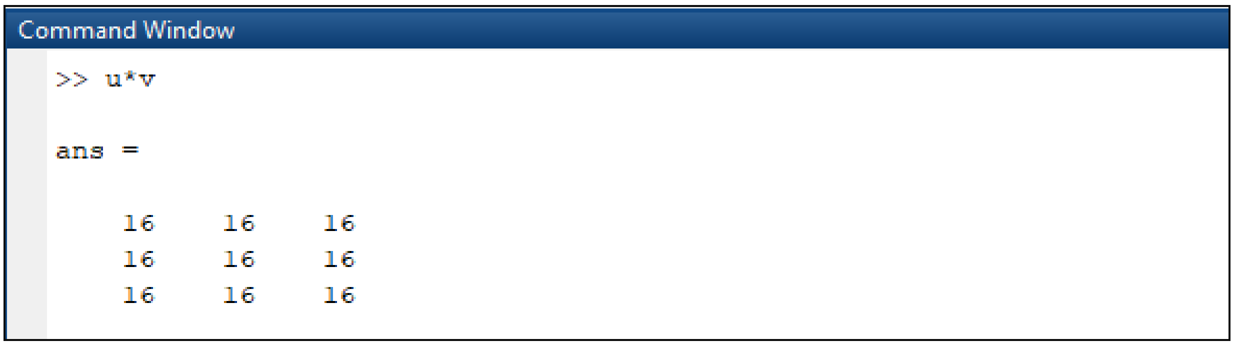
Now, let's do the following operations:

- Add four to the matrix v (v+4)



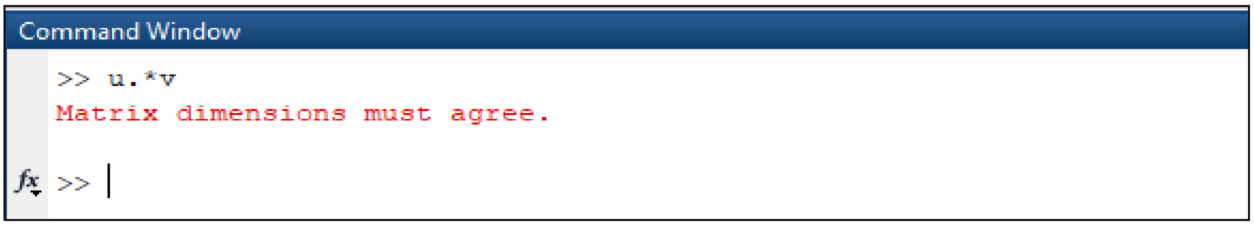
```
>> v = v+4

v =

     4     4     4
     4     4     4
     4     4     4
     4     4     4
```

- Multiply u*v



```
>> u*v

ans =

    16    16    16
    16    16    16
    16    16    16
```

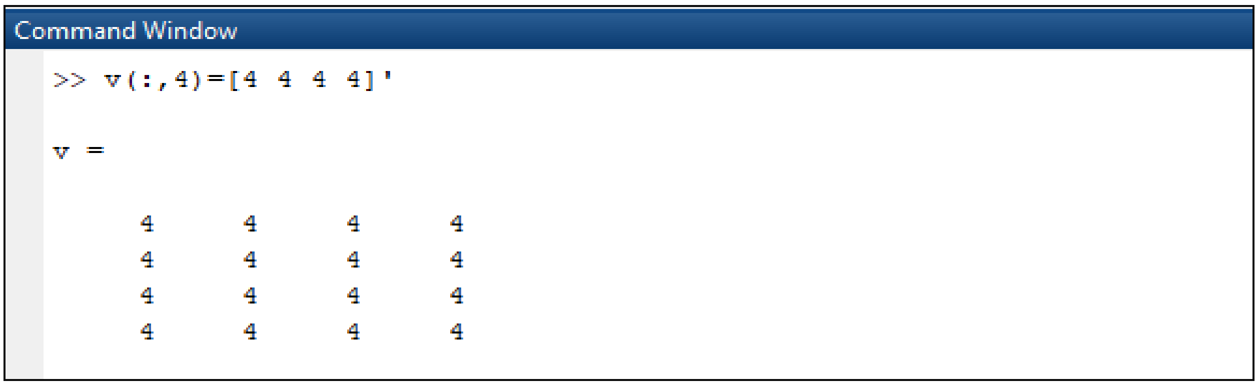
- u.*v



```
>> u.*v
Matrix dimensions must agree.

>> |
```

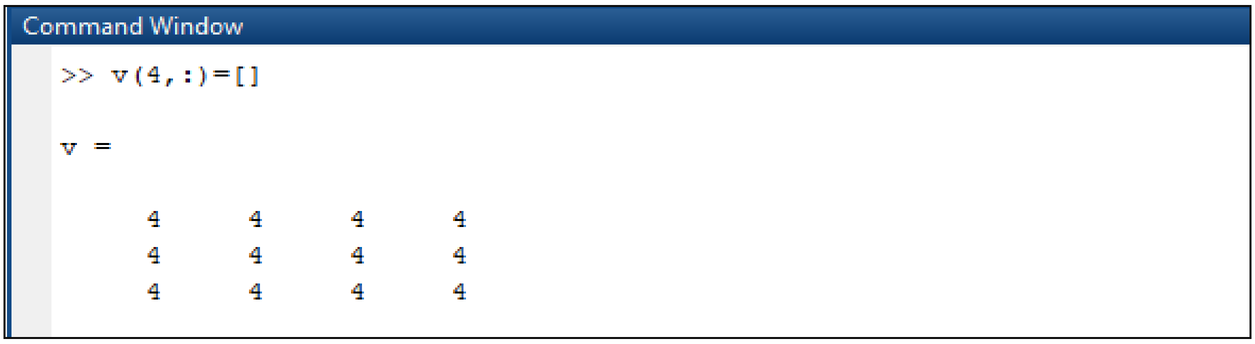
As reported in the above paragraph, to do element by element multiplication the matrix dimensions must agree.

Before, let’s do the following operations. Firstly, add a 4th column composed from 4 as seen:



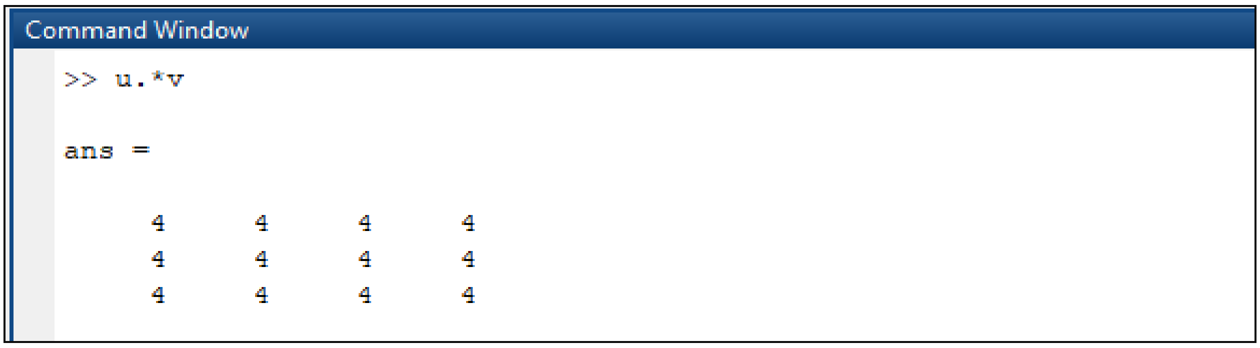
```
>> v(:,4)=[4 4 4 4]'

v =

     4     4     4     4
     4     4     4     4
     4     4     4     4
     4     4     4     4
```

Thereafter, delete the 4th row:



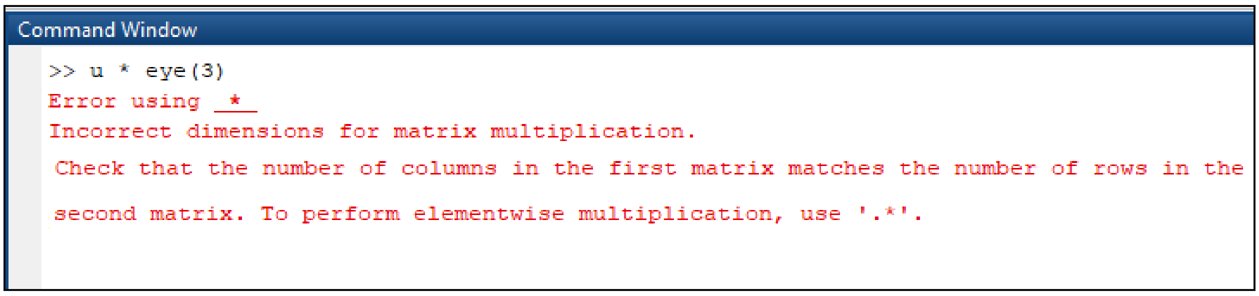
```
>> v(4,:)=[]

v =

     4     4     4     4
     4     4     4     4
     4     4     4     4
```

Now, we can see that the dimensions of matrix “u” and ‘v’ are similar and the multiplication can be done.



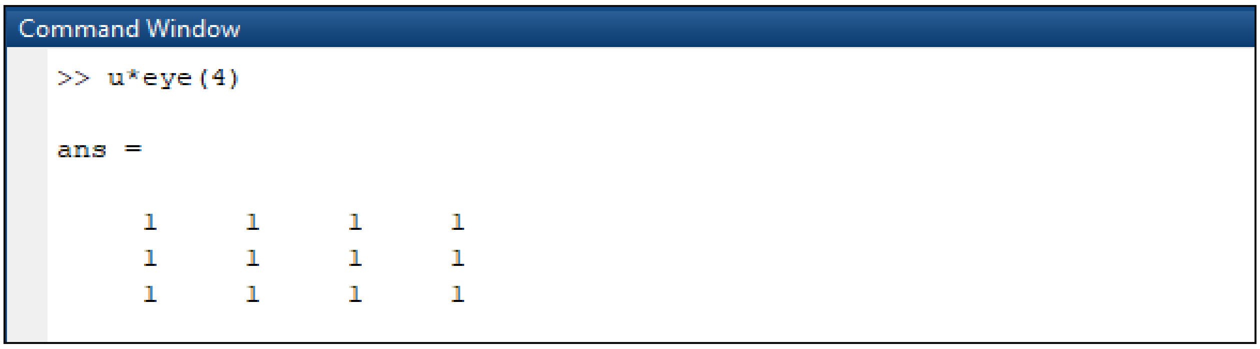
```
>> u.*v

ans =

     4     4     4     4
     4     4     4     4
     4     4     4     4
```

Another example is by multiplying the matrix ‘u’ with an identity matrix ’eye’. Here also a condition to be respected. The number of columns in the matrix ‘u’ must match with the number of rows of the matrix eye. For instance, if we would like to multiply u*eye(3) an error message will be generated:


Command Window

```
>> u * eye(3)
Error using  * 
Incorrect dimensions for matrix multiplication.
 Check that the number of columns in the first matrix matches the number of rows in the
 second matrix. To perform elementwise multiplication, use '.*'.
```

By contrast, the syntax will be written in this manner:

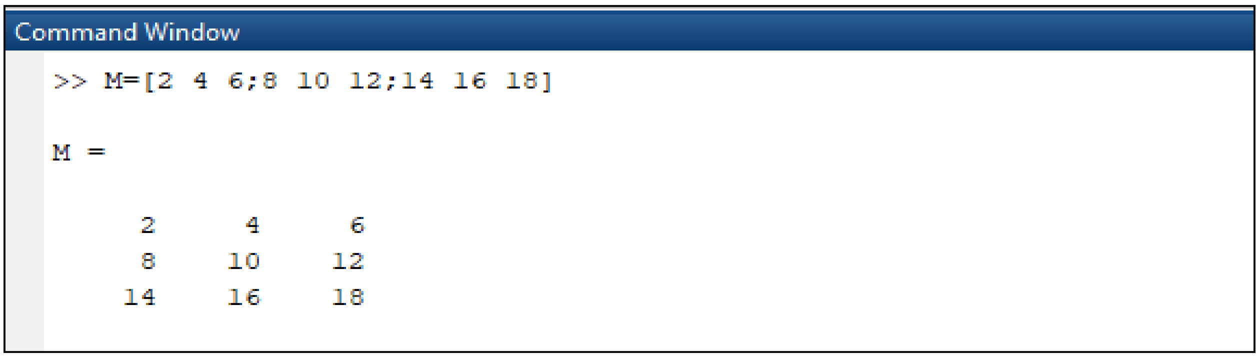

Command Window

```
>> u*eye(4)

ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```

2.2.6 *Additional useful matrices functions*

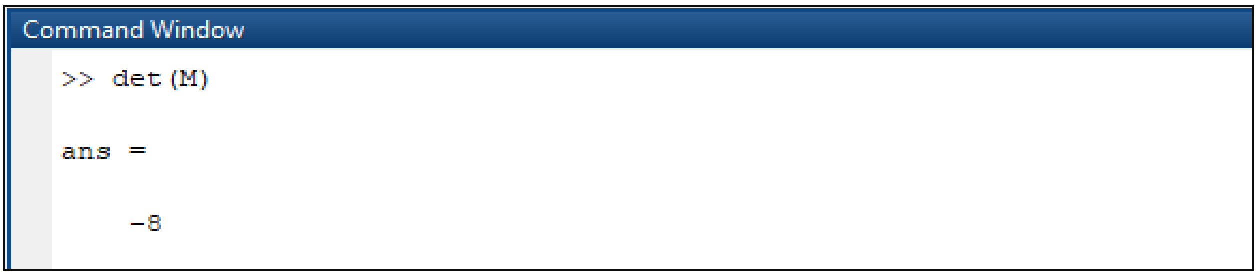
Matlab knows as a powerful tool when dealing with matrices operations, it has a set of other useful functions. These are some of most used one.

Firstly, let’s create a matrix ‘**M’** as shown:


Command Window

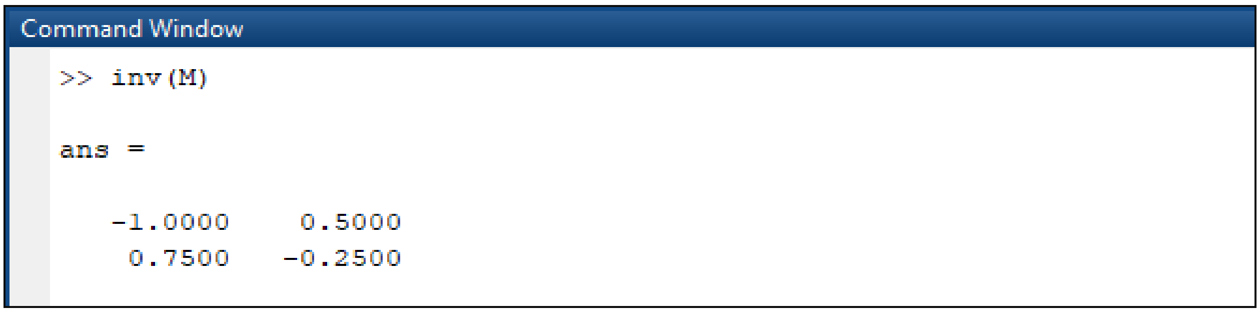
```
>> M=[2 4 6;8 10 12;14 16 18]

M =

     2     4     6
     8    10    12
    14    16    18
```

The determinant of the matrix M can be calculated by using the function “**det**” as follows:


Command Window

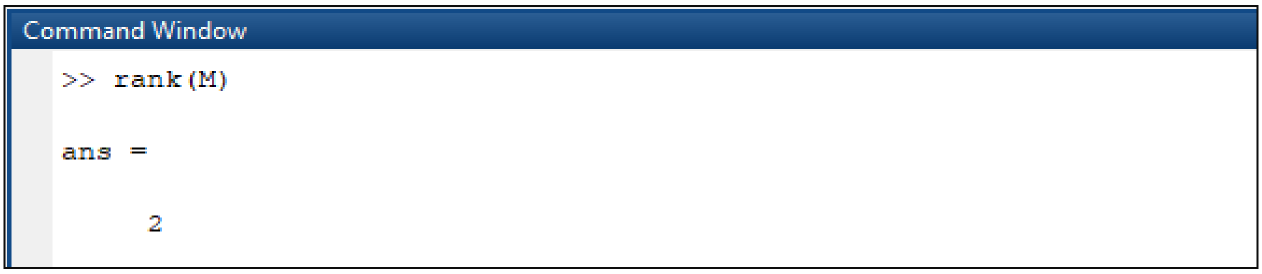
```
>> det(M)

ans =

    -8
```

To determine the inverse of the matrix M, we use the function “**inv**” like so:



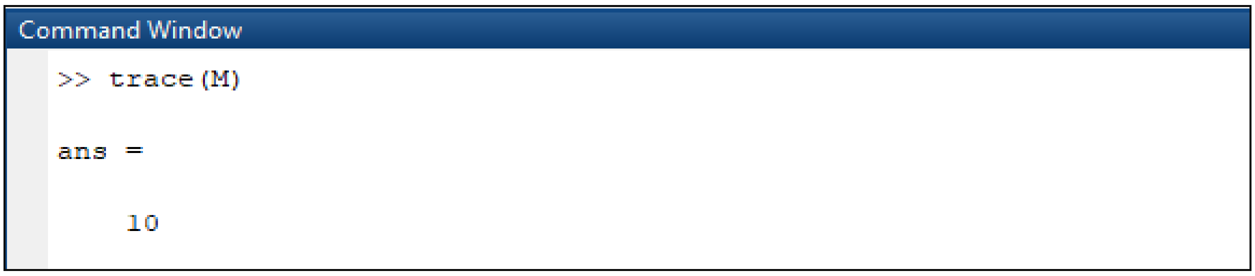
```
>> inv(M)

ans =

   -1.0000    0.5000
    0.7500   -0.2500
```

The rank of the matrix can be calculated as seen:



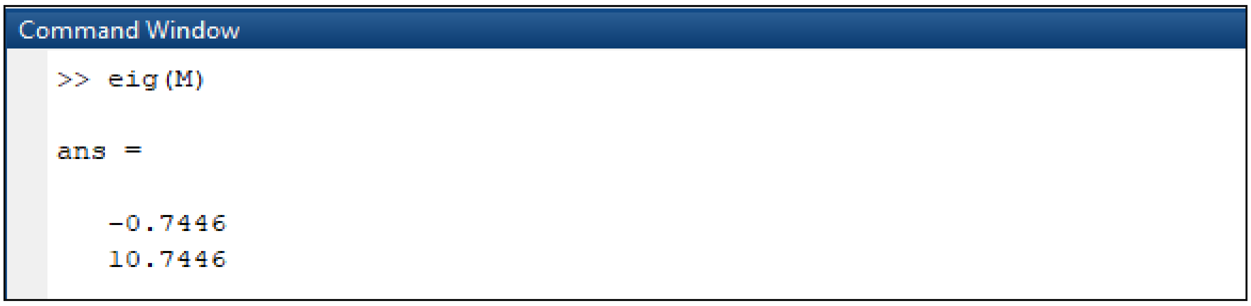
```
>> rank(M)

ans =

     2
```

To calculate the trace of matrix M, we use the function ‘**trace’** as follows:



```
>> trace(M)

ans =

    10
```

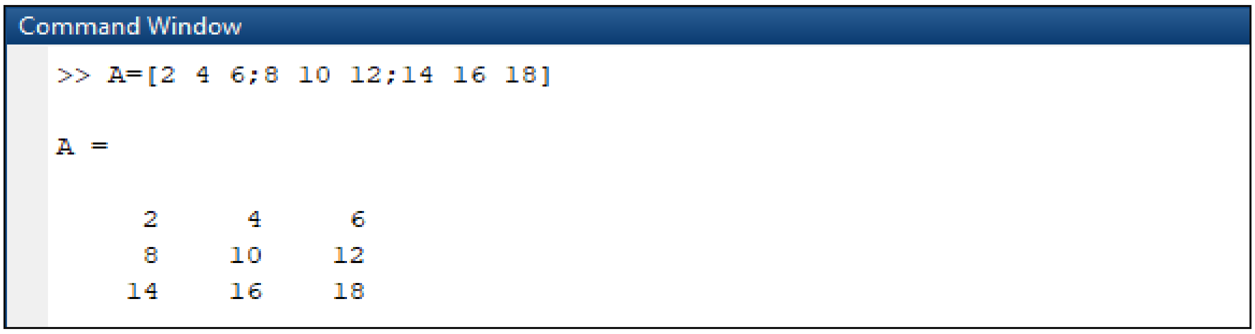
We use the syntax **eig** (M) to determinate the eigenvalues of the matrix M



