



# An introductory course in MATLAB

Santiago Pellegrini and Alejandro Rodriguez

Master in Business Administration and Quantitative Methods

September 2008

# Contents

<b>1</b>	<b>First steps in MATLAB</b>	<b>3</b>
1.1	The MATLAB environment . . . . .	3
1.2	Basic syntax: vectors and matrices . . . . .	5
1.3	Algebraical and logical operations . . . . .	7
1.4	Condition constructs: the <i>if</i> and <i>elseif</i> statements . . . . .	10
1.5	Loop constructs: the <i>for</i> and <i>while</i> statements . . . . .	11
1.6	Loading and saving data . . . . .	12
1.7	Introduction to Graphics: The <i>Plot</i> command . . . . .	13
<b>2</b>	<b>Using the <i>Statistics</i> and <i>Optimization</i> Toolboxes</b>	<b>15</b>
2.1	The <i>Statistics</i> toolbox. (Applications) . . . . .	15
2.1.1	Probability density function ( <i>PDF</i> ), Cumulative distribution function ( <i>CDF</i> ), Inverse CDF ( <i>ICDF</i> ) and random number generators. . . . .	16
2.1.2	Descriptive Statistics . . . . .	21
2.1.3	Statistical plots . . . . .	23
2.1.4	Linear regression analysis . . . . .	26
2.2	The <i>Optimization</i> toolbox. (Applications) . . . . .	28
2.2.1	Minimizing unconstrained nonlinear functions . . . . .	29
2.2.2	Minimizing constrained nonlinear functions. . . . .	32
<b>3</b>	<b>Programming in MATLAB</b>	<b>38</b>
3.1	M-files (Scripts) . . . . .	39
3.2	Functions . . . . .	40
3.3	Exercises and applications . . . . .	40

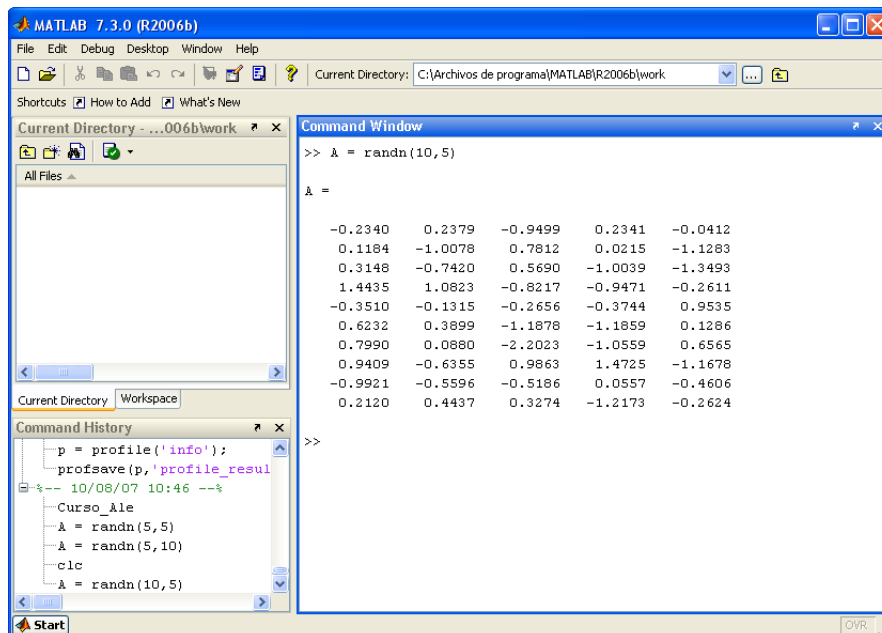
# 1 First steps in MATLAB

MATLAB is a program that was originally developed in FORTRAN<sup>1</sup> as a MATrix LABoratory for solving numerical linear algebra problems. It has since grown into something much bigger, nowadays it combines numeric computation, advanced graphics and visualization, and a high-level programming language. It has essentially the following features:

1. Matrices as fundamental data type.
2. Built-in support for complex numbers.
3. Powerful built-in math functions and extensive function libraries.
4. Extensibility in the form of user-defined functions.

## 1.1 The MATLAB environment

When MATLAB is started appears, by default, a window divided in three parts or sub-windows similar to the following figure:



- The most important part of this window is what is called “**Command Window**”, which is at the right side of the figure. This window is where MATLAB commands are executed

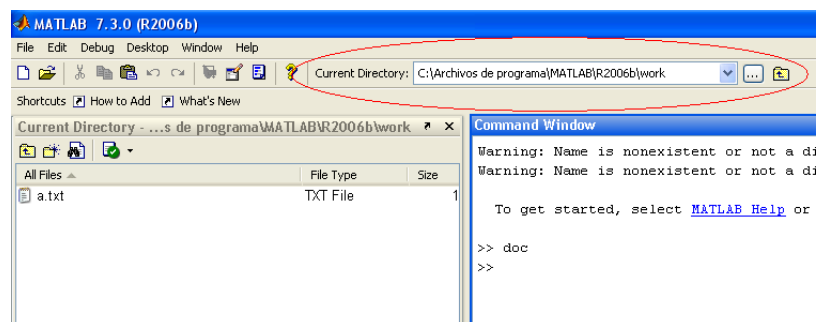
---

<sup>1</sup>Fortran is a general-purpose, procedural, imperative programming language that is especially suited to numeric computation and scientific computing.

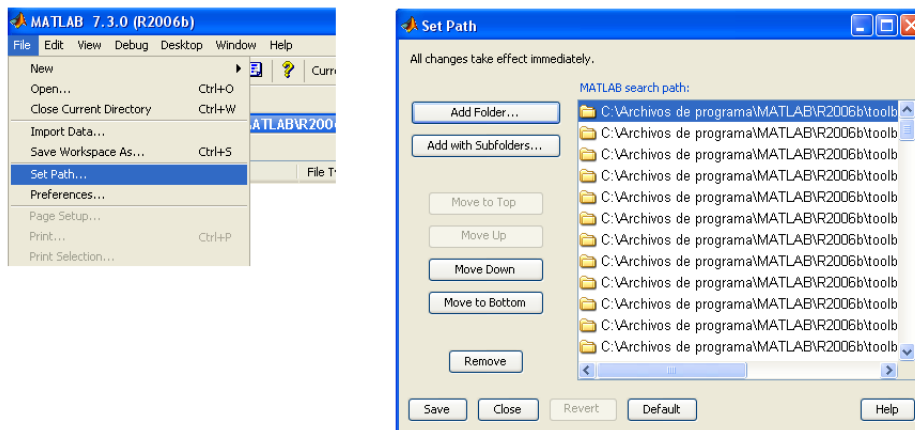
and the results are showed. For example, in the figure we can see the way of creating a matrix of random numbers using the command `randn`.

- At the left side of the figure, we can see two sub-windows. The upper left window shows the “**Current Directory**” and “**Workspace**” changing each other by clicking on the corresponding tab. The Current Directory window displays the current folder and its contents. The Workspace window provides an inventory of all variables in the workspace that are currently defined, either by assignment or calculation in the Command Window or by importation.
- Finally, the lower window called “**Command History**” contains a list of commands that have been executed within the Command Window.

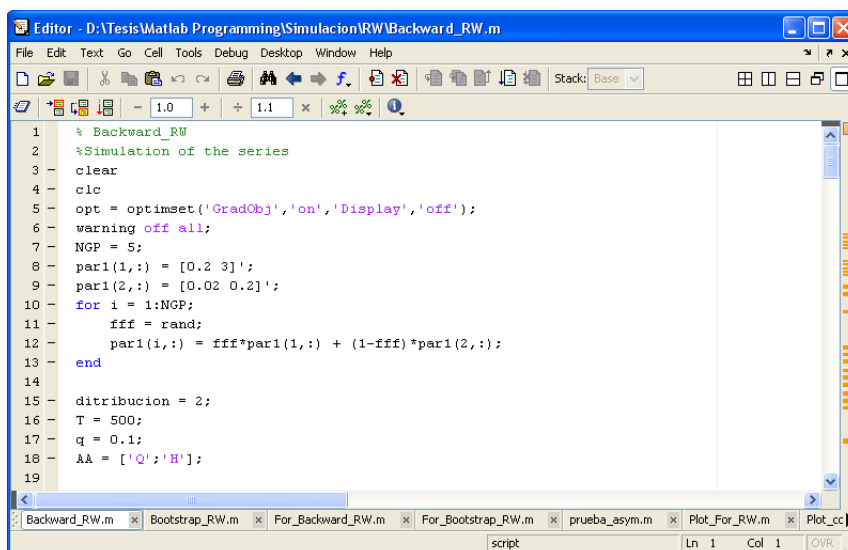
MATLAB file operations use the current directory and the MATLAB search path as reference points. Any file we want to run must be either in the **Current Directory** or on the **Search Path**. A quick way to view or change the current directory is by using the current directory field in the desktop toolbar. See the following figure:



MATLAB uses a Search Path to find *m-files* and other MATLAB related files, which are organized in directories on our file system. By default, the files supplied with MATLAB products are included in the search path. These are all of the directories and files under `matlabroot/toolbox`, where *matlabroot* is the directory in which MATLAB was installed. We can modify, add or remove directories through “**File**”, “**Set Path**”, and click on “**Add Folder**” or “**Add with subfolder**”, as showed in the next figure:



The “**m-files**” (named after its .m file extension) have a special importance. They are text (ASCII) files ending in “.m”, these files contain a set of command or functions. In general, after typing the name of one of these “**m-files**” in the Command Window and pressing *Intro*, all commands and functions defined into the file are sequentially executed. These files can be created in any text editor. MATLAB has its own *m-file* editor, which allows to create and modify these files. The following figure shows how this editor look likes.



## 1.2 Basic syntax: vectors and matrices

Let’s stress, before starting with the definition of matrices and vectors, that at any moment we can access to the help of any command of MATLAB by writing in the Command Window `help` followed by the `command-name` or `doc` followed by the `command-name`. The former shows the help in the Command Window and the latter open a new windows, the “*Help*” windows, where shows the help of the command.

MATLAB is fundamentally a matrix programming language. A matrix is a two-dimensional array of real or complex numbers. Linear algebra defines many matrix operations that are directly supported by MATLAB. Linear algebra includes matrix arithmetic, linear equations, eigenvalues, singular values, and matrix factorizations. MATLAB provides many functions for performing mathematical operations and analyzing data. Creation and modification of matrices and vectors are straightforward tasks. For example, if we want to create a row vector called “a” with elements  $\{1, 2, 3, 4\}$ , we enter

```
>> a = [1 2 3 4]
```

and the result will be

```
a =  
    1    2    3    4
```

or use the notation [start:stepsize:end].

```
>> a = [1:2:7]
```

and the result will be

```
a =  
    1    3    5    7
```

If we want a column vector, we have to write

```
>> a = [1;2;3;4]
```

or

```
>> a = [1 2 3 4]'
```

and the result will be

```
a =  
    1  
    2  
    3  
    4
```

Next, taking it in mind, we can define a matrix as follows

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

```
A =  
    1    2    3  
    4    5    6  
    7    8    9
```

Suppose that we are interested in one element of a vector or matrix. In this case we can get this element by putting the correspondent index between parenthesis after the name of the vector or matrix. For example, given the vector  $b = \{23, 45, 82, 19, 42, 4, 7\}$ , we are interested in the second element, then we enter

```
>> b(2)
```

the result will be

```
ans =
```

45

In the case that we can get an element from a matrix, for example the element in the second row, third column of matrix  $A$ , we have to enter

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

the result will be

```
ans =
```

6

Finally, in the case that we are interested in more elements of a vector or matrix we use the notation [start:end]. For example, the first three elements of the second column of the matrix  $B$

$$B = \begin{pmatrix} 4 & 3 & 6 & 0 \\ 5 & 7 & 0 & 4 \\ 9 & 8 & 2 & 5 \\ 5 & 1 & 8 & 7 \\ 4 & 1 & 9 & 2 \end{pmatrix}$$

can be obtained typing

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

```
ans =
```

3

7

8

There exist commands that create special matrices. For example, the command `zeros(N,M)` creates a matrix of zeros with dimension N-by-M, the command `eye(N)` creates an identity matrix of dimension N, the command `ones(N,M)` creates a matrix of ones with dimension N-by-M. For instance, if “ $b$ ” is a vector, then the command `diag(b)` creates a diagonal matrix with the elements of “ $b$ ”.

