An introductory course in MATLAB: MATLAB for beginners

Alba M. Franco-Pereira

Master in Business Administration and Quantitative Methods

September 2010

Contents

1 The Basics of MATLAB

The name MATLAB is an abbreviation for "Matrix Laboratory". As its name implies, this software package is designed for efficient vector and matrix computations. The interface follows a language that is designed to look like the notation used in linear algebra. The primary data type in MATLAB is an $N \times M$ two-dimensional array with integer, real, or complex elements. Of course, $N \times 1$ and $1 \times M$ arrays are vectors, and a 1×1 array represents a scalar quantity. MATLAB can also produce both planar plots and 3-D mesh surface plots. The syntax is simple and straightforward, and it is similar to C/C++ and FORTRAN. In general, it is easier to program in MATLAB than in C or FORTRAN, although MATLAB is usually slower. MATLAB is an interactive, high-level, high-performance matrix-based system for doing scientific and technical computation and visualization. It contains a wide range of basic built-in functions and also various specialized libraries (Toolboxes). These notes will only refer to the Statistics Toolbox and are based on the MATLAB 7.6.0 (R2008a) version, but most of the contents are applicable to any previous version.

1.1 The MATLAB Environment

To run MATLAB, click on the MATLAB icon and the **Command Window** will be automatically opened. This is where you can type, for example, the basic commands to compute simple operations, as shown below. From now on, the symbol >> represents a command line.

```
>> 9*8
>> 9*(1-4/5)
>> cos(pi/2)^2
```

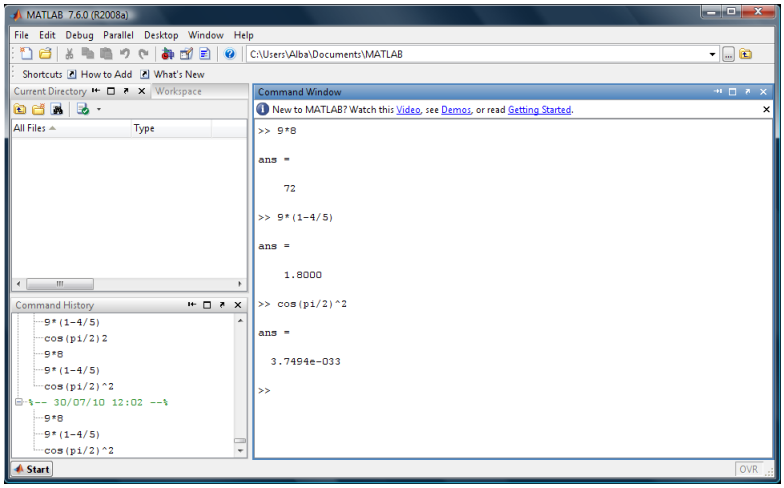

Figure 1: The MATLAB Environment

See Figure 1. On the left side of the Command Window, there are two sub-windows: either the **Current Directory** or the**Workspace** (clicking on the corresponding tab) and the **Command History**. 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. The Command History contains a list of commands that have been executed within the Command Window.

MATLAB 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 to change the current directory is by using the current directory field in the desktop toolbar. See Figure 2.

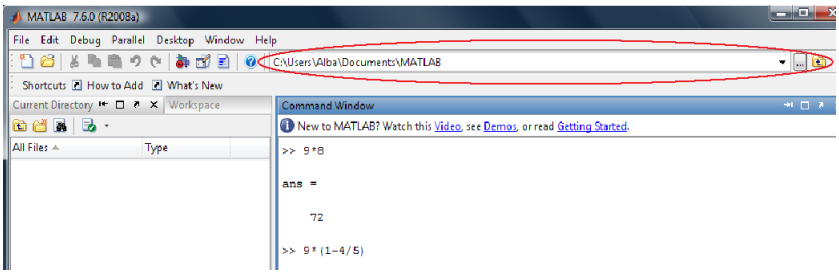

Figure 2: The Current Directory

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 Figure 3.

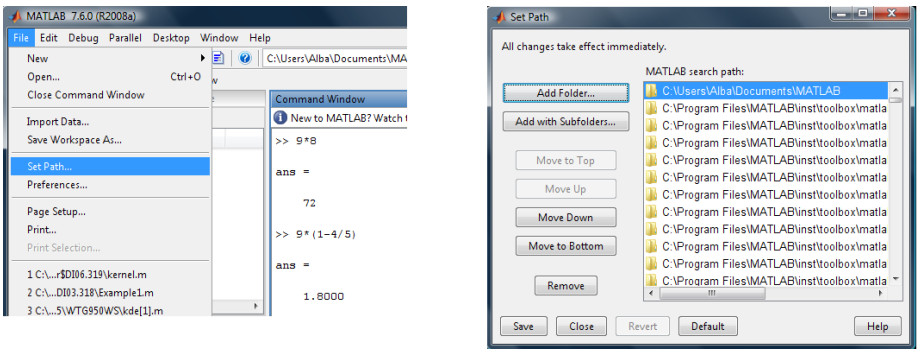

Figure 3: The Set Path

The *m-files* (named after its .m file extension) have a special importance. They are text (ASCII) files that contain a set of commands or functions. After typing the name of one of these *m-files* in the Command Window and pressing *Intro*, all commands and functions defined in the file are sequentially executed. MATLAB has its own *m-file* editor, which allows us to create and modify these files. Figure 4 shows how this editor looks like.

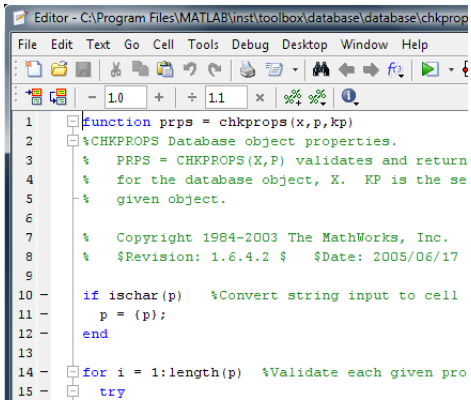

Figure 4: An m-file

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 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 opens a new window, the "Help" windows, where the help command is shown.

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.

A vector is defined by placing a sequence of numbers within square brackets. For example, to create a row vector called a with elements $\{1, 2, 3, 4\}$, we enter

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

and the result is

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

If you write a semi-colon (;) at the end of the line, the result will not be printed out. To visualize the vector, just type its label. You can view the individual entries of the vector. For example,

```
>> a (1)
```

and the result will be

```
ans =
        1
```

To simplify the creation of large vectors, you can define a vector by specifying the first entry, an increment, and the last entry. For example,

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

or

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

and the result will be

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

If you want to look at the first three entries of the vector you can use the following notation

```
>> a (1:3)
```

and the result will be

```
ans =
        1    3    5
```

If we want a column vector, we have to write

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

or

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

and the result is

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

Now, taking this into account, it is easy to remember how to create a matrix in MATLAB. Here we show an example,

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

We can get an element from a matrix; for example, the element in the second row and the third column of the matrix A. We 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 a matrix, we use the notation [start:end]. For example, the first three elements of the second column of a 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
```

It is also possible to create a matrix from vectors. For example, let

```
>> w=[1:2:6]
>> v=[1 2 3]'
```

and

```
>> b=[2 4 6]'
```

we can define the following matrices

```
>> A=[w;v';b']
```

and

```
>> B=[w',v,b]
```

Remark: Note that MATLAB distinguishes between capital letters and lower case letters, and that if we store two variables with the same name, only the second one remains.

There exist commands that create special matrices. For example, the command `zeros(N,M)` creates a N-by-M matrix of zeros, the command `eye(N)` creates an identity matrix of dimension N, the command `ones(N,M)` creates a N-by-M matrix of ones, and the command `rand(N,M)` creates a N-by-M matrix of random numbers in (0,1). Finally, 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

To carry out algebraical operations with scalar quantities in MATLAB just enter in the Command Window exactly the same expression as if one writes it in a paper. You can, for example, multiply two numbers

```
>> 3*5
```

divide two numbers

```
>> 15/3
```

or to elevate a number to some power

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

To execute algebraical operations with vectors or matrices in MATLAB, the procedure is the same, except for the multiplication operation. Let's see how to perform standard operations with vector. First, define

```
>> w=[1:2:8]
>> v=[1 2 3]'
>> b=[2 4 6]'
```

Now, type the following expressions and explain what happens.