```
>> eig(M)

ans =

   -0.7446
   10.7446
```

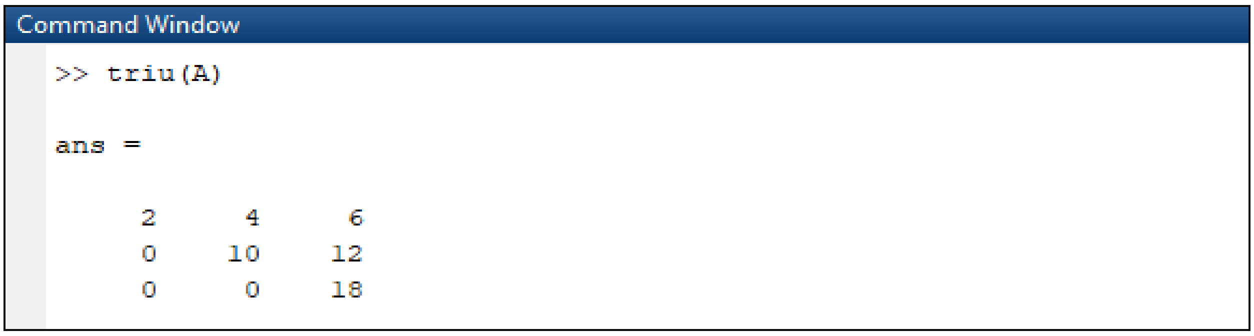
The second example aim to determine the upper and lower triangular part of the matrix **“A”**. To do this we use the syntax ‘triu’ and ‘tril’, respectively.

First of all, we create a matrix **“A”**:

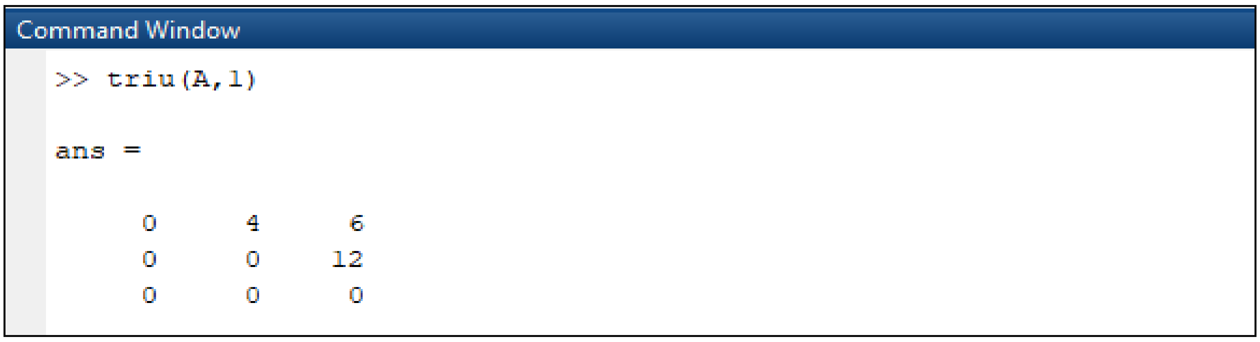


```
>> A=[2 4 6;8 10 12;14 16 18]

A =

     2     4     6
     8    10    12
    14    16    18
```

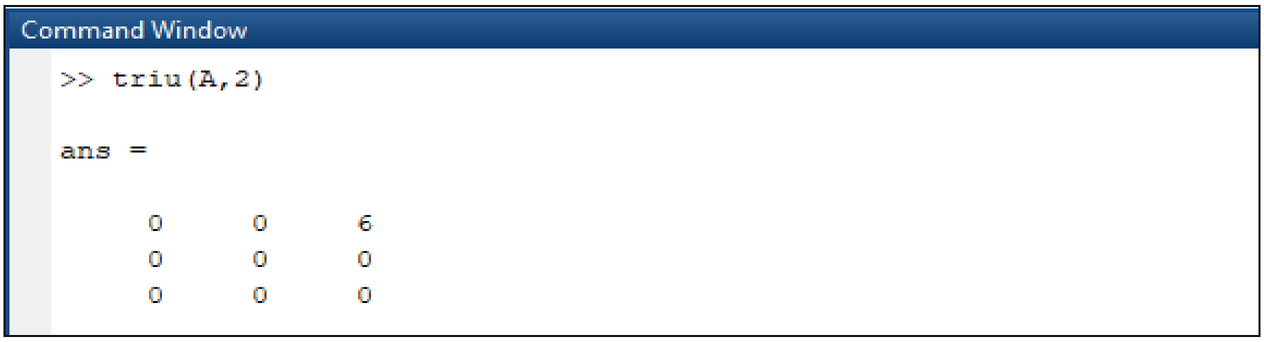
For example, the upper triangular part can be written as seen:



```
>> triu(A)

ans =

     2     4     6
     0    10    12
     0     0    18
```

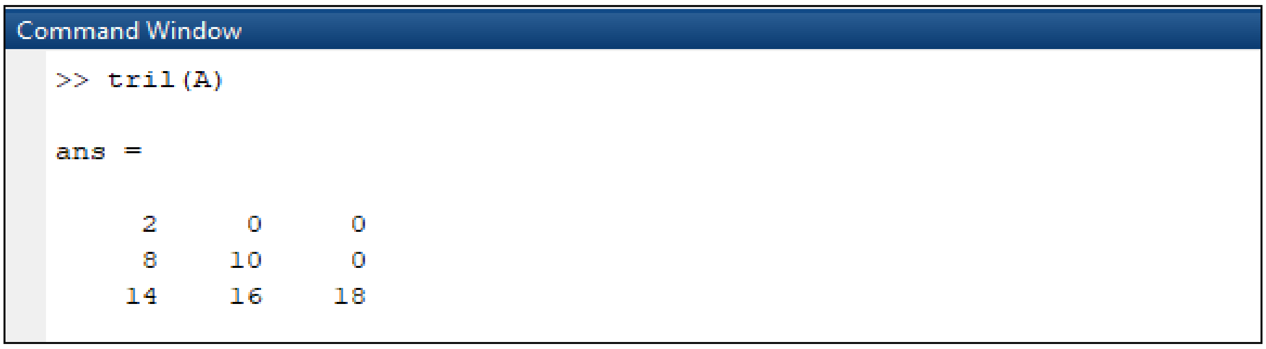


```
>> triu(A,1)

ans =

     0     4     6
     0     0    12
     0     0     0
```

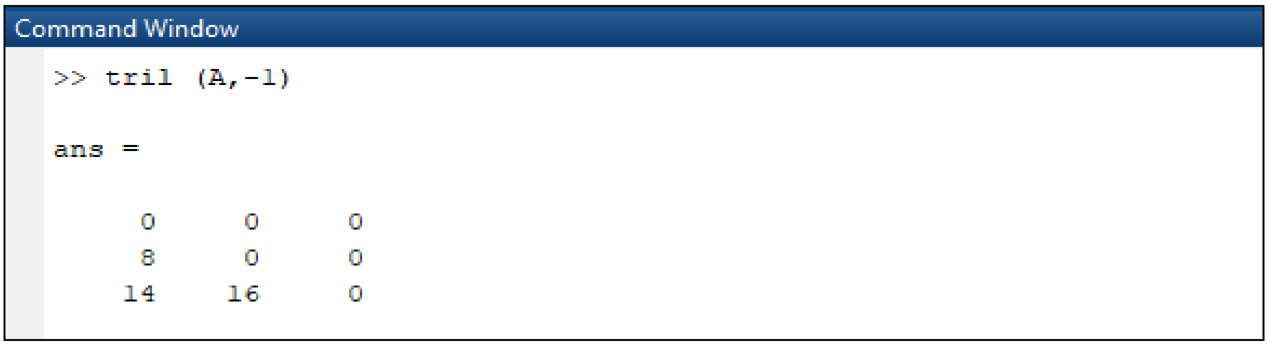


```
>> triu(A,2)

ans =

     0     0     6
     0     0     0
     0     0     0
```

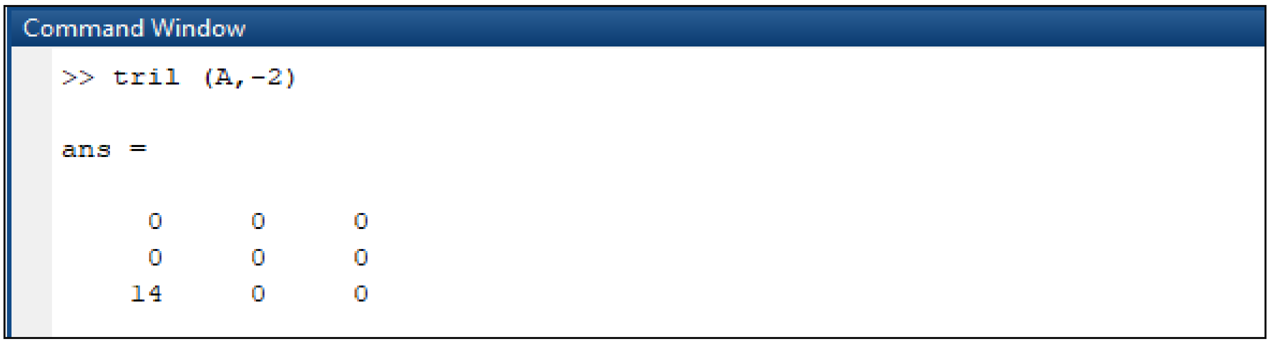
For instance, the lower triangular part can be generated as follows:



```
>> tril(A)

ans =

     2     0     0
     8    10     0
    14    16    18
```

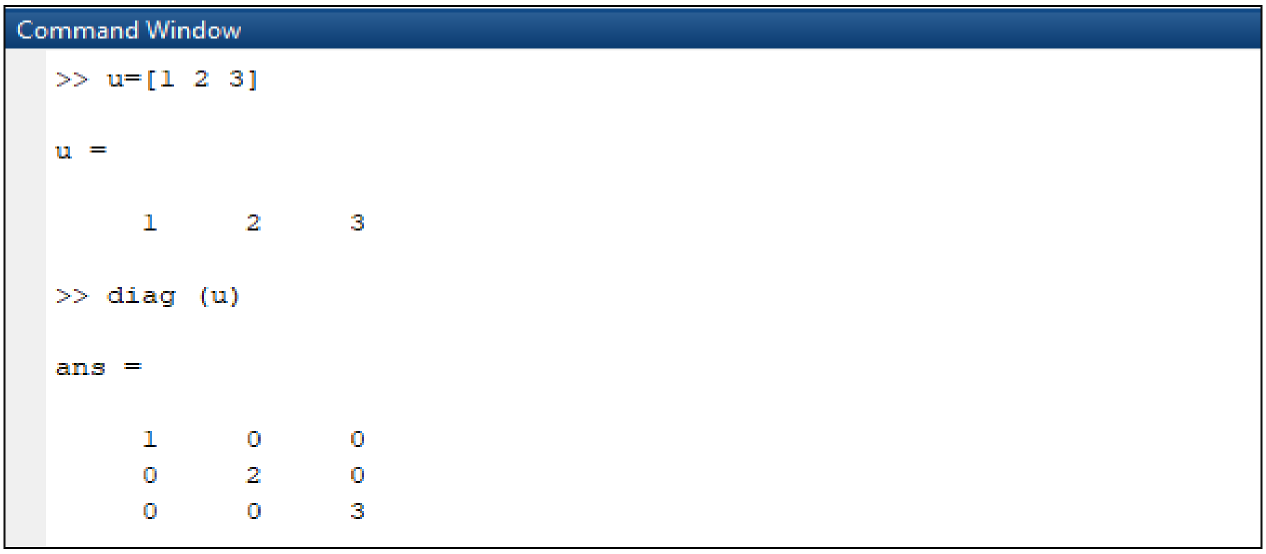


```
>> tril (A,-1)

ans =

     0     0     0
     8     0     0
    14    16     0
```



```
>> tril (A,-2)

ans =

     0     0     0
     0     0     0
    14     0     0
```

The third example aims to creates a matrix having the vector ‘u’ in the diagonal and 0 elsewhere.



```
>> u=[1 2 3]

u =

     1     2     3

>> diag (u)

ans =

     1     0     0
     0     2     0
     0     0     3
```

Chapter 3 **Matlab Plotting**

One of the most helpful Matlab commands is without a doubt the plot command. When writing this command, Matlab will open a new figure and plot the parameter (an array) vs its index. Matlab will interpret the first array as the x-coordinates and the second array as the y-coordinates[7].

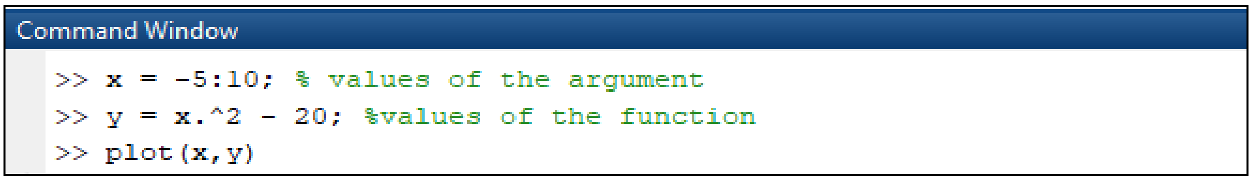
3.1 Matlab plot function

The **plot** function is easily used to plot any given data. The following example aim to simply to usage of this function in simple 2D plots.

For example, we would like to draw the following function:

- x = -5:10;
- y = x.^2 - 20;

The syntax in Matlab will be as follows:


Command Window

```
>> x = -5:10; % values of the argument
>> y = x.^2 - 20; %values of the function
>> plot(x,y)
```

Accordingly, a new window will be open illustating the figure bellow.

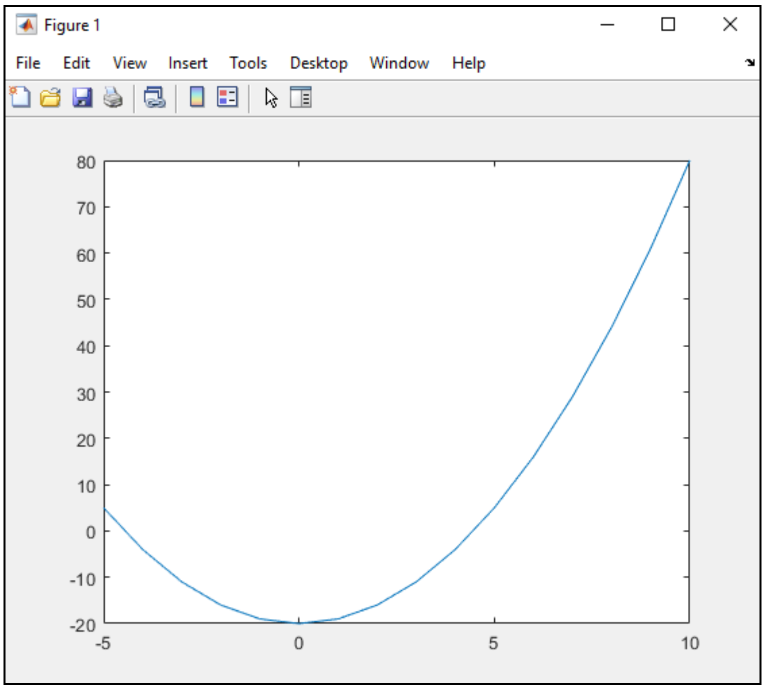
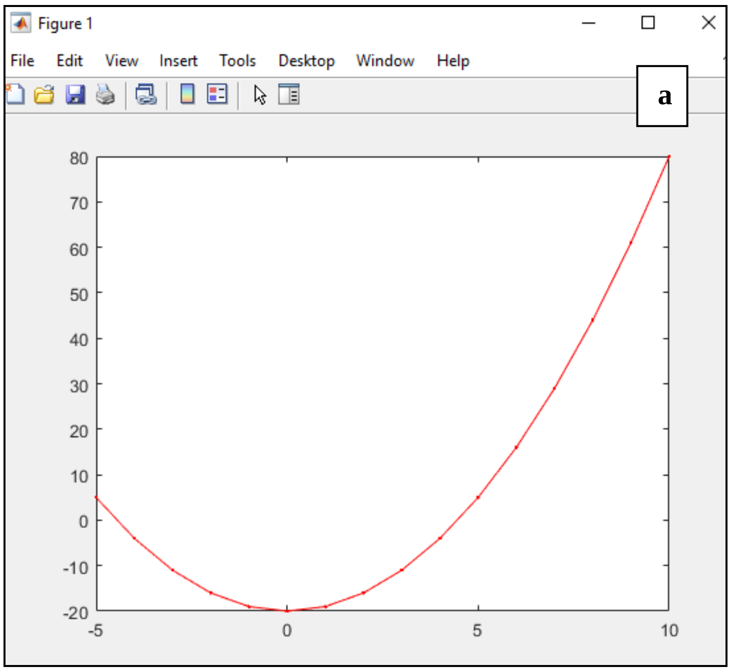


Figure 3.1 *Matlab 2D line plot*

Matlab gives us the opportunity to change the type of line, color, and marker of the plotted line. These laters can all be specified using a third string input. The pre-defined colour strings: **’k’** black **’r’** red **’g’** green **’b’** blue **’w’** white **’m’** magenta **’y’** yellow **’c’** cyan. Figure 5.2 displays the previous figure with different colors.



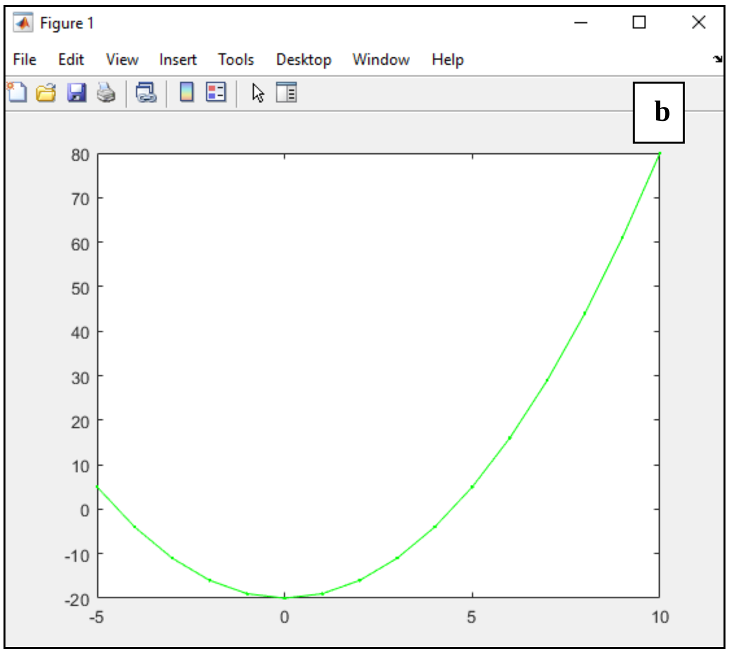


Figure 3.2 *Matlab’s plots style. (a) Red 2D line plot. (b) Green 2D line plot*

It is possible to change as well the appearance of a curve by changing the shape of the coordinate points, and the type of line connecting the points. The tables bellow summirize all the changes that can be made during plotting.

Table 3.1. *Matlab's curve color*

Operation	**Signification**
b or blue	Curve in blue
g or green	Curve in green
r or red	Curve in red
c or cyan	Curve in cyan
m or magenta	Curve in magenta
y or yellow	Curve in yellow
k or black	Curve in black

Table 3.2. *Matlab's curve style*

Character	**Effect**
-	Solid line
:	Dotted line
-.	Dash dotted line
--	Dashed line

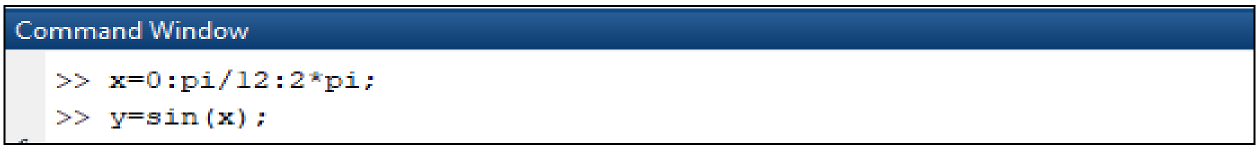
Table 3.3. *Matlab's curve effect*

Charactere	**Effect**
.	Point
+	Plus sign
*	Star
x	Cross
o	Circle
s	Square
d	Diamond
^	Upward pointing triangle

V	Downward pointing triangle
>	Right pointing triangle
<	Left pointing triangle
P	Pentagram

In aim to understand accuratly, let’s draw the function the function y = sin(x) for x = [0 ... 2π] with step of π/12.

The syntax in Matlab will be as seen:



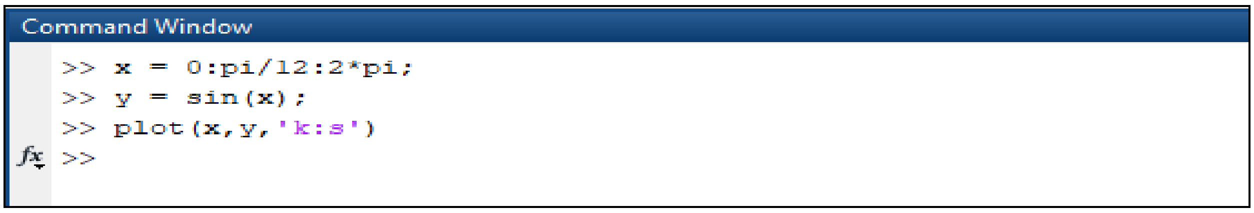
Command Window