### 1.3 Algebraical and logical operations

In this part of the tutorial we introduce simple algebraical and logical operations.

#### Algebraical operations

Algebraical operations in MATLAB are straightforward, just enter in the Command Window exactly the same expression as if one writes it in a paper. For example, multiply two numbers

```
>> 3*5
```

divide two numbers

```
>> 15/3
```

elevate a number to some power

```
>> 2^(-4)
```

When the algebraical operations are with vectors or matrices, the procedure is the same, except that we have to take into account that inner matrix dimension must agree. For example, multiply matrix  $A$  by  $B$

$$A = \begin{pmatrix} 4 & 3 & 6 \\ 5 & 7 & 0 \end{pmatrix}, B = \begin{pmatrix} 5 & 8 & 3 \\ 1 & 0 & 6 \end{pmatrix}$$

entering in the Command Window  $A * B$  is a mistake, the right way is enter

```
>> A*B'
```

or

```
>> A'*B
```

using the transpose operator ( $'$ ). On the other hand, it is possible to do the product  $A$  times  $B$ , element-by-element, by typing

```
>> A.*B
```

the result will be

```
ans =
    20    24    18
     5     0     0
```

The inverse matrix can be calculated with the command `inv`. For example, the inverse of matrix  $A$

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & 5 \\ 1 & 1 & 3 \end{pmatrix}$$

it is obtained through

```
>> inv(A)
```

```
ans =
   -1    -3     5
    1    1.5  -2.5
    0    0.5  -0.5
```

The next table summarizes a list of very useful commands.



Command	Description
<code>abs(a)</code>	absolute value of the elements of “a”
<code>clc</code>	clear the screen
<code>clear</code>	clear all variables and functions from the memory
<code>cumprod(a)</code>	creates a vector containing the cumulative product of the elements of “a”
<code>cumsum(a)</code>	creates a vector containing the cumulative sum of the elements of “a”
<code>det(A)</code>	the determinant of matrix A
<code>eig(A)</code>	eigen values and eigen vectors of matrix A
<code>inv(A)</code>	the inverse matrix of A
<code>length(a)</code>	the number of elements of “a”
<code>prod(a)</code>	multiplies the elements of “a”
<code>rank(A)</code>	the rank of matrix A
<code>size(A)</code>	the dimensions of A
<code>sum(a)</code>	returns the sum of the elements of “a”
<code>trace(A)</code>	the trace of matrix A
<code>who</code>	shows all variables in the Workspace
<code>whos</code>	shows all variables in the Workspace with more detail

### Relational and Logical operations

The symbols “&”, “|”, and “~” are the logical array operators AND, OR, and NOT. They work element by element on arrays, with logical 0 representing false, and logical 1 or any nonzero element representing true. The logical operators return a logical array with elements set to 1 (true) or 0 (false), as appropriate. For example, we would like to know if there exist elements of a vector  $a = \{1, 4, 7, 2, 8, 5, 2, 7, 9\}$  that are greater than or equal to those of a vector  $b = \{2, 5, 1, 4, 8, 0, 3, 4, 3\}$ , compared pairwise. In the Command Window we enter

```
>> a >= b
```

and the result will be

```
ans =
```

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

Now, suppose we are interested in to know how many elements of vector “a” are greater than or equal to those of vector “b”. Alternatively, we can enter

enter

```
>> sum(a >= b)
```

and we will find the result that we are looking for

```
ans =
```

```
5
```

This is just an example of a very simple logical operation, but one can construct more complex logical sentences. The following table shows some useful logical operator.

Command	Description
==	equal
~=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
&	and
	or
xor	exclusive or
~	logical complement
any	True if any element of a vector is a nonzero number
all	True if all elements of a vector are nonzero

For example, we would like to see if the number of the elements greater than 2, but lower than 7 of the vector “a” is equal to the number of the elements greater than 5, but lower than 10 of the vector “b”. Then, we enter

```
>> sum(a > 2 & a < 7) == sum(b > 5 & b < 10)
```

the result will be

```
ans =
```

```
0
```

## 1.4 Condition constructs: the *if* and *elseif* statements

There exist situations in which we want to apply different statements depending on some conditioning. This is easily done with the `if` statement. The basic structure for an if statement is the following

```
if Condition 1
```

```
    Statement 1
```

```

elseif condition 2
    Statement 2
elseif condition 3
    Statement 3
...
else
    Statement n
end

```

Each set of commands is executed if its corresponding condition is satisfied. The `else` and `elseif` parts are optional. Notice that the `if` command checks conditions sequentially, first condition 1, then condition 2, and so on. But when the one condition is verified the statement associated with it is executed and the `if` does not continue verifying the rest of conditions.

For example

```
>> n = 0; if -1 > 0, n = 1; elseif 2 > 0, n = 2; elseif 3 > 0, n = 3; else, n = 4; end
```

The value of “n” will be 2 even though the third condition is also true.

## 1.5 Loop constructs: the *for* and *while* statements

When we want to repeat statements, we can use loops. They are constructed using either the command `for` or the command `while`.

### The command `for`

This command repeats statements a specific number of times. Its structure is the following

```

for variable = expression
    Statement 1
    Statement 2
    Statement 3
...
    Statement n
end

```

For example

```
>> I = [1:5]; J = [1:5]; A = [];
>> for i = I
    for j = J
        A(i,j) = 1/(j+i-1)
    end
end
```

The result will be

```
ans =  
    1.00    0.50    0.33    0.25    0.20  
    0.50    0.33    0.25    0.20    0.17  
    0.33    0.25    0.20    0.17    0.14  
    0.25    0.20    0.17    0.14    0.13  
    0.20    0.17    0.14    0.13    0.11
```

### The command while

This command repeats statements an indefinite number of times depending on a condition. Its structure is the following

```
while Condition  
    Statement 1  
    Statement 2  
    Statement 3  
    ...  
    Statement n
```

```
end
```

While the condition is verified this command continues repeating the statements. For example

```
>> eps = 1; counter = 0;  
>> while (eps + 1) > 1  
    counter = counter + 1;  
    eps = eps/2;  
end
```

The result will be

```
eps =  
    1.1102e-016  
counter =  
    53
```

## 1.6 Loading and saving data

While working, MATLAB saves all variables created or loaded in the Workspace. However, when we close MATLAB, the Workspace cleans itself, and all these variables are eliminated. If we are interested in keeping that information, the command `save` can help us. This command saves all Workspace variables to a “file”. For example

```
>> save filename
```

creates a file called “filename.mat”, in the current directory, which contains all variables in the Workspace. Now, using the command `load` can recover all variables saved in that file. For example

```
>> load filename
```

Some times we want to save only certain variables. For example

```
>> save filename a b D
```

This expression creates a file called “filename.mat”, which contains only the variables “a”, “b” and “D”.

There are situations in which we are interesting in saving a variable in a text file. This can be done with the command `save`, and using some of its options, for example

```
>> save filename.txt a -ascii
```

This syntax creates a text file called “filename.txt”, in the current directory, which contains the variable “a”. This type of file are loaded using command `load` as follows

```
>> load filename.txt
```

For more details enter `help save` or `doc save` in the Command Window.

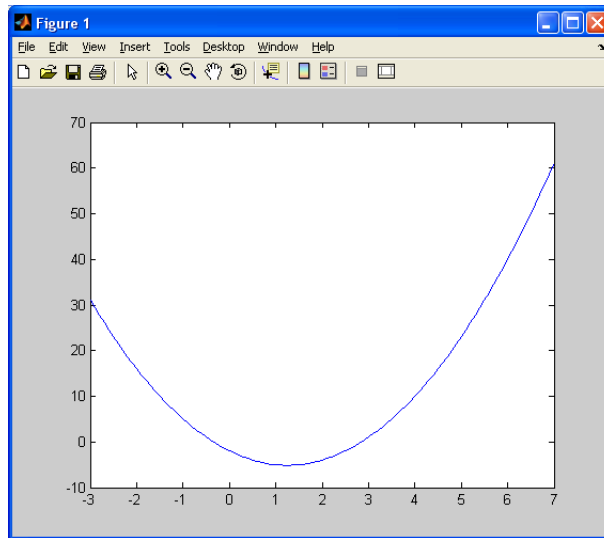
## 1.7 Introduction to Graphics: The *Plot* command

Now, we introduce a first notion to graphs, the command `plot`. It creates a graph in which the elements of the vector/s o matrix/ces used as input are plotted. This command can plot the elements of one vector “a” versus other vector “b” or the elements of a vector “c” versus their index. For example

```
>> x = -3:0.1:7
```

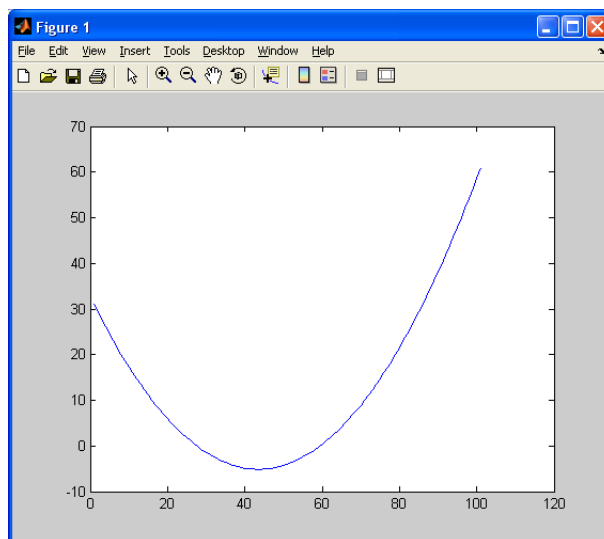
```
>> y = 2*x.^2 - 5*x -2
```

```
>> plot(x,y)
```



Alternatively, we can plot a vector “y” against its own index. In our example we enter

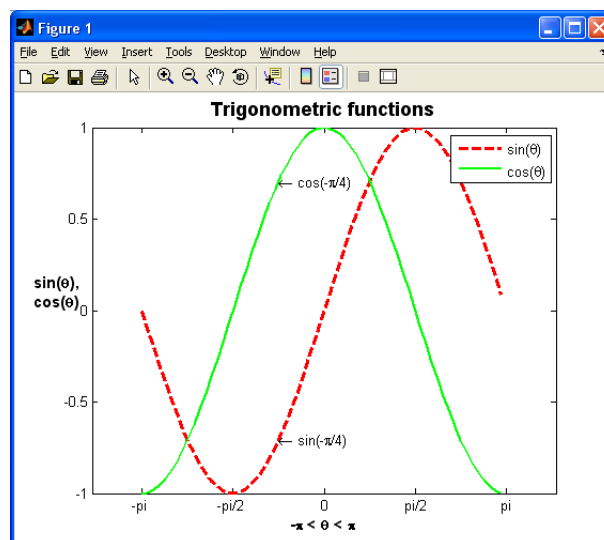
```
>> plot(y)
```



It is possible to modify some aspect of the graph like color, width, and shape, change the ticks of the axes, add other line, using the commands `set`, `hold on` and `hold off`. Moreover, we can add a title, label in the axis, legends, etc. Here is an example

```
>> x = -pi:0.1:pi; y = sin(x); y1 = cos(x);  
>> set(gcf,'Color','w')  
>> plot1 = plot(x,y);  
>> hold on  
>> plot2 = plot(x,y1)  
>> hold off  
>> set(plot1,'LineStyle','--','LineWidth',2.5, 'Color','r');
```

```
>> set(plot2,'LineStyle','-','LineWidth',2, 'Color','g');
>> set(gca,'XTick',-pi:pi/2:pi,'XTickLabel',{'-pi','-pi/2',...
'0','pi/2','pi'},'YTick',-1:0.5:1);
>> title('Trigonometric functions','FontWeight','bold','FontSize',13)
>> legend('sin(\theta)','cos(\theta)')
>> xlabel('-\pi < \theta < \pi','FontWeight','bold','FontSize',11)
>> ylabel(['sin(\theta)','\n','cos(\theta)'], 'Rotation',0.0,...
'FontWeight','bold','FontSize',11)
>> text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi/4)','HorizontalAlignment','left')
>> text(-pi/4,cos(-pi/4),'\leftarrow cos(-\pi/4)','HorizontalAlignment','left')
```



## 2 Using the *Statistics* and *Optimization* Toolboxes

MATLAB has several auxiliary libraries called *Toolbox*, which are collections of *m-files* that have been developed for particular applications. These include the Statistics toolbox, the Optimization toolbox, and the Financial toolbox among others. In our particular case, we will do a brief description of the Statistics and Optimization toolboxes and we will use some commands of the Financial toolbox.

