LABORATORY OF MATHEMATICS
Class notes

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DEL DISEÑO

DEPARTAMENTO DE MATEMÁTICA APLICADA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Contents

Chapter 1

Matlab

1.1 Introduction

During this course you will learn how to use Matlab, to design, and to perform mathematical computations. You will also get acquainted with basic programming. If you learn to use this program well, you will find it very useful in future, since many technical or mathematical problems can be solved using Matlab.

This text includes all material (with some additional information) that you need to know, however, many things are treated briefly. Therefore, "The Student Edition of Matlab. User's guide" should be used as a complementary book during the course.

It is important that you spend enough time to learn the Matlab basics that are introduced in the first few sections. If you notice that you need more time than laboratory hours, please consider doing some examples during the rest of the time. Since learning Matlab is a challenging task, you are encouraged to prepare each laboratory; You may read a part of text at home or try the examples yourself. This way, you can use laboratory time to ask questions in order to make things become more clear to you.

1.1.1 Preliminaries

Below you find a few basic definitions on computers and programming. Please get acquainted with them since they introduce key concepts needed in the coming sections:

- A bit (short for binary digit) is the smallest unit of information on a computer. A single bit can hold only one of two values: 0 or 1. More meaningful information is obtained by combining consecutive bits into larger units, such as byte.
- A byte - a unit of 8 bits, being capable of holding a single character. Large amounts of memory are indicated in terms of kilobytes (1024 bytes), megabytes (1024 kilobytes), and gigabytes (1024 megabytes).
- Binary system - a number system that has two unique digits: 0 and 1. Computers are based on such a system, because of its electrical nature (charged versus uncharged). Each digit position represents a different power of 2. The powers of 2 increase while moving from the right most to the left most position, starting from $2^0 = 1$. Here is an example of a binary number and its representation in the decimal system:

  $10110 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 16 + 0 + 4 + 2 + 0 = 24$
- Data is information represented with symbols, e.g. numbers, words, signals or images.
- A command is a instruction to do a specific task.
- An algorithm is a sequence of instructions for the solution of a specific task in a finite number of steps.
- A program is the implementation of an algorithm suitable for execution by a computer.
- A variable is a container that can hold a value. For example, in the expression: `x+y`, `x` and `y` are variables. They can represent numeric values, like `25.5`, characters, like `'c'` or character strings, like `'Matlab'`. Variables make programs more flexible. When a program is executed, the variables are then replaced with real data. That is why the same program can process different sets of data.

  Every variable has a name (called the variable name) and a data type. A variable's data type indicates the sort of value that the variable represents (see below).
- A constant is a value that never changes. That makes it the opposite of a variable. It can be a numeric value, a character or a string.
- A data type is a classification of a particular type of information. The most basic data types are:

- integer: a whole number; a number that has no fractional part, e.g. 3.
- floating-point: a number with a decimal point, e.g. 3.5 or 1.2e-16 (this stands for $1.2 * 10^{-16}$).
- character: readable text character, e.g. `'p'`.

- A bug is an error in a program, causing the program to stop running, not to run at all or to provide wrong results. Some bugs can be very subtle and hard to find. The process of finding and removing bugs is called debugging.
- A file is a collection of data or information that has a name, stored in a computer. There are many different types of files: data files, program files, text files etc.
- An ASCII file is a standardized, readable and editable plain text file.
- A binary file is a file stored in a format, which is computer-readable but not human-readable. Most numeric data and all executable programs are stored in binary files. Matlab binary files are those with the extension tt '*.m'.

1.2 How to work with Matlab

Matlab is a tool for mathematical (technical) calculations. First, it can be used as a scientific calculator; it provides simple mathematical operations, but handles complex numbers, powers, logarithms, trigonometric operations as well. Next, it allows you to plot or visualize data in many different ways, perform matrix algebra, work with polynomials or integrate functions. Like in a programmable calculator, you can create, execute and save a sequence of commands in order to make your computational process automatic. It can be used to store or retrieve data. In the end, Matlab can also be treated as a user-friendly programming language, which gives the possibility to handle mathematical calculations in an easy way. Running Matlab creates one or more windows on your screen. The most important is the Matlab Command Window, which is the place where you interact with Matlab. The string `>>` is the Matlab prompt (except for the Student Edition that has the `EDU>>` prompt). When the Command window is active, a cursor appears after the prompt, indicating that Matlab is waiting for your command, which can be e.g. `17/3` or `cos(x)`.

1.2.1 Input via the command-line

Matlab is an interactive system; commands followed by `Enter` are executed immediately. The results are, if desired, displayed on screen. However, execution of a command will not be possible or will go wrong when the command is not typed according to the rules. Table 1 shows a list of commands used to solve the indicated mathematical equation (a, b, x and y are numbers).

Mathematical notation	Matlab-command
$a+b$	`a+b`
$a-b$	`a-b`
ab	`a*b`
$3xy$	`3*x*y`
a^b	`a^ b`
$\sqrt{x}$	`sqrt(x)` or `x^ 0.5`
π	`pi`
e	`exp(1)`
i	`i`
$4 \cdot 10^3$	`4e3` or `4*10^ 3`
$3-4i$	`3-4*i` or `3-4*j`
$\sin(x)$, $\arctan(x)$,...	`sin(x)`, `atan(x)`, ...
e^x	`exp(x)`
$\ln(x)$	`log(x)`
$\log(x)$	`log10(x)`
$\lvert x\rvert$	`abs(x)`

Table 1: Translation of mathematical notation to Matlab commands.

There are also commands that perform a specific task, such as `help`; work with a file-system, like `cd`; or work with matrices, like `rank`. Below you find basic information that is important when starting with Matlab:

- Commands in Matlab are executed by pressing `Enter` or `Return`. The output will be displayed on screen immediately. Try the following:

```
3 + 7.5
18/4
3 * 7
```

  Note that spaces are not important in Matlab.

- The result of the last execution is called `ans`. It can be used on the next command line. Try, for instance:

  ```
   14/4
  ans =
  3.5000
   ans^(-6)
  ans =
  5.4399e-04
  ```

  Note that `ans` is always overwritten by the last command.

- `ans` is an example of a Matlab built-in variable. It stores the result of the recently performed computation. You can also define your own variables. Look how the information is stored in the variables `a` and `b`:

  ```
   a = 14/4
  a =
  3.5000
   b = a^(-6)
  b =
  5.4399e-04
  ```

- When the command is followed by a semicolon (;), the output is suppressed. Check the difference between the following expressions:

  ```
   3 + 7.5
   3 + 7.5;
   x = 3/4;
  ```

- It is possible to execute more than one command at the same time; the separate commands should then be divided by commas (to display the output) or by semicolons (to suppress the output display), e.g.:

  ```
   sin(pi/4), cos(pi); sin(0)
  ans =
  0.7071
  ans =
  0
  ```

  Note that the value of `cos(pi)` is not printed.

- The output may contain some empty lines; this can be suppressed by the command `format compact`. In contrast, the command `format loose` will insert extra empty lines.

- Matlab is case sensitive. There is a difference between capital and normal letters; for example, *a* is written as `a` in Matlab, `A` will result then in an error.

- All text after a percent sign % until the end of a line is treated as a comment. Enter e.g. the following:

  ```
   sin(3.14159) \% this is an approximation of sin(pi)
  ```

  You will notice that some examples in this text are followed by comments. They are meant for you and you should skip them while typing those examples.

1.2.2 Help-facilities

Matlab provides assistance through extensive online help. The `help` command is the simplest way to get help; that is, if you know the topic or command on which you want help. For example:

```
 help elfun
```

gives information on the elementary math functions of Matlab. The command help displays a list of all possible topics. The topic you want help on must be exact and spelled correctly. The command `lookfor` is more useful if you do not know the exact name of the command or topic. For example:

```
 lookfor inverse
```

displays all commands for which the word inverse is included in its help-text. This command results in a list of commands with a short description. Besides the `help` and `lookfor` commands, there is also a separate mouse driven help. The `helpwin` command opens a new window on screen which can be browsed in an interactive way.

1.2.3 Interrupting a command or program

Sometimes you might spot an error in your command or program. Due to this error it can happen that the command or program does not stop. Pressing `Ctrl-C` forces Matlab to stop the process. After this the Matlab-prompt () re-appears. This may take a while, though.

1.2.4 Workspace issues

If you work in the Command Window, Matlab memorizes all commands that you entered and all variables that you created. This can be verified with the commands `who`, which gives a list of variables present in the workspace, and `whos`, which includes information on name, number of allocated bytes and class of the variables. For example, assuming that you performed all commands given in the beginning, after typing who you should get the following information:

```
 who
Your variables are:
a ans b x
```

The command `clear <name>` deletes the variable `<name>` from the Matlab workspace, `clear` or `clear all` removes all variables. This is useful when starting a new exercise. For example:

```
 clear a, x
 who
Your variables are:
ans b
```

1.2.5 Saving and loading data

The command `save` allows for saving your workspace variables either into a binary file or an ASCII file. Binary files automatically get the `'.mat'` extension, which is not true for ASCII files. However, it is recommended to add a `'.txt'` or `'.dat'` extension.