```
>> x=0:pi/12:2*pi;
>> y=sin(x);
```

Firsly, draw it by trying different shapes:

- starting with black, dotted line with squares.

The code will be like so:



Command Window

```
>> x = 0:pi/12:2*pi;
>> y = sin(x);
>> plot(x,y,'k:s')
>>
```

The curve will appear as seen:

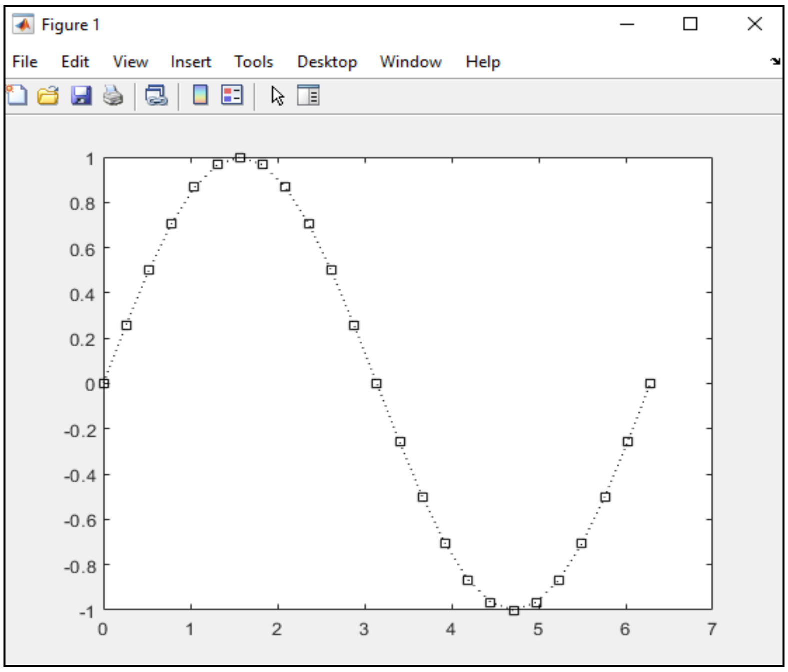
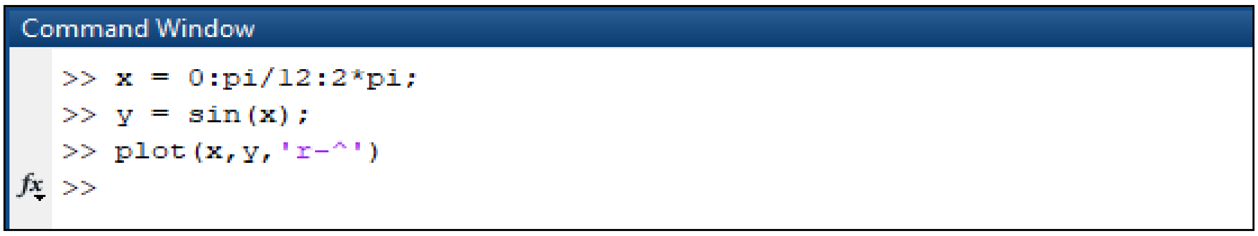


Figure 3.3 *Matlab customized 2D plot (Black, Square, Dotted line)*

- Red color, solid line and with triangles



```
>> x = 0:pi/12:2*pi;
>> y = sin(x);
>> plot(x,y,'r-^')
>>
```

The curve will be depicted as follows:

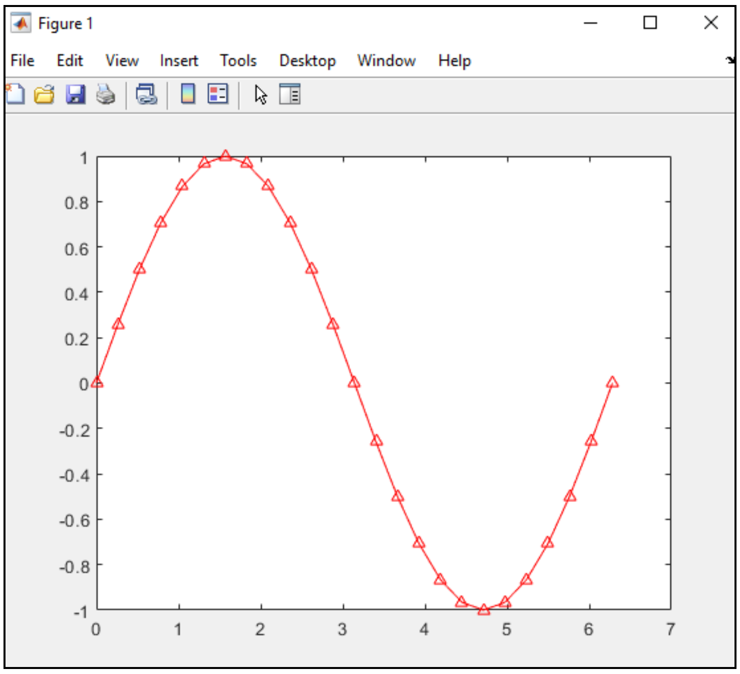
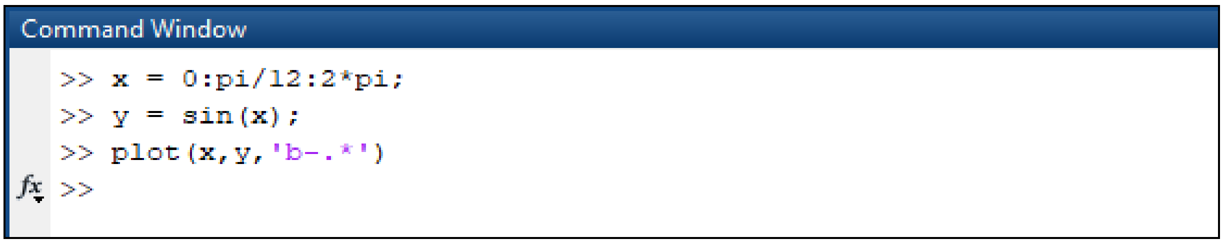


Figure 3.4 *Matlab customized 2D plot (Red, Triangle, Solid line)*

- Blue color, dash dotted line and with stars



```
>> x = 0:pi/12:2*pi;
>> y = sin(x);
>> plot(x,y,'b-.*')
>>
```

The outcome will be illustrated as seen:

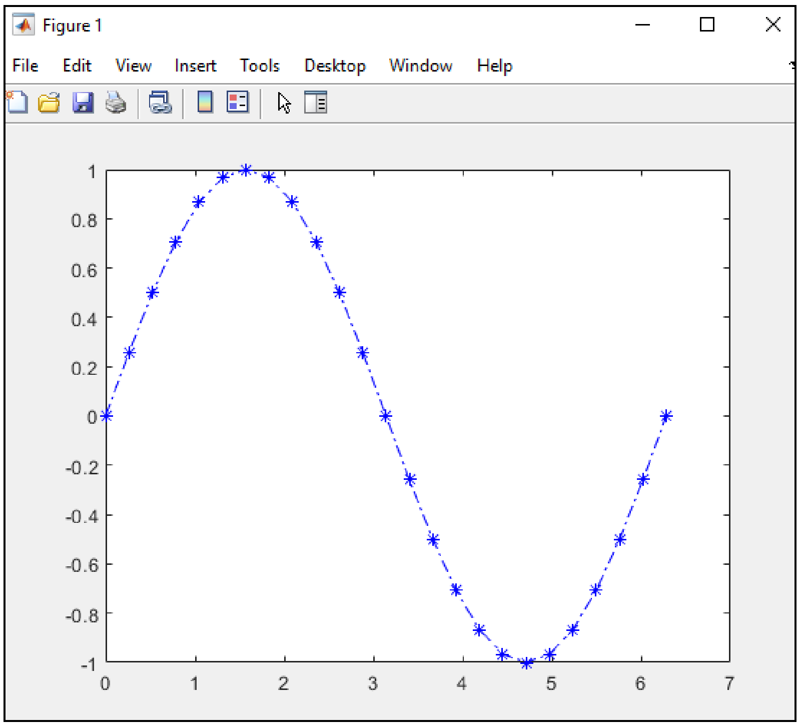
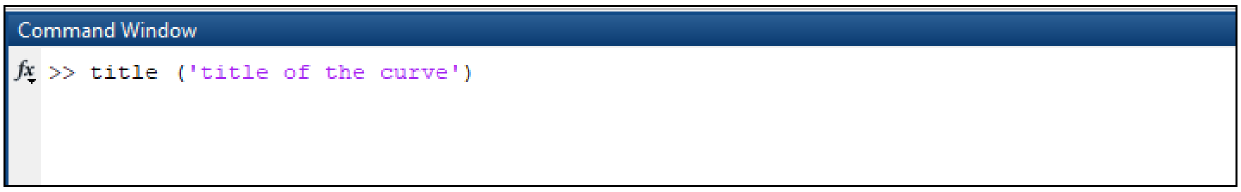


Figure 3.5 *Matlab customized 2D plot (Blue, Start, Dash-dotted line)*

3.2 Curves customzing

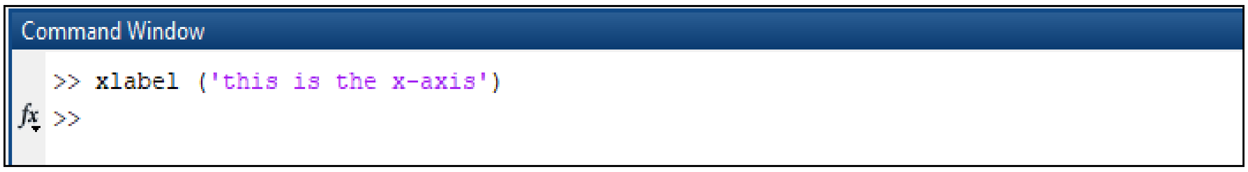
Matlab allows to customize the curve by adding a grid or by including a legend to to the axes. Another interesting point, is the ability to indicate the locations of significant points in a curve with a comment. All these can be done by using the following syntax:

In the beginning, to give a title to a figure containing a curve we use the "title" function as seen:



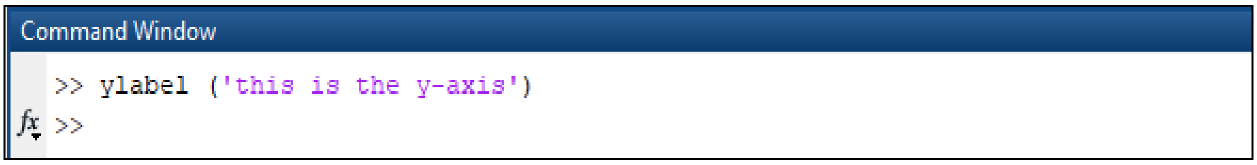
```
>> title ('title of the curve')
```

Subsequently, to give a title for the horizontal x-axis, we use the "**xlabel**" function:



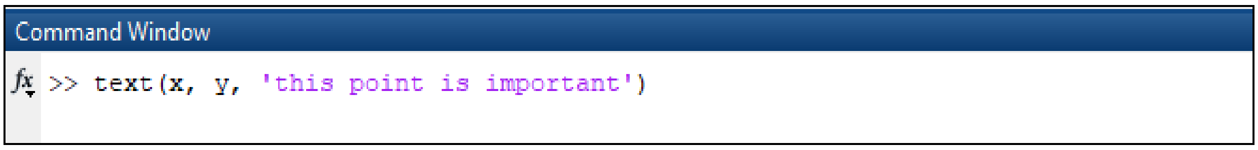
```
>> xlabel ('this is the x-axis')
>>
```

Besides, to give a title for the vertical y axis, we use the “**ylabel**” function like so:


Command Window

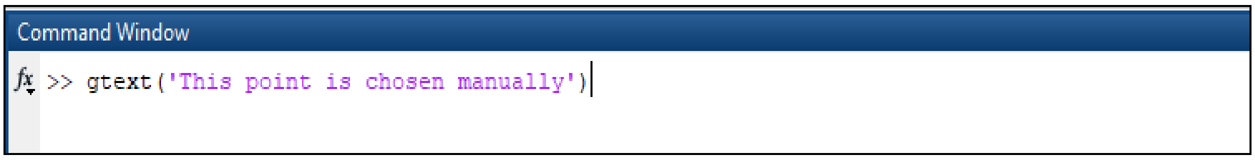
```
>> ylabel ('this is the y-axis')
>>
```

To write text (a message) to the graphics window at an indicated position by “x” and “y” coordinates, we use the “text” function:


Command Window

```
>> text(x, y, 'this point is important')
```

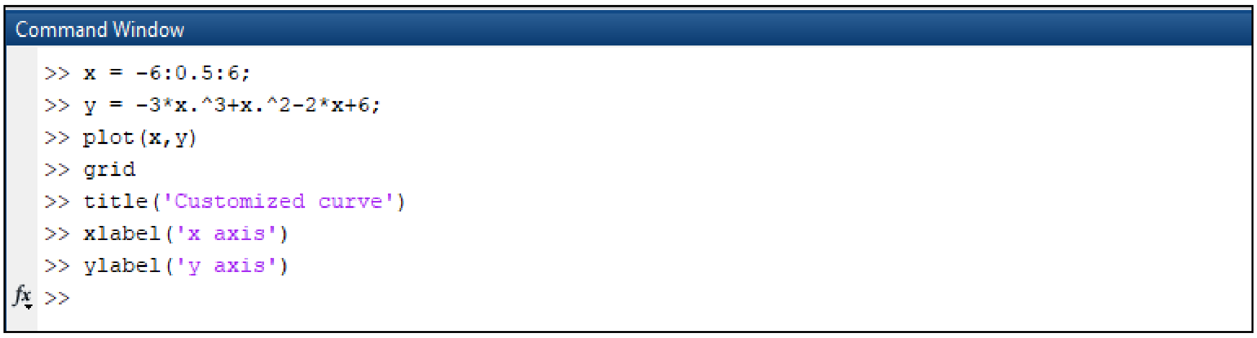
To put a text on a desired position chosen manually (with the mouse), we use the “gtext” function, which has the following syntax:


Command Window

```
>> gtext('This point is chosen manually')
```

To put a grid, use the “grid” (or grid on) command. To remove it, reuse the same “grid” (or grid off) command.

The following example helps you to understand better.


Command Window

```
>> x = -6:0.5:6;
>> y = -3*x.^3+x.^2-2*x+6;
>> plot(x,y)
>> grid
>> title('Customized curve')
>> xlabel('x axis')
>> ylabel('y axis')
>>
```

The curve will appear accordinagly:

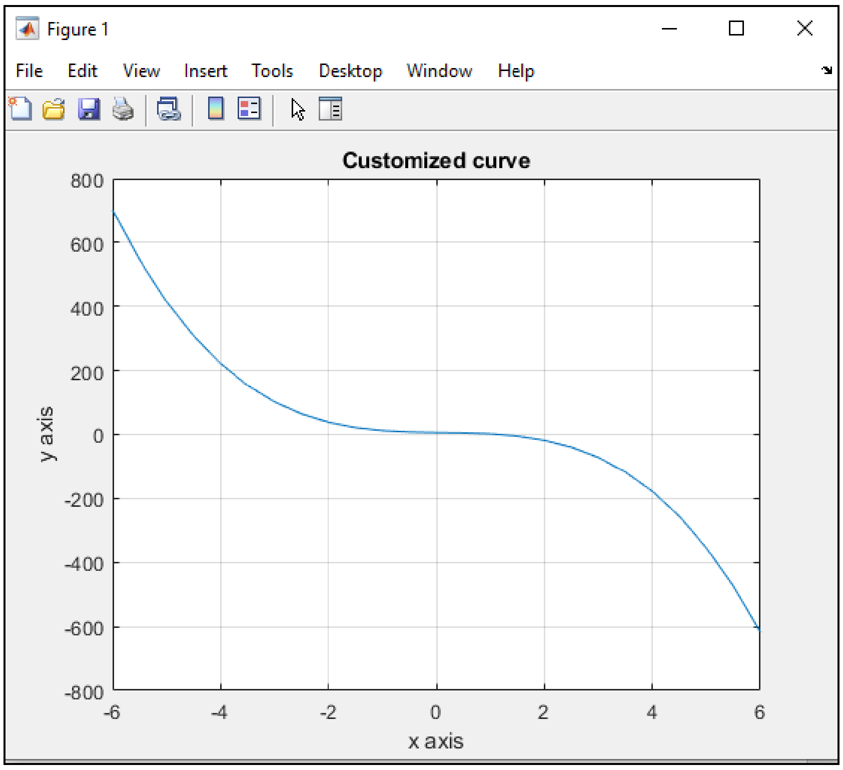


Figure 3.6 *Matlab 2D customized plot*

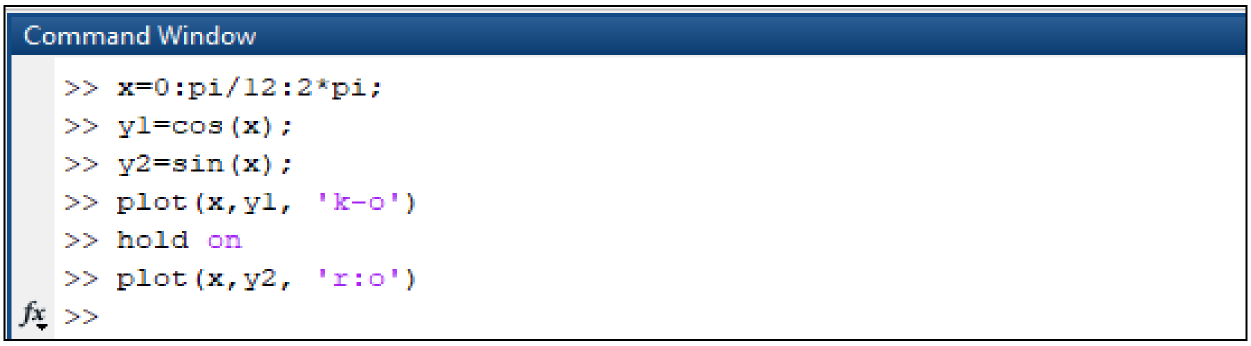
3.3 Multiple curves in a same graph

By default, in Matlab, each new drawing with the plot command erases the previous one. To force a new curve to coexist with previous curves, there are several methods, such as the hold command as well as the use plot with several arguments

3.3.1 The command ***hold***

The command "**hold**" or "**hold on**" enable the preservation of previous curves, which allows the display of different curves in the same figure. If we desire ti disable its effect, we can simply rewrite hold or hold off.

For instance, to draw the curve of the two functions cos(x) and sin(x) in the same figure, we can write:



```
Command Window
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>> plot(x,y1, 'k-o')
>> hold on
>> plot(x,y2, 'r:o')
>>
```

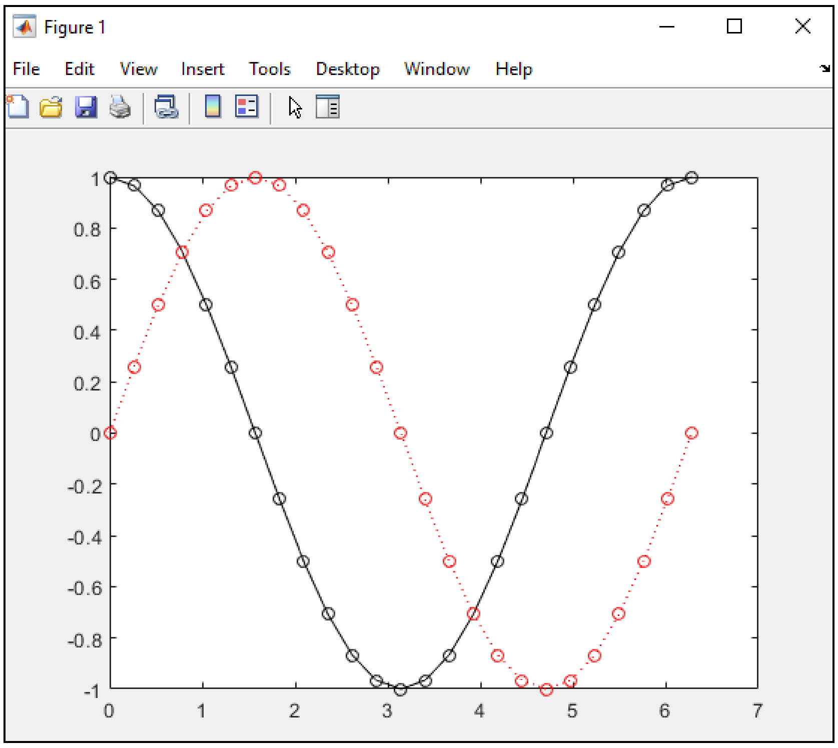
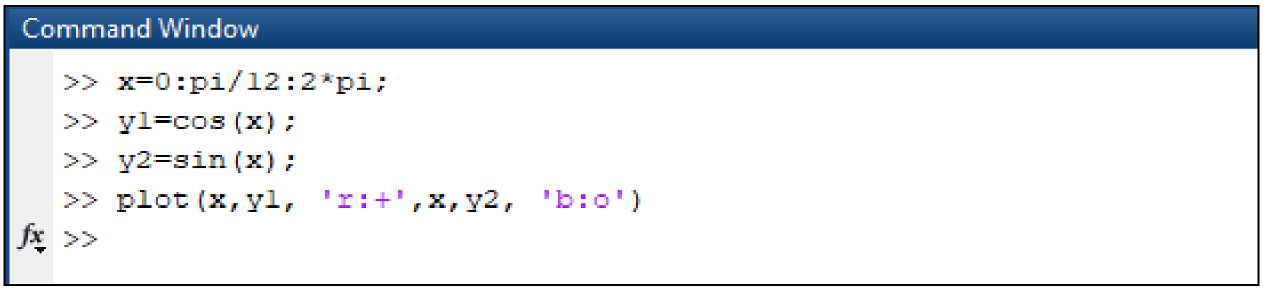


Figure 3.7 *Matlab's multiple curve (hold on command)*

3.3.2 Plot with several arguments

We can use plot with various couple (x,y) or triple (x,y, markor) as a arguments.