### 2.1 The *Statistics* toolbox. (Applications)

The *Statistics* toolbox, created in the version 5.3 and continuously updated in newer versions, is a collection of statistical tools built on the MATLAB numeric computing environment. The toolbox supports a wide range of common statistical tasks, from random number generation,

curve fitting, to design of experiments and statistical process control. By typing `help stats` on the command window, an enormous list of functions appears. They are classified according to the topic they are made for. We will revise just a few of them, taking into account their interest for the first-year courses.

### 2.1.1 Probability density function (*PDF*), Cumulative distribution function (*CDF*), Inverse CDF (*ICDF*) and random number generators.

Parametric inferential statistical methods are mathematical procedures for statistical hypothesis testing which assume that the distributions of the variables being assessed belong to known parametrized families of probability distributions. In that case we speak of parametric models. When working within this branch of statistics, it is of vital importance to know the properties of these parametric distributions. The *Statistics* toolbox has information of a great number of known parametric distributions, regarding their *PDFs*, *CDFs*, *ICDFs*, and other useful commands to work with. Moreover, the function names were created in a very intuitive way, keeping the distribution code name unchanged and adding at the end a specific particle for each function. Thus, for instance, `norm` is the root for the normal distribution, and consequently `normpdf` and `normcdf` are the commands for finding the density and cumulative distribution functions, respectively. Below we list some of the well-known parametric distributions and the specific commands that are already available for each of them.



Distributions	Root name	pdf	cdf	inv	fit	rnd	stat	like
Beta	beta	✓	✓	✓	✓	✓	✓	✓
Binomial	bino	✓	✓	✓	✓	✓	✓	
Chi square	chi2	✓	✓	✓		✓	✓	
Empirical	e		✓					
Extreme value	ev	✓	✓	✓	✓	✓	✓	✓
Exponential	exp	✓	✓	✓	✓	✓	✓	✓
F	f	✓	✓	✓		✓	✓	
Gamma	gam	✓	✓	✓	✓	✓	✓	✓
Geometric	geo	✓	✓	✓		✓	✓	
Generalized extreme value	gev	✓	✓	✓	✓	✓	✓	
Generalized Pareto	gp	✓	✓	✓	✓	✓	✓	✓
Hypergeometric	hyge	✓	✓	✓		✓	✓	
Lognormal	logn	✓	✓	✓	✓	✓	✓	
Multivariate normal	mvn	✓	✓			✓		
Multivariate t	mvt	✓	✓			✓		
Negative binomial	nbin	✓	✓	✓	✓	✓	✓	✓
Noncentral F	ncf	✓	✓	✓		✓	✓	
Noncentral t	nct	✓	✓	✓		✓	✓	
Noncentral Chi-square	ncx2	✓	✓	✓		✓	✓	
Normal (Gaussian)	norm	✓	✓	✓	✓	✓	✓	✓
Poisson	poiss	✓	✓	✓	✓	✓	✓	
Rayleigh	rayl	✓	✓	✓	✓	✓	✓	
T	t	✓	✓	✓		✓	✓	
Discrete uniform	unid	✓	✓	✓		✓	✓	
Uniform	unif	✓	✓	✓	✓	✓	✓	
Weibull	wbl	✓	✓	✓	✓	✓	✓	✓

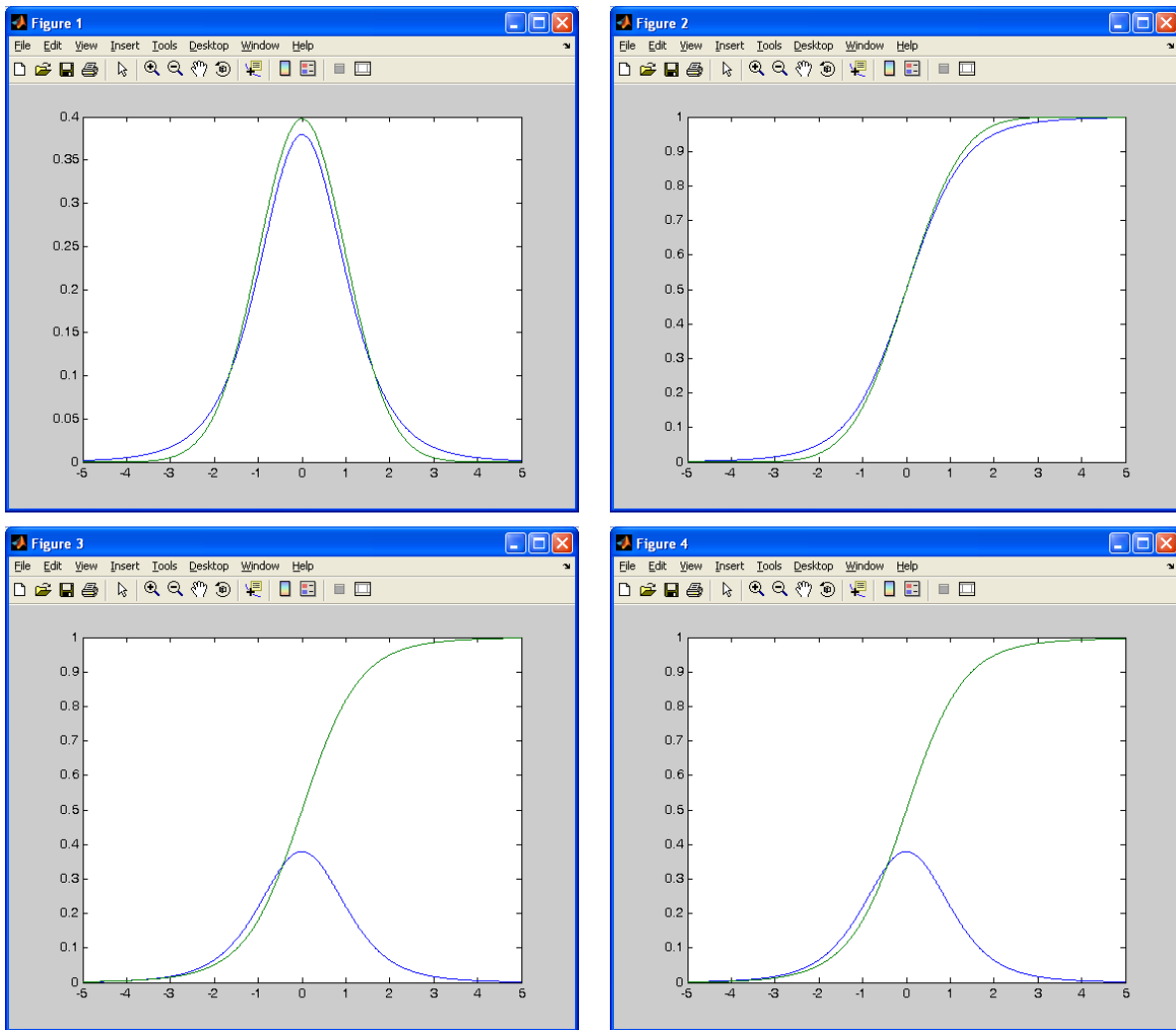
Particles	Description	Inputs
pdf	Probability density function	Argument, parameters
cdf	Cumulative distribution function	Argument, parameters
inv	Inverse CDF	Prob, parameters
fit	Parameter estimation	Data vector
rnd	Random Number generation	Parameters, matrix size
stat	Mean and variance	Parameters
like	Likelihood	Parameters, data vector

Let's do some examples in MATLAB. First, create a vector  $X$  with 1000 i.i.d. normal random variables with zero mean and unit variance and then estimate its parameters.

```
>> X = normrnd(0,1,1000,1);
>> plot(X)
>> [mu,sigma] = normfit(X);
>> mu,sigma
mu =
-0.0431
sigma =
0.9435
```

Now, let's find out how the PDF and CDF of certain distributions look like. First generate the domain of the function with the command  $S = \text{first}:\text{stepsize}:\text{end}$ . For example, set the interval  $(-5;5)$  with a step of size 0.01 and find the PDF and CDF of a Student-t distribution with 5, and 100 degrees of freedom:

```
>> Z = -5:0.01:5;Z=Z';
>> t5_pdf = tpdf(Z,5);t10_pdf = tpdf(Z,10);t100_pdf = tpdf(Z,100);
>> t5_cdf = tcdf(Z,5);t10_cdf = tcdf(Z,10);t100_cdf = tcdf(Z,100);
>> figure(1),plot(Z,[t5_pdf t100_pdf]),
>> figure(2),plot(Z,[t5_cdf t100_cdf]),
>> figure(3),plot(Z,[t5_pdf t5_cdf]),
>> figure(4),plot(Z,[t5_pdf cumsum(t5_pdf)*0.01]),
```



Note that Figures 3 and 4 above are pretty similar, as though they were plotting the same functions. This happens because the CDF of a given value  $x$  is not more than the area below the pdf up to this point, and this area can be easily approximated by the command `cumsum`. This command simply replaces the  $i^{\text{th}}$  element of a vector (in our case, the vector `t5_pdf`) by the partial sum of all observations up to this element. Thus if  $D = [1 \ 2 \ 3 \ 4]$ , then  $\text{cumsum}(D) = [1 \ 3 \ 6 \ 10]$ .

The Inverse CDF is pretty much used in hypothesis testing to find critical values of a given known distribution. For example, consider testing the mean equal to zero in the vector  $X$  created above. First, we construct the appropriate statistics, and then find the critical values for  $\alpha = 0.05$  significance level. Since this is a two-tailed test, we should obtain the value that leaves  $\alpha/2 = 0.025$  of probability in each tail.

```
>> zstat = mean(X)./std(X)
>> zstat =
-0.0457
>> zcritic = norminv(0.025)
>> zcritic =
-1.96
```

Note that the command `std`, by default, calculates the sample-corrected standard deviation. If you add an optional input you can change it, see the Help. Of course, we cannot reject the null of zero mean provided that we have generated  $X$  as a Normal random variable with zero mean!!