```
>> w(1:3)-w(2:4)
>> b+v
>> v-b
>> 3*b+v-1
>> b^2
>> sin(v)
```

Now, suppose we want to multiply the two following matrices

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

If we type in the Command Window $A * B$ we make a mistake. The right way to multiply these two matrices is entering

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

or

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

using the transpose operator (').

Remark: Note that inner matrix dimension must agree. On the other hand, it is possible to do the multiplication A times B element-by-element typing

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

and the result will be

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

Recall the previous example that showed an error message

```
>> b^2
```

Now, we know that if we want to obtain a vector whose components are the square of the components of b, we must type

```
>> b.^2
```

The inverse matrix can be calculated with the command `inv`. For example, consider the following matrix,

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

the inverse matrix of A is obtained this way

```
>> inv(A)
ans =
        -1     -3      5
         1    1.5   -2.5
         0    0.5   -0.5
```

The next table summarizes a list of useful commands in MATLAB to compute algebraic expressions.

Command	**Description**
`who`	shows all variables in the Workspace
`length(a)`	the number of elements of a
`abs(a)`	absolute value of the elements of a
`sum(a)`	returns the sum of the elements of a
`prod(a)`	multiplies the elements of a
`cumsum(a)`	creates a vector containing the cumulative sum of the elements of a
`cumprod(a)`	creates a vector containing the cumulative product of the elements of a
`size(A)`	the dimensions of A
`det(A)`	the determinant of matrix A
`rank(A)`	the rank of matrix A
`trace(A)`	the trace of matrix A
`eig(A)`	eigen values and eigen vectors of matrix A
`clear`	clear all variables and functions from the memory

Relational and Logical operations

As in many packages, in MATLAB you should think of 1 as true and 0 as false. Some common logical operators are the following. See also `help ops` and `help relop`.

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

The logical operators work element by element on arrays returning a logical array with a set of 1's (when the corresponding element verifies the logical condition) or 0's (when it does not). For example, let $a = \{4, 7, 2, 8, 5, 2\}$ and $b = \{5, 1, 4, 8, 0, 3\}$, and suppose we want to compare these two vectors; that is, we want to know how many elements of a are greater or equal than the corresponding elements of b. If we type

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

in the Command Window, the result will be

```
ans =
        0
        1
        0
        1
        1
        0
```

Therefore, to get the answer we were looking for, we must enter

```
>> sum(a >= b)
ans =
        3
```

Now, suppose we need to know whether the number of elements in a that are greater than 2 but lower than 7, is equal to the number of elements of b that are greater than 5 but lower or equal than 10. We type

```
>> sum(a > 2 & a < 7) == sum(b > 5 & b <= 10)
ans =
        0
```

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

There are situations in which you want certain parts of your program to be carried out only if some conditions hold. This can be done using an `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 the conditions sequentially, first condition 1, then condition 2, and so on. Then, when one condition is verified, MATLAB exits the if-block. For example, in the following case,

```
>> 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 wish to repeat some statements of our program several times, we use loops. The commands to construct loops in MATLAB are `for` and `while`. We will explain the main differences between these two loops.

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,

```
>> A = [];
>> for i = 1:5
     for j = 1:3
        A(i,j) = 1/(j+i-1);
     end
   end
```

The result will be

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

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
```

Remark: To stop a loop you can press **Ctrl** and **c**.

1.6 Loading and saving data

While we are working, MATLAB saves all the variables we have created or loaded in the Workspace. However, when we close MATLAB we loose all the information in the Workspace. To keep this information, the command `save` can help us. This command saves all the Workspace variables in a *file*. For example, typing in the Command Window

```
>> save filename
```

creates a file called *filename.mat* in the current directory that contains all variables in the Workspace. Now, using the command `load`, the variables that were saved in the file are recovered.

```
>> load filename
```

Some times we want to save only certain variables. The following expression creates a file called *filename.mat*, which contains only the variables a, b and D.

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

There are situations in which we are interested 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

MATLAB can produce both, planar plots and 3-D mesh surface plots. Here, a first notion of planar graphs is explained through the command `plot`.

The `plot` command creates linear x-y plots. Let x and y be two vectors of the same size. If we type

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

a graphics window is opened and the elements of a are plotted versus the vector b. For example,

```
>> x = -3:0.1:7
>> y = 2*x.^2 - 5*x -2
>> plot(x,y)
```

And, alternatively, we can plot a vector y against its own index. In our example we enter

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

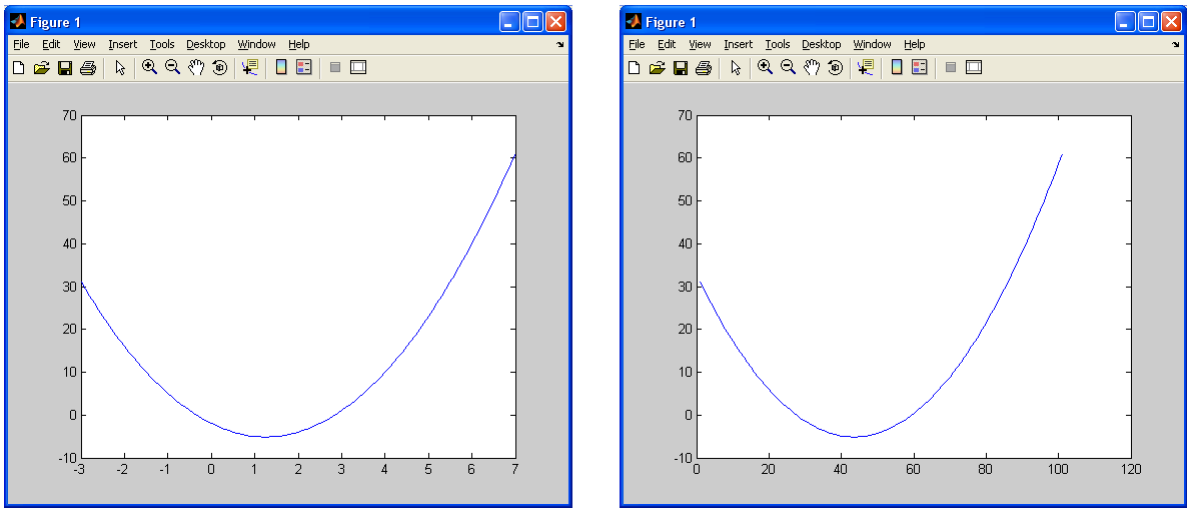

Figure 5: First illustration of the `plot` command

It is possible to modify some aspect of the graph like color, width, and shape, to change the ticks of the axes, to add another line... For that, we use the commands `set`, `hold on`, and `hold off`. We can also add a tittle, labels in the axis, legends... Here we show an example.

