Apress® | MATLAB SOLUTIONS SERIES

César Pérez López

MATLAB Mathematical Analysis

HANDS-ON MATLAB TRAINING AND EXERCISES

Springer

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

MATLAB is a platform for scientific computing that can work in almost all areas of the experimental sciences, engineering, mathematics and financial solutions. Logically, this software allows you to work in the field of mathematical analysis through a wide variety of commands and functions. *MATLAB Mathematical Analysis* is a reference to these commands and brief discussions of how to solve a very wide array of problems. It is not meant to be a tutorial on MATLAB, but a user of MATLAB should be able to follow the book easily and look up various types of problems or commands and see quick examples of how they work. This book covers a wide array of content and so it is a great place to find new features, topics or commands that you might not have known existed.

The book begins by introducing the reader to the use of numbers of all types and bases, operators, variables and functions in the MATLAB environment. Then it delves into working with complex variables. A large section is devoted to working with developing graphical representations of curves, volumes, and surfaces. MATLAB functions allow working with two-dimensional and three-dimensional graphics, statistical graphs, curves and surfaces in explicit, implicit, parametric and polar coordinates. Additional work implements twisted curves, surfaces, meshes, contours, volumes and graphical interpolation.

Vectors and matrices are a major feature of MATLAB and so they are treated throughout the book as the topics come up and include applications in many of the areas.

The following content block develops computation of limits and functions, continuity and numerical and power series. Then differentiability is addressed in one and several variables including differential theorems for vector fields. It continues to address integration in one or more variable, and multiple integrals and their applications. Finally differential equations are treated and some of their applications.

MATLAB Mathematical Analysis offers a broad resource for looking up commands. Unlike most references, it can be read from start to finish and jumping into chapters to look up is not an issue. So one of the aspects of the book is its flexibility and the other main benefit is the breadth of coverage.

CHAPTER 1

MATLAB Introduction and Working Environment

Introduction to Working with MATLAB

Whenever you use a program of this type, it is necessary to know the general characteristics of how the program interprets notation. This chapter will introduce you to some of the practices used in the MATLAB environment. If you are familiar with using MATLAB, you may wish to skip over this chapter.

The best way to learn MATLAB is to use the program. Each example in the book consists of the header with the prompt for user input ">>" and the response from MATLAB appears on the next line. See Figure 1-1.

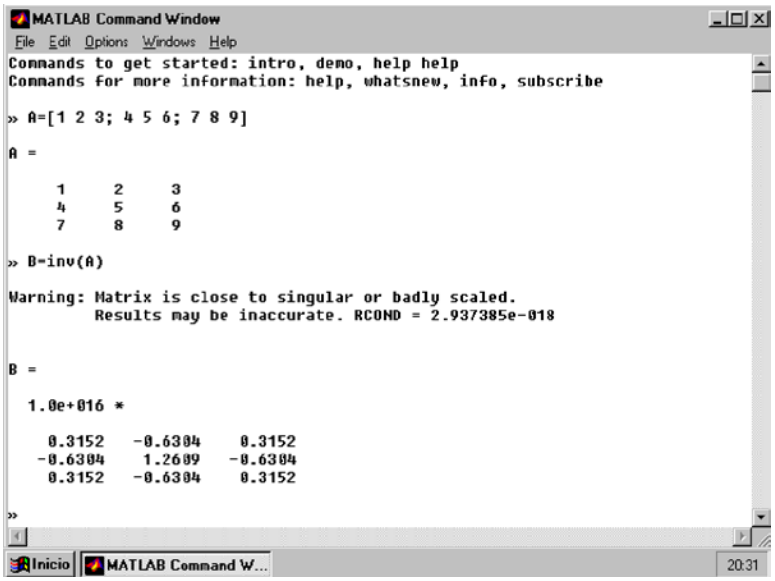

Figure 1-1.

At other times, depending on the type of user input used, MATLAB returns the response using the expression **"ans ="**. See Figure 1-2.

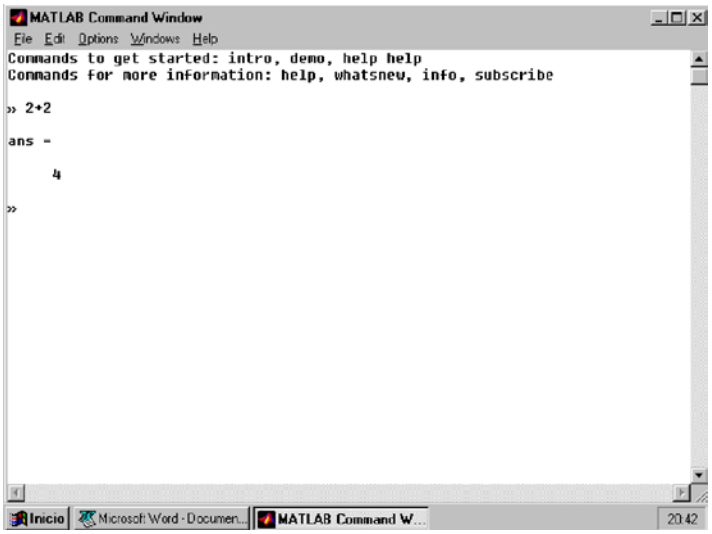

Figure 1-2.

In a program like MATLAB it is always necessary to pay attention to the difference between uppercase and lowercase letters, types of parentheses or square brackets, the amount of spaces used in your input and your choice of punctuation (e.g., commas, semicolons). Different uses will produce different results. We will go over the general rules now.