Finally, let's play a little more with the random number generators. We want to generate time series of size  $T = 1000$  with the following dynamic properties:

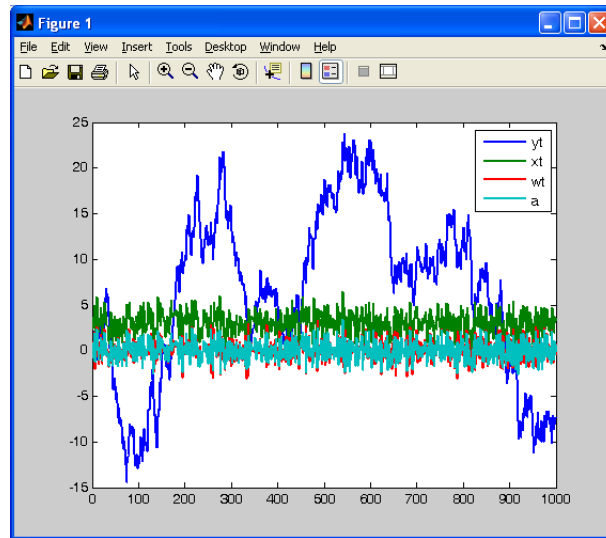
$$y_t = y_{t-1} + a_t, \quad (1)$$

$$x_t = 3 + 0.5 a_{t-1} + a_t, \quad (2)$$

$$w_t = -0.8 w_{t-1} + a_t + 0.5 a_{t-1}, \quad (3)$$

where  $a_t$  is an i.i.d. Gaussian process with zero mean and unit variance, commonly known as a white noise process. Since all series depend on their own past values, we need to construct the following loop to generate them:

```
>> close all % Closing all figures without saving them
>> T =1000; % Sample size
>> a = normrnd(0,1,T,1); % Generating the white noise process
>> yt = zeros(T,1); xt = 3*ones(T,1); wt = zeros(T,1); % Generating the empty vectors
>> for t = 2:T, % Starting the loop at t=2
>>     yt(t) = yt(t-1) + a(t); % Generating yt
>>     xt(t) = 3 + 0.5 * a(t-1) + a(t); % Generating xt
>>     wt(t) = -0.8 * wt(t-1) + 0.5 * a(t-1) + a(t); % Generating wt
>> end % Finishing the loop
>> figure(1),plot([yt xt wt a],'LineWidth',1.5); % Plotting the series against time
>> legend yt xt wt a % Adding the legend to the current figure
```



From the plot we can see that the only series without a constant mean is  $y_t$ , known as a random walk process. Check that its first difference given by  $\Delta y_t = y_t - y_{t-1}$ , is simply the white noise process,  $a_t$ . The command to do this in MATLAB is `diff(yt)`.

### 2.1.2 Descriptive Statistics

The *Statistics* toolbox provides functions for describing the features of a data sample. These descriptive statistics include measures of location and spread, percentile estimates and functions for dealing with data having missing values. The following table shows the most important ones with a brief description of their use.

Function Name	Description
corr	Linear or Rank Correlation Coefficient
corrcoef	Linear Correlation Coefficient with Confidence intervals
cov	Covariance Matrix
geomean	Geometric Mean
grpstats	Summary Statistics by Group
kurtosis	Kurtosis
mad	Median Absolute Deviation
median	50th Percentile of a Sample
moment	Moments of a Sample
nanstat	Descriptive Stastiscs ignoring NaNs (missing data)*
prctile	Percentiles
quantile	Quantiles
range	Range
skewness	Skewness

\* Replace the particle *stat* by one of the following names: *max*, *min*, *std*, *mean*, *median*, *sum* or *var*.

As an example, let's find some sample statistics of the time series  $x_t$ ,  $w_t$  and  $a_t$  generated above. Before doing it, we must know that in a data matrix, MATLAB automatically interprets columns as samples and rows as observations. Therefore, the output of all these functions applied to the data matrix is a row vector with its  $i^{th}$  element being the value of the statistics for the  $i^{th}$  column. Let's now calculate the median, skewness, kurtosis, and the 25<sup>th</sup> and 75<sup>th</sup> Percentile of the selected series by typing the following in the command window:

```
>> med = median([xt wt a]); % Calculating the median for each series
>> skew = skewness([xt wt a]); % Calculating the Skewness
>> kurt = kurtosis([xt wt a]); % Calculating the Kurtosis
>> prct25 = prctile([xt wt a],25); % Calculating the 25th Percentile
>> prct75 = prctile([xt wt a],75); % Calculating the 75th Percentile
>> med,skew,kurt,prct25,prct75,
>> med =
3.0026      -0.0438      -0.0001
>> skew =
-0.0854      0.0750      0.0311
>> kurt =
2.7254      2.8461      2.7629
>> prct25 =
2.2257      -0.7365      -0.6947
>> prct75 =
3.7530      0.7206      0.6734
```

Another way of obtaining the sample kurtosis or skewness is by using the command `moment(x,n)`, which finds the  $n^{th}$  central sample moment of  $x$ . For example, knowing that the kurtosis is defined as the ratio between the fourth central moment and the second central moment to the squares, we could have done

```
>> m4 = moment([xt wt a],4); % Calculating the 4th Central Moment
>> m2 = moment([xt wt a],2); % Calculating the 2nd Central Moment
>> kurtbis = m4./(m2).^2 % Calculating the Kurtosis
>> kurtbis =
2.7254      2.8461      2.7629
```

In the construction of `kurtbis`, we must remember adding the dot (`.`) before the mathematical operations, in order to ensure that we are dealing with element-by-element (scalar) operations.

### 2.1.3 Statistical plots

The graphical methods of data analysis are as important as the descriptive statistics to collect information from the data. In this section we will see some of these specialized plots available in the *Statistics* toolbox and also others from MATLAB main toolbox. The *Statistics* toolbox adds box plots, normal probability plots, Weibull probability plots, control charts, and quantile-quantile plots to the arsenal of graphs in MATLAB. There is also extended support for polynomial curve fitting and prediction. There are also functions to create scatter plots or matrices of scatter plots for grouped data, and to identify points interactively on such plots. It is true that the best way of plotting some data strongly depends on their own nature. Thus, for instance, qualitative data information might be summarized in pie charts or bars, while quantitative data in scatter plots. The table below shows some of the most known specialized graphs.

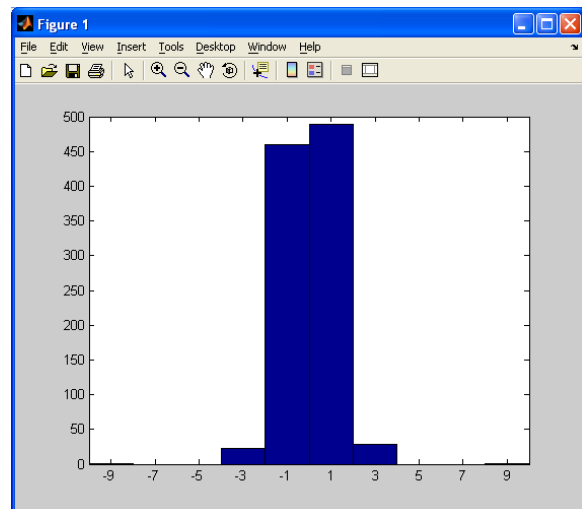
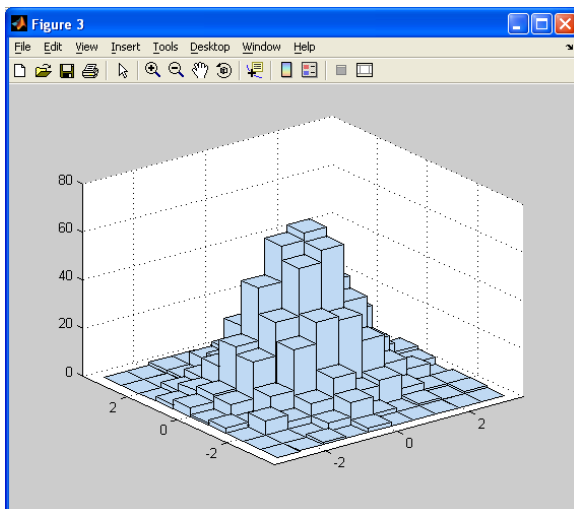
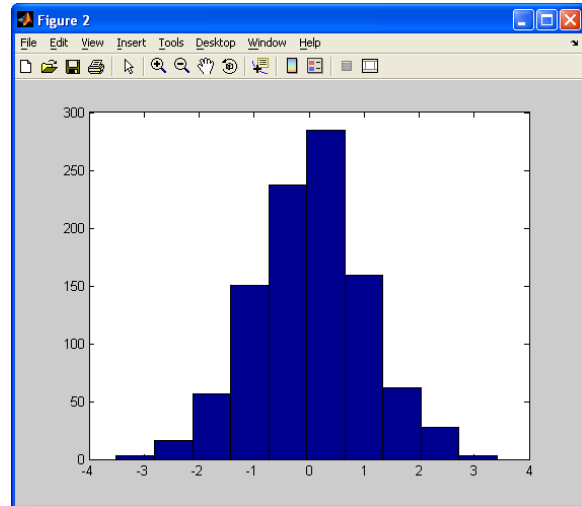
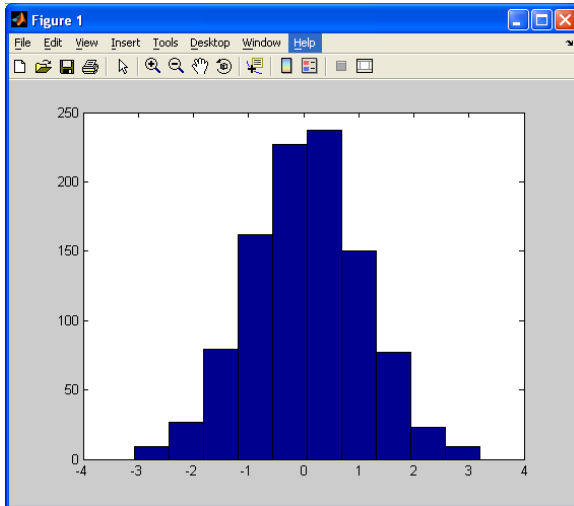
Statistical Plotting	Description
bar	Data plotting in vertical bars (in MATLAB toolbox)
barh	Data plotting in horizontal bars (in MATLAB toolbox)
boxplot	Boxplots of a data matrix (one per column)
cdfplot	Plot of empirical cumulative distribution function
ecdfhist	Histogram calculated from empirical cdf
gline	Point, drag and click line drawing on figures
gplotmatrix	Matrix of scatter plots grouped by a common variable
gscatter	Scatter plot of two variables grouped by a third
hist	Histogram (in MATLAB toolbox)
hist3	Three-dimensional histogram of bivariate data
ksdensity	Kernel smoothing density estimation
lsline	Add least-square fit line to scatter plot
normplot	Normal probability plot
parallelcoords	Parallel coordinates plot for multivariate data
pie	Pie chart of data (in MATLAB toolbox)
scatter	Scatter plot of two variables (in MATLAB toolbox)
stem	Plot of discrete data as lines (stems) with a marker
probplot	Probability plot
qqplot	Quantile-Quantile plot

Obviously, the histogram is the most popular in describing a sample. Let's find out how to use it. Consider the vector  $X$  of i.i.d. standard normal random variables, and the white noise series,  $a_t$ . We will obtain the histogram for each variable and a 3-D version of the histogram for both at the same time.

```

>> close all % Close all current figures
>> figure(1),hist(X) % Histogram of X
>> figure(2),hist(a) % Histogram of a
>> figure(3),hist3([X a]) % Histogram of the bivariate sample [X,a]
>> figure(4),hist([X;10;-10]) % Histogram of X contaminated with two outliers