```
>> x = -pi:0.1:pi; y = sin(x); y1 = cos(x);
>> plot0 = plot(x,y);
>> 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),',sprintf('\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')
```

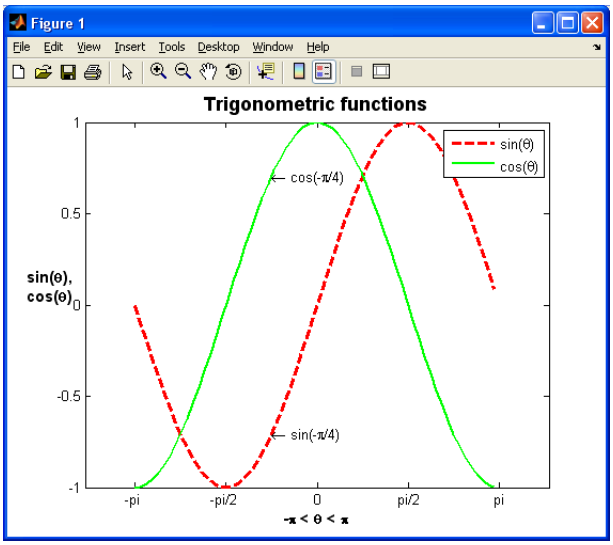

Figure 6: Second illustration of the `plot` command

1.8 Exercises: Part 1

1. Let $x = [3\ 2\ 6\ 8]'$ and $y = [4\ 1\ 3\ 5]'$.

   (a) Add the sum of the elements in x to y.

   (b) Raise each element of x to the power specified by the corresponding element in y.

   (c) Divide each element of y by the corresponding element in x.

   (d) Multiply each element of x by the corresponding element in y and call the result z.

   (e) Sum up the elements in z and assign the result to a variable called w.

   (f) Compute $x'*y - w$ and interpret the result.

2. Evaluate the following expressions:

   a) round(6/9) b) floor(6/9) c) ceil(6/9)

3. Given $x = [3\ 1\ 5\ 7\ 9\ 2\ 6]$, explain the meaning of the following commands:

   (a) x(3)

   (b) x(1:7)

   (c) x(1:end)

   (d) x(1:end-1)

   (e) x(6:-2:1)

   (f) x([1 6 2 1 1])

4. Plot the functions x, x^3, e^x and e^{x^2} over the interval $0 < x < 4$.

5. Create a vector x with the elements:

$$x_n = (-1)^n \frac{n+1}{2n-1}$$

6. Given the array $A = [2\ 4\ 1; 6\ 7\ 2; 3\ 5\ 9]$, provide the commands needed to:

   (a) Assign the first row of A to a vector $x_1 = [1\ 2\ 3]$.

   (b) Assign the last 2 rows of A to a matrix $Y = [2\ 3\ 1; 5\ 3\ 2]$.

   (c) Compute the sum over the columns of A and also over the rows of A.

7. Given the arrays $x = [1\ 4\ 8]$, $y = [2\ 1\ 5]$ and $A = [3\ 1\ 6; 5\ 2\ 7]$, determine which of the following statements will and will not be correctly executed.

   (a) A-[x ' y']

   (b) [x;y']

   (c) A-3

8. Given the array $A = [2\ 7\ 9\ 7; 3\ 1\ 5\ 6; 8\ 1\ 2\ 5]$, explain the results of the following commands:

   (a) A'

   (b) A(:,[1 4])

   (c) A([2 3], [3,1])

   (d) A(:)

   (e) [A A]

   (f) [A;A(1:2,:)]

9. Given the array A from the previous exercise, above, provide the command that will

   (a) Assign the even-numbered columns of A to an array $B = [2\ 3; 4\ 5; 5\ 4]$.

   (b) Compute the square-root of each element of A.

10. Given $x = [1\ 5\ 2\ 8\ 9\ 0\ 1]$ and $y = [5\ 2\ 2\ 6\ 0\ 0\ 2]$, carry out and explain the results of the following commands:

    (a) $x > y$

    (b) $x == y$

    (c) $y >= x$

    (d) $(x \sim= 0)\&(y \sim= 0)$

(e) $(x > y)|(x < y)$

(f) $(x > 0)|(y > 0)$

(g) $(x \sim= 0)\&(y == 0)$

(h) $(x > y)\&(x < y)$

11. Given $x = 1 : 10$ and $y = [3\ 1\ 5\ 6\ 8\ 2\ 9\ 4\ 7\ 0]$, execute and interpret the results of the following commands:

(a) $(x > 3)\&(x < 8)$

(b) $x(x > 5)$

(c) $y(x <= 4)$

(d) $x((x < 2)|(x >= 8))$

(e) $y((x < 2)|(x >= 8))$

(f) $x(y < 0)$

12. Create the vector $x = randperm(35)$ and the vector y which is a function of x given by:

$$f(x) = \begin{cases} 2, & \text{if } x < 6 \\ x - 4, & \text{if } 6 \leq x \leq 20 \\ 36 - x, & \text{if } 20 < x \leq 35 \end{cases}$$

13. Evaluate the following fragments of MATLAB code for $n = 7, 0, -10$, $z = 1, 9, 60, 200$, and $T = 50, 15, 0$; respectively.

(a)
```
if n > 1
  m=n+1
else
  m=n-1
end
```

(b)
```
if z < 5
  w=2*z
elseif z < 10
  w=9-z
elseif z = sqrt(z)
  w=9-z
else
  w=z
end
```

(c) `if` $T < 30$
*h=2*T+1*
`elseif` $T < 10$
h=T-2
`else`
h=0
`end`

14. Given $x = [3\ 15\ 9\ 12\ -1\ 0\ -12\ 9\ 6\ 1]$, provide the command(s) that

(a) Set the values of x that are greater or equal than zero.

(b) Extract the values of x that are greater than 10 into a vector called y.

(c) Extract the values of x that are greater than the mean of x.

15. Write a sequence of sentences to evaluate the following functions.

(a) Let

$$h(t) = \begin{cases} t - 10, & \text{if } 0 < t < 100, \\ 0.45t + 900, & \text{if } t \geq 100. \end{cases}$$

Test with $h(5) = -5$ and $h(110) = 949.5$.

(b) Compare the results of the following function with the function `sign`.

$$f(x) = \begin{cases} -1, & \text{if } 0 < x, \\ 1, & \text{if } x \geq 100. \end{cases}$$

(c) Let

$$g(y) = \begin{cases} 200, & \text{if } y < 10000, \\ 200 + 0.1(y - 10000), & \text{if } 10000 \leq y < 20000, \\ 1200 + 0.15(y - 20000), & \text{if } 20000 \leq y < 50000, \\ 5700 + 0.25(y - 50000), & \text{if } 50000 \leq y. \end{cases}$$

Test with $g(5000) = 200$, $g(17000) = 900$, $g(25000) = 1959$, and $g(75000) = 11950$

16. Read the following fragment of MATLAB code and, without running MATLAB, explain why the following if-block is not a correct solution to the previous exercise, part (c).

`if` $y < 10000$
t=200
`elseif` $10000 < y < 20000$
t=200+0.1(y-10000)*
`elseif` $20000 < y < 50000$