Any input to run MATLAB should be written to the right of the header (the prompt for user *input* ">>"). You then obtain the response in the lines immediately below the input.

When an input is proposed to MATLAB in the command window that does not cite a variable to collect the result, MATLAB returns the response using the expression ***ans***=.

If at the end of the input we put a semicolon, the program runs the calculation(s) and keeps them in memory (*Workspace*), but does not display the result on the screen (see the first input in Figure 1-3 as an example).

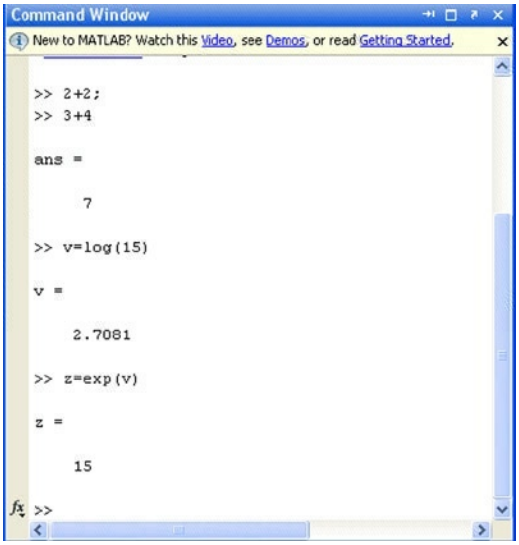

Figure 1-3.

The *input* prompt appears ">> " to indicate that you can enter a new entry.

If you enter and define a variable that can be calculated, you can then use that variable as the argument for subsequent entries. This is the case for the variable *v* in Figure 1-3, which is subsequently used as an input in the exponential function.

Like the C programming language, MATLAB is sensitive to the difference between uppercase and lowercase letters; for example, "*Sin (x)*" is different from *"sin (x)"*. The names of all built-in functions begin with lowercase. There should be no spaces in the names of symbols of more than one letter or in the names of functions. In other cases, spaces are simply ignored. They can be used in some cases to make *input* more readable. Multiple entries in the same line of command, separated by commas, can also be used. In this case, press *Enter* at the end of the last entry (see Figure 1-4). If you use a semicolon at the end of one of the entries of the line, as we stated before, its corresponding output is ignored.

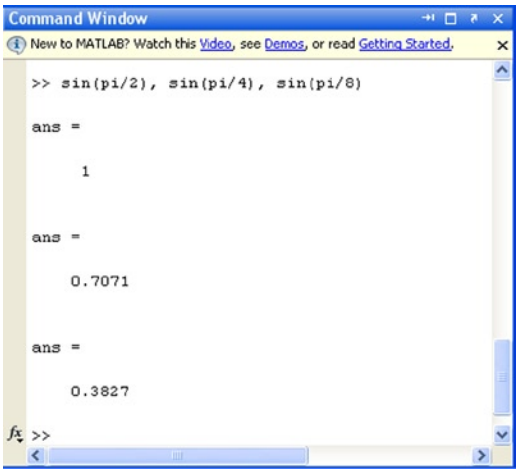

Figure 1-4.

It is possible to enter descriptive comments in a command input line by starting the comment with the "%" sign. When you run the input, MATLAB ignores the rest of the line to the right of the % sign:

```
>> L = log (123) % L is a natural logarithm

L =

        4.8122
>>
```

To simplify the introduction of a script to be evaluated by the MATLAB interpreter (via the command window), you can use the arrow computer keys. For example, if you press the up arrow once, MATLAB will recover the last entry submitted in MATLAB. If you press the up arrow key twice, MATLAB recovers the prior entry submitted, and so on. This can save you the headache of re-entering complicated formulae.

Similarly, if you type a sequence of characters in the *input* area and then click the up arrow, MATLAB recovers the last entry that begins with the specified string.

Commands entered during a MATLAB session are temporarily stored in the buffer (*Workspace*) until you end the session with the program, at which time they can be permanently stored in a file or you lose them permanently.

Below is a summary of the keys that can be used in the *input* area of MATLAB (command line), as well as their functions:

Up arrow (Ctrl-P)	Retrieves the previous line.
Arrow down (Ctrl-N)	Retrieves the following entry.
Arrow to the left (Ctrl-B)	Takes the cursor to the left, a character.
Arrow to the right (Ctrl-F)	Takes the cursor to the right, a character.
CTRL-arrow to the left	Takes the cursor to the left, a word.
CTRL-arrow to the right	Takes the cursor to the right, a word.
Home (Ctrl-A)	Takes the cursor to the beginning of the line.
End (Ctrl-E)	Takes the cursor at the end of the current line.
Exhaust	Clears the command line.
Delete (Ctrl-D)	Deletes the character indicated by the cursor.
BACKSPACE	Deletes the character to the left of the cursor.
CTRL-K	Deletes all of the current line.

The command ***clc*** clears the command window, but does not delete the content of the Workspace (that content remains in memory).

Numerical Calculations with MATLAB

You can use MATLAB as a powerful numerical calculator. Most calculators handle numbers only with a preset degree of precision, however MATLAB performs exact calculations with the necessary precision. In addition, unlike calculators, we can perform operations not only with individual numbers, but also with objects such as arrays.