```



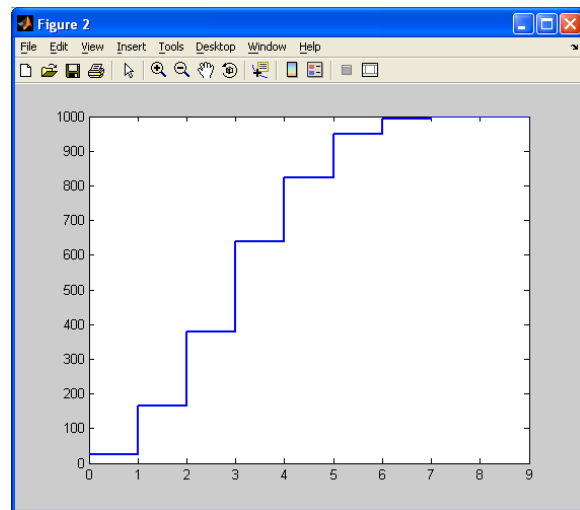
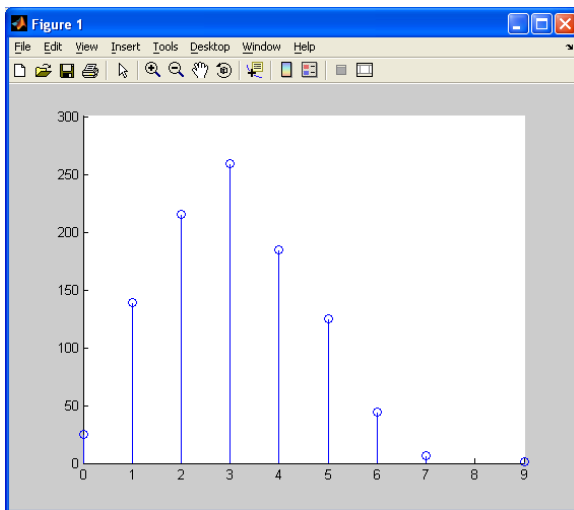
The histograms serve mainly for giving a first view of the shape of the distribution, and thus infer whether the data come from a given known distribution. Moreover, they also serve to detect outliers, as shown in the bottom-right figure above. This also shows the histogram of  $X$ , but after adding two “large” (in absolute value) observations. It is clear that these two outliers significantly change the shape of the histogram.

When dealing with discrete data, the `stem` function may be the appropriate graphic for plotting them. For example, let’s generate data from a binomial distribution with parameters  $p = 0.3$  and  $n = 10$ . First, we need to obtain the frequency table that counts the number observations for each value between 0 and 9. The right command is `tabulate`. Finally, the



commands `stem` and `stairs` produce the plots of the simple and cumulative frequencies given by `tabulate`.

```
>> close all % Close all current figures
>> B = binornd(10,0.3,1000,1); % Generating Binomial(10,0.3) data
>> Freq = tabulate(B) % Tabulate the frequencies of B
Freq =
0.0000    25.0000    2.5000
1.0000   139.0000   13.9000
2.0000   215.0000   21.5000
3.0000   259.0000   25.9000
4.0000   185.0000   18.5000
5.0000   125.0000   12.5000
6.0000    44.0000    4.4000
7.0000    7.0000    0.7000
9.0000    1.0000    0.1000
>> figure(1),stem(Freq(:,1),Freq(:,2)) % Stem plot of Freq
>> figure(2),stairs(Freq(:,1),cumsum(Freq(:,2))) % Stair plot of Freq
```



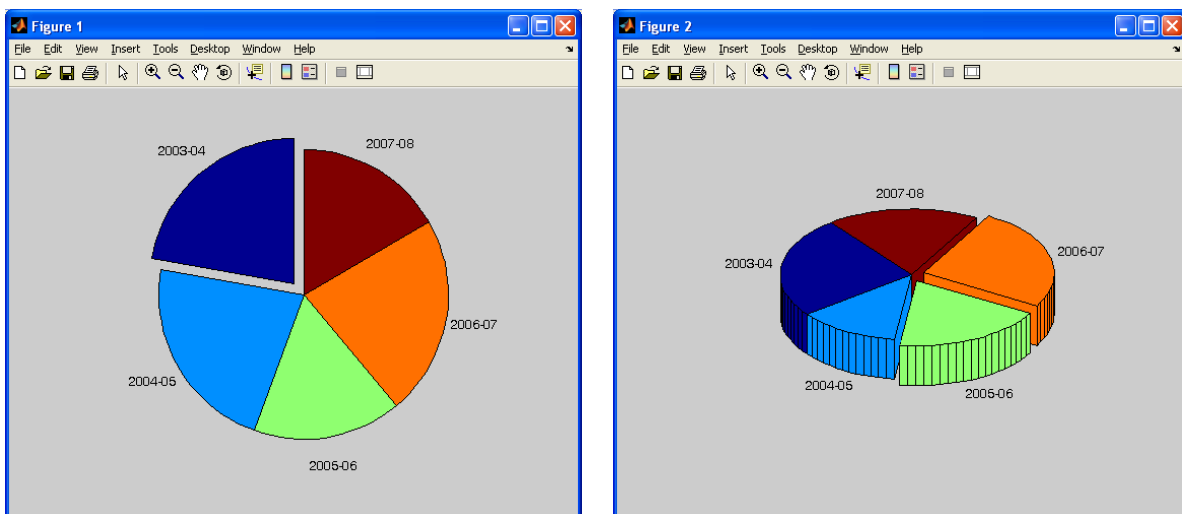
Finally, we will learn how to create the pie charts. Consider the following information about the number of students of the Statistics Department that have started and finished the first-year courses in the last four years:

Course	2003-04	2004-05	2005-06	2006-07	2007-08
Started	4	4	3	4	3
Finished	4	2	3	4	3

In order to introduce this table in MATLAB and then plot its figure, we must follow these steps:

```
>> Course = {'2003-04','2004-05','2005-06','2006-07','2007-08'}
>> Started = [4, 4, 3, 4, 3];
>> Finished = [4, 2, 3, 4, 3];

>> close all
>> figure(1), pie(Started,[1 0 0 0 0],Course),
>> figure(2), pie3(Finished,[0 0 1 1 0],Course),
```



Note that the vector of zeros and ones following the data input specifies whether to offset a slice from the center of the pie chart; see the Help for further information.

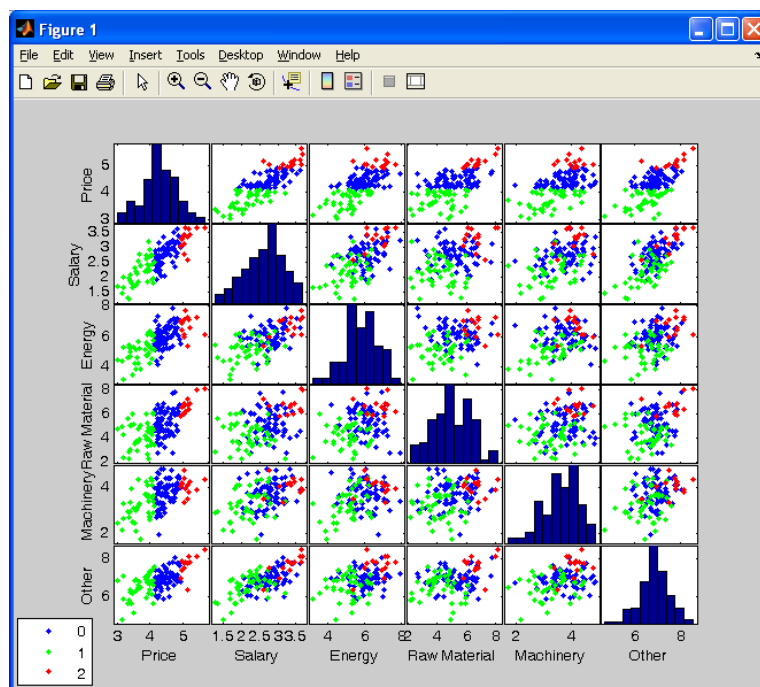
#### 2.1.4 Linear regression analysis

Regression analysis is other big branch of Statistics covered by the *Statistical* toolbox. For example, finding the Ordinary Least Squares (OLS) estimators of a given linear regression is certainly easy in MATLAB. In fact, if we knew the algebra behind this procedure we could use simple matrix algebra with the commands you have seen in previous sections. However, the *Statistics* toolbox goes beyond that and provides much more information for inference and testing with just a simple command named `regress`. In its simplest form, by typing `B = regress(Y,X)` we find the OLS estimators of the linear regression of the dependent variable  $Y$  on the regressors contained in matrix  $X$ . In other words, it estimates the vector  $\beta$  of the model

$$\begin{aligned} Y &= X\beta + U, \\ &= \beta_1 X_1 + \beta X_2 + \dots + \beta_p X_p + U, \end{aligned} \quad (4)$$

where  $U$  is a vector of zero-mean random errors. If we further assume that  $U$  is Normal and  $X$  is fixed (i.e. not random), then the OLS estimator,  $\hat{\beta}_{OLS}$ , is the best linear unbiased estimator (BLUE) of the parameter  $\beta$ . Let's see how the command `regress` works. First, we need to enter some data to apply model (4). We will work with production costs of several factories that produce steel. The data is located in a text-file called `regress_data.txt`, which contains seven variables with 129 observations. The variables are: *price per unit*, *hourly salaries*, *energy costs*, *raw material costs*, *machinery depreciation costs*, *other fixed costs* and a discrete variable for *factory type* (1=star, 0=neutral and 2=basic)<sup>2</sup>. All variables except the binary one are in logs. We wish to relate the unit price of steel with the different production costs by means of (4). First, remember that to load the data we should type `Data = load('regress_data.txt');` on the command window. Once the data have been loaded, it could be interesting to observe the cloud of points in a scatter plot. Since there is a categorical discrete variable referring to the geographical situation, we will use it to group the data into the three situations. Recall that the command to do this is `gplotmatrix` (the simple scatter plot matrix is `plotmatrix`). The command also allows to introduce the variable names and a histogram in the main diagonal, as shown in the figure below. For more information about the usage of this function; see the Help.

```
>> Data = load('regress_data.txt');
>> varNames = {'Price','Salary','Energy','Raw Material','Machinery',...
'Other','Situation'};
>> gplotmatrix(Data(:,1:6), [], Data(:,7), 'bgrcm', [], [], 'on', 'hist', varNames(1:6));
```

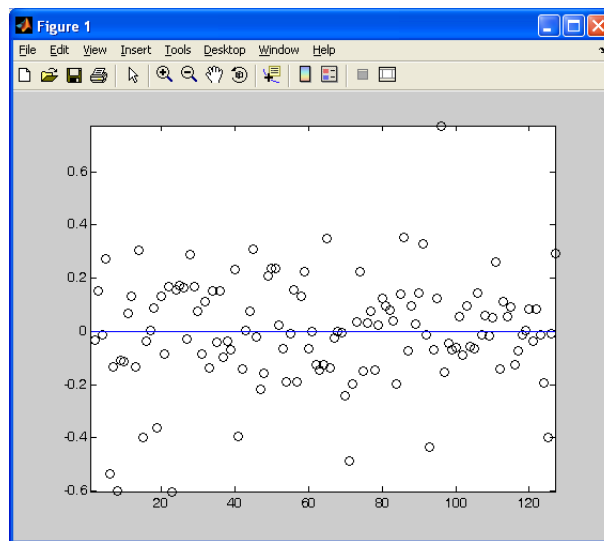
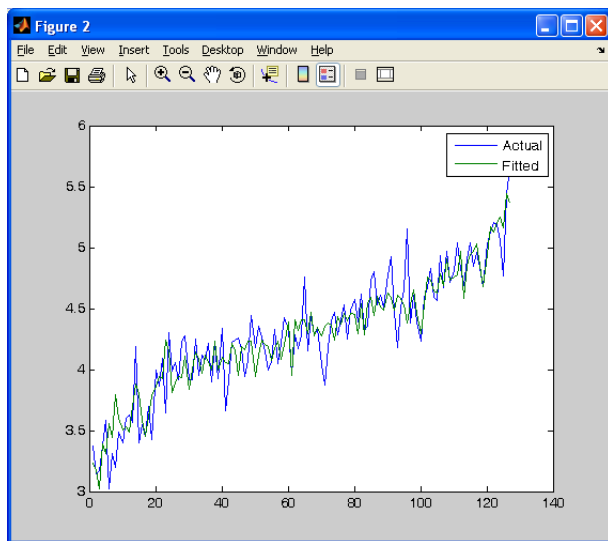