```
  t=1200+0.15*(y-20000)
elseif y > 50000
  t=5700+0.25*(y-50000)
end
```

17. Given the vector $x = [1\ 8\ 3\ 9\ 0\ 1]$, create a short set of commands that

    (a) Adds up the values of the elements of x.

    (b) Computes the running sum of the elements of x.

18. Create a 3×4 array of random numbers. Move through the array (element by element), and set any value that is less than 0.2 to 0 and any value that is greater or equal than 0.2 to 1.

19. Given $x = [4\ 1\ 6]$ and $y = [6\ 2\ 7]$, compute the following arrays.

    (a) A matrix A whose elements are $a_{ij} = x_i y_j$.

    (b) A matrix B whose elements are $b_{ij} = x_i / y_j$.

    (c) A vector c whose elements are $c_i = x_i y_j$.

20. Write a sequence of sentences that use the random generator command `rand` to determine:

    (a) The number of random numbers it takes to add up to 20 (or more).

    (b) The number of random numbers it takes before a number between 0.8 and 0.85 occurs

    (c) The number of random numbers it takes before the mean of those numbers is within 0.01 and 0.5

2 The *Statistics* Toolbox

MATLAB has several auxiliary libraries called ***Toolboxes***, 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 Toolbox.

The *Statistics* Toolbox was created in the version 5.3 and continuously updated in newer versions. It consists of a collection of statistical tools built on the MATLAB numeric computing environment. This Toolbox supports a wide range of common statistical tasks: random number generation, curve fitting, design of experiments, statistical process control... Typing `help stats` on the Command Window, a huge list of functions appears. They are classified according to the topic of the purpose they are made for. We will review a few of them.

2.1 Descriptive Statistics

The *Statistics* Toolbox provides functions for describing the features of a data sample. These descriptive statistics include measures of location and dispersion, and there exits the possibility of computing these measures when there are missing values among the data. The following table shows the most important ones and a brief description of their use.

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

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

The following example shows how the command `grpstats` works.

```
>> x=[3 5 4 8 9 10];
>> y=[1 0 0 1 1 1];
>> grpstats(x,y)
ans =
        4.5000
        7.5000
>> grpstats(x,y,'std')
ans =
        0.7071
        3.1091
```

There exists an alternative way of obtaining the sample kurtosis and the skewness (apart from using the commands `skewness` and `kurtosis`). The command `moment(x,n)` computes the n^{th} central sample moment of x. As the kurtosis is defined as the ratio between the fourth central moment and the square of the second central moment, we can write

```
>> m4 = moment(x,4); % Calculating the 4th Central Moment
>> m2 = moment(x,2); % Calculating the 2nd Central Moment
>> kurtbis = m4./(m2).^2 % Calculating the Kurtosis
>> kurtbis =
        1.3530
>> kurtosis(x)==kurtbis
>> ans =
        1
```

Remark: In the construction of `kurtbis`, we must remember to write the dot (.) before the mathematical operations, to ensure that we deal with element-by-element (scalar) operations.

2.2 Density and distribution functions, and random number generators

Parametric statistical methods are mathematical procedures which assume that the distributions of the underlying random variables belong to known parametrized families of probability distributions. When working within this branch of statistics, it is of importance to know the properties of these parametric distributions. The *Statistics* Toolbox has information of a great number of known parametric distributions, regarding their Probability Density Functions (*PDFs*), Cumulative Distribution Functions (*CDFs*), Inverse CDFs (*ICDFs*), and useful commands to work with them. 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 `normpdf` and `normcdf` are the commands for the density and cumulative distribution functions, respectively.

Below, we list some of the best-known parametric distributions and useful commands to work with them.

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

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 practice some of these commands. First, create a vector X of 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 what the PDF and the CDF of certain distributions look like. First, generate the domain of the function with the command `S = first:stepsize:end` and then draw the PDF and the CDF of a t-Student 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])
```

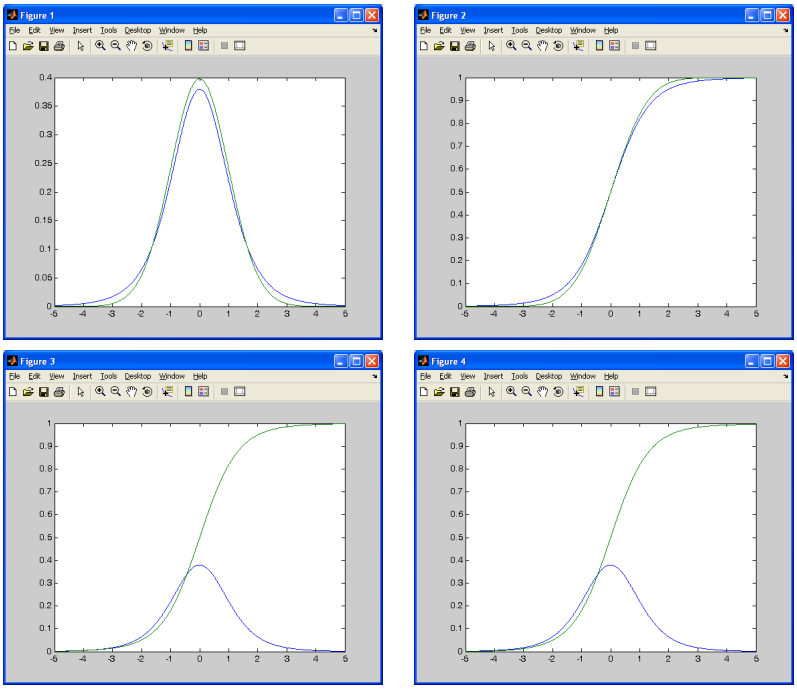

Figure 7: figure(1), figure(2), figure(3), and figure(4)

Note that figure(3) and figure(4) in Figure 7 are pretty similar, as they represent the same functions. This is because the CDF evaluated in a point x is 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^{th} element of a vector (in our case, the vector t5_pdf) by the partial sum of the all observations up to this element. Thus, if $D = [1\ \ 2\ \ 3\ \ 4]$, then $cumsum(D) = [1\ \ 3\ \ 6\ \ 10]$.

The Inverse CDF is used in the contexts of confidence intervals and hypothesis testing to find the critical values of a known distribution. For example, let X be the vector created before, and suppose we wish to test whether its mean is equal to zero (assuming that the variance is unknown). First, construct the appropriate statistics, and then compute the critical values for a fixed significance level, let's say $\alpha = 0.05$. Since this is a two-tailed test, we should obtain the value that leaves $\alpha/2 = 0.025$ of probability in each tail.

```
>> tstat = sqrt(1000)*mean(X)./std(X)
>> tstat =
        -1.4440
>> tcritic = tinv(0.025,999)
>> tcritic =
        -1.9623
```

As we expected, we do not reject the null of zero mean (we have generated X as a Normal random variable with zero mean!)

Remark: Note that the command `std`, by default, calculates the sample-corrected standard deviation. Adding an optional input it is possible to can change it. Use the command `Help` for more information.

Now we generate a time series of size $T = 1000$ with the following dynamic characteristics:

$$y_t = y_{t-1} + a_t, \tag{1}$$

$$x_t = 0.5\ a_{t-1} + a_t, \tag{2}$$

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

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 use the following loop to generate them.

```
>> close all % Closes all the figures without saving them
>> T =1000; % Stores the sample size
>> a = normrnd(0,1,T,1); % Generates the white noise process
>> yt = zeros(T,1); xt = 3*ones(T,1); wt = zeros(T,1); % Generates the empty vectors
>> for t = 2:T, % Starts the loop at t=2
>>     yt(t) = yt(t-1) + a(t); % Generates yt
>>     xt(t) = 3 + 0.5 * a(t-1) + a(t); % Generates xt
>>     wt(t) = -0.8 * wt(t-1) + 0.5 * a(t-1) + a(t); % Generates wt
>> end % Finishes the loop
>> figure(1),plot([yt xt wt a],'LineWidth',1.5); % Plots the series against time
>> legend yt xt wt a % Adds the legend to the current figure
```

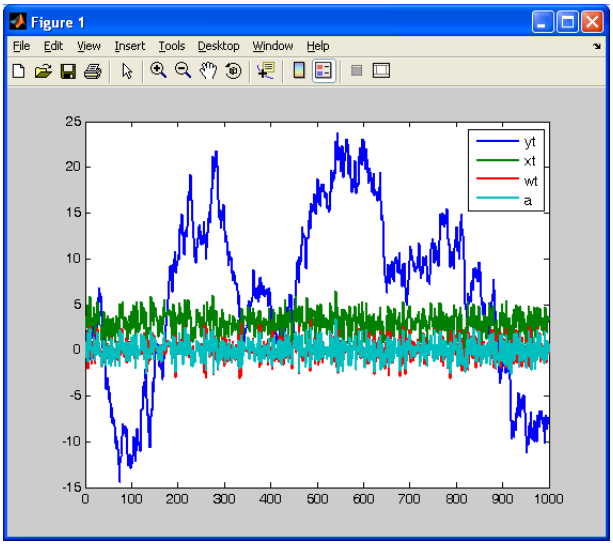

Figure 8: Plot of several time series we have generated in MATLAB

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`.