Most of the themes in classical numerical calculations, are treated in this software. It supports matrix calculations, statistics, interpolation, fit by least squares, numerical integration, minimization of functions, linear programming, numerical algebraic and resolution of differential equations and a long list of processes of numerical analysis that we'll see later in this book.

Here are some examples of numerical calculations with MATLAB. (As we all know, for results it is necessary to press Enter once you have written the corresponding expressions next to the prompt ">>")

We can simply calculate 4 + 3 and get as a result 7. To do this, just type 4 + 3, and then Enter.

```
>> 4 + 3

Ans =

7
```

Also we can get the value of 3 to the 100th power, without having previously set the level of precision. For this purpose press 3 ^ 100.

```
>> 3 ^ 100

Ans =

5. 1538e + 047
```

You can use the command "format long e" to pass the result of the operation with 16 digits before the exponent (scientific notation).

```
>> format long e

>> 3 ^ 100

ans =

5.153775207320115e+047
```

We can also work with complex numbers. We will get the result of the operation (2 + 3i) raised to the 10th power, by typing the expression (2 + 3i) ^ 10.

```
>> (2 + 3i) ^ 10

Ans =

-1. 415249999999998e + 005 - 1. 456680000000000e + 005i
```

The previous result is also available in short format, using the "format short" command.

```
>> format short
>> (2 + 3i) ^ 10

ans =

-1.4152e+005- 1.4567e+005i
```

Also we can calculate the value of the Bessel function found for 11.5. To do this type Besselj (0,11.5).

```
>> besselj(0, 11.5)

ans =

   -0.0677
```

Symbolic Calculations with MATLAB

MATLAB handles symbolic mathematical computation, manipulating formulae and algebraic expressions easily and quickly and can perform most operations with them. You can expand, factor and simplify polynomials, rational and trigonometric expressions, you can find algebraic solutions of polynomial equations and systems of equations, can evaluate derivatives and integrals symbolically, and find function solutions for differential equations, you can manipulate powers, limits and many other facets of algebraic mathematical series.

To perform this task, MATLAB requires all the variables (or algebraic expressions) to be written between single quotes to distinguish them from numerical solutions. When MATLAB receives a variable or expression in quotes, it is considered symbolic.

Here are some examples of symbolic computation with MATLAB.

1. Raise the following algebraic expression to the third power: (x + 1) (x+2)-(x+2) ^ 2. This is done by typing the following expression: expand ('((x + 1) (x+2) - (x+2) ^ 2) ^ 3'). The result will be another algebraic expression:

   ```
   >> syms x; expand (((x + 1) *(x + 2)-(x + 2) ^ 2) ^ 3)

   Ans =

   -x ^ 3-6 * x ^ 2-12 * x-8
   ```

   Note in this example, the *syms x* which is needed to initiate the variable *x*.
   You can then factor the result of the calculation in the example above by typing: factor ('((x + 1) *(x + 2)-(x + 2) ^ 2) ^ 3')

   ```
   >> syms x; factor(((x + 1)*(x + 2)-(x + 2)^2)^3)

   ans =

   -(x+2)^3
   ```

2. You can resolve the indefinite integral of the function (x ^ 2) sine (x) ^ 2 by typing: int ('x ^ 2 * sin (x) ^ 2', 'x')

   ```
   >> int('x^2*sin(x)^2', 'x')

   ans =

   x ^ 2 * (-1/2 * cos (x) * sin (x) + 1/2 * x)-1/2 * x * cos (x) ^ 2 + 1/4 *
   cos (x) * sin (x) + 1/4 * 1/x-3 * x ^ 3
   ```

You can simplify the previous result:

```
>> syms x; simplify(int(x^2*sin(x)^2, x))

ans =

sin(2*x)/8 - (x*cos(2*x))/4 - (x^2*sin(2*x))/4 + x^3/6
```

You can express the previous result with more elegant mathematical notation:

```
>> syms x; pretty(simplify(int(x^2*sin(x)^2, x)))

                      2             3
  sin(2 x)   x cos(2 x)   x  sin(2 x)   x
 -------- - ---------- - ----------- + --
    8           4             4        6
```

3. We can solve the equation 3ax-7 x ^ 2 + x ^ 3 = 0 (where a, is a parameter):

```
>> solve('3*a*x-7*x^2 + x^3 = 0', 'x')

ans =

[                       0]
[7/2+1/2 *(49-12*a)^(1/2)]
[7/2-1/2 *(49-12*a)^(1/2)]
```

4. We can find the five solutions of the equation x ^ 5 + 2 x + 1 = 0:

```
ans =

                                          -0.48638903593454300001655725369801
  0.94506808682313338631496614476119 + 0.85451751443904587692179191887616*i
  0.94506808682313338631496614476119 - 0.85451751443904587692179191887616*i
- 0.70187356885586188630668751791218 - 0.87969719792982402287026727381769*i
- 0.70187356885586188630668751791218 + 0.87969719792982402287026727381769*i
```

On the other hand, MATLAB may use the libraries of the Maple V program to work with symbolic math and can thus extend its field of application. In this way, you can use MATLAB to work on problems in areas including differential forms, Euclidean geometry, projective geometry, statistics, etc.

At the same time, you also can expand your options for numerical calculations using the libraries from MATLAB and libraries of Maple (combinatorics, optimization, theory of numbers, etc.).

Graphics with MATLAB

MATLAB produces graphs of two and three dimensions, as well as outlines and graphs of density. You can represent both the graphics and list the data in MATLAB. MATLAB allows you to control colors, shading and other graphics features, and also supports animated graphics. Graphics produced by MATLAB are portable to other programs.