<sup>2</sup>This exercise is a modification of a practice of Statistics II. You will find all information [here](#)

The first row of the scatter plot provides a good insight that the linear model can be a good alternative for relating these variables. Then, we finally estimate the parameter  $\beta$  of (4) and the residual variance with the function `regress`. We first leave aside the categorical data. Note also that we must add a vector of ones to the regressors matrix in order to include an intercept.

```
>> [Bols Bolsint R RINT STATS] = regress(Data(:,1),[ones(127,1) Data(:,2:6)]);
>> close all,figure(1),plot(R,'ok'),hold on,line([1 127],[0 0]),axis tight
>> figure(2),plot([Data(:,1) [ones(127,1) Data(:,2:6)]*Bols])
>> legend Actual Fitted
```

Regressor	$\hat{\beta}_{OLS}$	95% Conf. interv. of $\hat{\beta}_{OLS}$	$R^2$	$F-stat$	$p-value$	$\sigma_u^2$
Constant	0.4826	(0.0234 0.9417)	0.8485	135.5559	0	0.0413
Salary	0.3550	(0.2422 0.4678)				
Energy	0.1647	(0.1147 0.2147)				
Raw Material	0.1206	(0.0903 0.1509)				
Machinery	0.1183	(0.0556 0.1811)				
Other Costs	0.1273	(0.0491 0.20544)				



Note that in the residual plot, the zero-line is also drawn. To include it, we need first to hold on the current figure and then add the command `line([x1 x2],[y1 y2])`.

## 2.2 The *Optimization* toolbox. (Applications)

Optimization is the process of finding the minimum or maximum of a function, usually called the objective function sometimes subject to constraints. The Optimization Toolbox consists of a set of functions that perform minimization on general linear and nonlinear functions. The toolbox

also provides functions for solving nonlinear equation and least-squares (data-fitting) problems. The Optimization toolbox includes routines for many types of optimization procedures like

- Unconstrained nonlinear minimization
- Constrained nonlinear minimization, including goal attainment problems, minimax problems, and semi-infinite minimization problems
- Quadratic and linear programming
- Nonlinear least squares and curve-fitting
- Nonlinear system of equation solving
- Constrained linear least squares
- Sparse and structured large-scale problems
- Linear programming

### 2.2.1 Minimizing unconstrained nonlinear functions

Given a mathematical function coded in an *m-file*, one can find a local minimizer of the function in a given interval using an iterative algorithm called `fminunc`. It attempts to find a minimum of a scalar function of several variables, starting at an initial estimate. Its syntax is as follows

```
>> [X,FVAL,EXITFLAG,OUTPUT,GRAD,HESSIAN] = fminunc('ObjFunction',X0,...  
options,varargin)
```

Inputs:

- `'ObjFunction'`: is a function that accepts input `X` and returns a scalar function value evaluated at `X`.
- `X0`: is a initial guess value for the iterative algorithm. It must be of the same size as `X`.
- `options`: is a structure containing all options about the way in which the algorithm is searching the minimum point, e.g. the maximum number of iterations, gradient, hessian, tolerance, among others. The following table summarizes the most common options.

Option	Possible Values	Description
Display	'off'   'iter'   'final'   'notify'	Level of display. 'off' displays no output; 'iter' displays output at each iteration; 'final' displays just the final output; 'notify' displays output only if the function does not converge.
FunValCheck	'off'   'on'	Check whether objective function values are valid. 'on' displays an error when the objective function returns a value that is complex or NaN. 'off' displays no error.
GradObj	'on'   'off'	Use the gradient provided by the user.
MaxFunEvals	Positive integer	Maximum number of objective function evaluation allowed.
MaxIter	Positive integer	Maximum number of iterations allowed.
TolFun	Positive scalar	The minimum tolerance for the objective function.
TolX	Positive scalar	The minimum tolerance for the norm of parameters.
Hessian	'on'   'off'	Use the hessian provided by the user.

- `varargin`: are the additional input that could have the objective function ('ObjFunction').

The number of Output may vary according to our need, and they can be:

- `X`: is the value of parameters that minimize the objective function.
- `FVAL`: is the value of the objective function at the optimal point, (`X`).
- `EXITFLAG`: is the value that describes the exit condition of the command `fminunc` reaching the solution `X`. It can be: 1 magnitude of gradient smaller than the specified tolerance; 2 change in `X` smaller than the specified tolerance; 3 change in the objective function value smaller than the specified tolerance (only occurs in the large-scale method); 0 maximum number of function evaluations or iterations reached; -1 algorithm terminated by the output function; -2 line search cannot find an acceptable point along the current search direction (only occurs in the medium-scale method).
- `OUTPUT`: is a structure with: the number of iterations taken, the number of function evaluations, the number of CG iterations (if used) in `OUTPUT.cgiterations`, the first-order optimality (if used), and the exit message.
- `GRAD`: is the value of the gradient of the objective function at the solution `X`.
- `HESSIAN`: is the value of the Hessian of the objective function at the solution `X`.

**Application:** Consider, again, model (4). But now, we are going to minimize the sum squares of the residuals numerically. We mean that first create a function in MATLAB that

returns the value of the sum squares of the residuals given a particular value of parameters and the data, then, we use the command `fminunc` for finding a minimum. As we have seen, the parameters can be estimated using the command `regress`.

Let `OLS_b` be the function that returns the value of the sum squares of the residuals given a particular value of the parameters and the data. Now, we enter

```
>> [par,fval,exitflag,output] = fminunc('OLS_b',Ini_Guess,[],X,y)
```

with `[]`, we left the default options and we add the input `X` and `y` which are the additional input of the `OLS_b` function. The results will be

Iteration	f(x)	Norm of step	First-order optimality	CG-iterations
0	5372.74		1.14e+004	
1	5372.74	10	1.14E+004	3
2	203.197	2.5	1.52E+003	0
3	111.26	1.04324	193	1
4	43.8013	2.1383	200	3
5	22.2694	1.53945	63.6	3
6	6.15335	0.587813	5.45	3
7	5.53869	0.335366	4.41	3
8	5.27467	0.198886	1.81	3
9	5.27467	2.5	1.81	3
10	5.18068	0.625	4.55	0
11	4.99806	0.0778814	0.9	3
12	4.99307	0.0149232	0.141	3
13	4.99278	0.00528417	0.095	3
14	4.99261	0.00522933	0.0904	3
15	4.99259	0.0123942	0.231	3
16	4.9923	0.00291178	0.0181	3
17	4.99228	0.001871	0.0324	3
18	4.99227	0.00107856	0.0166	3
19	4.99227	0.00104532	0.0338	3
20	4.99226	0.00060263	0.0071	3
21	4.99226	0.00058137	0.0133	3
22	4.99226	0.0010612	0.0141	3
23	4.99226	0.00022532	0.00192	3
24	4.99226	0.00056045	0.00353	3
25	4.99226	0.00048948	0.00627	3
26	4.99226	6.92E-05	0.000684	3
27	4.99226	4.91E-05	0.00077	3
28	4.99226	2.20E-05	0.000202	3
29	4.99226	2.74E-05	0.000414	3
30	4.99226	3.51E-05	0.000414	3
31	4.99226	8.77E-06	0.000117	0
32	4.99226	8.77E-06	0.000299	3
33	4.99226	2.19E-06	8.67E-05	3
34	4.99226	5.48E-07	8.67E-05	3

Optimization terminated: norm of the current step is less than OPTIONS.TolX.

par =

```
0.4826 0.3550 0.1647 0.1206 0.1183 0.1273
```

```

fval =
    4.9923
exitflag =
    3
output =
    iterations: 40
    funcCount: 41
    stepsize: 1
    firstorderopt: 1.8323e-005
    algorithm: 'large-scale: trust-region Newton'
    message: [1x86 char]

```

### 2.2.2 Minimizing constrained nonlinear functions.

Now, we concern about finding an optimal point of a nonlinear function with several variables but, in this case, we add linear and nonlinear, constraints. The problem is the following

$$\begin{aligned}
 & \min_X f(X) \\
 & \quad s.t. \\
 & \quad AX = b \\
 & \quad C(X) = 0 \\
 & \quad A_{in}X \leq b_{in} \\
 & \quad C_{in}(X) \leq 0 \\
 & lb \leq X \leq ub
 \end{aligned}$$

A command for solving this type of problems is `fmincon`. It works in a similar way that the command `fminunc`, but the command `fmincon` has more inputs, that refer to the constraints.

```
>> [X,FVAL,EXITFLAG,OUTPUT,LAMBDA,GRAD,HESSIAN] = fmincon('ObjFunction',X0,...
Ain,bin,A,b,lb,ub,'NonConFunc',Options,Varargin)
```

Inputs:

- 'ObjFunction': is a function that accepts input  $X$  and returns a scalar function value  $F$  evaluated at  $X$ .
- $X0$ : is an initial guess value for the iterative algorithm, it can be a scalar, vector or matrix.
- $Ain, bin$ : are the matrix and vector, respectively, that form, together with “ $X$ ”, the linear



inequality constraints. (Set `Ain=[]` and `bin=[]` if no inequalities exist.)

- `A, b`: are the matrix and vector, respectively, that form, together with “`X`”, the linear equality constraints. (Set `A=[]` and `b=[]` if no equalities exist.)
- `lb, ub`: are the vector of lower bounds and the upper bounds, respectively, of the variables “`X`”. (Set `lb=[]` and/or `ub=[]` if no bounds exist.)
- `'NonConFunc'`: is a function that accepts “`X`” as input and returns the vectors  $C_{in}(X)$  and  $C(X)$ , representing the nonlinear inequalities and equalities constraints, respectively.
- `options`: is a structure in which are all options about the way in which the algorithm is looking for the minimum point, e.g. the maximum number of iterations, the gradient, the hessian, the tolerances among others.
- `varargin`: are the additional input that could have the objective function.

The number of Output's can vary according our need and they can be:

- `X`: is the value of parameters that minimize the function `'ObjFunction'`.
- `FVAL`: is the value of the objective function at the optimal points, (`X`).
- `EXITFLAG`: is value that describes the exit condition of the command `fminunc` reaching the solution `X`'s values. It can be: Both medium- and large-scale: 1 First order optimality conditions satisfied to the specified tolerance. 0 Maximum number of function evaluations or iterations reached. -1 Optimization terminated by the output function. Large-scale only: 2 Change in `X` less than the specified tolerance. 3 Change in the objective function value less than the specified tolerance. Medium-scale only: 4 Magnitude of search direction smaller than the specified tolerance and constraint violation less than `options.TolCon`. 5 Magnitude of directional derivative less than the specified tolerance and constraint violation less than `options.TolCon`. -2 No feasible point found.
- `OUTPUT`: is a structure with five blocks of information. Namely, the number of iterations taken, the number of function evaluations, the number of CG iterations (if used) in `OUTPUT.cgiterations`, the first-order optimality (if used), and the exit message.
- `LAMBDA`: is the Lagrange multipliers at the solution `X`. `LAMBDA.lower` for `lb`, `LAMBDA.upper` for `ub`, `LAMBDA.ineqlin` is for the linear inequalities, `LAMBDA.eqlin` is for the linear equalities, `LAMBDA.ineqnonlin` is for the nonlinear inequalities, and `LAMBDA.eqnonlin` is for the nonlinear equalities.
- `GRAD`: is the value of the gradient of the objective function at the solution `X`.