```
Command Window
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>> plot(x,y1, 'r:+',x,y2, 'b:o')
>>
```

The curve will be illustrated like so:

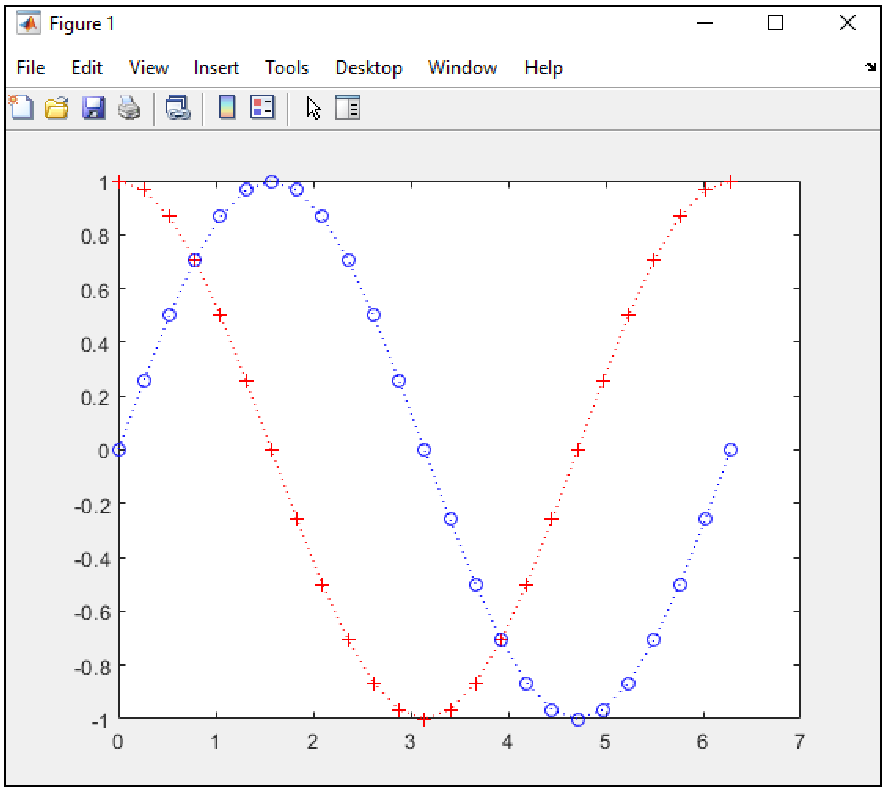
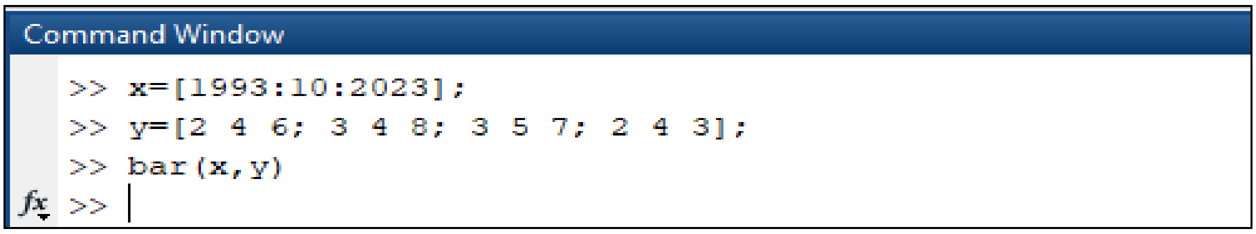


Figure 3.8 *Matlab's multiple curve (Several arguments)*

3.4 *Histogram and bar graphs in Matlab*

The Matlab not only allows the display of points to draw curves, but it also offers the possibility of drawing bar graphs and histograms. To draw a bar graph, we use the "bar" function, which has the same operating principle as the plot function.

The following example explains how we can draw bar graphs. First of all, write this code in the Matlab as seen:



```
>> x=[1993:10:2023];
>> y=[2 4 6; 3 4 8; 3 5 7; 2 4 3];
>> bar(x,y)
>>
```

The drawn bar graph will be as shown:

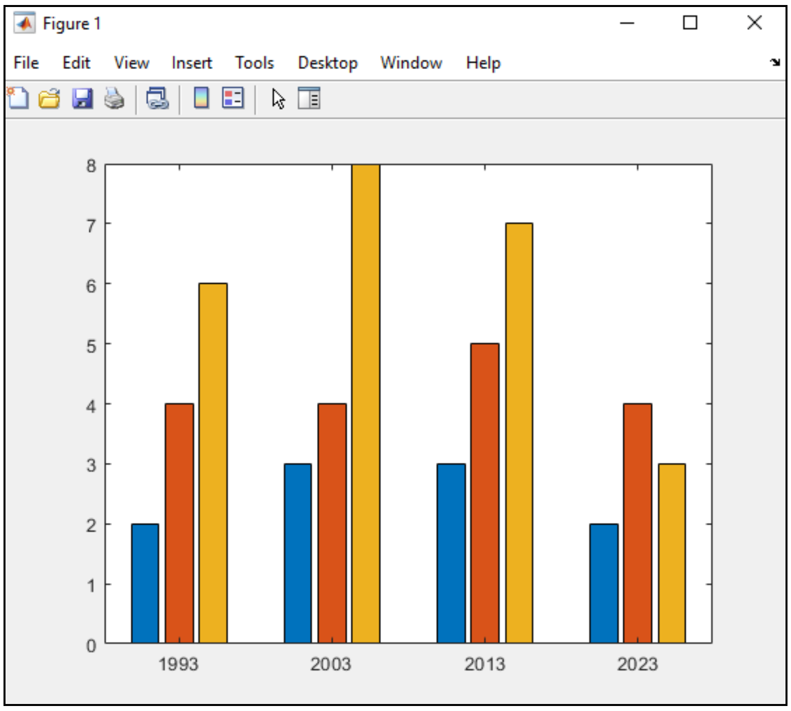


Figure 3.9 *Matlab's bar chart*

A huge variaty of functions exist that can for instance change the shape of bar graphs, for example the "bar3" to give a 3D aspect, as seen:

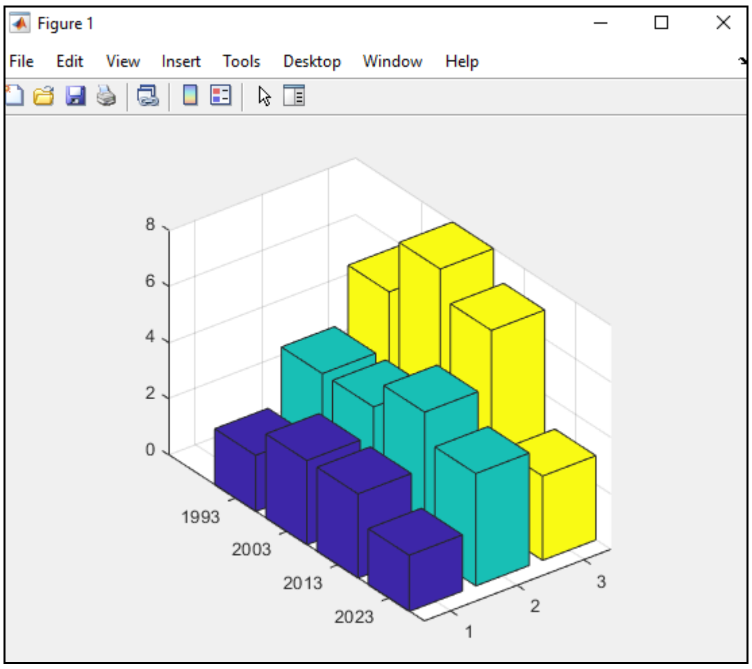
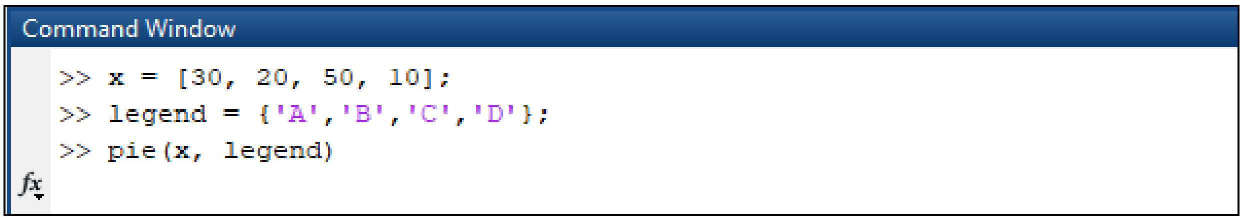


Figure 3.10 *Matlab's 3D bar chart*

The pie charts can also be drawn in Matlab as seen in the following example:



```
Command Window
>> x = [30, 20, 50, 10];
>> legend = {'A','B','C','D'};
>> pie(x, legend)
fx
```

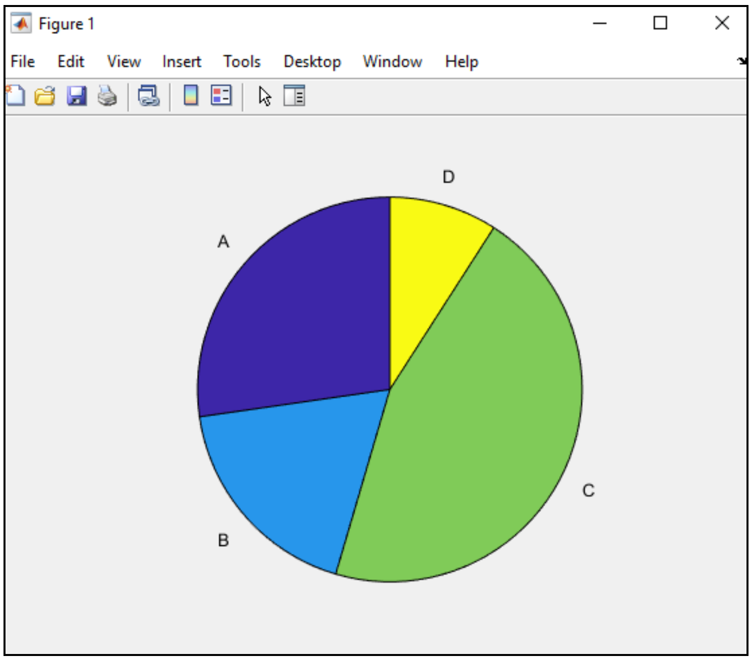
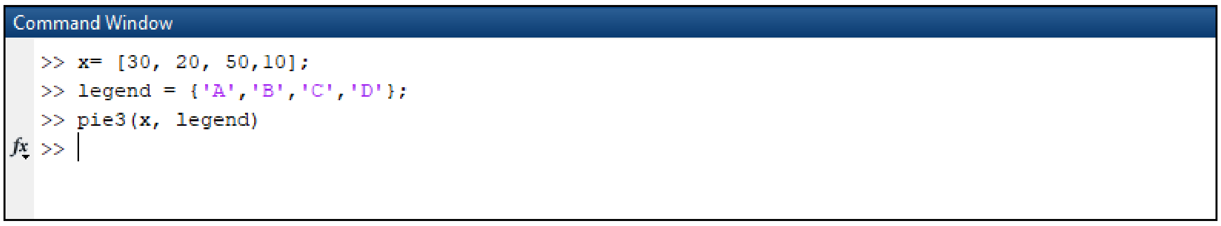


Figure 3.11 *Matlab's pie chart*

To give the 3D aspect, we use the "pie3" function as given:



```
Command Window
>> x= [30, 20, 50,10];
>> legend = {'A','B','C','D'};
>> pie3(x, legend)
fx >> |
```

And the 3D aspect is added as seen,

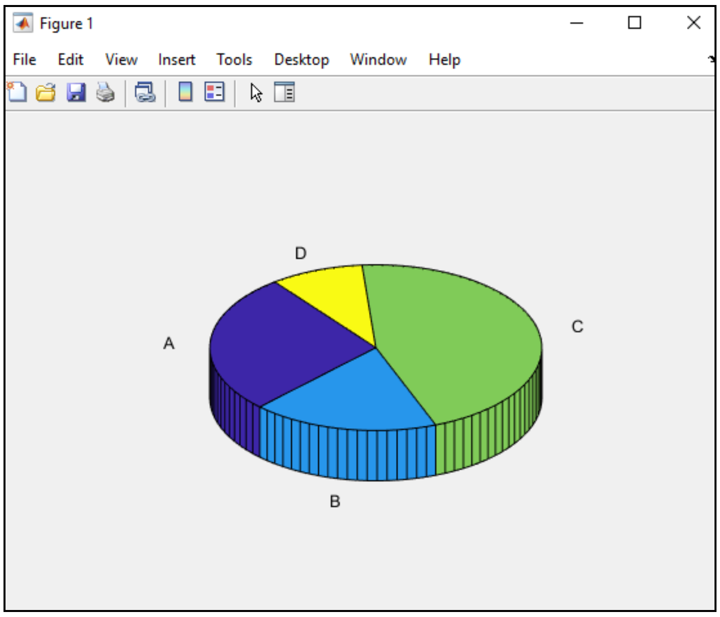
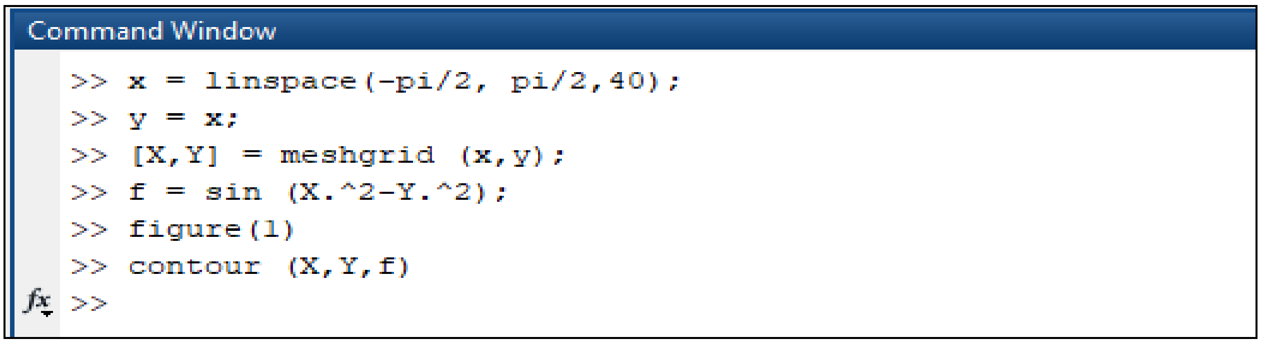


Figure 3.12 *Matlab's 3D pie chart*

3.5 Plotting 3D curves in Matlab

The way MATLAB handles two- and three-dimensional graphics is one of its outstanding characteristics. Although we won't often need to use MATLAB's powerful graphical rendering, we should be familiar with the fundamental functions. Examples demonstrate a few of the numerous options:

- Firstly, try to write the following code



```
Command Window
>> x = linspace(-pi/2, pi/2,40);
>> y = x;
>> [X,Y] = meshgrid (x,y);
>> f = sin (X.^2-Y.^2);
>> figure(1)
>> contour (X,Y,f)
fx >>
```

- The figure displayed is as follows:

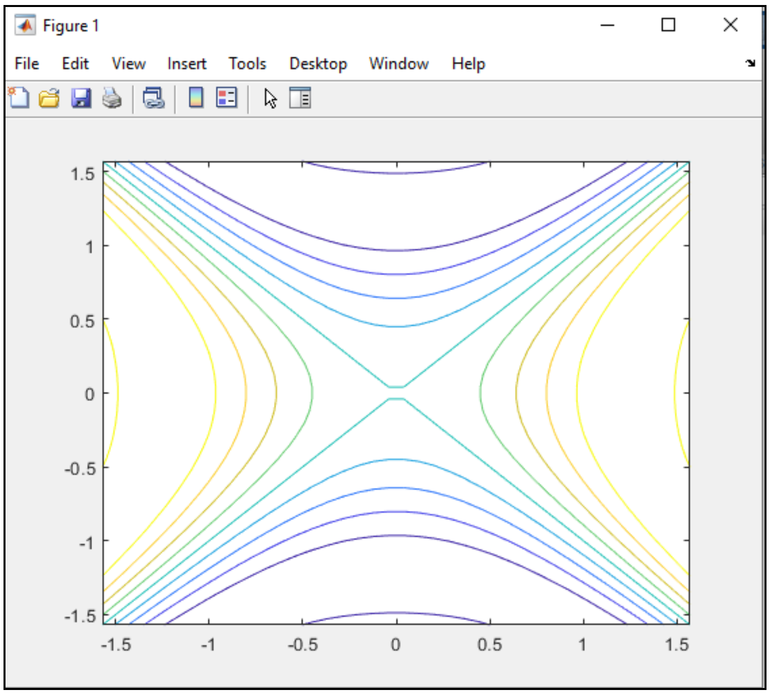
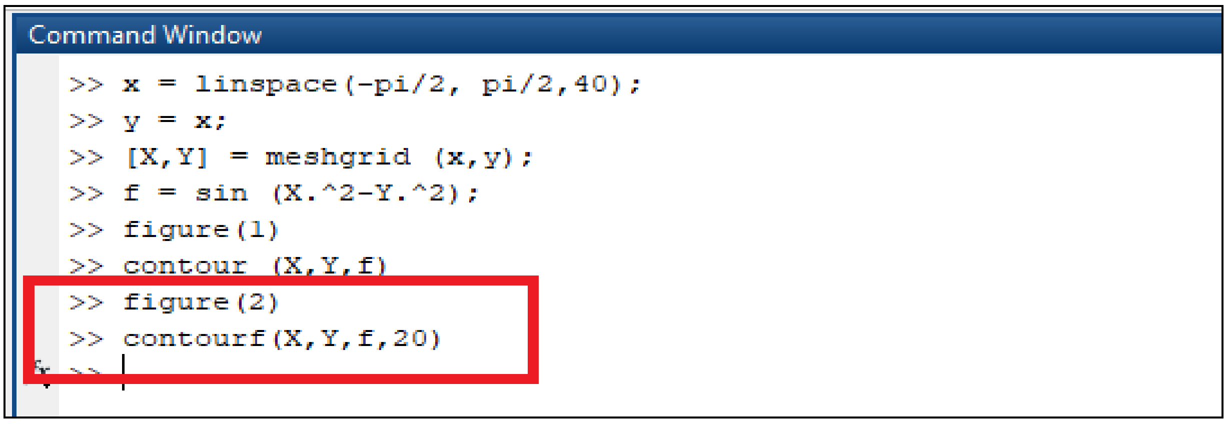


Figure 3.13 *Matlab contour plot*

- The second one, try to add the sentences highlighted in red:



```
>> x = linspace(-pi/2, pi/2,40);
>> y = x;
>> [X,Y] = meshgrid (x,y);
>> f = sin (X.^2-Y.^2);
>> figure(1)
>> contour (X,Y,f)
>> figure(2)
>> contourf(X,Y,f,20)
>> 
```

- The figure displayed is as shown:

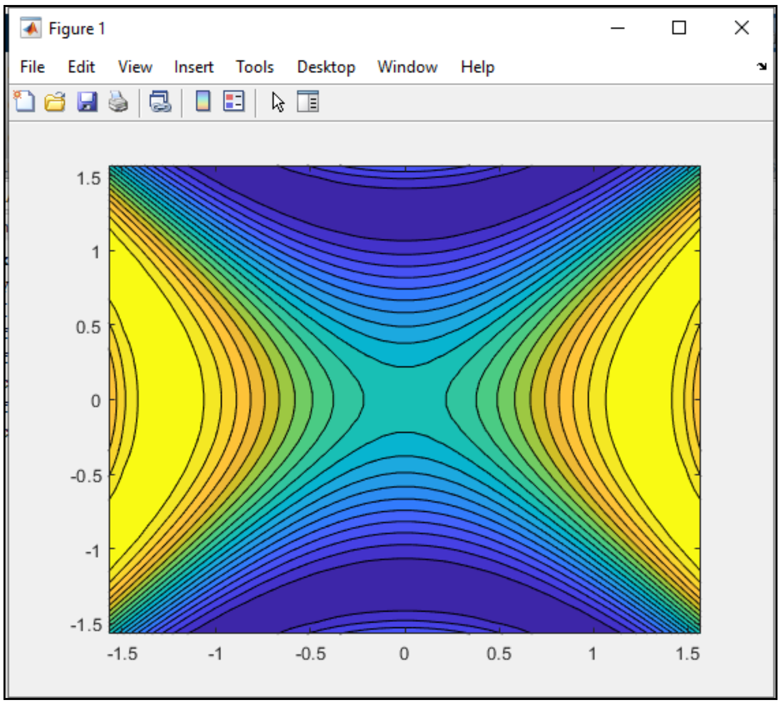
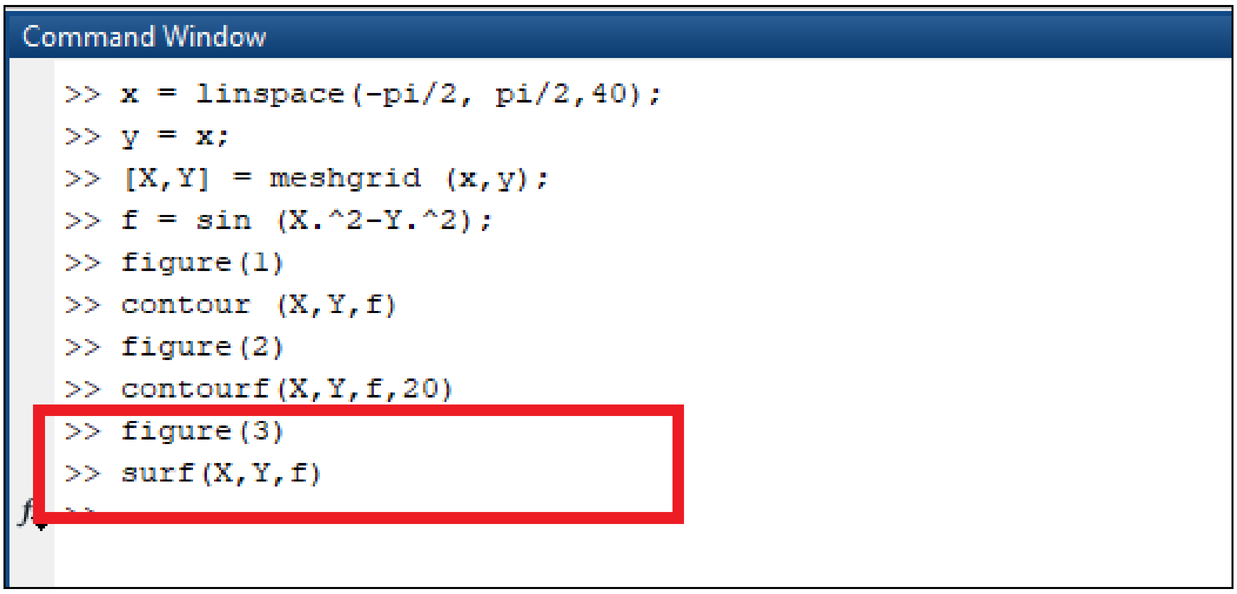


Figure 3.14 *Matlab filled contour plot*

- The third example is by using the surf (X,Y,f) command as seen:



```
>> x = linspace(-pi/2, pi/2,40);
>> y = x;
>> [X,Y] = meshgrid (x,y);
>> f = sin (X.^2-Y.^2);
>> figure(1)
>> contour (X,Y,f)
>> figure(2)
>> contourf(X,Y,f,20)
>> figure(3)
>> surf(X,Y,f)
>>
```

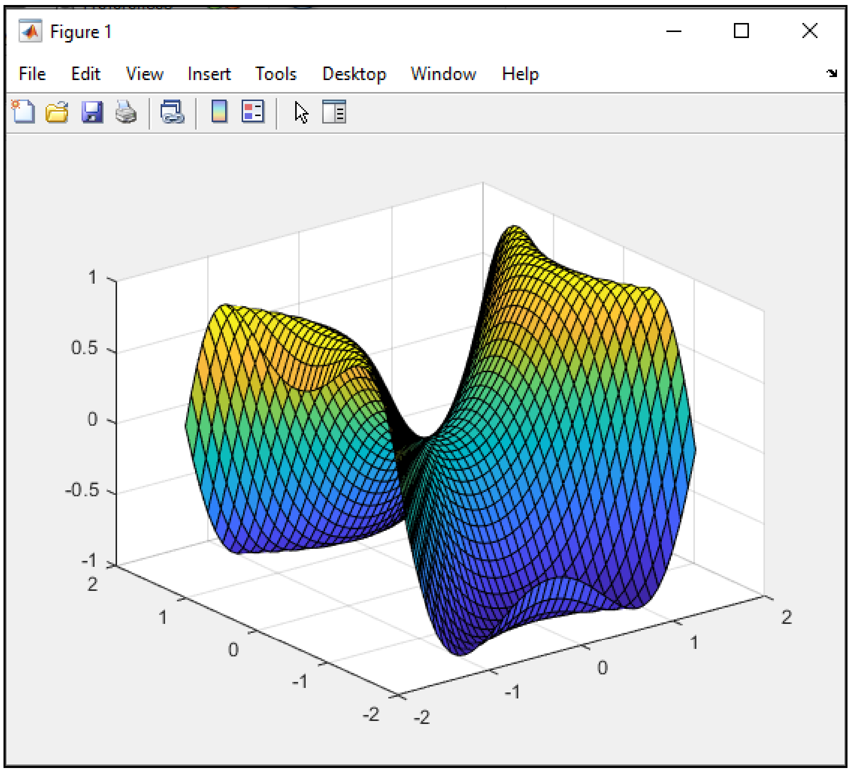


Figure 3.15 *Matlab 3D surface plot*

These three graphics, generated with the functions contour(X,Y,f), contour(X,Y,f,20), and surf(X,Y,f), represent a contour plot, a filled contour plot with 20 contour levels, and a surface plot, respectively. Using the command meshgrid(x,y), a grid is created and used to visualize the function f.

Chapter 4 Matlab's loops and conditional statements

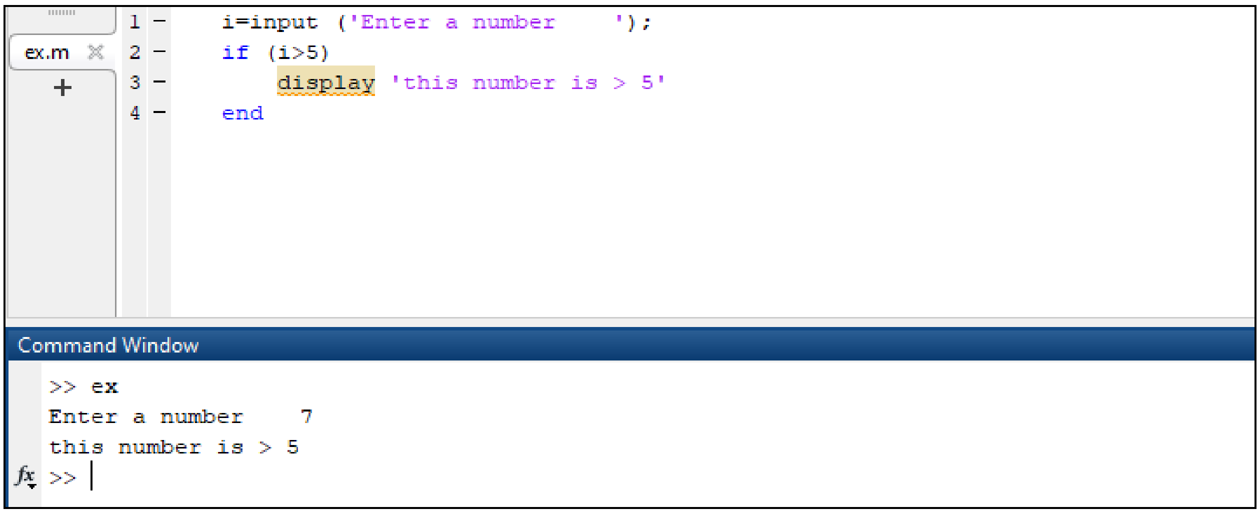
As several programs, in Matlab you can define set of code that either conditionally execute or repeat in a loop. Conditional statements use "if" or "switch", while loops use a "for" or "while" keyword [8].

4.1 *The "if, else, elseif" condition*

The if statement is among the simplest and most used conditional statements in Matlab. It executes statements "if" condition is true. Its general syntax can be written accordingly:

```
if (condition)
     instruction 1
     instruction 2
          .........
     instruction N
end
```

The condition "if" evaluates an expression, as well as executing a group of statements when the expression is true. Let's see the following example to understand better. We would like to display a given message **"this number is > 5"**, when the number is superior then 5.



```
i=input ('Enter a number     ');
if (i>5)
    display 'this number is > 5'
end
```

Command Window

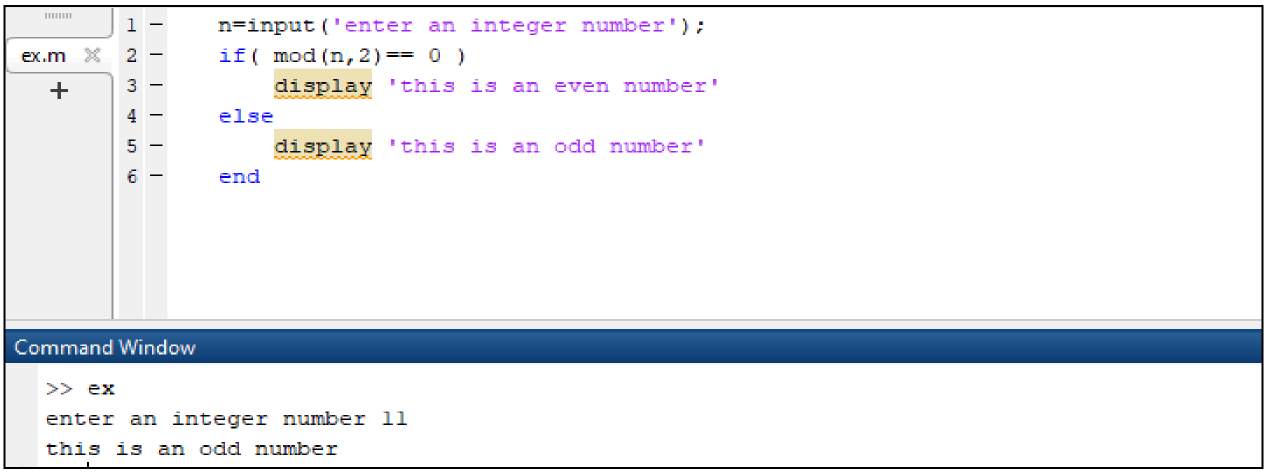
```
>> ex
Enter a number     7
this number is > 5
>> 
```

The condition “if” evaluates an expression, as well as executing a group of statements when the expression is true. Let’s see the following example to understand better. We would like to display a given message **“this number is > 5”**, when the number is superior then 5.

The second conditional statement is by using if – else – end. The syntax is written as seen:

if (condition)
instruction set 1
else
instruction set 2
end

The given example will make us understand better. We would like to give a random number and the Matlab will display if it is an even or odd number.



```
n=input('enter an integer number');
if( mod(n,2)== 0 )
    display 'this is an even number'
else
    display 'this is an odd number'
end
```

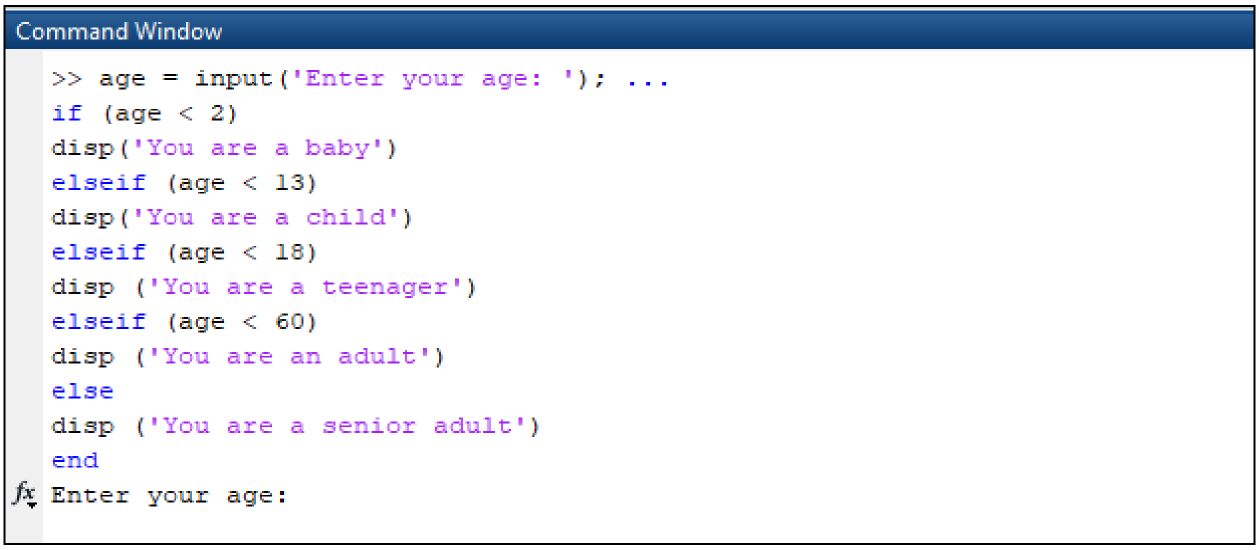
```
>> ex
enter an integer number 11
this is an odd number
```

The given example will make us understand better. We would like to give a random number and the Matlab will display if it is an even or odd number.

The third conditional statement is “elseif”. Be aware that if it is necessary to check several conditions, the “elseif” can be used for each additional condition, and at the end we can put an “else” in the case where no condition has been evaluated true. The syntax can be written thusly:

if (expression_1)
Instruction Set 1
elseif (expression_2)
Instruction Set 2
....
elseif (expression_n)
Instruction Set n
Else
Set of instructions if all expressions were false
End

The following example explains how we can use this syntax. For instance, we would like to write a program that defines the stage of your life according to your age:



```
>> age = input('Enter your age: '); ...
if (age < 2)
disp('You are a baby')
elseif (age < 13)
disp('You are a child')
elseif (age < 18)
disp ('You are a teenager')
elseif (age < 60)
disp ('You are an adult')
else
disp ('You are a senior adult')
end
Enter your age:
```

4.2 The "switch case otherwise end" condition

The following example explains how we can use this syntax. For instance, we would like to write a program that defines the stage of your life according to your age:

In Matlab, this condition evaluates an expression and chooses to execute one of several groups of statements. Accordingly, the syntax takes the following form:

Switch (expression d)

case (expression 1)

Perform task a1 to an

case (expression 2)

Perform task b1 to bn

.

.

.

case (expression k)

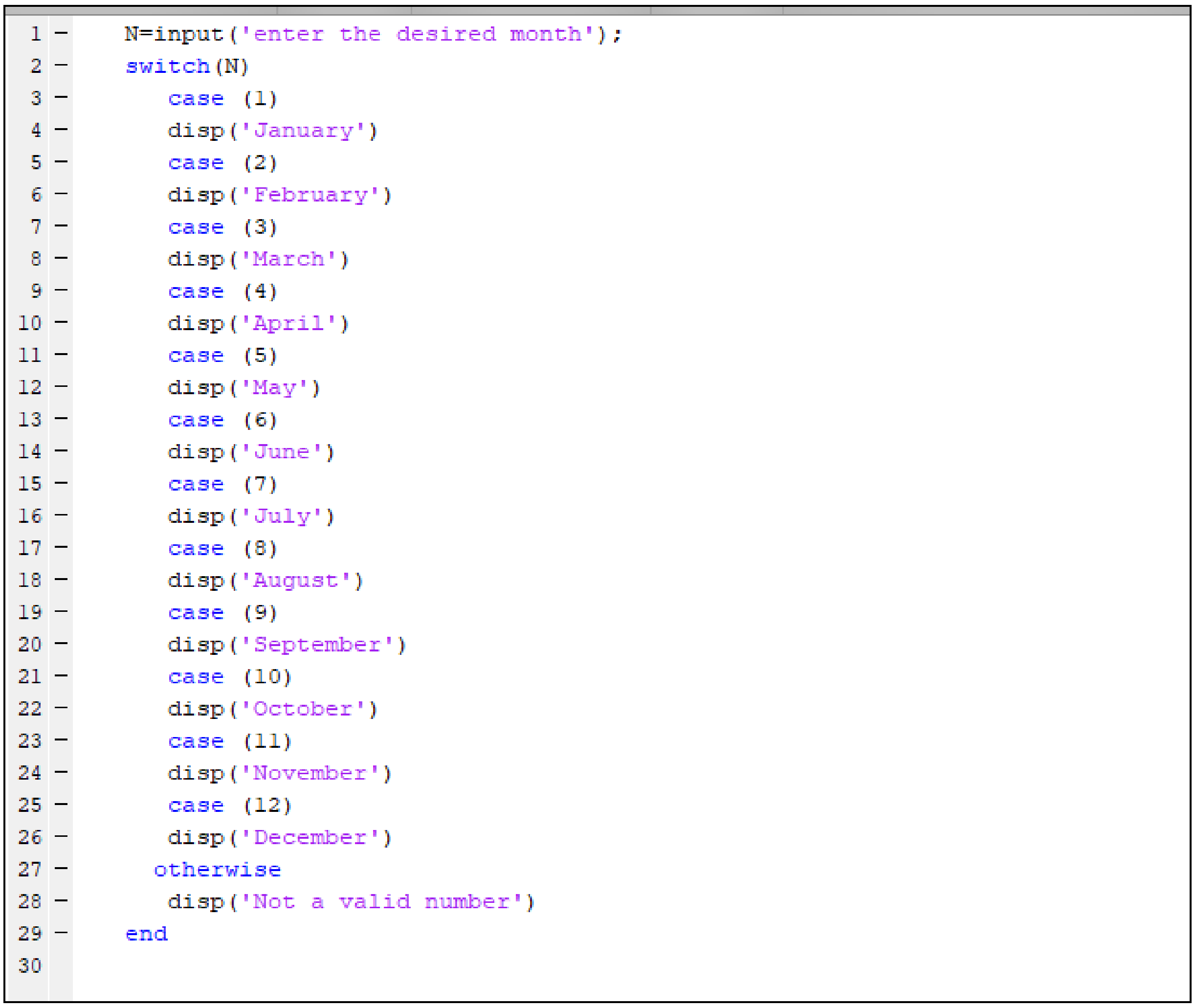
Perform task x1 to xn

otherwise

Perform task z1 to zn

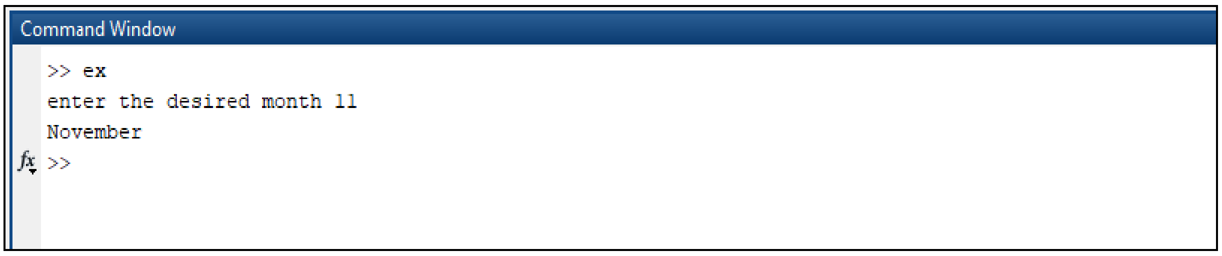
end

A really good example is by trying to figure out the month.