Here are some examples of MATLAB graphics:

1. You can represent the function xsine (x) for x ranging between $-\pi/4$ and $\pi/4$, using 300 equidistant points for the intervals. See Figure 1-5.

```
>> x = linspace(-pi/4,pi/4,300);
>> y = x.*sin(1./x);
>> plot(x,y)
```

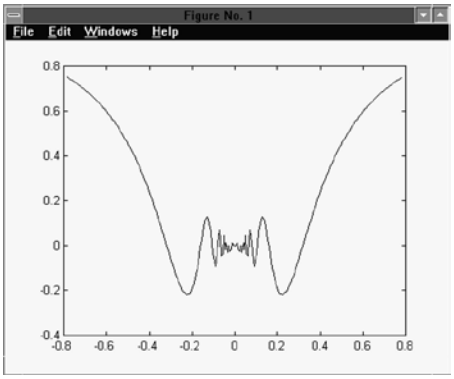

Figure 1-5.

Note the dots (periods) in this example in the second line. These are not decimal points and should not be confused as such. These indicate that if you are working with vectors or matrices that the following operator needs to be applied to every element of the vector or matrix. For an easy example, let's use *x .*3*. If x is a matrix, then all elements of x should be multiplied by 3. Again, be careful to avoid confusion with decimal points. It is sometimes wise to represent .6 as 0.6, for instance.

2. You can add the options frame and grille to the graph above, as well as create your own chart title and labels for axes. See Figure 1-6.

```
>> x = linspace(-pi/4,pi/4,300);
>> y = x.*sin(1./x);
>> plot(x,y);
>> grid;
>> xlabel('Independent variable X');
>> ylabel ('Independent variable Y');
>> title ('The functions y=sin(1/x)')
```

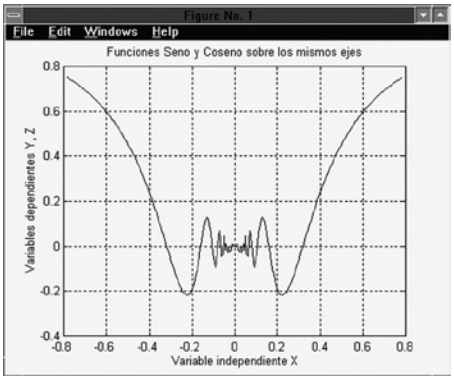

Figure 1-6.

3. You can generate a graph of the surface for the function z = sine (sqrt(x^2+y^2)) / sqrt(x^2+y^2), where x and y vary over the interval (- 7.5, 7.5), taking equally spaced points in 5 tenths. See Figure 1-7.

```
>> x =-7.5: .5:7.5;
>> y = x;
>> [X, Y] = meshgrid(x,y);
>> Z = sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
>> surf (X, Y, Z)
```

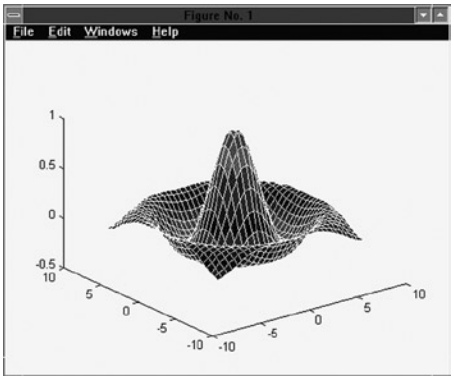

Figure 1-7.

In the first line of this example, note that the range is specified using colons. We did not use 0.5 and you can see that it can make the input somewhat confusing to a reader.

These 3D graphics allow you to get an idea of the figures in space, and are very helpful in visually identifying intersections between different bodies, generation of developments of all kinds and volumes of revolution.

4. You can generate a three dimensional graphic corresponding to the helix in parametric coordinates: x = sine (t), y = cosine (t), z = t. See Figure 1-8.

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t)
```

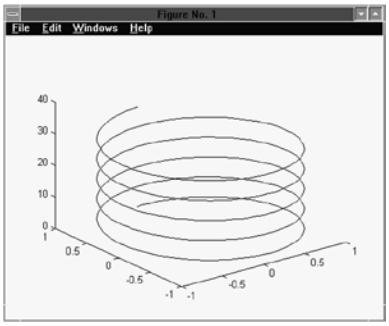

Figure 1-8.

We can represent a planar curve given by its polar coordinates r = cosine (2t) * sine (2t) for t varying between 0 and 2π, taking equally spaced points in one-hundredths of the considered range. See Figure 1-9.

```
>> t = 0:.01:2 * pi;
>> r = sin(2*t).* cos(2*t);
>> polar(t,r)
```

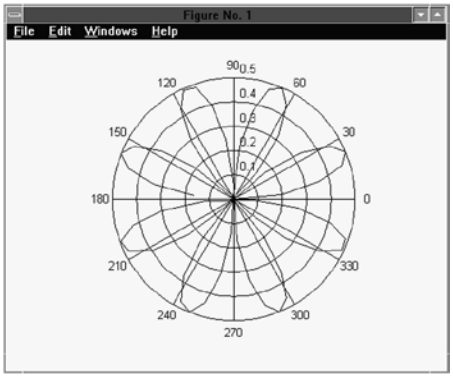

Figure 1-9.

5. We can also make a graph of a function considered as symbolic, using the command "ezplot". See Figure 1-10.