- **HESSIAN**: is the value of the Hessian of the objective function at the solution  $\mathbf{X}$ .

**Application:** Analyzing Portfolios

Portfolio managers concentrate their efforts on achieving the best possible trade-off between risk and return. For portfolios constructed from a fixed set of assets, the risk/return profile varies with the portfolio composition. Portfolios that maximize the return, given the risk, or, conversely, minimize the risk for the given return, are called optimal.

Suppose that we have money to invest on two different assets. *Asset*<sub>1</sub> has a mean return  $r_1 = 0.07$  and a standard deviation  $\sigma_1 = 3$ ; while *Asset*<sub>2</sub> has a mean return  $r_1 = 0.3$  and a standard deviation  $\sigma_2 = 5$ . The correlation between both assets is  $\rho = 0.1$ . In this virtual economy we must invest the totality of our money. Given this information our problem reduces to minimize the risk ( $\sigma_P$ ) of our portfolio given a particular value of the return.

The portfolio is composed as follows, the return

$$r_P = w_1 r_1 + w_2 r_2,$$

where,  $w_1$  and  $w_2$  are the weights invested in *Asset*<sub>1</sub> and *Asset*<sub>2</sub>, respectively. Its risk

$$\sigma_P = \sqrt{w_1^2 \sigma_1^2 + w_2^2 \sigma_2^2 + 2w_1 w_2 \sigma_1 \sigma_2 \rho}.$$

Hence, the problem is

$$\begin{aligned} \min_{w_1, w_2} f(w_1, w_2) &= \sqrt{w_1^2 \sigma_1^2 + w_2^2 \sigma_2^2 + 2w_1 w_2 \sigma_1 \sigma_2 \rho} \\ & \text{s.t.} \\ w_1 r_1 + w_2 r_2 &= r_0 \\ r_1 + r_2 &= 1 \\ lb \leq w_1 \leq ub \\ lb \leq w_2 \leq ub \end{aligned}$$

We present two situations, with and without short-selling.

**Without short-sale operations**

An interesting optimal point is such that minimizes the risk. That is

$$\begin{aligned} \min_{w_1, w_2} f(w_1, w_2) &= \sqrt{w_1^2 \sigma_1^2 + w_2^2 \sigma_2^2 + 2w_1 w_2 \sigma_1 \sigma_2 \rho} \\ &s.t. \\ w_1 + w_2 &= 1 \\ 0 \leq w_1 &\leq 1 \\ 0 \leq w_2 &\leq 1 \end{aligned}$$

We are looking for the weights that make minimum the risk without the possibility of take loan.

Now, we find the optimal weights entering in the Command Window the following,

first the options

```
opt = optimset('Display','iter','GradObj','off',...
'Hessian','off','LargeScale','off');
```

After that

```
>> [optweights,fval,exitflag,output,lambda] = fmincon('risk',[0.1 0.1],...
[],[],[1 1],1,[0 0],[1 1],[],opt,sig1,sig2,ro)
```

The results will be

Iter	F-count	f(x)	max constraint	Line search steplength	Directional derivative	First-order optimality Procedure
0	3	2.81759	0.01206			Infeasible start point
1	6	4.33549	0	1	3.26	7.16
2	9	2.69701	1.11E-16	1	-0.343	5.47
3	12	2.68879	0	1	0.0396	5.61
4	15	2.68058	0	1	2.95E-05	5.35
5	18	2.68058	0	1	1.61E-10	5.36

Optimization terminated: magnitude of directional derivative in search direction less than 2\*options.TolFun and maximum constraint violation is less than options.TolCon.

optweights =

```
0.7581    0.2419
```

fval =

```
2.6806
```

exitflag =

```
5
```

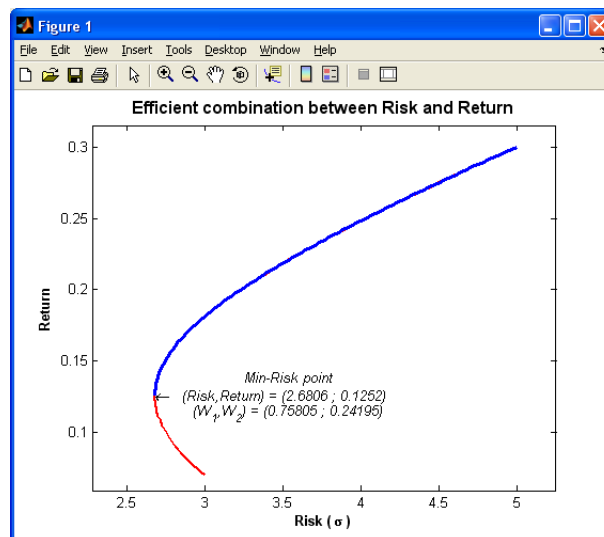
output =

```

iterations: 5
funcCount: 18
lssteplength: 1
stepsize: 4.7398e-004
algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
firstorderopt: 1.3411e-006
message: [1x172 char]
lambda =
lower: [2x1 double]
upper: [2x1 double]
eqlin: -2.6806
eqnonlin: [0x1 double]
ineqlin: [0x1 double]
ineqnonlin: [0x1 double]

```

The following figure shows the situation



### With short-sale operations

In this case, it is possible to borrow money, let assume that the interest rate at which we take the loan is the same as  $Asset_1$ . For example, we can borrow money by means of selling  $Asset_1$  (“leverage”) and using it for buying  $Asset_2$ , this situation can be summarized as a negative weight for  $Asset_1$  and a positive and greater than one weight for  $Asset_2$  (their sum must be equal to one).

Now, our problem consists in minimizing the risk subject to the level of return being fixed,

e.g.  $r_0 = 0.6$ .

$$\begin{aligned} \min_{w_1, w_2} f(w_1, w_2) &= \sqrt{w_1^2 \sigma_1^2 + w_2^2 \sigma_2^2 + 2w_1 w_2 \sigma_1 \sigma_2 \rho} \\ & \text{s.t.} \\ w_1 r_1 + w_2 r_2 &= 0.6 \\ w_1 + w_2 &= 1 \end{aligned}$$

Now, we find the optimal weights entering in the Command Window, first the options

```
opt = optimset('Display','iter','GradObj','off',...
'Hessian','off','LargeScale','off');
```

After that, we enter

```
>> [optwieghts,fval,exitflag,output,lambda] = fmincon('risk',[0.1 0.1],...
[],[],[r1 r2;1 1],[R0;1],[],[],[],opt,sig1,sig2,ro)
```

The results will be

Iter	F-count	f(x)	max constraint	Line search steplength	Directional derivative	First-order optimality Procedure
0	3	0.6205	0.8			Infeasible start point
1	6	11.599	1.11E-16	1	11.2	11.3
2	9	11.599	0	1	-2.55E-15	9.43

Optimization terminated: first-order optimality measure less than options.TolFun and maximum constraint violation is less than options.TolCon.

```
optwieghts =
```

```
    -1.3043     2.3043
```

```
fval =
```

```
    11.5990
```

```
exitflag =
```

```
     1
```

```
output =
```

```
    iterations: 2
```

```
    funcCount: 9
```

```
    lssteplength: 1
```

```
    stepsize: 6.8265e-016
```

```
    algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
```

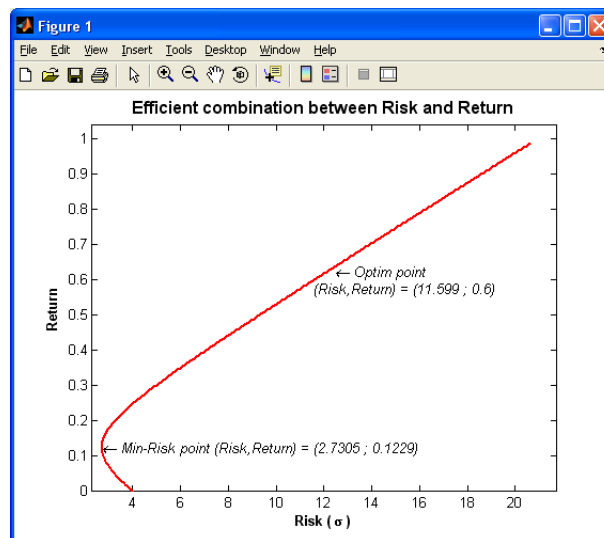
```
    firstorderopt: 1.9984e-015
```

```
    message: [1x143 char]
```

```
lambda =
```

```
lower:      [2x1 double]
upper:      [2x1 double]
eqnlin:     [2x1 double]
eqnonlin:   [0x1 double]
ineqlin:    [0x1 double]
ineqnonlin: [0x1 double]
```

The following figure shows the situation



### 3 Programming in MATLAB

MATLAB is a high-level language that includes matrix-based data structures, its own internal data types, an extensive catalog of functions, an environment in which to develop our own functions and scripts. MATLAB provides a full programming language that enables us to write a series of MATLAB statements into a file and then execute them with a single command. We write our program in an ordinary text file, giving the file a name of `filename.m`. The term we use for `filename` becomes the new command that MATLAB associates with the program. The file extension of `.m` makes this a MATLAB *m-file*. For example, when we write a program in MATLAB, we save it to an *m-file*. There are two types of *m-files*: the *scripts* that simply execute a sequence of MATLAB statements, and the *functions* that also accept input arguments and produce outputs.

### 3.1 M-files (Scripts)

The simplest *m-files* are the *Scripts*. They are useful for automating blocks of MATLAB commands, such as computations we have to perform repeatedly from the command line. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, they record all the created variables in the workspace, so that we can use them in further computations. In addition, scripts can produce graphical output using commands like `plot`.

Like any other *m-files*, scripts accept comments. Any text followed by a percent sign (%) on a given line is a comment. Comments may appear on separate lines by themselves, or simply at the end of an executable line.

For example, let's create an *m-file* to construct the plot made in section 1. First, in “*File*”, “*New*”, click on “*M-File*”. This action opens a new window in which we can edit the *m-file*. Next, inside this window we enter all the necessary commands.

```
clear % Clear all variables from memory
clc %Clear the screen
x = -pi:0.1:pi; % Define a variable 'x'
y = sin(x); % Define a function of the variable 'x'
y1 = cos(x); % Define other function of the variable 'x'
gr = plot(x,y); % Create a first plot
hold on
gr1 = plot(x,y1,'g'); % Create a second plot using the command hold on
hold off
set(gcf,'Color','w'); % Set the color of the background the current figure
set(gr,'LineStyle','--','LineWidth',2.5, 'Color', 'r'); % Set the styles, color and width for the first plot
set(gr1,'LineStyle','-','LineWidth',2, 'Color','g'); % Set the styles, color and width for the second plot
set(gca,'XTick',-pi:pi/2:pi,'XTickLabel',{'-pi','-pi/2','0', 'pi/2','pi'}, 'YTick',-1:0.5:1);
% Set the marks for the axes
text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi/4)','HorizontalAlignment','left'); % Insert a text inside the graph
text(-pi/4,cos(-pi/4),'\leftarrow cos(-\pi/4)', 'HorizontalAlignment','left');
title('Trigonometric functions','FontWeight','bold','FontSize',13); % Insert a title
legend('sin(\theta)','cos(\theta)'); % Add legends
xlabel('-\pi < \theta < \pi','FontWeight','bold','FontSize',11); % Label for the x axe
ylabel(['sin(\theta)',',sprintf('\n'),'cos(\theta)'],'Rotation',0.0,'FontWeight','bold','FontSize',11);
```

### 3.2 Functions

Function `[out1, out2, ...] = funname(in1, in2, ...)` defines the function `funname` that accepts inputs `in1`, `in2`, etc. and returns outputs `out1`, `out2`, etc. The name of a function, defined in the first line of the *m-file*, should be the same as the name of the file without the “.m” extension. For example, we create a function with the weights, the standard deviations associated to each assets and the correlation between each other as input and returns the risk, measured as the standard deviation of the portfolio.

```
function [R Var] = risk(weights,sig1,sig2,ro)
% [R Var] = risk(weights,sig1,sig2,ro)
% weights(1): the weight associated to the first asset
% weights(2): the weight associated to the second asset
% sig1: the Standard Deviation associated to the first asset
% sig2: the Standard Deviation associated to the second asset
% ro: the correlation between the assets.
w1 = weights(1);
w2 = weights(2);
sig12 = sig1^2;
sig22 = sig2^2;
Var = (w1^2)*sig12 +(w2^2)*sig22 + ...
      2*w1*w2*sqrt(sig12)*sqrt(sig22)*ro;% The Variance of the portfolio
R = sqrt(Var); % The Risk of the portfolio
```

### 3.3 Exercises and applications

Now we leave you constructing your own *m-files* to solve some exercises we have prepared specially for you!