2.3 Statistical plots

The graphical methods are as important as the descriptive statistics to summarize and to understand the information provided by the data. In this section we review some of the specialized plots in the *Statistics* Toolbox and some others from the MATLAB main Toolbox. Apart from the arsenal of graphs in MATLAB, the *Statistics* Toolbox allows us to construct box plots, scatter plots, normal probability plots, Weibull probability plots, control charts, and quantile-

quantile plots, among others. The best way of plotting data strongly depends on their own nature. Thus, for instance, qualitative data information should 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**
scatter	Scatter plot of two variables (in MATLAB Toolbox)
gscatter	Scatter plot of two variables grouped by a third
gplotmatrix	Matrix of scatter plots grouped by a common variable
hist	Histogram (in MATLAB Toolbox)
hist3	Three-dimensional histogram of bivariate data
boxplot	Boxplots of a data matrix (one per column)
bar	Data plotting in vertical bars (in MATLAB Toolbox)
barh	Data plotting in hertical bars (in MATLAB Toolbox)
cdfplot	Plot of empirical cumulative distribution function
ecdfhist	Histogram calculated from empirical cdf
gline	Point, drag and click line drawing on figures
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)
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 create histograms in MATLAB.

Consider a 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 % Closes all the current figures
>> figure(1),hist(X) % Draws the histogram of X
>> figure(2),hist(a) % Draws the histogram of a
>> figure(3),hist3([X a]) % Draws the histogram of the bivariate sample [X,a]
>> figure(4),hist([X;10;-10]) % Draws the histogram of X contaminated with 2 outliers
```

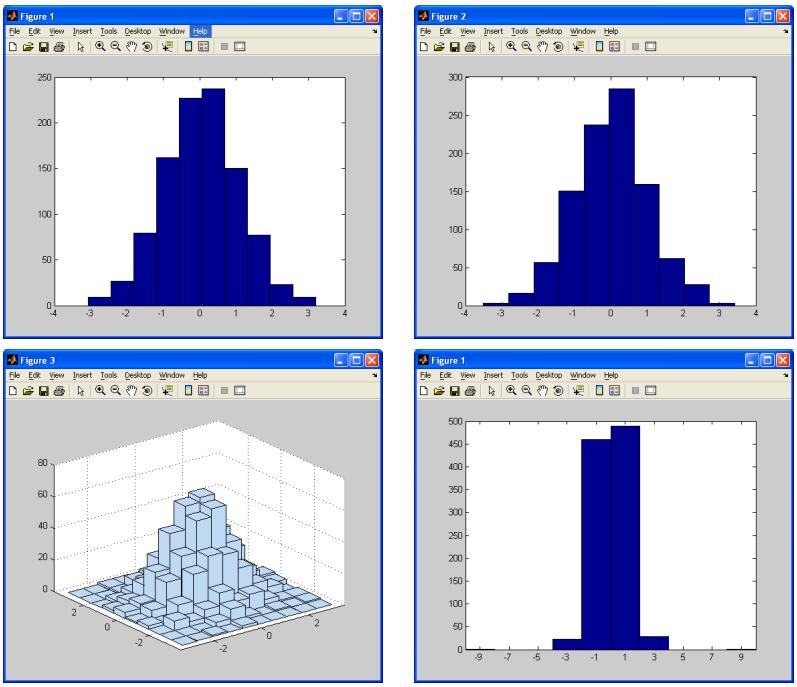

Figure 9: figure(1), figure(2), figure(3), and figure(4) that illustrate the command `hist`

The histograms give us an idea of the shape of the distribution, and allow us to infer whether the data come from a given distribution. Moreover, they are useful for detecting outliers: figure(4) in Figure 9 shows the histogram of X after we have added two "large" (in absolute value) observations to the original sample. It is clear that these two outliers change significantly the shape of the histogram.

When dealing with discrete data, the `stem` command may be more appropriate than the histogram to plot 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 for that is `tabulate`. Then, the commands `stem` and `stairs` produce the plots of the simple and cumulative frequencies given by `tabulate`.

```
>> close all % Closes all the current figures
>> B = binornd(10,0.3,1000,1); % Generates a vector of size 1000 from a Binomial(10,0.3)
>> Freq = tabulate(B) % Tabulates 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)) % Draws the stem plot of Freq
>> figure(2),stairs(Freq(:,1),cumsum(Freq(:,2))) % Draws the stair plot of Freq
```

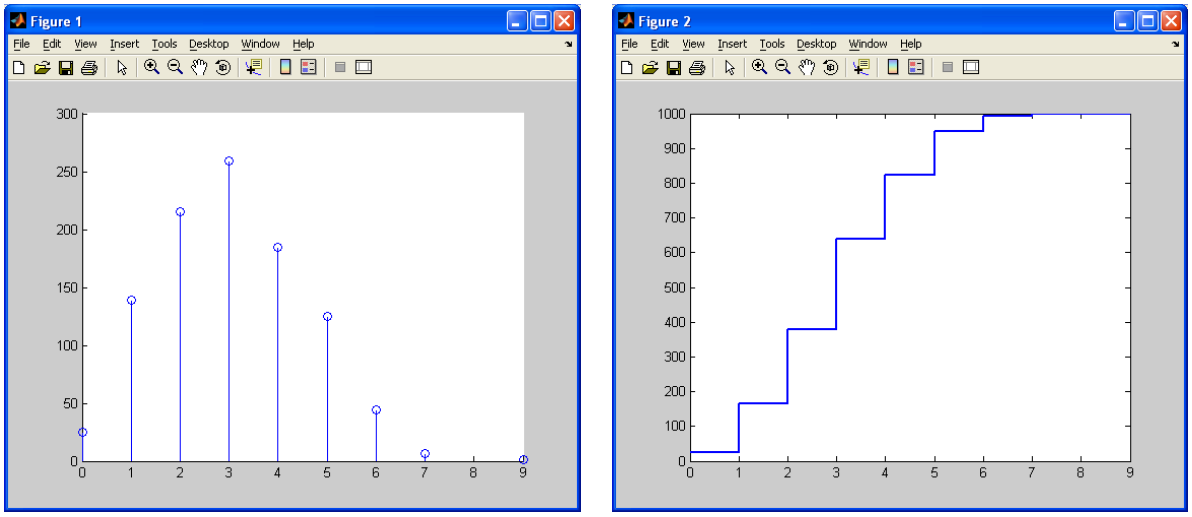

Figure 10: figure(1) and figure(2) that illustrate the commands `stem` and `stairs`

Finally, we learn how to create the pie charts. Consider the following information about the number of marriages and divorces in Spain in the last nine years [1]:

Year	**2000**	**2001**	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**	**2008**
Marriages	194.022	201.579	203.453	208.146	216.149	212.300	211.522	208.057	216.451
Divorces	37.743	39.242	41.621	45.448	50.974	72.848	126.952	125.777	110.036

In order to introduce this table in MATLAB and then plot the pie charts, we must follow these steps:

[1] Source INE (Instituto Nacional de Estadística)