```
>> y ='x ^ 3 /(x^2-1)';
>> ezplot(y,[-5,5])
```

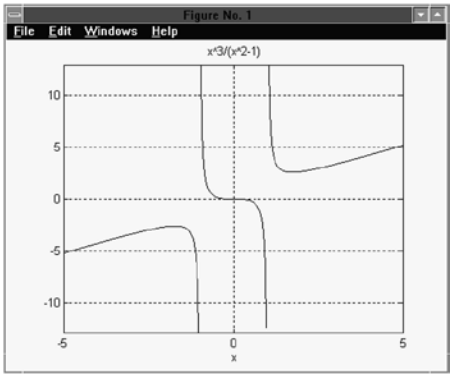

Figure 1-10.

In the corresponding chapter on graphics we will extend these concepts.

MATLAB and Programming

By properly combining all the objects defined in MATLAB appropriate to the work rules defined in the program, you can build useful mathematical research programming code. Programs usually consist of a series of instructions in which values are calculated, assigned a name and are reused in further calculations.

As in programming languages like C or Fortran, in MATLAB you can write programs with loops, control flow and conditionals. In MATLAB you can write procedural programs, (i.e., to define a sequence of standard steps to run). As in C or Pascal, a Do, For, or While loop may be used for repetitive calculations. The language of MATLAB also includes conditional constructs such as If Then Else. MATLAB also supports logic functions, such as And, Or, Not and Xor.

MATLAB supports procedural programming (iterative, recursive, loops…), functional programming and object-oriented programming. Here are two simple examples of programs. The first generates the order n Hilbert matrix, and the second calculates the Fibonacci numbers.

```
% Generating the order n Hilbert matrix
t = '1/(i+j-1)';
for i = 1:n
for j = 1:n
a(i,j) = eval(t);
end
end

% Calculating Fibonacci numbers
f = [1 1]; i = 1;
while f(i) + f(i-1) < 1000
f(i+2) = f(i) + f(i+1);
i = i+1
end
```

CHAPTER 2

Numbers, Operators, Variables and Functions

Numbers

The numerical scope of MATLAB is very wide. Integer, rational, real and complex numbers, which in turn can be arguments of functions giving rise to whole, rational, real and complex functions. Therefore, the complex variable is a treatable field in MATLAB. This chapter will look at the basic functionality of MATLAB.

Arithmetic operations in MATLAB are defined according to the standard mathematical conventions. The following table presents the syntax of basic arithmetic operations:

x + y	*Sum*
	`>> 4 + 8`
	`Ans =`
	`12`
x - y	*Difference*
	`>> 4 - 8`
	`Ans =`
	`-4`
x * y or x y	*Product*
	`>> 4 * 8`
	`Ans =`
	`32`

(*continued*)

x/y	*Division*
	`>> 4/8`
	`Ans =`
	`0.5000`
x ^ y	*Power*
	`>> 4 ^ 8`
	`Ans =`
	`65536`

MATLAB performs arithmetic operations as if it were a conventional calculator, but with total precision in the calculation. The results of operations presented are accurate for the default or for the degree of precision that the user prescribes. But this only affects the presentation. The feature that differentiates MATLAB from other numerical calculation programs in which the word length of the computer determines the accuracy, is that the calculation accuracy is unlimited in the case of MATLAB. MATLAB can represent the results with the accuracy that is required, although internally it always works with exact calculations to avoid rounding errors. MATLAB offers different approximate representation formats, which sometimes facilitate the interpretation of the results.

The following are the commands relating to the presentation of the results:

short format	*It offers results with 4 decimal. It's the format default of MATLAB*
	`>> sqrt(23)`
	`ans =`
	`4.7958`
long format	*It delivers results with 16 decimal places*
	`>> format long; sqrt(23)`
	`Ans =`
	`4.795831523312719`
format long e	*Provides the results to 16 decimal places using the power of 10*
	`>> format long e; SQRT (23)`
	`Ans =`
	`4.795831523312719e + 000`
format short e	*Provides the results to four decimal places with the power of 10*
	`>> format short e; SQRT (23)`
	`Ans =`
	`4.7958e + 000`

(*continued*)

format long g *It offers results in optimal long format*

```
>> format long g; SQRT (23)

Ans =

     4.79583152331272
```

format short g *It offers results in optimum short format*

```
>> format short g; SQRT (23)

Ans =

     4.7958
```

bank format *It delivers results with 2 decimal places*

```
>> bank format; SQRT (23)

Ans =

     4.80
```

format rat *It offers the results in the form of a rational number approximation*

```
>> format rat; SQRT (23)

Ans =

     1151/240
```

format + *Offers a binary result using the sign (+, -) and ignores the imaginary part of the complex numbers*

```
>> format +; sqrt(23)

Ans =

     +
```

format hex *It offers results in hexadecimal system*

```
>> format hex; sqrt(23)

ans =

     40132eee75770416
```

VPA 'operations' n *It provides the result of operations with n exact decimal digits*