Learn how to use the save command by exercising:

```
 s1 = sin(pi/4); s2 = sin(pi/2);
```

```
 c1 = cos(pi/4); c2 = cos(pi/2);
 str = 'hello world'; % this is a string example;
      % you will learn about strings later!
 save % saves all variables in binary
      % format to the file matlab.mat
 save data % saves all variables in binary
   % format to the file data.mat
 save numdata s1, c1 % saves numeric variables:
     %s1 and c1 to the file numdata.mat
 save strdata str % saves a string variable: str
  % to the file strdata.mat

 save allsin s* % saves all variables that
% match the pattern to allsin.mat
 save allcos.dat c* -ascii % saves c1,c2 in
   % 8-digit ascii format to allcos.dat
```

The `load` command allows for loading variables into the workspace. It uses the same syntax as save.

Assuming that you have done the previous exercise, try to load variables from the created files. Important: before each load command, clear the workspace and after loading check which variables are present in the workspace (using `who`).

```
 load % loads all variables from the file matlab.mat
 load data s1 c1 % loads only specified
 % variables from the file data.mat
 load strdata  % loads all variables
 % from the file strdata.mat
```

It is also possible to read ASCII files that contain rows of space separated values. Such a file may contain comments that begin with a percent character. The resulting data is placed into a variable with the same name as the ASCII file (without the extension). Check, for example:

```
 load allcos.dat % loads data from the
 % file allcos.dat into variable allcos
 who % lists variables present in the workspace now
```

1.2.6 Path

In Matlab, commands or programs are contained in m-files, which are just plain text files and have an extension `'.m'`. The m-file must be located in one of the directories which Matlab automatically searches. The list of these directories can be listed by the command path. One of the directories that is always taken into account is the current working directory, which can be identified by the command pwd. Use `path`, `addpath` and `rmpath` functions to modify the path. Here is an example:

```
path
h:\matlab
h:\matlab\toolbox\matlab\general
h:\matlab\toolbox\matlab\ops
...............................
addpath h:\mywork % assume that this is your directory
path
h:\mywork
h:\matlab
h:\matlab\toolbox\matlab\general
h:\matlab\toolbox\matlab\ops
...............................
```

It is also possible to access the path browser from the File menu-bar, instead.

1.3 Mathematics with numbers, vectors and matrices

The basic element of Matlab is a matrix (or an array). Special cases are:

- a 1×1-matrix: a scalar or a single number;
- a matrix existing only of one row or one column: a vector.

1.3.1 Single numbers

You have already got some experience with Matlab and you know that it can be used as a calculator. For example, simply type:

```
312/56
```

The result will be:

```
ans =
5.5714
```

By default, Matlab displays only 5 digits. The command `format long` increases this number to 15, `format short` reduces it to 5 again. For example:

```
format long
312/56
ans =
5.57142857142857
```

1.3.2 An introduction to floating-point numbers

In a computer, numbers can be represented only in a discrete form. It means that numbers are stored within a limited range and with a finite precision. Integers can be represented exactly with the base of 2. The typical size of an integer is 16 bits, so the largest positive integer, which can be stored, is $2^{16} = 65536$. If negative integers are permitted, then 16 bits allow for representing integers between -32768 and 32767. Within this range, operations defined on the set of integers can be performed exactly. However, this is not valid for other real numbers. In practice, computers are integer machines and are capable of representing real numbers only by using complicated codes. The most popular code is the floating point standard. The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, meaning that the decimal point can float. Note that most floating-point numbers that a computer can represent are just approximations. Therefore, care should be taken that these approximations lead to reasonable results. If a programmer is not careful, small discrepancies in the approximations can cause meaningless results. Note the difference between e.g. the integer arithmetic and floating-point arithmetic:

Integer arithmetic	Floating-point arithmetic
$2 + 4 = 6$	$18/7 = 2.5714$
$3 * 4 = 12$	$2.5714 * 7 = 17.9998$
$25/11 = 2$	$10000/3 = 3.3333e+03$

When describing floating-point numbers, precision refers to the number of bits used for the fractional part. The larger the precision, the more exact fractional quantities can be represented. Floating-point numbers are often classified as single precision or double precision. A double-precision number uses twice as many bits as a single-precision value, so it can represent fractional values much better. However, the precision itself is not double. The extra bits are also used to increase the range of magnitudes that can be represented. Matlab relies on a computer's floating point arithmetic. You could have already noticed that in the last 408-410 exercise since the value of $\sin(\pi)$ was almost zero, and not completely zero. It came from the fact that both the value of π is represented with a finite precision and the sin function is also approximated.

The fundamental type in Matlab is double, which stands for a representation with a double precision. It uses 64 bits. The single precision obtained by using the single type offers 32 bits. Since most numeric operations require high accuracy the double type is used by default.

The exact machine precision might be defined as the smallest positive number ϵ that added to 1 does not change the result, so $1 + \epsilon = 1$, in floating-point arithmetic this value is larger than zero (in exact arithmetic of course $\epsilon = 0$ holds). Matlab machine precision is stored in the built-in variable `eps`$\approx 2.2204e - 16$. This means that the relative accuracy of individual arithmetic operations is about 16 digits.

1.3.3 Assignments and variables

Working with complex numbers is easily done with Matlab. For example we can type

```
z = -3 + 2*i;
w = 5 - 7*i;
y1 = z + w, y2 = z - w, y3 = z * w, y4 = z / w
```

Formally, there is no need to declare (i.e. define the name, size and the type of) a new variable in Matlab. A variable is simply created by an assignment (the way it has been done above). Each newly created numerical variable is always of the `double` type. You can change this type by converting it into e.g. the `single` type [1]

[1] a variable `a` is converted into a different type by performing e.g. `a = single(a)`, `a = int16(a)` etc.

In some cases, when huge matrices should be handled and precision is not very important, this might be a way to proceed. Also, when only integers are taken into consideration, it might be useful to convert the double representations into e.g. `int16` or `int321` integer types. Since most computations should be performed with a high accuracy, Matlab automatically chooses the double type. It means that real numbers are approximated with the highest possible precision. However, it is important to emphasize that integer numbers are always represented exactly, no matter which numeric type is used. Bear in mind that undefined values cannot be assigned to a variable. So, the following is not possible:

```
clear x;    % to make sure that x does not exist
fun = x^2 + 4 * sin (x)
```

It becomes possible by:

```
x = pi / 3; fun = x^2 + 4 * sin (x)
```

1.3.4 Vectors

Row vectors are lists of numbers separated either by commas or by spaces. They are examples of simple arrays. The number of entries is known as the length of the vector (the command length exists as well). Their entities are referred to as elements or components. The entries must be enclosed in [ ]:

```
v = [-1 sin(3) 7]
v =
-1.0000 0.1411 7.0000
 length(v)
ans =
3
```

A number of operations can be done on vectors. A vector can be multiplied by a scalar, or added/subtracted to/from another vector with the same length, or a number can be added/subtracted to/from a vector. All these operations are carried out element-by-element. Vectors can be also built from the already existing ones.

```
v = [-1 2 7]; w = [2 3 4];
z = v + w
```

```
z =
1 5 11
9
v3 = v + 2 % add 2 to all elements of vector v3
v3 =
1 4 9
t = [2*v, -w]
ans =
-2 4 14 -2 -3 -4
```

Also, a particular value can be changed or displayed:

```
v(2) = -1, w(2) % change the 2nd value of v,
% display the 2nd value of w
v =
-1 -1 7
ans =
3
```

The colon notation is an important shortcut, used when producing row vectors:

```
2:5
ans =
2 3 4 5

-2:3
ans =
-2 -1 0 1 2 3
```

In general, `first:step:last` produces a vector of entities with the value first, incrementing by the step until it reaches last:

```
 0.2:0.5:2.4
ans =
0.2000 0.7000 1.2000 1.7000 2.2000

-3:3:10
ans =
-3 0 3 6 9
```

```
1.5:-0.5:-0.5 % negative step is also possible
ans =
1.5000 1.0000 0.5000 0 -0.5000
```

Parts of vectors can be extracted by using a colon notation:

```
r = [-1:2:6, 2, 3, -2] % -1:2:6 => -1 1 3 5
r =
-1 1 3 5 2 3 -2

r(3:6) % get elements of r on the positions from 3 to 6
ans =
3 5 2 3

r(1:2:5) % get elements of r on the positions from 1 to 5
 % with the step of 2
ans =
-1 3 2
 r(5:-1:2) % what will you get here?
```