```
>> Year = {'2000','2001','2002','2003','2004','2005','2006','2007','2008'};
>> M = [194.022, 201.579, 203.453, 208.146, 216.149, 212.300, 211.522, 208.057, 216.451];
>> D = [37.743, 39.242, 41.621, 45.448, 50.974, 72.848, 126.952, 125.777, 110.036];

>> close all
>> figure(1), pie(M,[1 0 0 0 0 0 0 0 0],Year),
>> figure(2), pie3(D,[0 0 1 1 0 0 0 0 0],Year)
```

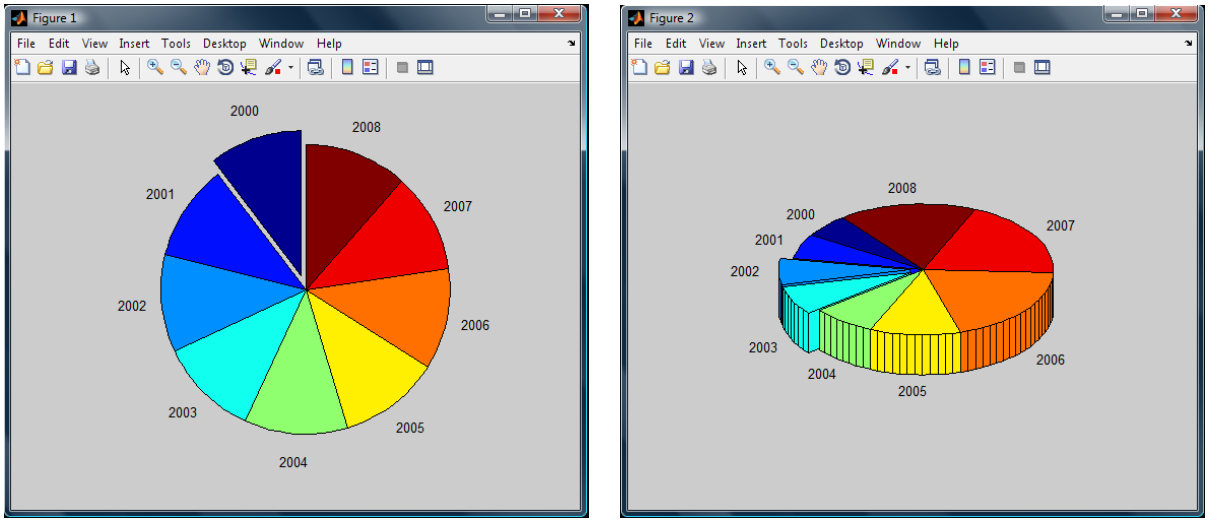

Figure 11: figure(1) and figure(2) illustrate different pie charts

Remark: 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. Use the command `Help` for further information.

2.4 Linear regression analysis

Regression analysis is another big branch of Statistics covered by the *Statistical* Toolbox. For example, the Ordinary Least Squares (OLS) estimators of a given linear regression is easy to be obtained in MATLAB. If we know the algebra behind the procedure, we can simply use matrix algebra and some of the commands that were introduced in the previous sections. However, the *Statistics* Toolbox provides very useful information for inference and testing with just a simple command named `regress`. In its simplest form, by typing `B = regress(Y,X)`, we obtain 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 β in the model

$$\begin{aligned} Y &= X\,\beta + U, \\ &= \beta_1\,X_1 + \beta\,X_2 + ... + \beta_p\,X_p + U, \end{aligned} \tag{4}$$

where U is a vector of zero-mean random errors. If we 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 β. 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 steel factories. The data file is a text-file called *regress_data.txt* and refers to seven variables with 129 observations. The variables are: *price per unit*, *salaries per hour*, *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)[2]. All the variables, except the binary one, are in $\log' s$. We wish to study the relationship between the unit price of steel with the different production costs by means of (4). As we have seen in Section 1.6, to load the data we have to 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. We will use the categorical discrete variable, that refers to the geographical situation, to group the data. Recall that the command to draw a matrix of scatter plots grouped by a common variable is `gplotmatrix`. The command also allows us to introduce the variable names and a histogram in the main diagonal, as shown in Figure 12. For more information about the usage of this function use the command `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));
```

The first row of the scatter plot provides a good insight that the linear model can be a good way to relate these variables. Then, we estimate the parameter β in (4) and the residual variance with the function `regress`. We first leave aside the categorical data.