```
N=input('enter the desired month');
switch(N)
    case (1)
    disp('January')
    case (2)
    disp('February')
    case (3)
    disp('March')
    case (4)
    disp('April')
    case (5)
    disp('May')
    case (6)
    disp('June')
    case (7)
    disp('July')
    case (8)
    disp('August')
    case (9)
    disp('September')
    case (10)
    disp('October')
    case (11)
    disp('November')
    case (12)
    disp('December')
  otherwise
    disp('Not a valid number')
end
```

The output will be as shown:



```
>> ex
enter the desired month 11
November
>>
```

4.3 Loop Control Statements

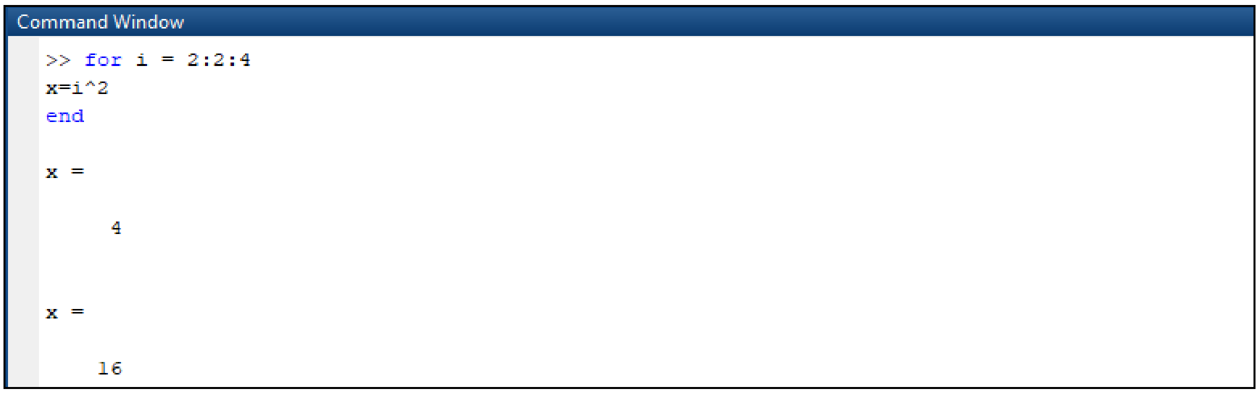
In Matlab, you may run a block of code repeatedly via using loop control commands. As known, a loop is a structure for repeating a calculation (or calculations) a predefined number of times. Accordingly, two different kinds of loops exist: the "**for**" and "**while**" loop.

*4.3.1 The "**for**" loop*

In Matlab, you may run a block of code repeatedly via using loop control commands. As known, a loop is a structure for repeating a calculation (or calculations) a predefined number of times. Accordingly, two different kinds of loops exist: the "**for**" and "**while**" loop. The syntax of "**for**" loop is as follows:

for variable = i:s:n
statement(s);
end

with **'i'** is the initial value, '**s**' the steps (incremental value) and '**n**' the final value. For example:



```
>> for i = 2:2:4
x=i^2
end

x =

     4


x =

    16
```

The bellow flowchart explains accurately the loop "for"

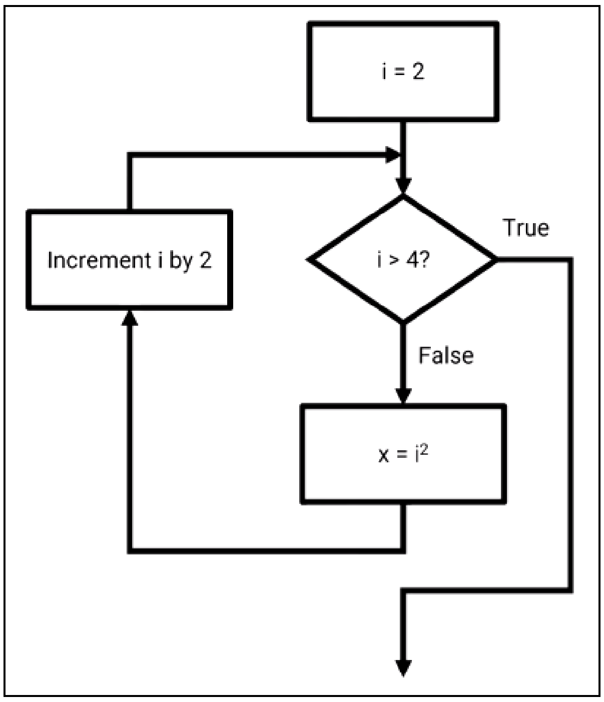


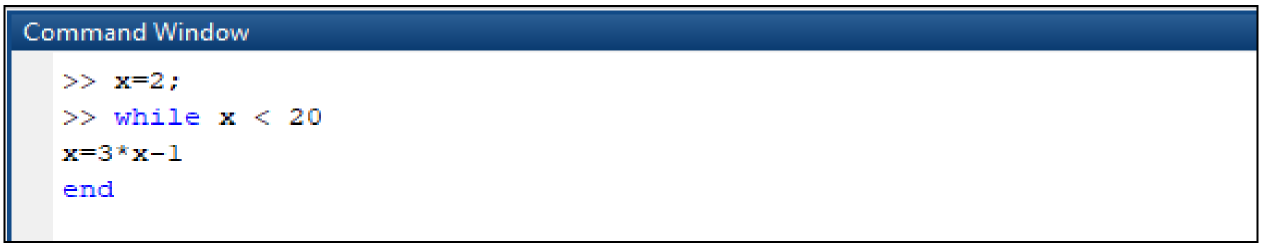
Figure 4.1 *for loop flowchart*

4.3.2 *The "**while**" loop*

By contrast, the '**while**' loop is used when the looping process terminates because a prescribed condition is met. Unlike in the for loop, the number of passes is not known in advance. The syntax of while loop is like so:

while logical expression
statement (s)
end

On the other hand, the loop must have a value before the while statement and the loop variable must be changed by the statements. The following example make us understand better.



```
>> x=2;
>> while x < 20
x=3*x-1
end
```

The while loop can be as illustartes the following flowchart:

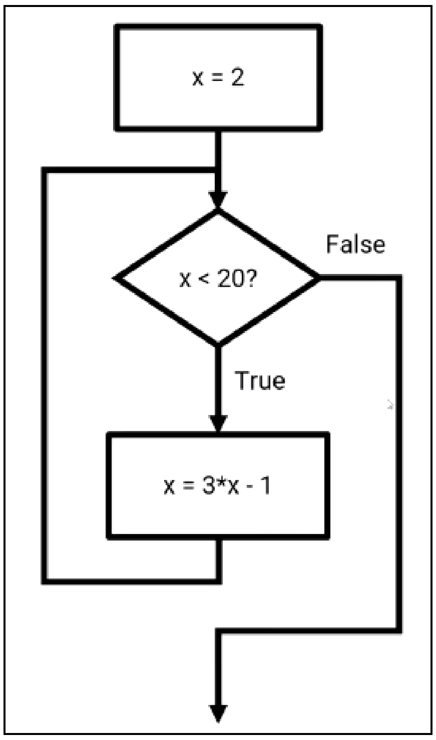
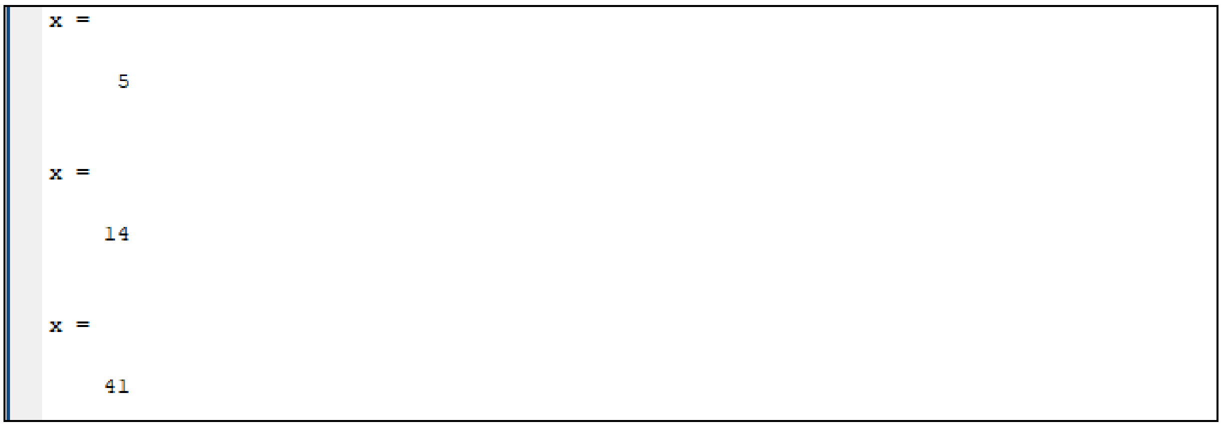


Figure 4.1 *while loop flowchart*

The output will be accordinagly:



```
x =

     5


x =

    14


x =

    41
```

Overall, the following table summirize the loops and conditional statements in Matlab.

Table 4.1. *Matlab Language Syntax*

Syntax	**Signification**
if, elseif, else	Execute statements if condition is true
switch, case, otherwise	Execute one of several groups of statements
for	for loop to repeat specified number of times
While	while loop to repeat when condition is true

Chapter 5 Matlab Toolbox

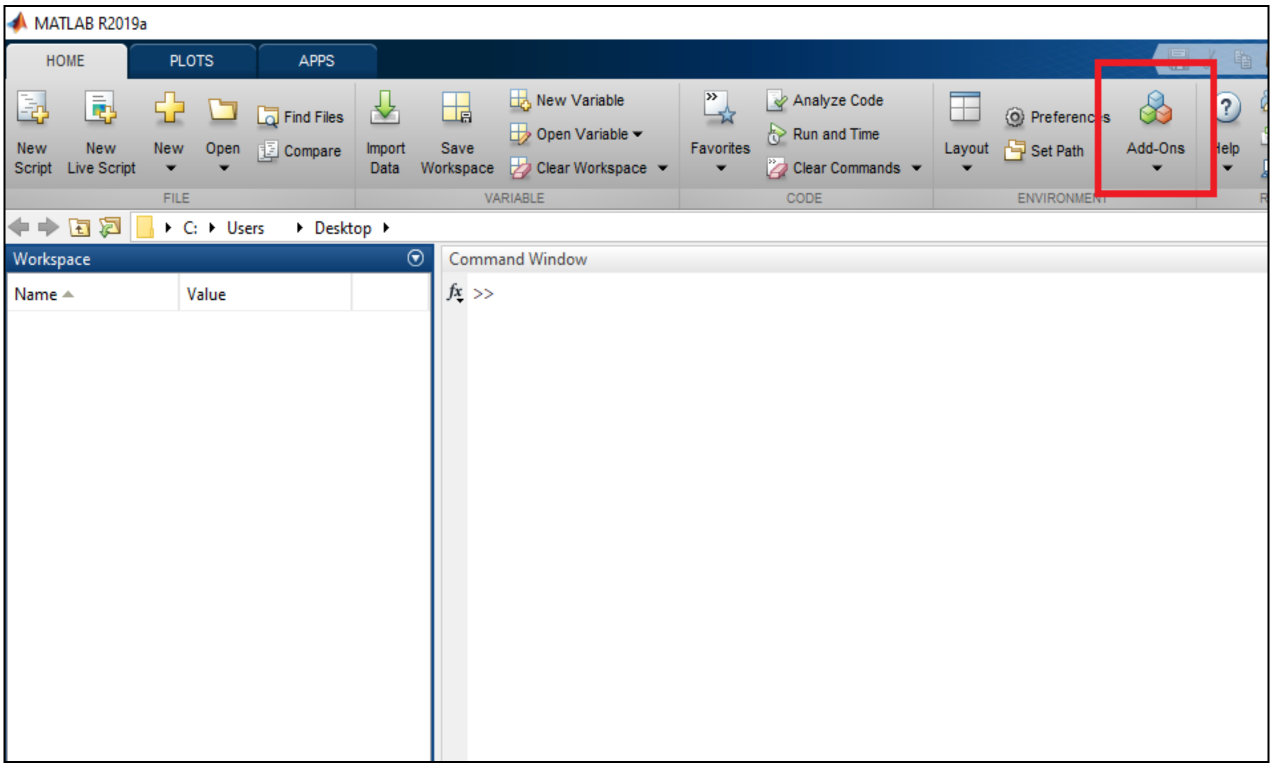
As known, Matlab has an interface that offers an interactive console-type window for executing commands, as well as, an integrated development environment (IDE) for programming applications.

Matlab finds its applications in many disciplines. It constitutes a powerful digital tool for the modeling of physical systems, the simulation of mathematical models, the design and validation (simulation tests and experimentation) of applications. The basic software can be supplemented by multiple toolboxes. We can cite for example: Curve Fitting Toolbox; Image Processing Toolbox; Data Acquisition Toolbox…etc [8].

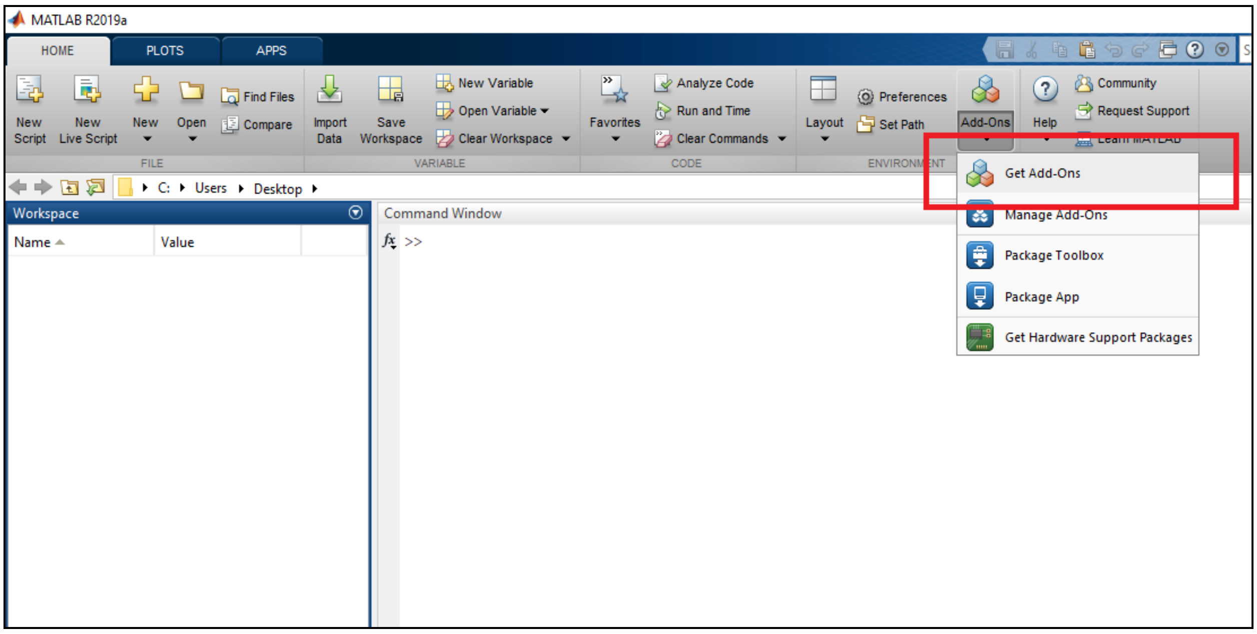
5.1 *Add additional Toolbox*

As talked above, Matlab has a limitless variety of Toolboxes. To add any desired toolbox, you can follow these steps:

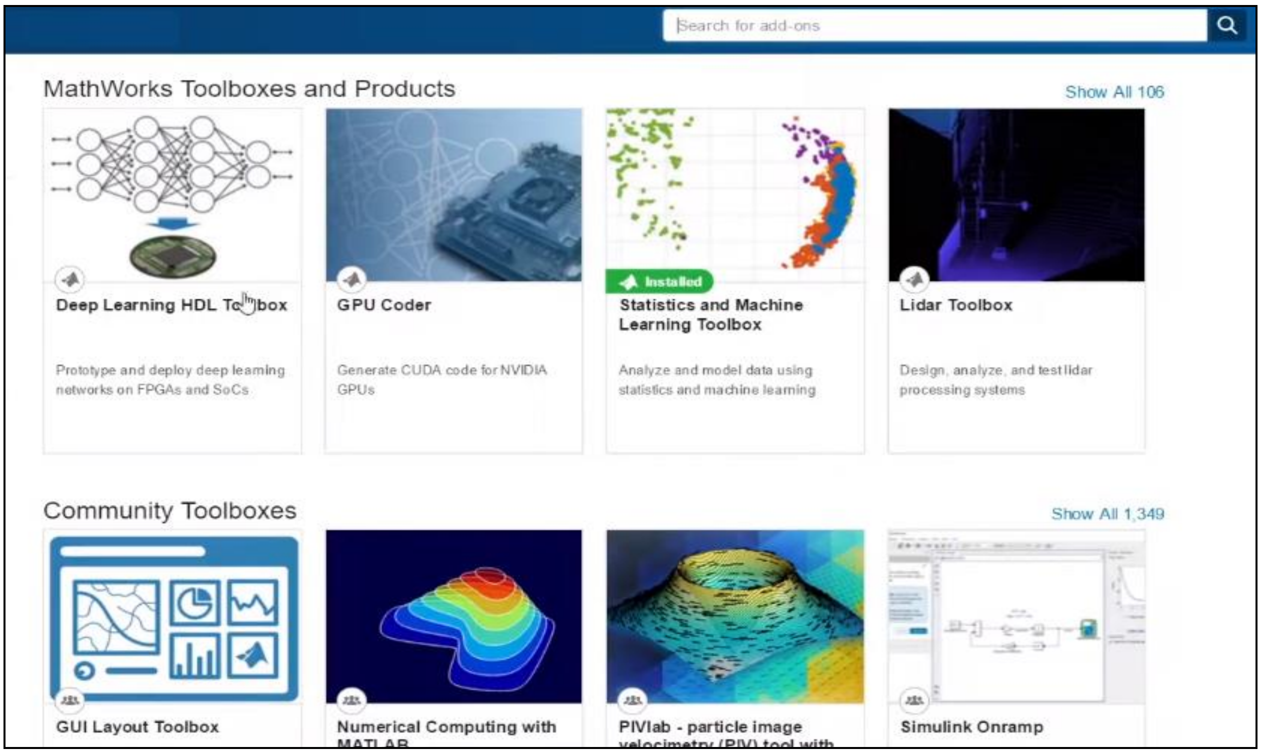
- Step 1: Click on Add-Ons



- Step 2: Get Add-Ons



A window will appear as shows in the figure bellow. By using the research tool above at the right, you can download any desired toolbox.

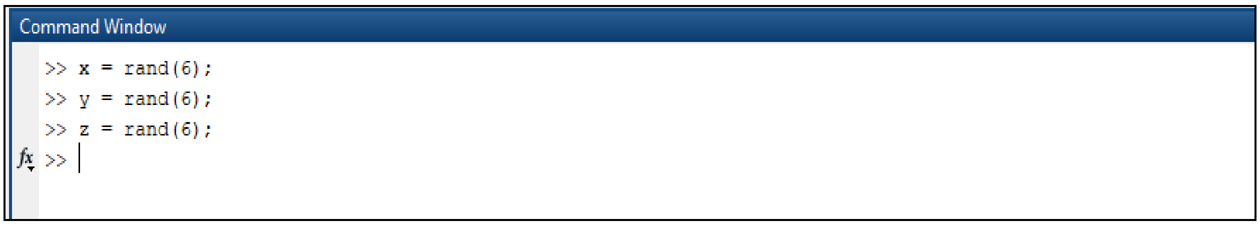

Search for add-ons
MathWorks Toolboxes and Products
Show All 106
Deep Learning HDL Toolbox
Prototype and deploy deep learning networks on FPGAs and SoCs
GPU Coder
Generate CUDA code for NVIDIA GPUs
Installed
Statistics and Machine Learning Toolbox
Analyze and model data using statistics and machine learning
Lidar Toolbox
Design, analyze, and test lidar processing systems
Community Toolboxes
Show All 1,349
GUI Layout Toolbox
Numerical Computing with MATLAB
PIVlab - particle image velocimetry (PIV) tool with
Simulink Onramp

5.2 Using of the “Curve Fitting Toolbox”

Among the powerful Toolbox that we will use in this course is “**Curve Fitting Toolbox**”. An application and features for fitting curves and surfaces to data are offered by Curve Fitting Toolbox. You can use the toolbox to perform exploratory data analysis, pre- and post-process data, remove outliers and compare candidate models. You can perform regression analysis using the given library of linear and nonlinear models, or you can create your own custom equations.

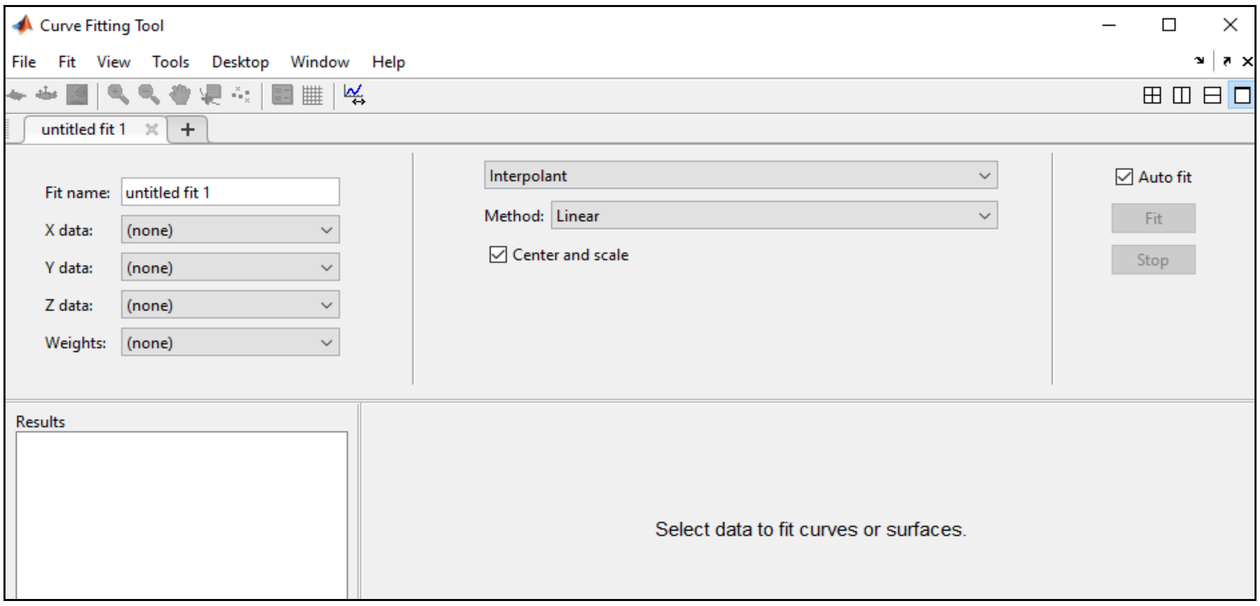
The library offers optimized solver parameters as well as starting conditions to enhance the quality of your fits. The toolbox supports as well non-parametric modeling methods, including splines, smoothing and interpolation. Now, let’s understand how to use this toolbox via the following steps:

- Firstly, we will create three random matrices, by using the rand function as seen:

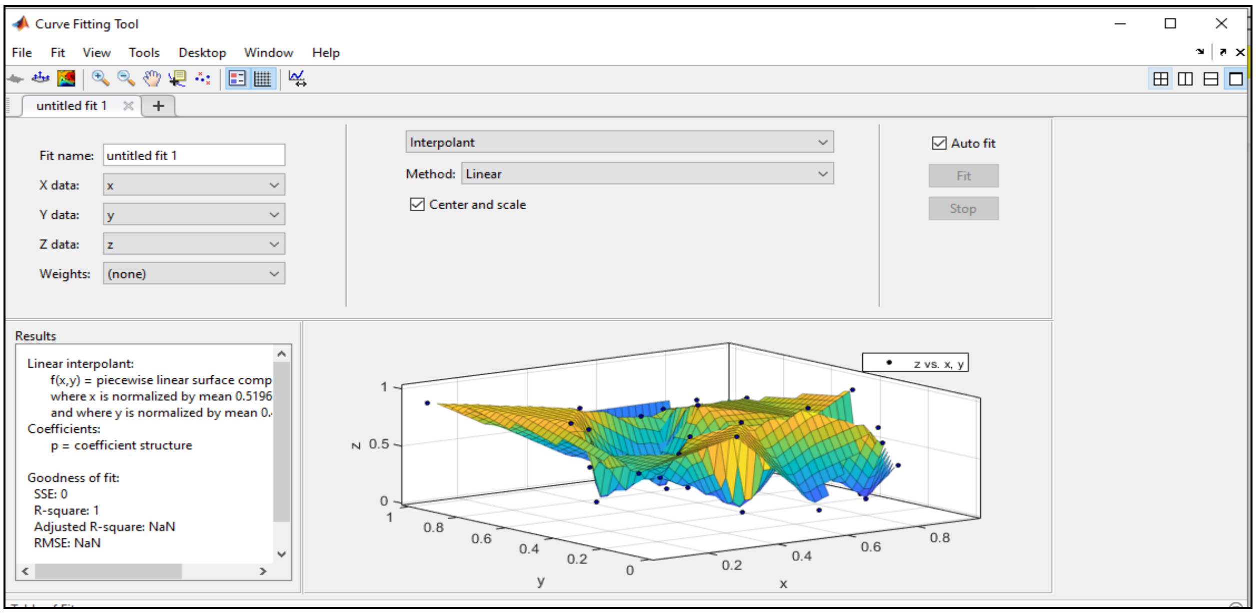