To create column vectors, you should separate entries by a ; or by new lines:

```
f = [-1; 3; 5]
f =
-1
3
5
```

The same operations as on row vectors can be done on column vectors. However, you cannot for example add a column vector to a row vector. To do that, you need an operation called transposing, which converts a column vector into a row vector and vice versa:

```
f = [-1; 3; 5]; % a column vector
f' % f' is a row vector
ans =
-1 3 5

v = [-1 2 7]; % a row vector
f + v % you cannot add a column vector f to a row vector v
```

```
??? Error using ==> +
Matrix dimensions must agree.

f' + v
ans =
-2 5 12

f + v'
ans =
-2
5
12
```

If x is a complex vector than `x'` gives the conjugate transpose of `x`, e.g.:

```
x = [1+2i, -1+i]
x =
1.0000 + 2.0000i -1.0000 + 1.0000i

x' % this is the conjugate transpose
ans =
1.0000 - 2.0000i
-1.0000 - 1.0000i

x.' % this is the 'normal' transpose
ans =
1.0000 + 2.0000i
-1.0000 + 1.0000i
```

1.3.5 Matrices

Defining a matrix is similar to defining a vector. The generalization is straightforward, if you know that a matrix consists of row vectors (or column vectors). Commas or spaces are used to separate elements in a row, and semicolons are used to separate individual rows. For example, the matrix

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

is defined as:

```
 A = [1 2 3; 4 5 6; 7 8 9] % row by row input
A =
1 2 3
4 5 6
7 8 9
```

A matrix can be also automatically extended, e.g.:

```
A(5,2) = 5  % assign 5 to the position (5,2);
 % the uninitialized
 % elements become zeros
A =
1  2  3
4  5  6
7  8  9
0  0  0
0  5  0
```

If needed, the other zero elements of the matrix `A` can be also defined, by e.g.:

```
 A(4,:) = [1, 2, 3];  % assign vector [1, 2, 3]
      % to the 4th row of A
 A(5,[1,3]) = [4, 5]; % a 'shortcut' when assigning:
      % A(5,1) = 4 and A(5,3) = 5
 A % how does the matrix A look like now?
```

As you have seen an example above, it is possible to manipulate (groups of) matrix-elements.

We have

```
 E = [] % this is an empty matrix of 0-by-0 elements!
E =
[]

B = rand(2,3) % a matrix of random numbers;
      % you will get a different one!
0.0227 0.9101 0.9222
0.0299 0.0640 0.3309
```

```
O = ones(3,2)
O =
1 1
1 1
1 1

B + O' % add 2 matrices;
       % why O' is needed instead of O?
ans =
1.0227 1.9101 1.9222
1.0299 1.0640 1.3309

S = ans./4 % divide all elements
   % of the matrix ans by 4
S =
0.2557 0.4775 0.4806
0.2575 0.2660 0.3327

C = [1 -1 4; 7 0 -1];
B * C % matrix multiplication is not defined!
??? Error using ==> *
Inner matrix dimensions must agree.

B .* C % but you can multiply element by element
ans =
0.0227 -0.9101 3.6888
0.2093 0 -0.3309

D = B * C' % or you can make the
   % multiplication legal; C' is now 3-by-2
D =
2.8014 -0.7633
1.2895 -0.1216

D^3 % this is equivalent to D * D * D
ans =
16.5899 -4.9902
8.4304 -2.5198

D.^3 - 2 % perform for all elem.:
 % raise to the power 3 and subtract 2
```

```
ans =
19.9849 -2.4447
0.1442 -2.0018

r = [1 3 -2];
13

R = diag(r) % create a diagonal matrix with r on the diagonal
R =
1 0 0
0 3 0
0 0 -2

r * O  % this is a legal operation:
 % r is a 1-by-3 matrix and O is
 % 3-by-2 matrix; O * r is an illegal operation
ans =
2 2

v = linspace(1,2,4)
v =
1.0000 1.3333 1.6667 2.0000
```

You can now compute the inner product between two vectors x and y,

$$x^T y = \sum_i x_i y_i,$$

in a simple way:

```
 f = [-1; 3; 5] % a column vector
 v = [-1; 2; 7] % a column vector
 f' * v % this is the inner product!
ans =
42
 f * v'   % be careful! now a 3-by-1
  % matrix is multiplied by a 1-by-3
  % matrix, which results in a 3-by-3
  % matrix (the outer product)
ans =
1 -2 -7
-3 6 21
```

```
-5 10 35

f .* v % this is element-by-element multiplication
1
6
35

sum (f .* v) % this is an another way to compute the inner product
ans =
42
```

1.4 Control flow

As you know, it is possible to execute more than one command at the same time. The separate commands should then be divided by commas (to display the output) or by semicolons (to suppress output display). Control flow loops increase the number of possibilities. A control flow structure is a block of commands that allows conditional code execution and making loops.

1.4.1 Logical and relational operators

To use control flow commands, it is necessary to perform operations that result in logical values: TRUE or FALSE. In Matlab the result of a logical operation is 1 if it is true and 0 if it is false. Table 2 shows the relational and logical operations. Another way to get to know more about them is to type help `relop`. The relational operators <, <=, >, >=, == and = can be used to compare two arrays of the same size or an array to a scalar. The logical operators &, | and allow for the logical combination or negation of relational operators. In addition, three functions are also available: `xor`, `any` and `all` (use help to find out more).

Important: The logical & and | have the equal precedence in Matlab, which means that those operators associate from left to right. A common situation is:

```
 b = 10;
 1 | b > 0 & 0
ans =
0
```

```
(1 | b > 0) & 0 % this indicates the same as above
ans =
0

1 | (b > 0 & 0)
ans =
1
```

This shows that you should always use brackets to indicate in which way the operators should be evaluated.

Command	Result
`a=(b>c)`	`a` is 1 if `b` is larger than `c`. Similar are: `<`, `>=` and `<=`
`a=(b==c)`	`a` is 1 if `b` is equal to `c`
`a=(b =c)`	`a` is 1 if `b` is not equal `c`
`a=˜b`	logical complement: `a` is 1 if `b` is 0
`a=(b&c)`	logical AND: `a` is 1 if `b` = TRUE AND `c` = TRUE
`a=(b|c)`	logical OR: `a` is 1 if `b` = TRUE OR `c` = TRUE

Table 2: Relational and logical operations.

1.4.2 Conditional code execution

Selection control structures, if-blocks, are used to decide which instruction to execute next depending whether expression is TRUE or not. The general description is given below. In the examples below the command `disp` is frequently used. This command displays on the screen the text between the quotes.

- `if` ... `end`

  One example of this structure is the following one:

  ```
  if (a > 0)
  b = a;
  disp ('a is positive');
  end
  ```

- `if` ... `else` ... `end` One example of this structure is the following one:

```
if (temperature > 100)
disp ('Above boiling');
toohigh = 1;
else
disp ('Temperature is OK');
toohigh = 0;
end
```

- `if ... elseif ... else ... end` One example of this structure is the following one:

  ```
  if (height > 190)
  disp ('very tall');
  elseif (height > 170)
  disp ('tall');
  elseif (height < 150)
  disp ('small');
  else
  disp ('average');
  end
  ```

Remark: To check the examples below, the notion of a script m-file will be useful. A script is an external file that contains a sequence of Matlab commands. What you need to do is to open an editor, (Matlab environment usually includes ones in one in the frontend) enter all commands needed for the solution of a task, save it with the extension `'.m'` (e.g. myscript.m) and then run it from the Command Window, by typing myscript. All commands in the script will be executed in Matlab. You will learn more on script m-files later.

Important: m-script file must be saved in one of the directories in Matlab's path.

Another selection structure is `switch`, which switches between several cases depending on an expression, which is either a scalar or a string. An example where this structure is used is the following one

```
switch method
case {1,2}
disp('Method is linear');
case 3:
disp('Method is cubic');
```

```
case 4:
disp('Method is nearest');
otherwise:
disp('Unknown method');
end
```

The statements following the first `case` where the expression matches the choice are executed. This construction can be very handy to avoid long `if .. elseif ... else ... end` constructions. The expression can be a scalar or a string. A scalar expression matches a choice if `expression == choice`. A string expression matches a choice if `strcmp(expression, choice)` returns 1 (is true) (`strcmp` is a command comparing two strings).

Note that the `switch`-construction only allows the execution of one group of commands.