Remark: Note 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
```

[2]This exercise is a modification of a practice of Statistics II. You will find all the information at http://www.est.uc3m.es/esp/nueva_docencia/leganes/ing_industrial/estadistica_II/practicas.html

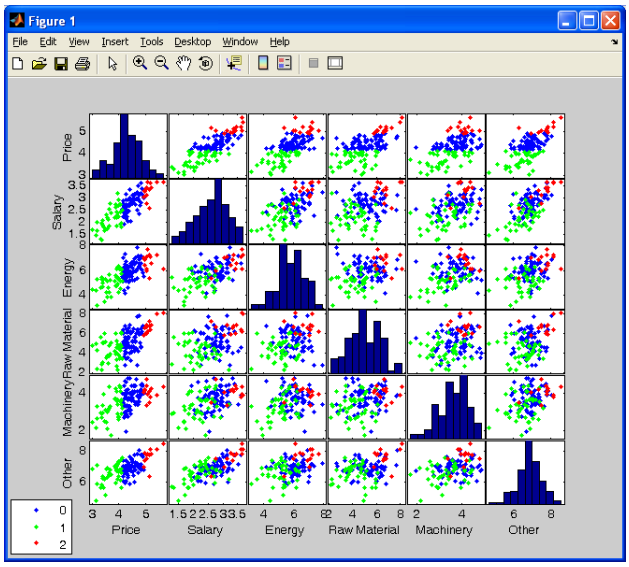

Figure 12: Illustration of the command `gplotmatrix`

Regressor	$\hat{\beta}_{\mathbf{OLS}}$	**95% Conf. interv. of** $\hat{\beta}_{\mathbf{OLS}}$	$\mathbf{R^2}$	***F-stat***	***p-value***	$\sigma_{\mathbf{u}}^2$
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)				

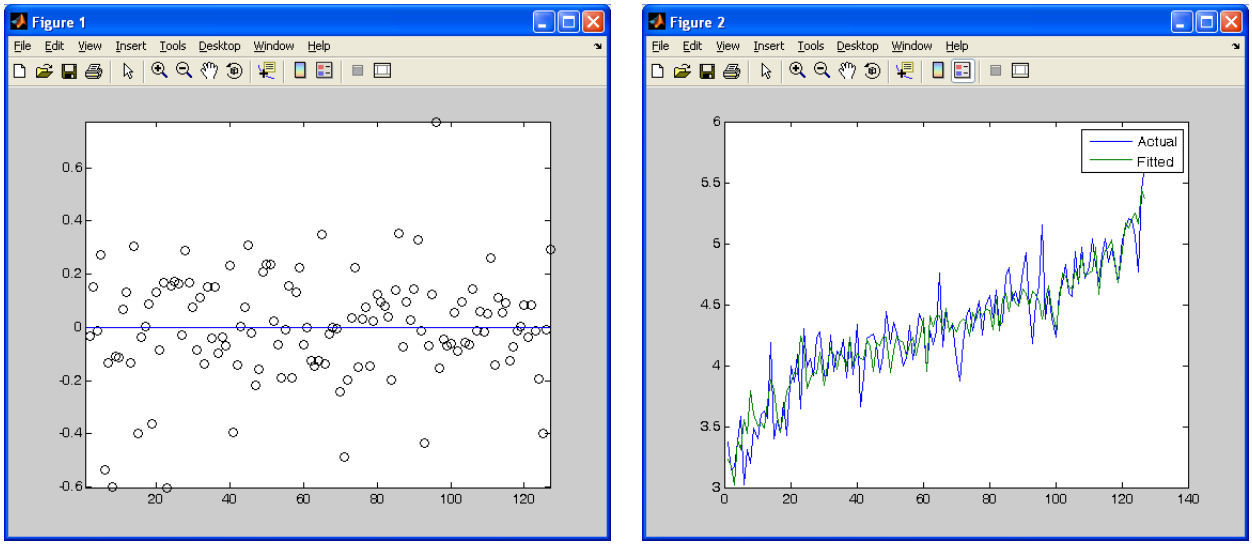

Figure 13: figure(1) and figure(2) in the regression analysis

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

2.5 Exercises: Part 2

1. Generate 1000 samples of size 50 from an Exponential distribution with mean $\lambda^{-1} = 2$. Note that the columns represent different samples. Define a row vector containing the mean value of each column and plot a histogram of these mean values, what do you expect to see?

2. Given a random sample $X_1, X_2, \ldots, X_n$ from a normal population $N(\mu, \sigma)$, let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ be the sample mean, and let $S^2 = \frac{1}{n-1}\left[\frac{1}{n}\sum_{i=1}^{n} X_i^2 - n\bar{X}\right]$ be the sample variance. Show by simulation the following properties of the normal distribution:

   (a) $\bar{X} \sim N(\mu, \frac{\sigma}{\sqrt{n}})$

   (b) $\frac{(n-1)S^2}{\sigma^2} \sim \chi_{n-1}$

   (c) $\bar{X}$ and S^2 are independent

3. Consider the data of the example in Section 2.3 regarding the number of marriages and divorces in Spain from 2000 to 2008. Test the null hypothesis that the mean of divorces per year in Spain is 200.

4. In television's early years, most commercials were 60 seconds long. However, nowadays commercials can be any length. The objective of commercials remains the same; that is, to encourage viewers to remember the product favorably and eventually buy it. An experiment was carried out in order to determine how the length of a commercial affects people's memory: 60 randomly selected people were asked to watch a 1 hour television programme. In the middle of the show, a commercial advertising a brand of toothpaste appeared. The essential content of the advert was the same but the duration was different. After the show, each person was given a test to measure how much he or she remembered about the product. The commercial times and test scores (on a 30-point test) are given in the following table:

Length	**Score**	**Length**	**Score**
52	24	52	15
40	20	28	7
36	16	56	26
28	11	20	11
44	10	52	18
16	4	16	16
48	24	20	8
52	18	40	12
60	16	16	10
44	15	44	14
36	14	32	19
44	15	20	8
60	24	56	11
24	10	56	24
32	1	60	15
40	8	24	9
24	9	48	18
32	0	16	14
52	17	32	14
36	9	16	11
60	26	40	15
56	28	24	11
20	15	36	27
40	8	28	5
48	2	56	17
20	0	32	8
24	11	28	15
36	8	48	8
60	24	28	24
44	10	48	21

(a) Obtain a scatter diagram of the data to determine whether a linear model appears to be appropriate.

(b) Determine the least square line and compute the covariance and the correlation coefficient.

(c) Plot the line and the residuals.

3 Programming in MATLAB

MATLAB can carry out a set of statements stored in a file. Such files are called *m-files*, named after its .m file extension, as we highlighted in Section 1.1. Often, the work of a researcher in the field of Statistics involves the creation and the adaptation of *m-files*.

The *m-files* allow us to store and to carry out a long sequence of sentences. The first step is to create an *m-file*. The easiest editor on our system is the MATLAB editor (*m-file* editor). There are two types of *m-files*: **script files**, that simply execute a sequence of MATLAB statements, and **function files**, that also accept inputs to produce outputs.

3.1 Script files

The simplest *m-files* are the *Scripts*. They are useful for automating blocks of MATLAB commands, such as calculations 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 calculations. Besides, scripts can produce graphical outputs.

Like any other *m-file*, the scripts accept comments. Any text followed by the percent sign (%) on a given line is a comment. Comments may appear on separate lines, or at the end of an executable line. For example, imagine we define a set of sentences to compute the Body Mass Index (BMI) in an *m-file*, say BMI.m. First, go to *File->New->M-File*. This action opens a new window in which we edit the *m-file*. Then, go to *File->Save As...* and write BMI. Next, inside the script window, enter all the necessary commands. See Figure 14.

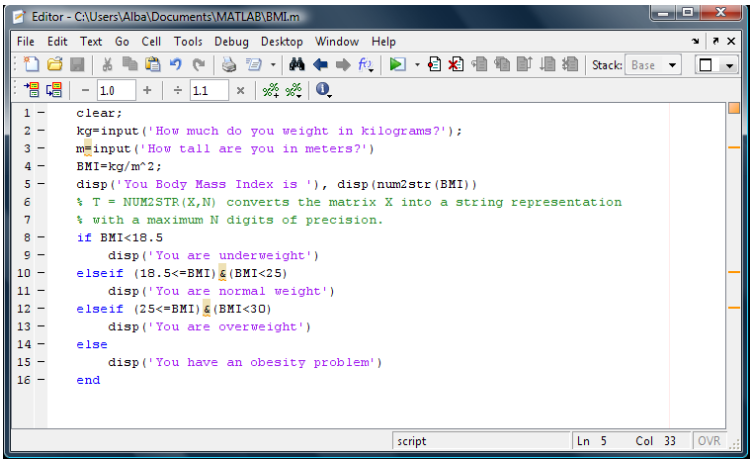

Figure 14: Example of a script file that calculates the Body Mass Index

The MATLAB command `BMI` will carry out the statements in the file.

Now, let's create an *m-file* to construct the plot shown in Figure 6 given in Section 1.7.

```
clear % Clears all variables from the memory

clc % Clears the screen

x = -pi:0.1:pi; % Defines a variable x

y = sin(x); % Defines a function of the variable x and calls it y

y1 = cos(x); % Defines another function of the variable x and calls it y1

gr = plot(x,y); % Creates a plot of y versus the values of x

hold on% Allows us to draw another plot in the same frame as the previous one

gr1 = plot(x,y1,'g'); % Creates a of y1 versus the values of x

hold off

set(gcf,'Color','w' ); % Sets the color of the background of the current figure

set(gr,'LineStyle','--','LineWidth',2.5, 'Color', 'r'); % Sets the styles, color and width for the first plot

set(gr1,'LineStyle','-','LineWidth',2, 'Color','g'); % Sets 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);