```
>> vpa 'sqrt(23)' 20

ans =

     4.7958315233127195416
```

Integers and Integer Variable Functions

MATLAB works with integers and exact integer variable functions. Regardless of the format of presentation of the results, calculations are exact. Anyway, the command *vpa* allows exact outputs with the precision required.

In terms of functions with integer variables, the most important thing that MATLAB includes are as follows (the expressions between quotation marks have format string):

rem (n, m)	*Remainder of the division of n by m (valid function for n and m-real)*
	`>> rem(15,2)`
	`ans =`
	`1`

sign (n)	*Sign of n (1 if n > 0; -1 if n < 0)*
	`>> sign(-8)`
	`ans =`
	`-1`

max (n1, n2)	*Maximum of the numbers n1 and n2*
	`>> max(17,12)`
	`ans =`
	`17`

min (n1, n2)	*Minimum of the numbers n1 and n2*
	`>> min(17,12)`
	`ans =`
	`12`

gcd (n1, n2)	*Greatest common divisor of n1 and n2*
	`>> gcd(17,12)`
	`ans =`
	`1`

lcm (n1, n2)	*Least common multiple of n1 and n2*
	`>> lcm(17,12)`
	`ans =`
	`204`

(*continued*)

factorial (n)	*N factorial (n(n-1) (n-2)...)1)*
	`>> factorial(9)`
	`ans =`
	`362880`

factor (n)	*It decomposes the factorization of n*
	`>> factor(51)`
	`ans =`
	`3 17`

dec2base (decimal, n_base)	*Converts a specified decimal (base-10) number to the new base given by n_base*
	`>> dec2base(2345,7)`
	`ans =`
	`6560`

base2dec(numero,b)	*Converts the given base b number to a decimal number*
	`>> base2dec('ab12579',12)`
	`ans =`
	`32621997`

dec2bin (decimal)	*Converts a specified decimal number to base 2 (binary)*
	`>> dec2bin(213)`
	`ans =`
	`11010101`

dec2hex (decimal)	*Converts the specified decimal number to a base 16 (hexadecimal) number*
	`>> dec2hex(213)`
	`ans =`
	`D5`

bin2dec (binary)	*Converts the binary number to a decimal based number*
	`>> bin2dec('1110001')`
	`ans =`
	`113`

hex2dec (hexadecimal)	*It converts the specified base 16 number to decimal*
	`>> hex2dec('FFAA23')`
	`ans =`
	`16755235`

Real Numbers and Functions of Real Variables

A rational number is of the form *p/q*, where *p* is an integer and *q* another integer. The way in which MATLAB processes rationals is different from that of the majority of calculators. The rational numbers are ratios of integers, and MATLAB also can work with them so that the result of expressions involving rational numbers is always another rational or whole number. So, it is necessary to activate this format with the command *format rat*. But MATLAB also returns solutions with decimals as the result if the user so wishes by activating any other type of format (e.g. *format short* or *long format*). We consider the following example:

```
>> format rat;
>> 1/6 + 1/5-2/10

Ans =

       1/6
```

MATLAB deals with rationals as ratios of integers and keeps them in this form during the calculations. In this way, rounding errors are not dragged into calculations with fractions, which can become very serious, as evidenced by the theory of errors. Once enabled in the rational format, operations with rationals will be exact until another different format is enabled. When the rational format is enabled, a number in floating point, or a number with a decimal point is interpreted as exact and MATLAB is exact in how the rational expression is represented with the result in rational numbers. At the same time, if there is an irrational number in a rational expression, MATLAB retains the number in the rational form so that it presents the solution in the rational form. We consider the following examples:

```
>> format rat;
>> 2.64/25+4/100

Ans =

      91/625

<Note that this is the result of 66/625 + 25/625.>

>> 2.64/sqrt(25)+4/100

Ans =

      71/125

>> sqrt(2.64)/25+4/100

Ans =

     204/1943
```

MATLAB also works with the irrational numbers representing the results with greater accuracy or with the accuracy required by the user, bearing in mind that the irrational cannot be represented as exactly as the ratio of two integers. Below is an example.

```
>> sqrt (235)