1.4.3 Loops

Iteration control structures, loops, are used to repeat a block of statements until some condition is met. Two types of loops exist:

- the `for` loop that repeats a group of statements a fixed number of times; An example of this loop is the following one:

  ```
  sumx = 0;
  for i=1:length(x)
  sumx = sumx + x(i);
  end
  ```

  You can specify any step, including a negative value. The index of the `for`-loop can be also a vector. See some examples of possible variations:

  ```
  %Example 1
  for i=1:2:n
  ....
  end

  %Example 2
  for i=n:-1:3
  ....
  end
  ```

```
%Example 3
for x=0:0.5:4
disp(x^2);
end

%Example 4
for x=[25 9 81]
disp(sqrt(x));
end
```

- `while` loop, which evaluates a group of commands as long as expression is TRUE. One example of this kind of loop is the following one

```
N = 100;
iter = 1;
msum = 0;
while iter <= N
msum = msum + iter;
iter = iter + 1;
end;
```

1.4.4 Evaluation of logical and relational expressions in the control flow structures

The relational and logical expressions may become more complicated. It is not difficult to operate on them if you understand how they are evaluated. To explain more details, let us consider the following example:

```
if (~isempty(data)) & (max(data) < 5)
....
end
```

This construction of the `if`-block is necessary to avoid comparison if data happens to be an empty matrix. In such a case you cannot evaluate any expression and Matlab gives an error. The & operator returns 1 only if both expressions: `isempty (data) and max(data) < 5` are true, and 0 otherwise. When data is an empty matrix, the next expression is not evaluated since the whole &-expression is already known to be false. The second expression is checked only if data is a non-empty matrix. Remember to put logical expression units between brackets to avoid wrong evaluations!

Important: The fact that computers make use of floating-point arithmetic means that often you should be careful when comparing two floating-point numbers just by:

```
if (x == y)
....
end
```

(Of course, such a construction is allowed e.g. when you know that `x` and `y` represent integers.) Instead of the above construction, you may try using this:

```
if (abs (x - y) < tolerance) % e.g. tolerance = 1e-10
....
end
```

1.5 Script and function m-files

1.5.1 Script m-files

Matlab commands can be entered at the Matlab prompt. When a problem is more complicated this becomes inefficient. A solution is using *script m-files.* They are useful when the number of commands increases or when you want to change values of some variables and re-evaluate them quickly. Formally, a script is an external file that contains a sequence of Matlab commands (statements). However, it is not a function, since there are no input/output parameters and the script variables remain in the workspace. So, when you run a script, the commands in it are executed as if they had been entered through the keyboard.

It is possible to create a script opening the Matlab editor, and writing, for example:

```
x = input('write your name ');
disp(x)
```

saving the file as `'name.m'` and then run it by:

```
 name
```

The `name` script affects the workspace. Check:

```
 clear % all variables are removed from the workspace
 who % no variables present
 name
 who
Your variables are:
x
```

These generic name, `x`, may be easily used in further computations and this can cause side effects. Side effects occur in general when a set of commands change variables other than the input arguments. Since scripts create and change variables in the workspace (without warning), a bug, hard to track down, may easily appear. So, it is important to remember that the commands within a script have access to all variables in the workspace and all variables created in this script become a part of the workspace. Therefore, sometime it is better to use *function m-files*, when there is a specific problem to be solved.

1.5.2 Function m-file

Functions m-files are true subprograms, since they take input arguments and/or return output parameters. They can call other functions, as well. Variables defined and used inside a function, different from the input/output arguments, are invisible to other functions and the command environment. The general syntax of a function is presented below:

```
function [outputArgs] = function_name (inputArgs)
```

`outputArgs` are enclosed in `[ ]`:

- a comma-separated list of variable names;
- [ ] is optional when only one argument is present;
- functions without outputArgs are legal.

`inputArgs` are enclosed in `( )`:

- a comma-separated list of variable names;

- functions without `inputArgs` are legal.

Matlab provides a structure for creating your own functions. The first line of the file should be a definition of a new function (also called a header). After that, a continuous sequence of comment lines should appear. Their goal is to explain what the function does, especially when this is not trivial. Not only a general description, but also the expected input parameters, returned output parameters and synopsis should appear there. The comment lines (counted up to the first non-comment line) are important since they are displayed in response to the help command. Finally, the remainder of the function is called the body. Function m-files terminate execution and return when they reached the end of the file or, alternatively, when the command return is encountered. As an example, the function average is defined as follows:

```
funtion y=add5(x)
% add5 function returns y=x+5
y=x+5;
```

Important: The name of the function and the name of the file stored on disk should be identical. In our case, the function should be stored in a file called `add5.m`. When we type `help add5` we obtain the message

```
add5 function returns y=x+5
```

Another example of a simple function is the following one:

```
function [media,desv]=estad(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% estad function computes the
% mean and the standard deviation
% of the elements of vector x
% INPUTS:
%     vector x
% OUTPUTS:
%     media: mean of the elements of x
%     desv: standard deviation of the elements of x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[m,n]=size(x);
if  m==1
m=n;
end
```

```
media =sum(x)/m;
desv =sqrt(sum(x.^2)/m-media^2);
```

To calculate the standard deviation it takes into account that

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \overline{x}^2.$$

Warning: The functions `mean` and `std` already exist in Matlab. As long as a function name is used as variable name, Matlab can not perform the function. Many other, easily appealing names, such as `sum` or `prod` are reserved by Matlab functions, so be careful when choosing your names.

The `return` statement can be used to force an early return. An exemplary use of the return is given below:

```
function d = determinant(A)
%DETERMINANT Computes the determinant of a matrix A
[m,n] = size(A);
if (m ~= n)
disp ('Error. Matrix should be square.');
return;
else
d = det(A); % standard Matlab function
end
return;
```

Special function variables

Each function has two internal variables: `nargin` - the number of function input arguments that were used to call the function and `nargout` - the number of output arguments. Analyze the following function:

```
function [out1,out2] = checkarg (in1,in2,in3)
%CHECKARG Demo on using the nargin and nargout variables.
if (nargin == 0)
disp('no input arguments');
return;
elseif (nargin == 1)
s = in1;
p = in1;
```

```
disp('1 input argument');
elseif (nargin == 2)
s = in1+in2;
p = in1*in2;
disp('2 input arguments');
elseif (nargin == 3)
s = in1+in2+in3;
p = in1*in2*in3;
disp('3 input arguments');
else
error('Too many inputs.');
end
if (nargout == 0)
return;
elseif (nargout == 1)
out1 = s;
else
out1 = s;
out2 = p;
end
```

Local and global variables

Each m-file function has access to a part of memory separate from Matlab's workspace. This is called the function workspace. This means that each m-file function has its own local variables, which are separate from those of other function and from the workspace. However, if several functions and/or the workspace, all declare a particular variable as `global`, then they all share this variable (see `help global`). Any assignment to that variable is available to all other functions and/or the workspace. However, you should be careful when using global variables. It is very easy to get confused and end up with serious errors.

Indirect function evaluation

Using indirect function evaluation makes programming even more general, since functions can become input arguments. The crucial Matlab command here is `feval`, an abbreviation of function evaluation. The `feval` command allows execution of a function specified by a string. The general definition is as follows:

```
[y1,..,yn] = feval (F,x1,...,xn),
```

where `F` is a name of a function defined in Matlab, `x1,...,xn` are input arguments and `y1,...,yn` are possible output parameters. Consider an example:

```
 x = pi;
 y = cos(x);
 z = feval('cos',x);
```

The last command is also equivalent to the following two expressions:

```
 F = 'cos';
 z = feval(F,x)
```

1.6 Plots with Matlab

Matlab can be used to visualize both curves and surfaces. It allows to group and superpose graphics and it has several options that control colors and appearance.

1.6.1 Plots 2D

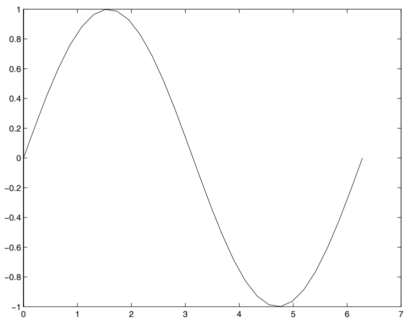
The 2D representation of a function can be obtained when a representation of this function is given in Cartesian or parametric coordinates.

The command `plot(x,y)` represents the couples (x_i, y_i) stored in vectors `x` and `y`, which should be vectors of equal length. `plot(y)` draws the points $(1, y_1), (2, y_2), \ldots, (n, y_n)$.