```
>> x = rand(6);
>> y = rand(6);
>> z = rand(6);
>>
```

- Secondly, on the Apps icon, we will select ‘Curve Fitting Tool’. A pop-up window will appear:



- Thirdly, we will set the ‘X, Y, and Z Data’ in this window to our inputs, ‘x, y and z’, respectively.


Curve Fitting Tool

Fit name: untitled fit 1
X data: x
Y data: y
Z data: z
Weights: (none)

Interpolant
Method: Linear
[✓] Center and scale
[✓] Auto fit

Results

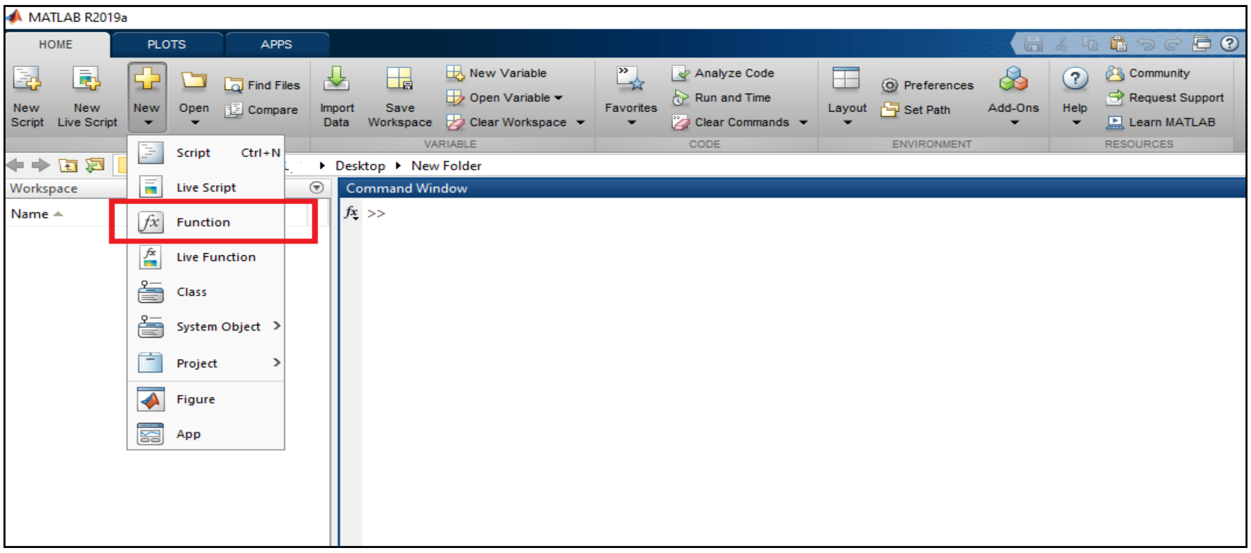
```
Linear interpolant:
    f(x,y) = piecewise linear surface comp
    where x is normalized by mean 0.5196
    and where y is normalized by mean 0.
Coefficients:
    p = coefficient structure

Goodness of fit:
  SSE: 0
  R-square: 1
  Adjusted R-square: NaN
  RMSE: NaN
```

Therefore, a curve be will created as seen in the figure 5.5. In the Result section, you can see the curve’s equation. Be aware, that by using the dropdown on the top of the curve, you can use a custom equation.

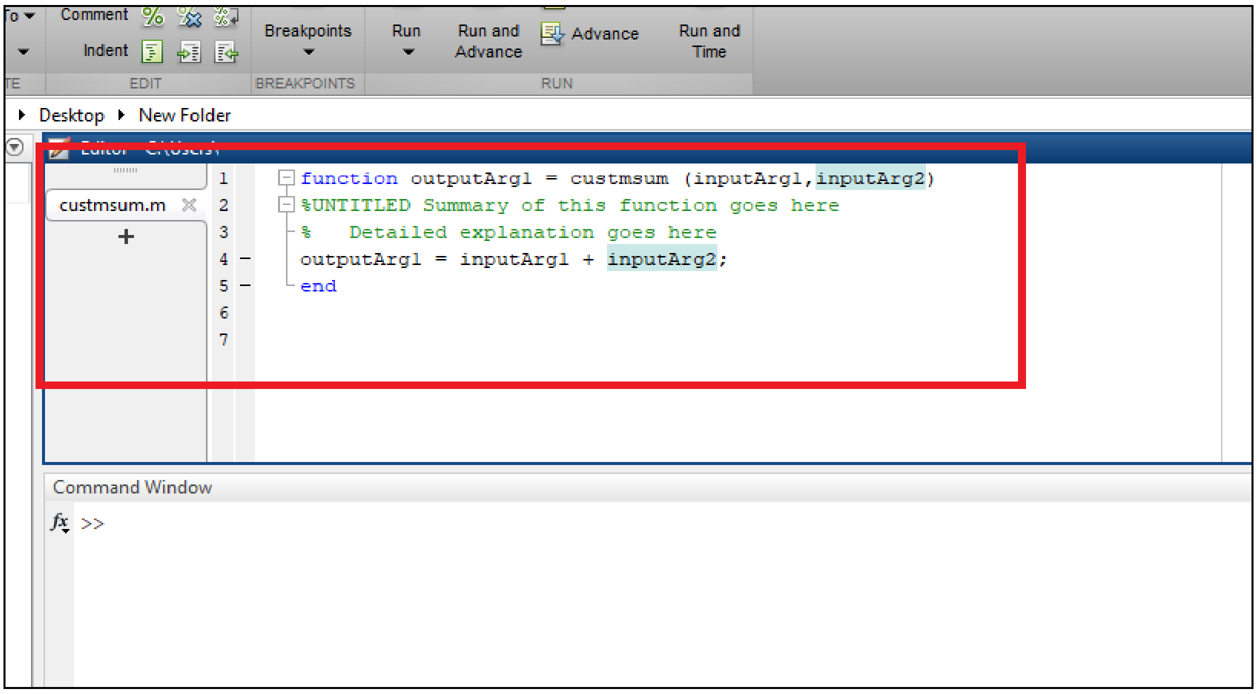
5.3 *Creating your own Toolbox*

Matlab allows us to create our own customed Toolbox environment. The following example will show us how do this. Firstly, we would like to create two Toolbox that will do the sum and the multiplication between given two numbers, which will be named “**custmsum**” and “**custmprod**”, respectively. Thereafter, we will create these two functions in our device as seen:

- **Step 1**: Click on New, then Functions:



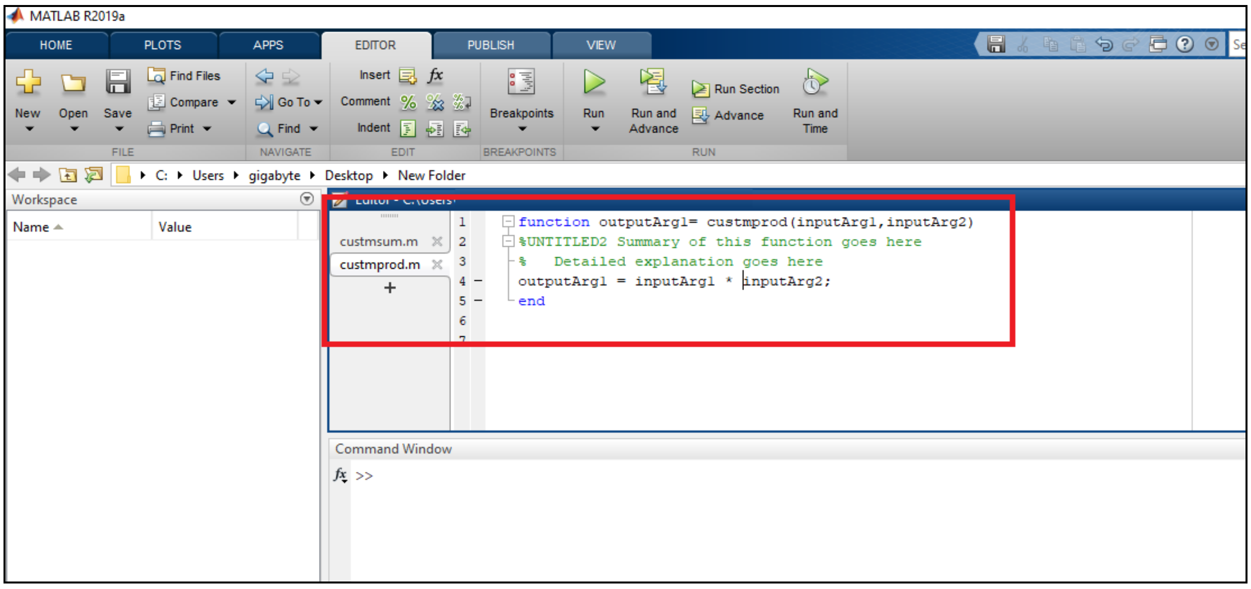
- **Step 2:** Do the required changes as it can be seen here:



```
function outputArg1 = custmsum (inputArg1,inputArg2)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1 + inputArg2;
end
```

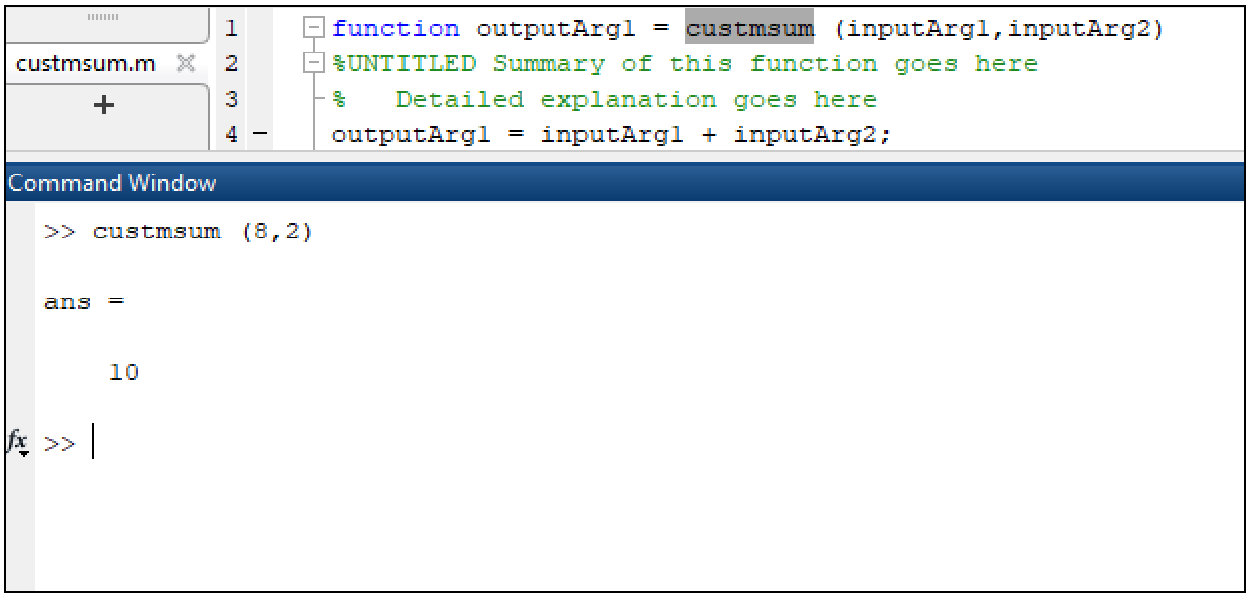
Thereafter, save the file in a specified folder.

Repeat the same steps to create the "**custmprod**" and do not forget to modify as seen previously. The required changes are highlighted in red as seen:



```
function outputArg1= custmprod(inputArg1,inputArg2)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1 * inputArg2;
end
```

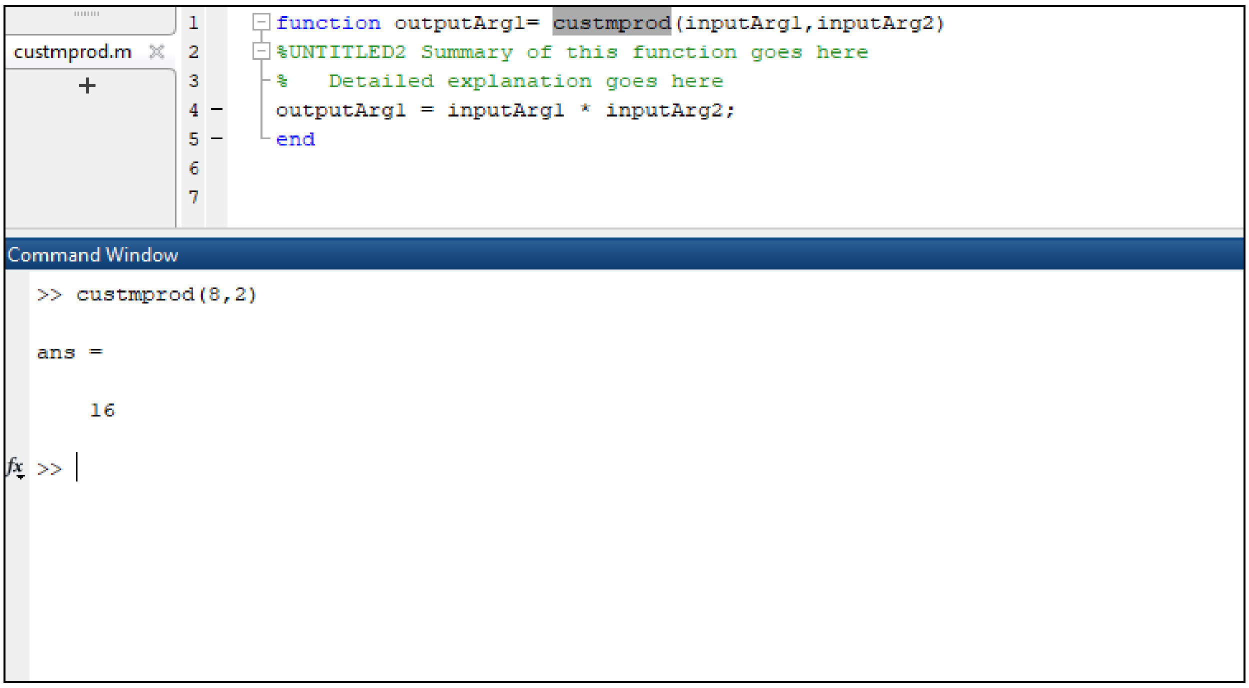
Now, let’s check if our created functions work accurately. To do this let’s do some sample operations and see the results in the “**Command Window**”:


custmsum.m

```
function outputArg1 = custmsum (inputArg1,inputArg2)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1 + inputArg2;
```

Command Window