1. A Chinese factory specialized in producing plastic dolls wishes to analyze some data on exports to Spain. The database is given in a text file called `chinese_data.txt`. At the end of this tutorial you will find the whole database with all the information you need. It contains information about the 2005 and 2006 sales of three different types of dolls, A, B and C. In particular, the share-holders wish to know which is the type of doll with best position at the Spanish market, and how these sales are spanned among the provinces. As their head of the sales department, you need to summarize the information by
  - (a) Obtaining the main descriptive statistics, for each year and type of doll.



- (b) Making different plots to have an intuition about the type of data we have and to account for possible outliers.
- (c) Testing whether the mean sales of each type are equal for a given year.
- (d) Checking, for three provinces chosen **at random**, whether the sales of each type have increased from 2005 to 2006. Then indicate in how many provinces the sales of type B have been increased.

We encourage you to put all the statements in a script *m-file* and then run them all at once to check the results. Don't worry, we will help you on this!!

2. This Chinese factory is in fact part of a big holding of multinational firms. The CEOs of this holding, after your excellent performance as the head of the sales department, offer you a position at their financial department. But, of course, to get this job they ask you to find the optimal portfolio composition of four different indexes, according to their expected returns. In the file `holding_shares.txt` you will find information about the daily returns of these four different indexes in the last two months. In particular, you need to:
  - (a) Find the mean and the variance-covariance matrix of these returns.
  - (b) Use this information as inputs to minimize the risk of the portfolio (measured as its standard deviation), subject to a fixed value of the portfolio return.
  - (c) Construct a loop to repeat the last minimization but with different portfolio returns.
  - (d) Plot the efficient frontier and compare it to the free risk return,  $r_f = 0.01\%$

Let's find out which of you deserves most this position!!

Province	Code	Type A Sales		Type B Sales		Type C Sales	
		2005	2006	2005	2006	2005	2006
Almería	1	303.8167	138.5333521	302.721389	240.784593	262.525503	73.8732537
Cádiz	2	464.5854	233.831308	465.984023	545.597393	517.66098	159.547614
Córdoba	3	303.4283	154.0214673	311.9839	811.220536	636.252724	142.47686
Granada	4	344.3596	161.7446758	344.899332	375.578128	364.81213	96.2035294
Huelva	5	209.0394	111.0995321	208.511201	178.610695	189.105729	45.6400125
Jaén	6	232.9159	123.7788657	227.792121	56.5949524	43.3245178	10.5630605
Málaga	7	631.4482	315.0965665	636.361654	917.378961	818.793071	204.232977
Sevilla	8	745.975	379.4407969	719.62946	722.723866	215.593884	59.6190273
Huesca	9	97.4746	49.1871286	99.3148782	205.725804	168.415217	46.3196872
Teruel	10	64.6877	32.02463933	67.4636531	231.764634	174.221687	45.2794701
Zaragoza	11	451.3192	234.3174915	443.066886	16.7036216	144.805528	40.740219
Asturias	12	428.7349	207.700932	438.457614	1002.75256	804.931532	223.697194
Balears (Illes)	13	525.1434	292.874503	542.548913	1563.08342	1205.49974	254.45755
Palmas (Las)	14	481.7652	239.1457546	461.471868	641.699706	253.682516	63.0471577
S.C.Tenerife	15	442.7193	222.6924456	425.857505	494.633492	170.941365	48.2352585
Cantabria	16	256.2794	135.3658683	260.147852	483.00351	404.849838	101.653321
Ávila	17	69.7995	35.00915026	69.06174	27.6350553	42.1817102	11.8033682
Burgos	18	167.2275	83.19120112	170.277277	346.51426	284.718821	81.3336798
León	19	198.7539	95.00468052	203.03844	451.435668	364.352052	93.6024425
Palencia	20	68.6278	34.84790983	69.9191098	144.582231	118.403214	26.616826
Salamanca	21	132.0427	61.35800838	136.545501	400.802011	308.213706	76.0058053
Segovia	22	67.5408	35.04554102	68.7343591	137.668048	113.496593	26.4047709
Soria	23	42.6622	23.16186945	44.0049447	122.606577	95.0632881	26.700042
Valladolid	24	248.0758	119.5018657	240.191178	193.05366	40.7540921	10.6846656
Zamora	25	71.4929	37.36774323	71.4554855	69.3332576	70.0780814	18.005544
Albacete	26	168.9492	90.07430365	168.24946	128.702425	142.58451	40.2182316
Ciudad Real	27	204.7915	93.76934453	196.108801	275.758391	109.786628	31.2752332
Cuenca	28	79.2367	36.60047166	79.775562	110.56494	99.7629521	22.4853302
Guadalajara	29	99.7914	51.40201792	97.0048042	56.7235592	2.69417559	0.67274635
Toledo	30	282.2868	138.1597124	292.84498	914.452377	696.691731	186.051911
Barcelona	31	2580.3633	1337.240501	2525.45449	524.410624	547.119892	125.128865
Girona	32	360.8789	188.2189212	365.921821	656.116121	554.341767	135.192101
Lleida	33	201.6594	104.62415	202.828276	269.548637	246.139641	67.9742358
Tarragona	34	365.8116	195.3803784	356.89801	136.638403	36.7873246	7.89431022
Alicante	35	787.27	407.5473804	742.101582	1674.21827	823.609905	199.600431
Castellón	36	269.8087	143.3962749	269.386528	245.465005	253.861018	53.5421812
Valencia	37	1156.5687	541.5249945	1125.67556	580.73602	18.9609822	4.169931
Badajoz	38	256.3402	128.0359492	260.503618	500.622827	416.419673	111.63435
Cáceres	39	156.3133	77.50924463	158.41086	279.048651	236.738494	67.2683041
Coruña (A)	40	499.6348	228.6340088	521.984262	1847.09351	1383.04096	348.425098
Lugo	41	136.9351	69.39879925	139.0752	262.427948	219.170266	56.7076858
Ourense	42	133.5857	63.06257509	131.313286	4.54343971	49.0724689	13.3622898
Pontevedra	43	426.075	228.9736919	430.35776	675.855344	589.740056	139.675636
Madrid	44	3036.5721	1453.669347	2955.58252	1518.13496	54.0937837	11.3952698
Murcia	45	639.0084	328.4337021	638.679054	619.997692	626.554158	165.688781
Navarra	46	289.9437	146.6524474	289.574321	268.638098	275.986218	75.173583
Álava	47	153.1765	72.85585736	153.1765	153.1765	153.1765	39.2912921
Guipúzcoa	48	330.434	146.2588523	327.657571	171.414058	226.271658	52.6028063
Vizcaya	49	510.8332	254.6172956	525.617631	1388.94459	1086.38451	296.123695
Rioja (La)	50	146.4271	69.3023311	139.174355	252.044757	114.385641	32.2283142
Ceuta	51	22.8769	11.81001126	23.1358138	37.995947	32.7836555	8.29980726
Melilla	52	21.8273	11.20654945	22.3439845	52.3608934	41.8385726	10.5318538

Daily returns in percentages				
Date	Nasdaq 100	S&P 500	FTSE	IBEX 35
02/07/2007	0.565405709	0.213522245	1.078999626	0.435793586
03/07/2007	0.803803368	0.148171917	-0.678011393	0.362421369
05/07/2007	0.36529767	0.469090714	0.97672173	1.007847277
06/07/2007	0.044751617	-0.008502726	0.387202016	-0.508126907
09/07/2007	-0.874007417	-1.519625119	-0.696372933	-0.911838568
10/07/2007	0.616285694	0.484825138	-0.457148446	-0.389935406
11/07/2007	1.845195182	1.648329932	1.906960943	1.2065808
12/07/2007	0.549198433	0.298224576	0.448532736	0.545079571
13/07/2007	-0.203931916	-0.413371554	0.110794242	0.577134035
16/07/2007	0.676202881	-0.018239882	-0.798818379	0.290023526
17/07/2007	-0.186285522	-0.20948219	-1.606535568	-0.630635537
18/07/2007	0.733814589	0.564299426	1.310508937	0.982774135
19/07/2007	-0.836910925	-1.391209152	-1.335718492	-1.839836455
20/07/2007	0.022101021	0.206847858	0.791063894	0.844517682
23/07/2007	-1.772702531	-2.108553109	-1.632920941	-0.957182339
24/07/2007	0.527958272	0.104138953	-2.325760998	0.163259144
25/07/2007	-1.227709067	-2.510894093	-2.4700203	-2.695332237
26/07/2007	-1.542593089	-1.651890391	-1.030050108	0.323382829
27/07/2007	0.902777548	1.146225849	0.340799863	-0.547510203
30/07/2007	-2.143968818	-1.209118429	1.787980196	2.009923603
31/07/2007	0.671631612	0.487708309	-1.537605621	-1.19946408
01/08/2007	1.100305603	0.518203421	0.74738431	0.566221085
02/08/2007	-2.473125871	-2.963263435	-0.604594075	-0.987272522
03/08/2007	1.849298622	2.121994394	0.242721268	-0.705861295
06/08/2007	0.37129672	0.479997438	0.592078544	1.51832957
07/08/2007	1.301620309	1.205104422	1.598946191	2.369223789
08/08/2007	-2.578058736	-2.857509344	-2.775443431	-1.111726144
09/08/2007	-0.601778144	0.264065253	-1.780864345	-2.625300125
10/08/2007	0.480367806	0.163009645	1.296471579	1.904950734
13/08/2007	-1.725398863	-2.148451584	-0.903266913	-1.220869677
14/08/2007	-1.933022506	-1.701793017	-0.541341024	-0.223901335
15/08/2007	-1.014826778	0.24482203	-3.070894112	-3.795604863
16/08/2007	2.286122904	2.31014938	2.205048173	1.827362545
17/08/2007	0.225816698	0.159609032	0.174688593	0.221741799
20/08/2007	0.94321474	0.314911866	0.405661474	-0.205414008
21/08/2007	1.340020467	1.325069093	1.342698486	0.834890778
22/08/2007	-0.252801499	-0.309945789	0.63213753	-0.466240162
23/08/2007	1.515468507	1.140389566	0.436335036	0.293243756
24/08/2007	-0.710694316	-1.108675717	-0.24142722	-0.201842096
27/08/2007	-2.508755781	-2.471118025	-0.818018854	-1.271272862
28/08/2007	2.8859658	2.272446475	0.146566426	0.415957912
29/08/2007	0.454244644	-0.44701085	1.04675445	0.910642658
30/08/2007	1.264033346	1.180409769	1.542185377	1.156168019