The command `linspace(a,b,N)` generates a vector of `N` equally spaced points between `a` and `b`. Hence, to generate the plot of a function we can write, for example,

```
x=linspace(0,2*pi,30);
y=sin(x);
plot(x,y)
```

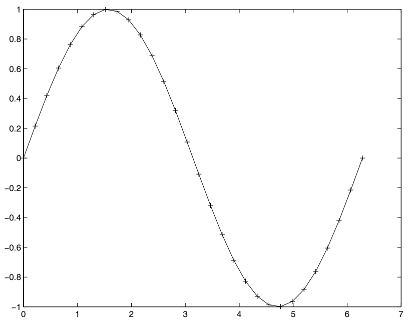
obtaining the following plot



It is possible to generate the same plot with 'lines' and 'crosses' writing

```
 plot(x,y,x,y,'+')
```

obtaining:



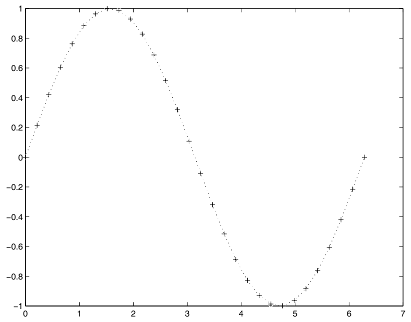
On the other hand, it is possible to control colors and styles for the lines in the plot using the symbols of the following table:

Symbol	Color	Symbol	Style
y	yellow	.	dots
m	magenta	o	circles
c	cyan	x	x-s
r	red	+	crosses
g	green	*	stars
b	blue	-	line
w	white	:	dots line
k	black	- .	lines and dots
		- -	dash line

For example, it is possible to write:

```
y=sin(x)
plot(x,y,'g:',x,y,'wo',x,y,'r:',x,y,'c+')
```

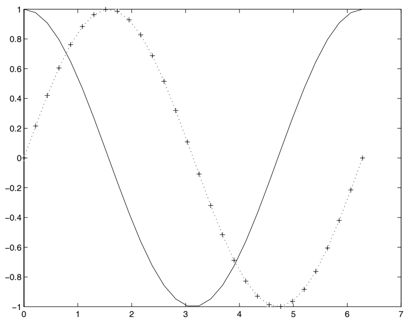
obtaining



Each time the command `plot` is executed the figure we has previously obtained disappears. If we want to superpose two or more plots, the command `hold` allows to keep the plots obtained in previous executions. For example, if we write:

```
plot(x,y,'g:',x,y,'wo',x,y,'r:',x,y,'c+')
hold on
z=cos(x)
plot(x,z)
```

the following plot is obtained,

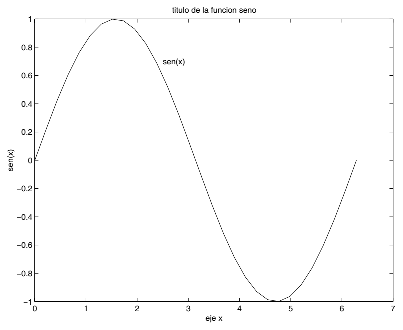


There are different commands to control the polt appearance. Thus, `grid on` adds a grid to the plot and `grid off` removes the grid. The commands `xlabel` and `ylabel` generate a label for the axes x and y, respectively. The command `title` generates a label for the plot. The command `text` allows to place a text in a determinate position of the plot.

An example of use of these functions is the following one

```
plot(x,y)
title('title of the function sin')
xlabel('axe x')
ylabel('sin(x)')
text(2.5,0.7,'sin(x)')
```

getting the following plot:



The command `axis` controls the axes appearance, for example

```
axis([xmin, xmax, ymin, ymax])
```

sets the axis x and the axis y in such a way that the maximum value for the x-s is `xmin`, the maximum value for the x-s is `xmax`, the minimum value for the y-s is `ymin`, the maximum value for the y-s is `ymax`.

`axis auto`: returns the axes scale to their default values.

`axis equal`: sets the same scale for the x-s and for the y-s.

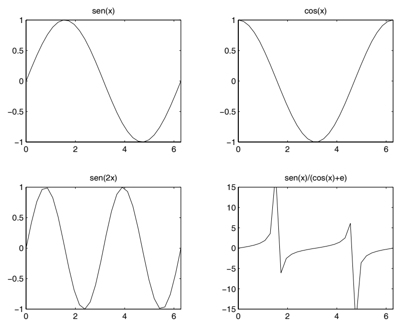
`axis normal`: returns the scales to their default values.

`axis off`: removes the axes.

`axis on`: plots the axes.

The command `subplot` allows to divide the plot region in different regions and in each region to plot a different graphic. An example of the use of the command `subplot` is the following one:

```
x=linspace(0,2*pi,30);
y=sin(x);
z=cos(x);
a=2*sin(x).*cos(x);
b=sin(x)./(cos(x)+eps);
subplot(2,2,1) % Divides the plot region
       % in 2 x 2 graphics and the first
       % region is selected (up left).
plot(x,y)
axis([0, 2*pi, -1, 1])
title('sin(x)')
subplot(2,2,2) % the second region is selected
plot(x,z)
axis([0, 2*pi, -1, 1])
title('cos(x)')
subplot(2,2,3) % the third region is selected
plot(x,a)
axis([0, 2*pi, -1, 1])
title('sin(2x)')
subplot(2,2,4) % the fourth region is selected
plot(x,b)
axis([0, 2*pi, -15, 15])
title('sin(x)/(cos(x)+e'))
```



Matlab can make plots of curves using logarithmic and semi-logarithmic scales. Thus,

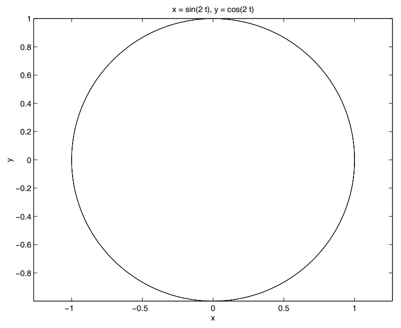
`loglog` : Is a function similar to `plot` but uses a logarithmic scale for axes x and y.

`semilogx` : Is a function similar to `plot` but uses a logarithmic scale for axis x and a linear scale for axis y.

`semilogy` : Is a function similar to `plot` but uses a logarithmic scale for axis y and a linear scale for axis x.

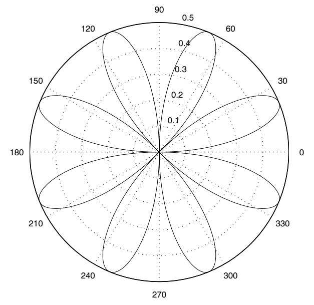
If we want to plot curves expressed in parametric coordinates, we can use the command `ezplot`. An example where this command is used is the following one:

```
ezplot('sin(2*t)','cos(2*t)',[0,2*pi])
```



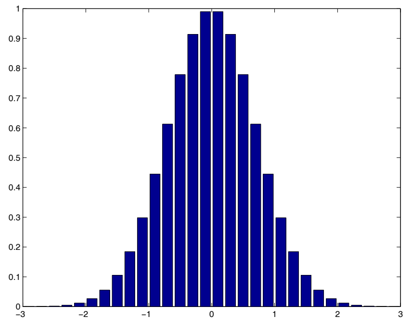
A curve in polar coordinates can be plot using the command `polar`, as it is shown in the following example:

```
t=0:0.01:2*pi;
r=sin(2*t).*cos(2*t);
polar(t,r)
```



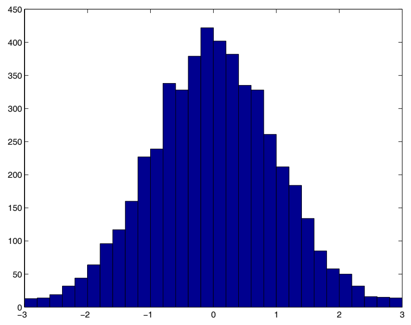
The command `bar` generates a bar diagram. An example where a bar diagram associated with a Gaussian function is generated, is the following one:

```
x=-2.9:0.2:2.9; % sets the number of divisions
y= exp(-x.*x);
bar(x,y)
```



The command `hist` generates the histogram associated with the data stored in a given vector. An example for a random numbers vector is the following one

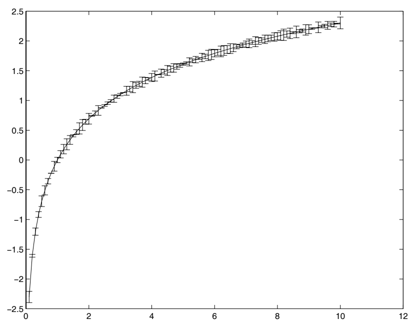
```
x=-2.9:0.2:2.9; % sets the number of divisions
y= randn(5000,1);
hist(y,x)
```



What happens if we use `randn` instead of `rand`?.

It is possible to plot graphics of data with an error bar associated. To do this, we make use the command `errorbar`, as it is shown in the following example

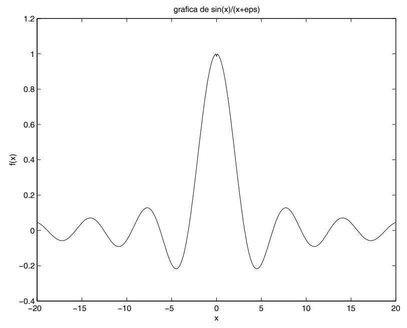
```
x=0.1:0.1:10;
y= log(x);
e=rand(size(x))/10;  % creates a vector with random errors
errorbar(x,y,e)
```



The function `fplot` generates the plot of a function of one variable, without generating vectors with the numeric data. The general form of the command is `fplot('fun',[xmin xmax])` or `fplot('fun',[xmin xmax ymin ymax])`.