```
>> custmsum (8,2)

ans =

    10

>> 
```

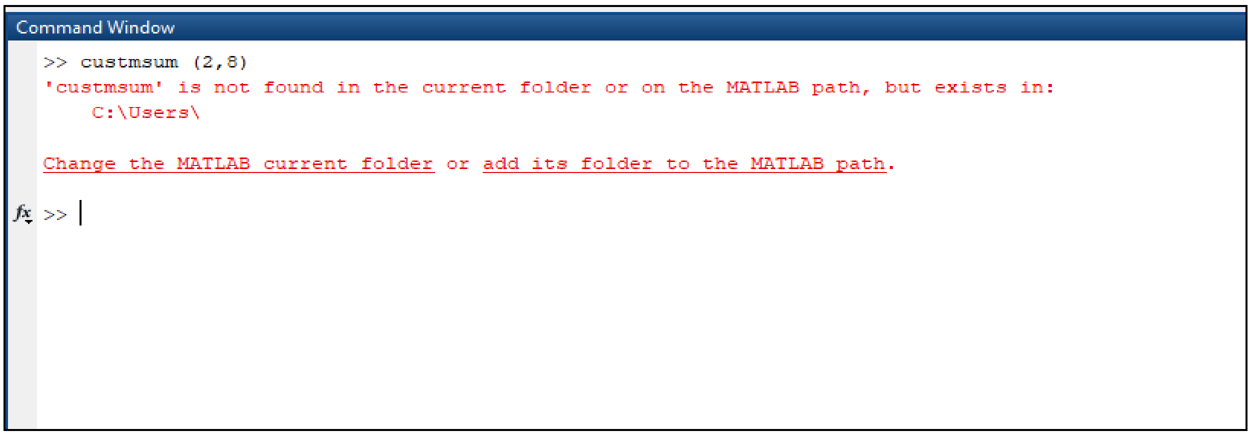

custmprod.m

```
function outputArg1= custmprod(inputArg1,inputArg2)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1 * inputArg2;
end
```

Command Window

```
>> custmprod(8,2)

ans =

    16

>> 
```

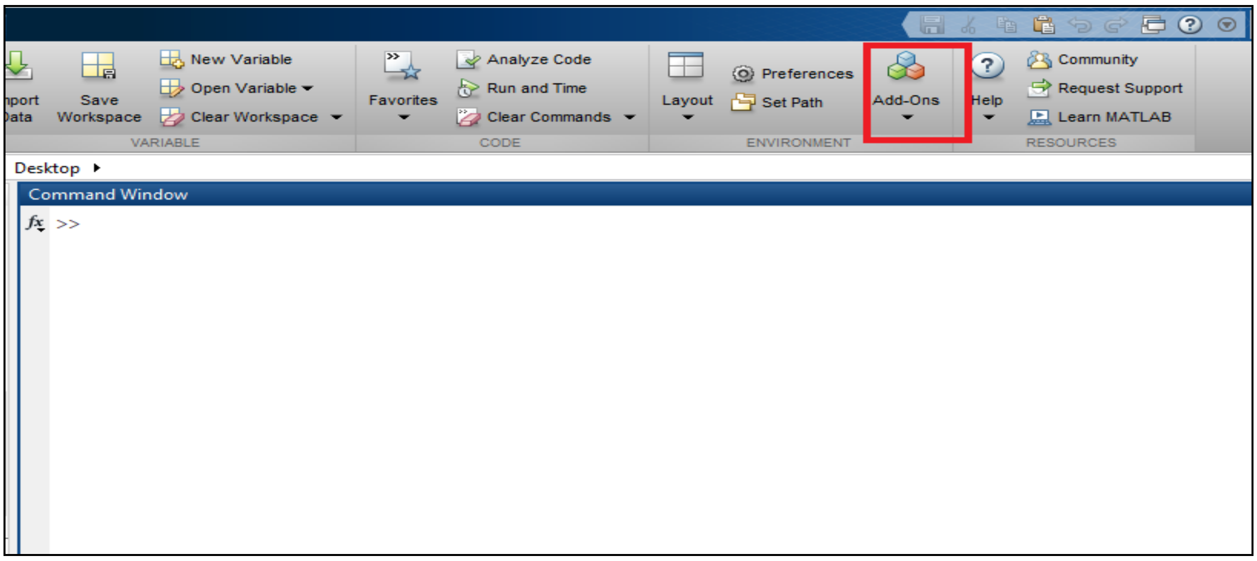
The two functions work accurately whenever I write the functions path and I will get the output in my Command Window. However, if I will change the folder, the operations will no longer work and an error message will be displayed accordingly:



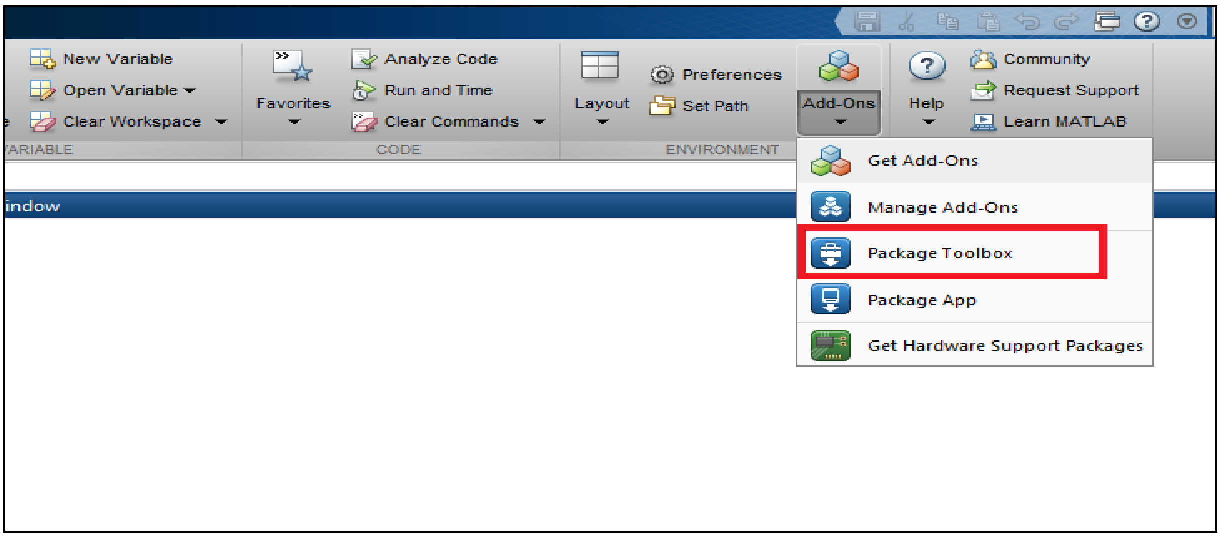
```
>> custmsum (2,8)
'custmsum' is not found in the current folder or on the MATLAB path, but exists in:
    C:\Users\

Change the MATLAB current folder or add its folder to the MATLAB path.

>>
```

Now, let’s create our customed Toolbox, by following these steps:

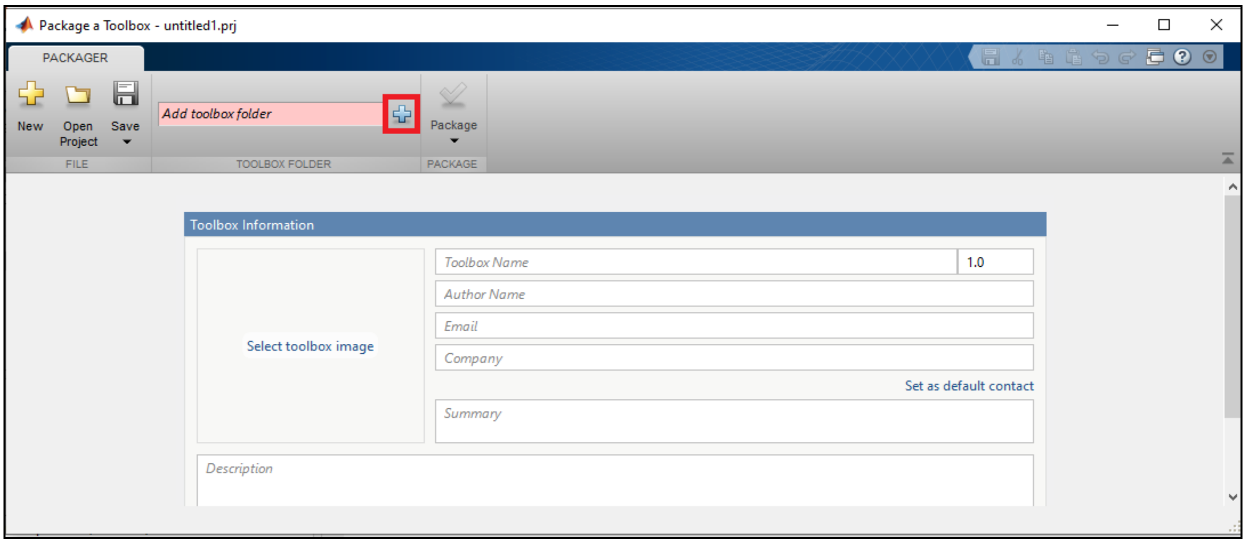
- **Step 1:**



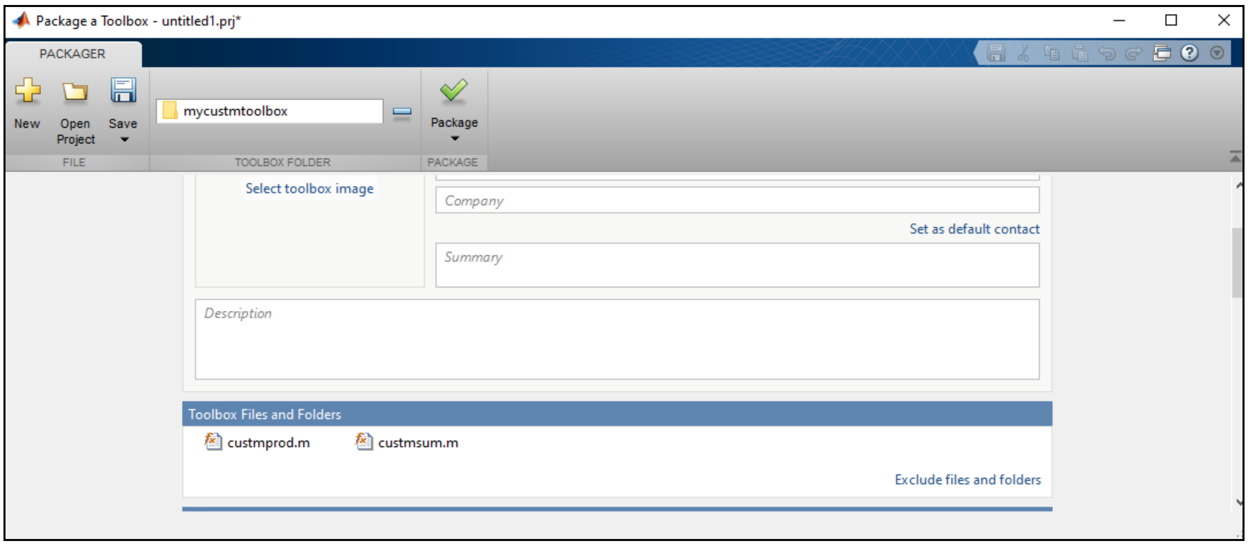
- **Step 2:**



- **Step 3:**



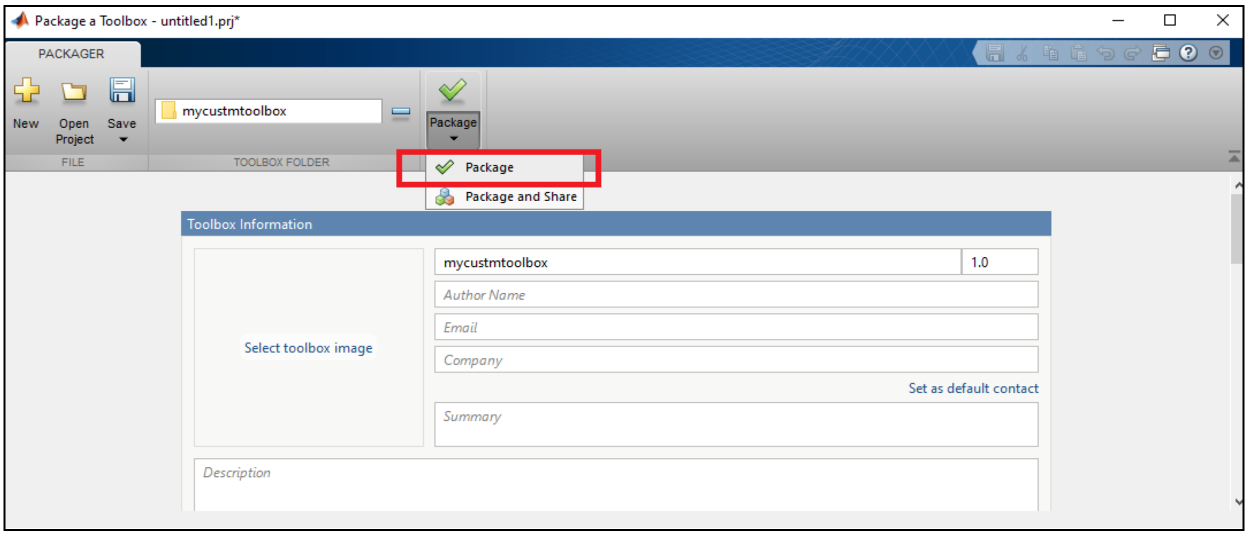
- **Step 4:**



mycustmtoolbox

custmprod.m   custmsum.m

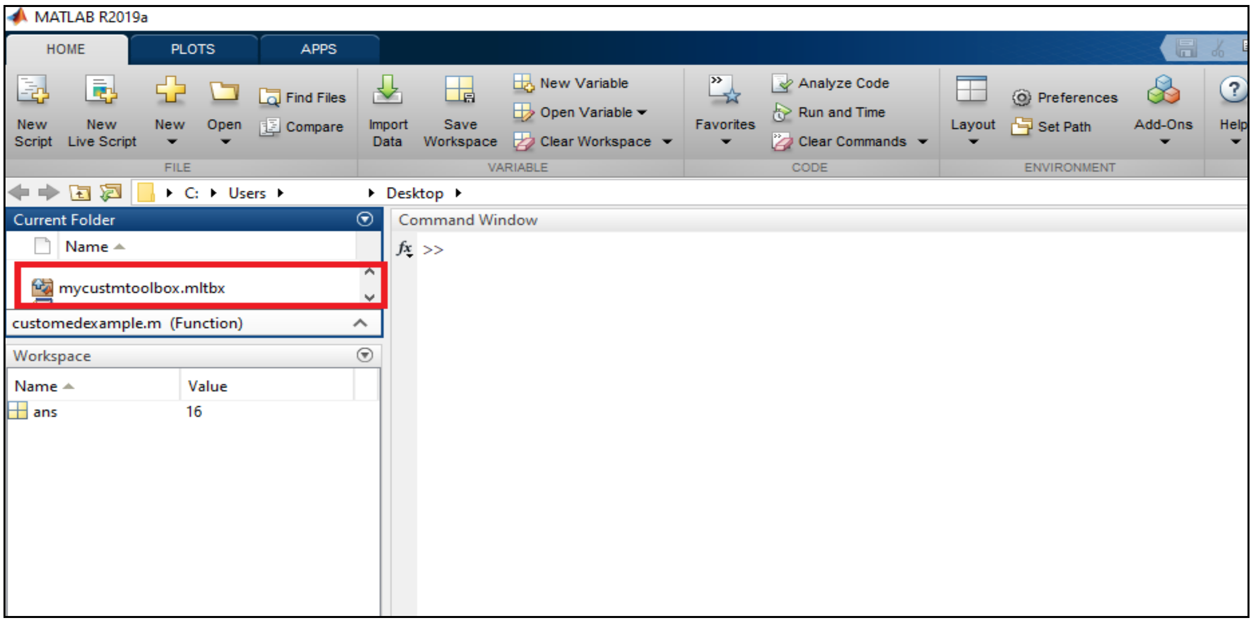
- **Step 5:**



mycustmtoolbox

mycustmtoolbox   1.0

After this we will save it. If we check on Matlab it will appear with an extension "**.mltbx**" as seen in the following figure:

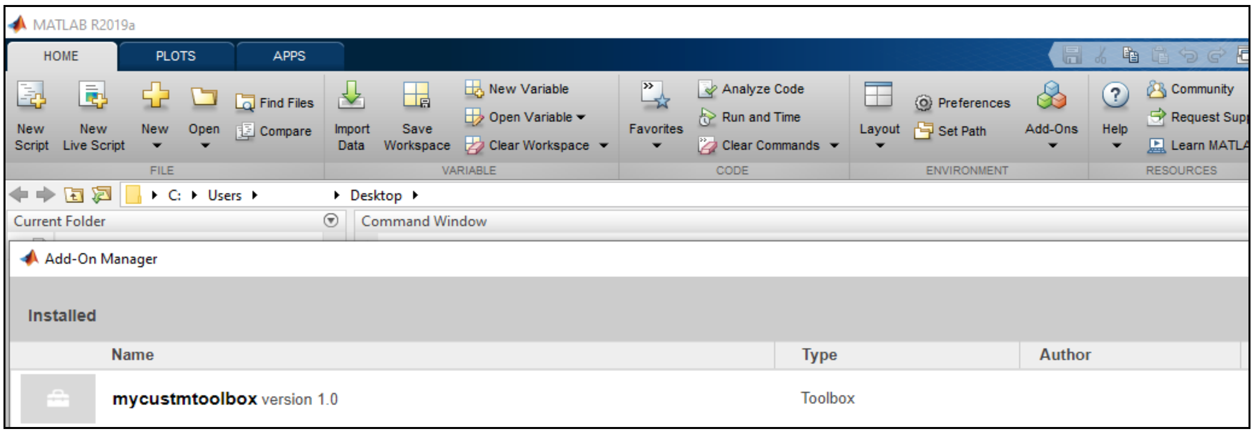


```
mycustmtoolbox.mltbx
customedexample.m (Function)
```

Name	Value
ans	16

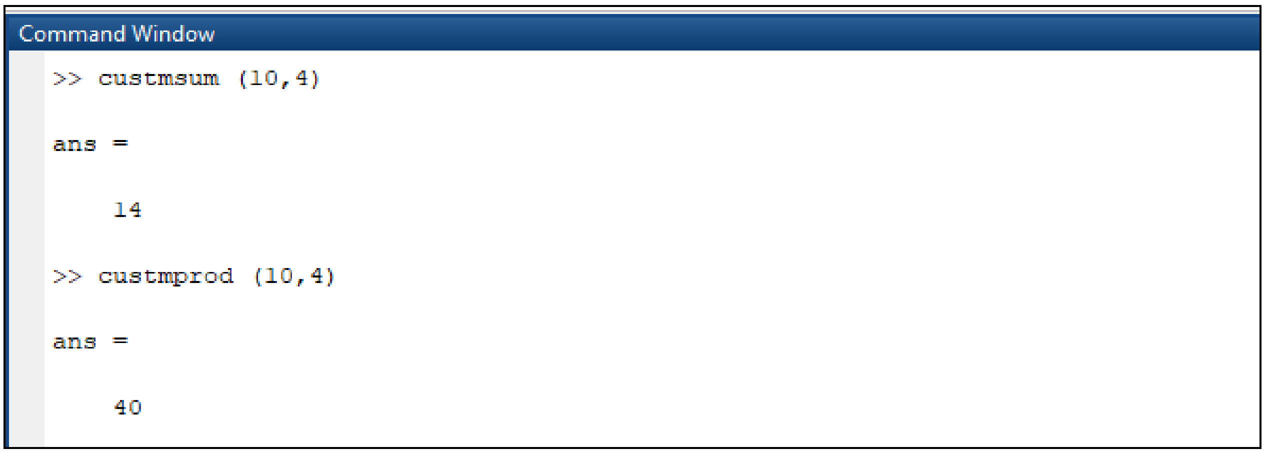
```
fx >>
```

After this, double click to install it. To double check, you can click on the Add-Ons icon, then Manage Add-Ons icon. It will show us that it has been successfully installed.



Name	Type	Author
mycustmtoolbox version 1.0	Toolbox	

Now, we can easily do the **sum** or **product** of any given numbers (even if we will change the folder) as seen:



```
>> custmsum (10,4)

ans =

    14

>> custmprod (10,4)

ans =

    40
```

References

[1] T. SALAHUDDIN, Numerical Techniques in MATLAB: Fundamental to Advanced Concepts. CRC Press, 2023

[2] Y. A. SHARDT, Using MATLAB to Solve Statistical Problems: A Practical Guide to the Book “Statistics for Chemical and Process Engineers”. Springer Nature, 2023

[3] S. LOCKHART, T. ERIC, An Engineer's Introduction to Programming with MATLAB 2019, Sdc Publications, 2019.

[4] B.R. HUNT, R. L. LIPSMAN, J. M. ROSENBERG, A guide to MATLAB: for beginners and experienced users Cambridge university press, 2014.

[5] D. HOUCQUE, Introduction to Matlab for engineering students, Northwestern University, 2005.

[6] Z. MANSOURI, Cours Matlab, Université de Skikda: 20 Aout 1955, 2014

[7] J. CHAKRAVORTY, Introduction to MATLAB Programming, Toolbox and Simulink, India: Universities Press Pvt. Ltd, 2021

[8] MATHWORKS INC, Matlab: The Language of Technical Computing, Accessed: January November, 2023. Available: https://www.mathworks.com/help/matlab/index.html?s_tid=hc_panel