% Sets the marks for the axes

text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi/4)','HorizontalAlignment','left'); % Inserts a text inside the graph

text(-pi/4,cos(-pi/4),'\leftarrow cos(-\pi/4)', 'HorizontalAlignment','left');

title('Trigonometric functions','FontWeight','bold','FontSize',13); % Inserts a title in the graph

legend('sin(\theta)','cos(\theta)'); % Adds legends

xlabel('-\pi < \theta < \pi','FontWeight','bold','FontSize',11); % Writes a label for the x axes

ylabel(['sin(\theta),',sprintf('\n'),'cos(\theta)'],'Rotation',0.0,'FontWeight','bold','FontSize',11);
```

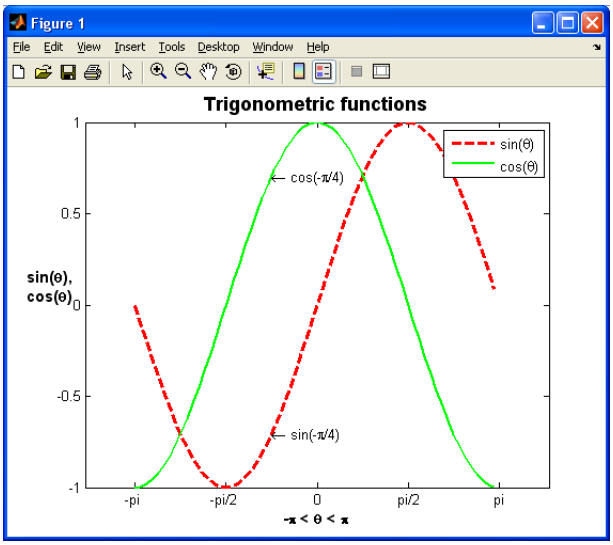

Figure 15: Figure 6, output of the previous script

3.2 Functions

Function files provide extensibility to MATLAB. It is possible to create new functions to a specific problem that then will have the same status as other MATLAB functions. Function `[out1, out2, ...] = funname(in1, in2, ...)` defines the function `funname` that accepts inputs `in1, in2`, ... and returns outputs `out1, out2`, ... 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. We first illustrate with a simple example a function file.

```
function y = randint(m,n)
% RANDINT Randomly generated integral matrix.
% randint(m,n) returns an m-by-n such matrix with
% entries between 0 and 9.
y = floor(10*rand(m,n));
```

A more general version of this function is the following.

```
function y = randint(m,n,a,b)
% RANDINT Randomly generated integral matrix.
% randint(m,n) returns an m-by-n such matrix with
% entries between 0 and 9.
% randint(m,n,a,b) returns entries between integers a and b.
if nargin < 3
a=0; b=9;
end
y = floor((b-a+1)*rand(m,n))+a;
```

The function must be placed in a diskfile with filename *randint.m* (corresponding to the function name). The first line declares the function name, the input arguments and the output arguments; without this line the file would be a script file. Now, if we write in the Command Window the statement `z = randint(4,5)`, in the function file the variables `m` and `n` will be the numbers 4 and 5, respectively; and the output result will be assigned to the variable `z`. Since variables in a function file are local, their names are independent of those in the current MATLAB environment.

```
>> z = randint(4,5)
ans =
     8   6   9   9   4
     9   0   9   4   9
     1   2   1   8   7
     9   5   9   1   9
```

Remark: Note that the command `nargin` ("Number of ARGuments provided as INputs")

permits to set a default value for omitted input variables (such as `a` and `b` in the example given above).

Now, we create a function that, given the inputs (of two assets): weights, standard deviations, and correlation, returns two outputs: the variance, and the risk of the portfolio measured as its standard deviation.

```
function [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
```

We illustrate how this function works carrying out the following statement:

```
>> [a b]=risk([.4 .6],.1,0.5,.9)
a =
        0.3365
b =
        0.1132
```

Figure 16 shows the *m-file* that computes the mean of a vector or a matrix (*mean.m*).

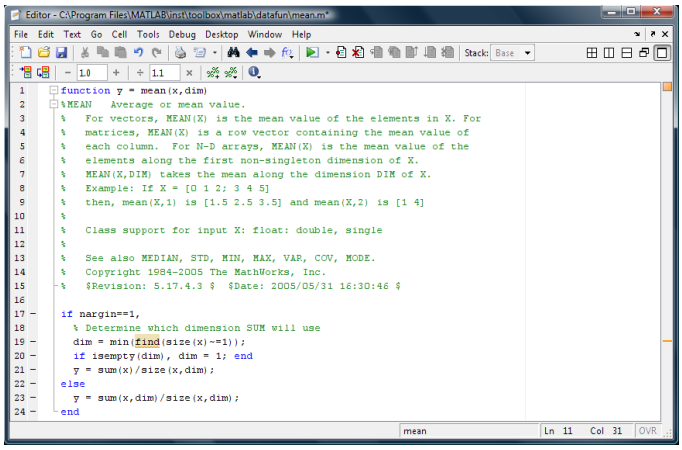

Figure 16: The file mean.m

3.3 Exercises: Part 3

1. The Fibonacci numbers are computed according to the following relation:

$$F_n = F_{n-1} + F_{n-2},$$

with $F_0 = F_1 = 1$.

(a) Compute the first 10 Fibonacci numbers.

(b) For the first 50 Fibonacci numbers, compute the ratio

$$\frac{F_n}{F_{n-1}}.$$

It is claimed that this ratio approaches the value of the golden mean $\frac{1+\sqrt{(5)}}{2}$. What can you say about it?

2. The Legendre polynomials, P_n, are defined by the following recurrence relation

$$(n+1)P_{n+1}(x) - (2n+1)P_n(x) + nP_{n-1}(x) = 0,$$

with $P_0(x) = 1$, $P_1(x) = x$, and $P_2(x) = (3x^2 - 1)/2$. Compute the next three Legendre polynomials and plot all 6 over the interval $[-1, 1]$.

3. Below, it is shown the Gauss-Legendre algorithm to approximate π:

(a) Set $a = 1$, $b = 1/\sqrt{(2)}$, $t = 1/4$, and $x = 1$.

(b) Repeat the following commands until the difference between a and b is within some desired accuracy:

```
>> y = a;
>> a = (a+b)/2;
>> b = sqrt(b*y);
>> t = t-x*(y-a)^2;
>> x = 2*x;
```

(c) From the resulting values of a, b, and t, an estimate of π is

$$\hat{\pi} = \frac{(a+b)^2}{4t}.$$

How many repetitions are needed to estimate π to an accuracy of 1e-8? How many repetitions are needed to estimate π to an accuracy of 1e-12?

4. Write a function that computes the cumulative product of the elements of a vector (do not use the command `cumprod` but compare the results with this built-in function).

5. Use the function `randint` created in Section 2.2 to generate random arrays of integers between a and b. Test this function with the following piece of code:

   (a)
   ```
   >> x = randint(100000,1,10,17);
   >> hist(x,10:17)
   ```

   (b)
   ```
   >> x = randint(100000,1,-30,5);
   >> hist(x,-30:5)

   >> x = randint(100000,1,-45,-35);
   >> hist(x,-45:-35)

   >> x = randint(100000,1,7,-2);
   >> hist(x,-2:7)
   ```

6. Explain what the following function does.

```
function y = gcd(a,b)
a=round(abs(a));b=round(abs(b));
if a==0 & b==0
error('The gcd is not defined when both numbers are zero ')
else
while b~=0
r=rem(a,b);
a=b; b=r;
end
end
```

7. 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 section 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 type of doll is best positioned in the Spanish market, and how these sales are spread out among the provinces. As the 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.

8. 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, $r_f = 0.01\%$

		Type A Sales		Type B Sales		Type C Sales	
Province	**Code**	**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

4 References

. C. Ausín (2005). MATLAB Tutorial: Part I. Departamento de Estadística. Universidad Carlos III de Madrid.
http://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/Matlab_Tutorial_PartI_2005.pdf

. C. Ausín (2005). MATLAB Tutorial: Part I. Departamento de Estadística. Universidad Carlos III de Madrid.
http://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/Matlab_Tutorial_PartII_2005.pdf

. S. Pellegrini and A. Rodriguez (2008). An introductory course in MATLAB. Departamento de Estadística. Universidad Carlos III de Madrid.
http://www.est.uc3m.es/afrodrig/Matlab/Tutorial_completo.pdf