An example is the following one

```
fplot('sin(x)./(x+eps)',[-20 20 -0.4 1.2]);
title('Plot of  sin(x)/(x + eps)');
xlabel('x')
ylabel('f(x)')
```



1.6.2 Plots 3D

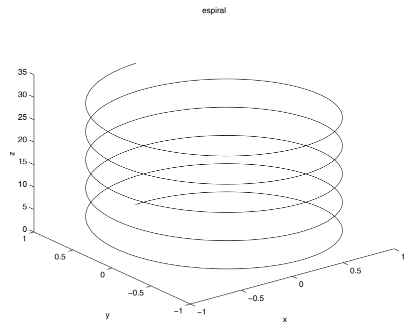
To generate the plot of a curve in the space it is possible to use the function `plot3`. Let us suppose that we want to plot the spiral

$$E = \begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) \\ z(t) = t \ , \quad t \in [0, 10\pi] \ . \end{cases}$$

We can write

```
t=0:pi/50:10*pi;
x=sin(t);
y=cos(t);
z=t;
plot3(x,y,z,'r');
title('espiral')
xlabel('x'), ylabel('y'),zlabel('z')
```

obtaining



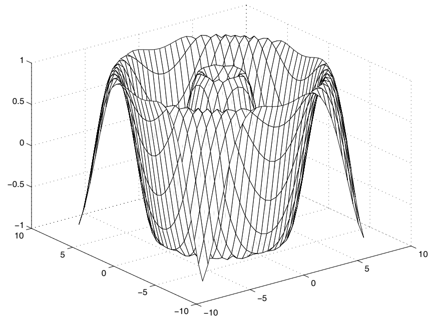
It is possible to control the axes scale using `axis([xmin, xmax, ymin,ymax,zmin,zma` in a similar way to what it is done for 2D plots. It is also possible to use `text(x,y,z,'texto')` to write a label in the position (x, y, z) of the graphic.

If we want to obtain a representation of the surface $Z = z(x, y)$, we can use different functions implemented in Matlab. Let us consider the function

$$z(x, y) = \sin\left(\sqrt{x^2 + y^2}\right) \ .$$

A possible representation is obtained in the following example:

```
x=-7.5:0.5:7.5;
y=x;
[X,Y]=meshgrid(x,y);
Z=sin(sqrt(X.^2+ Y.^2));
mesh(X,Y,Z)
```



Another possibility is

```
mesh(Z)
```

An alternative to the function `mesh` is the function `surf`. Hence, it is possible to use

```
surf(X,Y,Z)
```

It is possible to generate level curves for a determinate function using the commands `contour` y `contour3`. The following example

```
contour(X,Y,Z,20)
```

generates 20 level curves of the function $z(x, y)$ in the plane and

```
contour3(X,Y,Z,20)
```

gives the rame representation in the space.

It is possible to obtain a color bidimensional representation of a function $z(x, y)$ using the command `pcolor`. Thus,

```
pcolor(Z)
```

produces this representation using the default colormap. This colormap can be changed using the command `colormap`. Different possibilities are shown in the following table

option	Description
hsv	Tones saturation
hot	Black, red, yellow and white
pink	pink shadows
gray	grey scale
bone	grey scale with blue tones
jet	A variant of hsv
copper	Copper tones
prism	prism
flag	red, white, blue and black

This options can be used as it is done in the following example:

```
colormap(hot)
pcolor(X,Y,Z)
shading flat   % removes the grid
hold on
contour(X,Y,Z,20,'k')
hold off
```

It is possible to change the view point of the 3D plots using the command `view`. One possibility is to use the azimuth and the elevation in degrees of the direction we want to look at

```
view(phi,theta)
```

or we can use a vector in this direction

```
view([x1,x2,x3])
```

1.7 Storing and retrieving information

If we want to save the contents of a variable to an ascii file, we can make use of the function `save( )`. For example, the following code

```
x=1:100;
save vector.dat x -ascii
```

saves the contents of the variable `x` into the file `vector.dat` in ascii format. You can inspect the contents of this file using Notepad editor, for example.

On the other hand, if we have a file `vector.dat` and we want to assign its contents to a variable `y`, we can use the following commands

```
load vector.dat
y=vector
```

the `load` command reads the data in file `vector.dat` and assigns its contents to the variable `vector`.

To store and retrieve information from the hard disk we can use structures similar to the ones existing in the C language. In this way, the following code

```
x=1:10;
A=[x;log(x)];
fid=fopen('fichero.dat','w');
fprintf(fid,'%g\t%g',A);
fclose(fid);
```

stores in the file `fichero.dat` the contents of matrix `A`. If you inspect the file `fichero.dat` you will observe that the matrix is stored in 2 columns and 10 rows, which is different from the real structure of `A` which has 10 columns and 2 rows.

Now, we can read the contents of the file `fichero.dat` and assign them to the variable `B` with the following instructions

```
fid=fopen('fich','r');
Bt=fscanf(fid,'%g',[2,10]);
B=Bt;
```

now, the variable `B` is stored with the same structure as the file `fichero.dat`. To learn more about the different options of `fscanf( )` and `fprintf( )` see the Reference Manual pages.

Another useful function to read ascii files with a tabular structure is `textread( )`. Let us suppose that we have a file called `mydata.dat` with the following structure

```
  Name type score Y Y/N
  Sally Type1 12.34  45  Yes
  Joe  Type2  23.54  60  No
  Bill  Type1  34.90  12  No
```

we can assign the different columns of data to Matlab variables using the instruction

```
[names,types,score,y,answer]=
textread('mydata.dat','%s  %s  %f  %d  %s','headerlines',1);
```

The following set of instructions creates 10 figures and stores them in 10 files called `fig1.ps`, `fig2.ps`, ... , `fig10.ps`, which are in postscript format

```
x=0:0.1:1;
y=sin(x*pi);
z=[y];
hold on
for j=1:10
  z=[z;y.*exp(-0.05*j)];
  plot(x,z(j,:))
  filename=['fig',int2str(j),'.ps'];
  eval(['print ',filename,'-dps']);
end
hold off
```

You can learn more about `print( )` and `eval( )` functions looking at the Reference Manual pages.

1.8 Exercises

1. If $z_1 = 3+5i$ and $z_2 = 1-2i$. Obtain

$$|z_1|,\ z_1+z_2,\ z_1*z_2,\ z_1/z_2$$

Check that $\arg(z_1/z_2) = \arg(z_1) - \arg(z_2)$

2. Calculate

$$(8)^{\frac{1}{5}}\ ,\quad e^7+\ln(5)\ ,\quad \frac{\ln(9)+\sin(\pi/5)}{45-\cos(23)}\ .$$

3. Given

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 6 \\ 0 & 1 & 9 \end{pmatrix} \text{ y } b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},$$

obtain

   (a) $A+A$

   (b) $2A$

   (c) AA

   (d) A^2

   (e) $A\backslash b$ and check that the result is the same as the result of $A^{-1}b$.

   (f) b^T/A and check that the result coincides with b^TA^{-1}

   (g) Calculate $b\cdot b$ and $b\wedge b$.

4. Introduce the following matrices

$$D = \begin{bmatrix} 1 & -3 & 4 \\ 2 & -5 & 7 \\ 0 & -1 & 1 \end{bmatrix},\ E = \begin{bmatrix} 2 & 4 & 6 \\ 4 & 5 & 6 \\ 3 & 1 & 2 \end{bmatrix},\ c = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}.$$

   (a) Calculate $D+E, DE, D^TE$.

   (b) From matrices D and E extract their row and column vectors and a submatrix 2×2 constituted by the rows 1 and 2 and the columns 2 and 3.

   (c) Solve the system
$$E\ x = c$$

   (d) Construct a diagonal matrix whose diagonal contains the diagonal elements of matrix E.

   (e) Construct a matrix with block structure $[DE]$.

5. Calculate, using the function `sum( )`

$$\sum_{i=1}^{10} 2^i\ .$$

And, using the function `prod( )`

$$\prod_{j=1}^{10} \frac{j}{j+1}\ .$$

6. (a) Write a loop to calculate the sum of the first 25 natural numbers.

   (b) Calculate the largest natural number satisfying

$$3^n + \ln(n) < 2000.$$

   (c) Calculate the product of the first 10 odd numbers.