Ans =

    15.3297
```

There are very typical real constants represented in MATLAB as follows:

pi	*Number π = 3.1415926*
	`>> 2 * pi`
	`Ans =`
	`6.2832`
exp (1)	*Number e = 2.7182818*
	`>> exp (1)`
	`Ans =`
	`2.7183`
inf	*Infinity (for example 1/0)*
	`>> 1/0`
	`Ans =`
	`INF`
nan	*Uncertainty (for example 0/0)*
	`>> 0/0`
	`Ans =`
	`NaN`
realmin	*Least usable positive real number*
	`>> realmin`
	`Ans =`
	`2.2251e-308`
realmax	*Largest usable positive real number*
	`>> realmax`
	`Ans =`
	`1.7977e + 308`

MATLAB has a range of predefined functions of real variables.The following sectionss present the most important.

Trigonometric Functions

Below is a table with the trigonometric functions and their inverses that are incorporated in MATLAB illustrated with examples.

Function	Inverse
sin (x) *sine*	**asin (x)** *arcsine*
`>> sin(pi/2)`	`>> asin (1)`
`ans =`	`ans =`
`1`	`1.5708`
cos (x) *cosine*	**acos (x)** *arccosine*
`>> cos (pi)`	`>> acos (- 1)`
`ans =`	`ans =`
`-1`	`3.1416`
tan(x) *tangent*	**atan(x) atan2 (x) and** *arctangent*
`>> tan(pi/4)`	`>> atan (1)`
`ans =`	`ans =`
`1.0000`	`0.7854`
csc(x) *cosecant*	**acsc (x)** *arccosecant*
`>> csc(pi/2)`	`>> acsc (1)`
`ans =`	`ans =`
`1`	`1.5708`
sec (x) *secant*	**asec (x)** *arcsecant*
`>> sec (pi)`	`>> asec (- 1)`
`ans =`	`ans =`
`-1`	`3.1416`
cot (x) *cotangent*	**acot (x)** *arccotangent*
`>> cot(pi/4)`	`>> acot (1)`
`ans =`	`ans =`
`1.0000`	`0.7854`

Hyperbolic Functions

Below is a table with the hyperbolic functions and their inverses that are incorporated in MATLAB illustrated with examples.

Function	Reverse
sinh (x) *hyperbolic sine*	**asinh (x)** *hyperbolic arcsine*
`>> sinh (2)`	`>> asinh (3.6269)`
`ans =`	`ans =`
`3.6269`	`2.0000`
cosh(x) *hyperbolic cosine*	**acosh(x)** *hyperbolic arccosine*
`>> cosh (3)`	`>> acosh (10.0677)`
`ans =`	`ans =`
`10.0677`	`3.0000`
tanh(x) *hyperbolic tangent*	**atanh (x)** *hyperbolic arctangent*
`>> tanh (1)`	`>> atanh (0.7616)`
`ans =`	`ans =`
`0.7616`	`1.0000`
csch (x) *hyperbolic cosecant*	**acsch (x)** *hyperbolic arccosecant*
`>> csch (3.14159)`	`>> acsch (0.0866)`
`ans =`	`ans =`
`0.0866`	`3.1415`
sech (x) *hyperbolic secant*	**asech (x)** *hyperbolic arcsecant*
`>> sech (2.7182818)`	`>> asech (0.1314)`
`ans =`	`ans =`
`0.1314`	`2.7183`
coth (x) *hyperbolic cotangent*	**acoth (x)** *hyperbolic arccotangent*
`>> coth (9)`	`>> acoth (0.9999)`
`ans =`	`ans =`
`1.0000`	`4.9517 + 1.5708i`

Exponential and Logarithmic Functions

Below is a table with the exponential and logarithmic functions that are incorporated in MATLAB illustrated with examples.

Function	Meaning

exp (x)	*Exponential function in base e (e ^ x)*
	`>> exp (log (7))`
	`Ans =`
	`7`

log (x)	*Logarithmic function base e of x*
	`>> log (exp (7))`
	`Ans =`
	`7`

log10 (x)	*Base 10 logarithmic function*
	`>> log10 (1000)`
	`Ans =`
	`3`

log2 (x)	*Logarithmic function of base 2*
	`>> log2(2^8)`
	`Ans =`
	`8`

pow2 (x)	*Power function base 2*
	`>> pow2 (log2 (8))`
	`Ans =`
	`8`

sqrt (x)	*Function for the square root of x*
	`>> sqrt(2^8)`
	`Ans =`
	`16`

Numeric Variable-Specific Functions

MATLAB incorporates a group of functions of numerical variable. Among these features are the following:

Function	Meaning

abs (x)	*Absolute value of the real x*
	`>> abs (- 8)`
	`Ans =`
	`8`

floor (x)	*The largest integer less than or equal to the real x*
	`>> floor (- 23.557)`
	`Ans =`
	`-24`

ceil (x)	*The smallest integer greater than or equal to the real x*
	`>> ceil (- 23.557)`
	`Ans =`
	`-23`

round (x)	*The closest integer to the real x*
	`>> round (- 23.557)`
	`Ans =`
	`-24`

fix (x)	*Eliminates the decimal part of the real x*
	`>> fix (- 23.557)`
	`Ans =`
	`-23`

rem (a, b)	*It gives the remainder of the division between the real a and b*
	`>> rem (7.3)`
	`Ans =`
	`1`

sign (x)	*The sign of the real x (1 if x > 0; - 1 if x < 0)*
	`>> sign (- 23.557)`
	`Ans =`
	`-1`

One-Dimensional, Vector and Matrix Variables

MATLAB is software based on matrix language, and therefore focuses especially on tasks for working with arrays.

The initial way of defining a variable is very simple. Simply use the following syntax:

```
Variable = object
```

where the object can be a scalar, a vector or a matrix.

If the variable is an array, its syntax depends on its components and can be written in one of the following ways:

```
Variable = [v1 v2 v3... vn]
Variable = [v1, v2, v3,..., vn]
```

If the variable is an array, its syntax depends on its components and can be written in one of the following ways:

```
Variable = [v11 v12 v13... v1n; v21 v22 v23... v2n;...]
Variable = [v11, v12, v13,..., v1n; v21, v22, v23,..., v2n;...]
```

Script variables are defined in the command window from MATLAB in a natural way as shown in Figure 2-1.

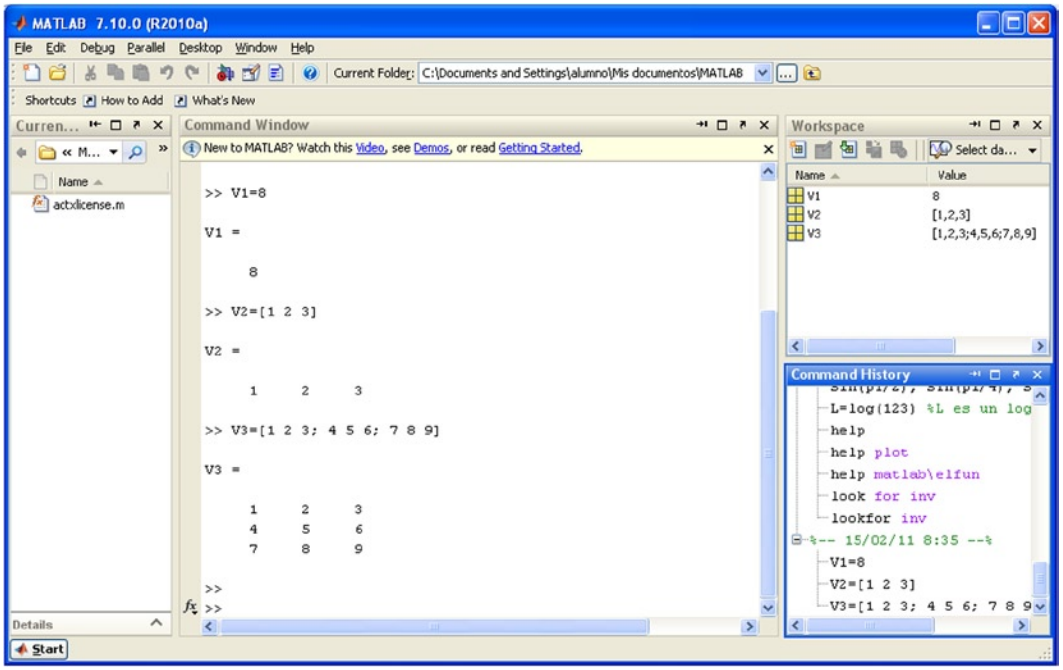

Figure 2-1.

The workspace (*Workspace*) window contains all variables that we will define in the session. The value of any of these variables is recoverable by typing their names on the command window (Figure 2-2).

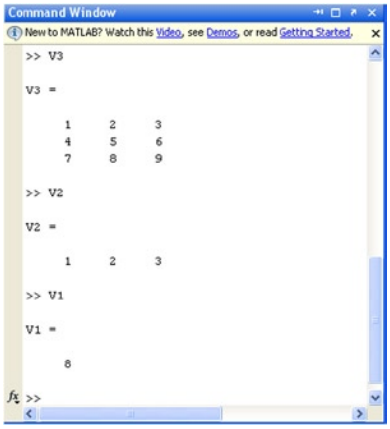

Figure 2-2.

Also, in the command history window (*Command History*) we can find all the syntax we execute from the command window.

Once a variable is defined, we can operate on it using it as a regular variable in mathematics, bearing in mind that the names of the variables are sensitive to upper and lower case. Figure 2-3 shows some operations with one-dimensional, vector and matrix variables. It is important to note the error that occurred when calculating the logarithm of V2. The mistake was changing the variable to lowercase so that it was not recognized.

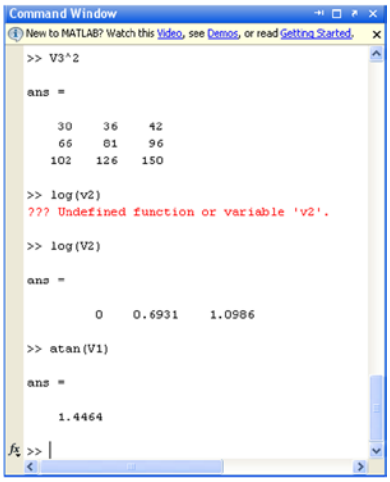

Figure 2-3.

MATLAB variables names begin with a letter followed by any number of letters, digits or underscores with a 31 character maximum.

There are also specific forms for the definition of vector variables, among which are the following:

variable = [a: b] *Defines the vector whose first and last elements are a and b, respectively, and the intermediate elements differ by one unit*

```
>> vector1 = [2:6]

vector1 =

         2     3     4     5     6
```

variable = [a: s:b] *Defines the vector whose first and last elements are a and b, and the intermediate elements differ in the amount specified by the increase in s*

```
>> vector2 = [2:2:8]

Vector2 =

         2 4 6 8
```

Nvariable = linespace [a, b, n] *Defines the vector whose first and last elements are a and b, and has in total n evenly spaced elements*

```
>> vector3 = linespace (10,30,6)

Vector3 =

         10 14 18 22 26 30
```

Elements of Vector Variables

MATLAB allows the selection of elements of vector variables using the following commands:

x (n) *Returns the nth element of the vector x*

```
>> X =(2:8)

X =

   2     3     4     5     6     7     8

>> X (3)

Ans =

     4
```

x(a:b) *Returns the elements of the vector x between the a-th and the b-th elements, both inclusive*

```
>> X (3:5)

Ans =

     4 5 6
```

(*continued*)

x(a:p:b) *Returns the elements of the vector x located between the a-th and the b-th elements, both inclusive, but separated by units where a < b.*

```
>> X (1:2:6)

Ans =

     2-4-6
```

x(b:-p:a) *Returns the elements of the vector x located between the b- and a-th, both inclusive, but separates by p units and starting with the b-th element (b > a)*

```
>> X(6:-2:1)

Ans =

     7-5-3
```

Elements of Matrix Variables

The same as in the case of vectors, MATLAB allows the selection of elements of array variables using the following commands:

A(m,n) *Defines the (m, n) element of the matrix A (row m and column n)*

```
>> A = [3 5 7 4; 1 6 8 9; 2 6 8 1]

A =

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

>> A (2,4)

Ans =

     9
```

A(a:b,c:d) *Defines the subarray of A formed by the intersection of the a-th and the b-th rows and the the c-th and the d-th columns*

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

Ans =

     5 7
     6 8
```

(*continued*)