7. Given the function

$$F(s) = \frac{\frac{b}{m}s + \frac{k}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}}\ ,$$

associated with the transfer function of a simplified model of an automovil suspension mechanism. Define this function as a Matlab function. Use it to store in a vector the values of $F(s)$ when $s = 0, 1, 2, \ldots, 100$. Consider $b = 1$, $m = 2$, $k = 3$.

8. Write a program to take in a set of 10 exam marks from the keyboard together with the name of the student. Sort the marks into increasing numerical order and print them out. The program should produce in the output the mean and the standard deviation of the marks.

9. La following expression

$$\log(n!) \approx n\log(n) - n\ ,$$

is the Stirling's approximation of $\log(n!)$. Write a program to check this approximation for $n = 100$, $n = 1000$ y $n = 5000$, bearing in mind that 5000! is much too large number to be stored in a standard way by the computer.

10. Plot the following curves:

a) $y = x^5 + 3x + 5$

b) $y = e^{3x} + \sin(x)$

c) $\left\{ \begin{array}{l} x(t) = t - \sin(t) \\ y(t) = t - \cos(t) \end{array} \right.$.

d) $\rho = \cos(\theta) + \mathrm{sen}(\theta/4)$

11. Using the proper function divide the plotting window in four sub-windows and plot in each sub-window one of the following functions:

$$\begin{aligned} y &= \tan(x) \\ y &= \cosh(5 * x) \\ y &= e^{2x} + 5\sin(2x) \\ y &= x^2 + 3x + 4 \end{aligned}$$

12. Estimate the crossing point of the following functions in the interval $[0, 2]$

$$\begin{aligned} y &= 2x^3 + 5.4x^2 + 4.8x + 1.4 \\ y &= 7x - 10 \end{aligned}$$

13. Obtain a representation of the surface

$$z(x, y) = \sqrt{x^2 + y^2} \ ,$$

for $(x, y) \in [-1, 1] \times [-1, 1]$.

14. Plot the surface

$$z(x, y) = x^2 + y^2 \ ,$$

in the square $[-1, 1] \times [-1, 1]$ and superpose to the plot 5 level curves.

15. Create a matrix of the form

$$M = \left(x_i, x_i^2, x_i^3\right) \ ,$$

being x the vector

$$x = (0.1, 0.2, \ldots, 10)^T \ .$$

Store this matrix in a file called `matrix.dat`. Use to do this the functions `save( )` and `fprintf( )`.

Read the data from the file `matrix.dat`, assign them to the variable A and compute MM^T and M^TM.

16. Create a file in ascii format with the following data corresponding to the marks obtained in a test

Name	mark
María	8
Pepe	5
Juan	2.5
Luisa	7
Samuel	3
Rafa	100

Read the data with Matlab and calculate the mean and the variance of the marks.

17. A traveling pulse has the equation

$$y(x, t) = \exp(-20(x - 0.1t)^2) \ .$$

create 10 files called `pulse1.bmp`, `pulse2.bmp`, ... , `pulse10.bmp` containing the plot of the shape of the pulse for the each one of the first 10 seconds of evolution in windows bitmap format.

18. Solve the following assignment[2]: ACME (the company you work for) is currently under contract to help the Corporation Of Nuclear Electrical Devices, one of the nation's largest commercial producers of electricity, to build a laser fusion reactor. This reactor will be powered by a single strong laser pulse targeted at a some small object in the middle of this device. The details of how this works are unimportant, but the characteristics of the laser pulse, at least for this particular project, are crucial.

The people in CON ED's laboratory have made several measurements on the laser pulse during the time when it is turned on, but they still have no idea what the pulse looks like as a function of time. Because of your excellent reputation in doing numerical computations, the people at CON ED have decided that they would like to have you produce a picture of this function. Your boss has agreed that you are just the person for the job.

The power produced by the pulse can be considered a function of time, say $P(t)$. The measurements produced by CON ED are the total energy

[2]From the book: T.A. Grandine, *The Numerical Methods programming Projects Book.* Oxford University Press, (1990).

of the pulse during the first second of operation, the first and second moments of the power function, and two pieces of spectral information. More precisely, CON ED has measured the following numbers which estimate the following integrals

$$\begin{aligned} 0.43843 &= \int_0^1 P(t)\,dt \ , \\ 0.16475 &= \int_0^1 tP(t)\,dt \ , \\ 0.07844 &= \int_0^1 t^2P(t)\,dt \ , \\ 0.33298 &= \int_0^1 \sin(\pi t)P(t)\,dt \ , \\ 0.14434 &= \int_0^1 \cos(\pi t)P(t)\,dt \ , \end{aligned}$$

You will approximate $P(t)$ by a function $S(t)$ which has the form

$$S(t) = a_1 + a_2t + a_3t^2 + a_4\sin(\pi t) + a_5\cos(\pi t) \ .$$

Thus, it will your job to compute a_i. This is done by **projecting** $P(t)$ onto the space of functions which are of the same form as $S(t)$ above. That is you insist that the five equations

$$\begin{aligned} \int_0^1 S(t)\,dt &= \int_0^1 P(t)\,dt \ , \\ \int_0^1 tS(t)\,dt &= \int_0^1 tP(t)\,dt \ , \\ \int_0^1 t^2S(t)\,dt &= \int_0^1 t^2P(t)\,dt \ , \\ \int_0^1 \sin(\pi t)S(t)\,dt &= \int_0^1 \sin(\pi t)P(t)\,dt \ , \\ \int_0^1 \cos(\pi t)S(t)\,dt &= \int_0^1 \cos(\pi t)P(t)\,dt \end{aligned}$$

be satisfied. Once you have computed the a_i, you have a function which should be a good approximation to $P(t)$. Produce a graph of this function in the interval $[0, 1]$. Be sure this will be a contribution to provide future energy to the World.

Chapter 2

Interpolation and fitting of data

Matlab has implemented several functions for the interpolation and fitting of data. We will review these functions and we will study some examples.

2.1 Polynomials in Matlab

To represent a polynomial Matlab uses a vector whose components are the coefficients of the polynomials ordered in decreasing powers of the variable x. Thus, the vector

```
  1 -4 -3 -4 10
```

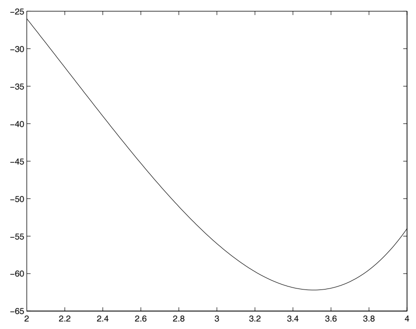
represents the polynomial

$$P(x) = x^4 - 4x^3 - 3x^2 - 4x + 10 \ .$$

If p is a vector of the coefficient of a given polynomial, we can evaluate the polynomial for a given value of the variable s_0 using the command `polyval(p,s0)`. This function allows to obtain a graphic representation of the polynomial in a certain interval. For example the following instructions

```
p=[1   -4  -3  -4  10];
x=2:0.01:4;
y=polyval(p,x);
plot(x,y)
```

allow to obtain the plot of the polynomial $P(x)$ in the interval $[2, 4]$, which is of the form,



The roots of a given polynomial can be obtained using the function `roots( )`. Thus, the command

```
 raices=roots(p)
```

provides the result

```
4.7200
-0.8600+1.1743i
-0.8600-1.1743i
1.0000
```

A polynomial can be reconstructed from its roots using the instruction

```
p=poly(raices)
```

Let us suppose that we have the following polynomials

$$\begin{aligned} a(x) &= x^2 + 3x + 2 \ , \\ b(x) &= x^2 + 5x + 1 \ , \end{aligned}$$

and we write the vectors of the coefficients

```
a = [1 3 2]
b = [1 5 1]
```

the product of polynomials can be calculated by means of the function `conv( )`, hence,

```
    conv(a,b)
```

provides the result

```
 1 8 18 13 2
```

which corresponds to the polynomial

$$x^4 + 8x^3 + 18x^2 + 13x + 2 \ .$$

2.2 Interpolation

Given a table with a series of data, as the following one

x_0	x_1	$\cdots$	$\cdots$	$\cdots$	x_n
y_0	y_1	$\cdots$	$\cdots$	$\cdots$	y_n

the function `interp( )` allows to calculate interpolated values for data placed between the extrema values collected in the table.

The function works in the following way

```
      yi=interp(x,y,xi,'method')
```

where `x` is the vector of xs from the table, whose components must be ordered in an increasing or a decreasing way. `y` is the vector of ys from the table. `xi` is the value or a vector of values we want to interpolate. `'method'` is the particular technique we want to use to obtain the interpolation. This argument is an optional one, and it can take the following values:

`'linear'` A linear interpolation is performed.

`'spline'` The interpolation is performed constructing a cubic spline for the data of the table.

`'cubic'` A cubic interpolation is calculated for the data of the table. To be able to use this option the data in vector `x` must be equally spaced.

Let us see an example. In the following table it is shown the population of the U.S.A., expressed in millions of people from 1900 to 1990

año	1900	1910	1920	1930	1940
Pobl.	75.995	91.972	105.711	123.203	131.669
año	1950	1960	1970	1980	1990
Pobl.	150.697	179.323	203.212	226.505	249.633

We can write the following commands

```
tiem=1900:10:1990;
pob=[75.995   91.972 105.711 123.203 131.669 ...
     150.697 179.323 203.212 226.505 249.633];
```

obtaining two vectors with the data of the table.

We can estimate the population of 1975 writing

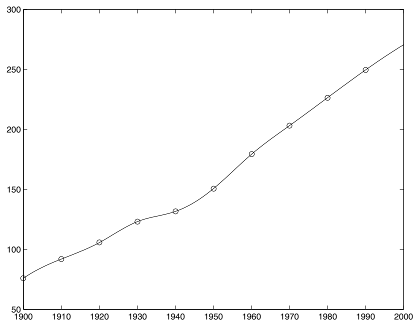
```
interp1(tiem,pob,1975)
```

obtaining as a result 214.8585 millions.

On the other hand, we can plot the evolution of the population with the following instructions:

```
xi=1900:2000;
yi=interp(tiem,pob,xi,'spline')
plot(tiem,pob,'o',xi,yi)
```

obtaining



2.3 Fitting of data

The function `polyfit( )` provides a polynomial fitting for the data in a table. This function works in the following way:

```
p=polyfit(x,y,n)
```

where `x` and `y` are vectors with the data of the table, `n` is the degree of the polynomial used in the interpolation, and `p` is the coefficients vector obtained from the fitting.

Let us see an example of use of this function.

It is known that the data in the following table

x	1	2	3	4	5
y	4.52	4.09	3.70	3.35	3.03
x	6	7	8	9	10
y	2.74	2.48	2.25	2.03	1.84

correspond with an exponential decay of the form

$$y = a\exp(bx) \ .$$

To estimate the values of a and b from the data in the table we write the following instructions.

```
x=1:10;
y=[ 4.52,4.09,3.70,3.35,...
3.03,2.74,2.48,2.25,2.03,1.84];
y1=log(y);
p=polyfit(x,y1,1)
```

and we obtain

```
       -0.0999 1.6082
```

which are the parameters of a model of the form

$$\ln(y) = bx + \ln(a) \ ,$$

thus, we have that $b = -0.0999$ and $a = 4.9938$.

2.4 Exercises

1. Calculate:

   a) The fifth roots of -1.

   b) $(x^4 + 3x^3 + 2x + 0.5)(x^6 + 3x + 0.3)$

   c) The real roots of the polynomial $x^6 - 3x^5 - x^4 + 7x^3 - 14x^2 + 10x - 12$.

2. The following table shows the time evolution of the mineralization nitrogen in the ground

Time (días)	Nmin (mg/kg)
7	9.466
14	8.211
27	15.590
41	17.615
55	20.215
83	21.734

   Obtain the time necessary to obtain a value of mineralization nitrogen of 18.5 mg/Kg. To do this, use a linear interpolation and a spline and compare the obtained results.

3. Interpolate the function

$$f(x) = \frac{1}{1+x^2} \ ,$$

   in two ways. First, using a set of points $x_n = -5 + n$, $y_n = f(x_n)$ with $n = 0, 1, 2, \dots, 10$, and second, using the set $x_n = 5\cos(\pi n/10)$, $y_n = f(x_n)$ with $n = 0, 1, 2, \dots, 10$. Use a polynomial of degree 9 to perform the interpolation. Make a graphic representation of both results.

4. It is known that the relation between the live weight of a butterfly, W (g), and the consumed oxygen, R in ml/h, is approximately of the form

$$R = bW^a \ .$$

   Obtain the values of a and b using the data in the following table

W	R
0.017	0.154
0.087	0.296
0.174	0.363
1.11	0.531
1.74	2.23
4.09	3.58
5.45	3.52
5.96	2.40

5. It is known that, under certain conditions, the time evolution of a given population can be modeled using the logistic equation that is of the form

$$P(t) = \frac{1000}{1 + Ce^{At}} \ .$$

   Obtain C and A for the data in the following table:

$P(t)$	200	400	650	850	950
t	0	1	2	3	4

6. From the calibration of a motor the following data have been obtained

Time (milliseconds)	Distance (cm)
0.00	0.0
1.40	1.0
2.37	2.0
3.30	3.0
3.37	4.0
4.11	5.0
5.42	6.0
5.71	7.0
6.39	8.0
7.26	9.0
7.82	10.0
8.67	11.0
9.12	12.0
9.66	13.0
10.70	14.0
11.23	15.0
11.25	16.0
12.47	17.0
12.79	18.0
13.20	19.0
14.12	20.0
14.83	21.0
15.83	22.0
16.04	23.0

Perform a fitting using a polynomial of degree 4 to obtain a model of the distance as a function of time. Make a plot of the fitted polynomial and the data of the table.

Chapter 3

Roots, integrals and differential equations

Now, we will show some Matlab functions to perform approximate calculations.

3.1 Finding roots of equations

Let us suppose that we want to find the roots of the function

$$f(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.4} - 6 \ .$$

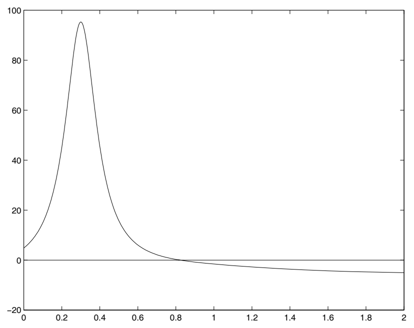
First, we make a plot of the function together with the axis of x-s. To do this we define the function

```
function y=func(x)
y =1/((x-0.3).^2 +0.01) +  1/((x-0.9).^2 + 0.4)- 6;
```

and we use the function `fplot( )`, in the following way:

```
fplot('func', [0 2])
hold on
fplot('0', [0 2], 'g')
hold off
```

obtaining



We see that there is a zero near $x = 0.8$. We can improve this result using the function `fzero( )`, Hence, we can write

```
fzero('func', 0.8)
```

obtaining the result $x = 0.8222$.

Let us suppose now, that we want to obtain a root of the system of equations

$$\begin{aligned} x_1^2 - 10x_1 + x_2^2 + 8 = 0 \ , \\ x_1 x_2^2 + x_1 - 10x_2 + 8 = 0 \ . \end{aligned} \tag{3.1}$$

The following Matlab function is a possible implementation of Newton's method for a system of equations,

```
function [xr,k]=newtonsi(x,tol,imax)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Metodo de Newton para sistemas de ecuaciones
%  Uso: [xr,k]=newtonsi(x,tol,imax)
%
%  Input:
% x = vector x1,x2,...,xn inicial,
% tol=tolerancia
%
%  Se ha de disponer de las funciones:
%      f.m  funcion y=f(x) donde se define el sistema
%      jac.m funcion df=jac(x) donde se define la matriz
```

```
%     derivada del sistema.
%
%  Output: xr= raiz,  k= numero de iteraciones.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k=1;
epi=1;
x1=x;
while norm(epi)>tol
   x=x1;
   fxn=f(x);
   axn=jac(x);
   epi=axn\fxn';
   x1=x-epi';
   k=k+1;
   if k>imax
      disp('no converge')
      break
   end
end
xr=x1;
```

This function uses two auxiliary functions, `f.m` where it is defined the function associated with the system of equations and `jac.m` that defines the derivative of this function. For the system (3.1) these functions are of the form

```
function y=f(x)
y(1)=x(1)^2-10*x(1)+x(2)^2+8;
y(2)=x(1)*x(2)^2+x(1)-10*x(2)+8;
```

and

```
function df=jac(x)
% matriz jacabiana para usar con newtonsi.m
df(1,1)=2*x(1)-10;
df(1,2)=2*x(2);
df(2,1)=x(2)^2+1;
df(2,2)=2*x(1)*x(2)-10;
```

Once we have written these functions, we can execute the following instruction

```
[xr,k]=newtonsi([0 0],1.e-5,100)
```

obtaining as results the root `xr`= $[1.00, 1.00]$ in `k`= 10 iterations.

3.2 Approximate integration

To calculate definite integral numerically Matlab has the functions `trapz`, `quad` y `quad8`.

If we want to calculate

$$\int_0^{\frac{\pi}{2}} \operatorname{sen}(x)\, dx \ ,$$

we can write

```
x=0:pi/10:pi/2;
y=sin(x);
integral=trapz(x,y)
```

The function `trapz( )` uses the trapezoidal rule based on the divisions of the interval defined by the elements of vector `x`.

Other possibilities are

```
    integral=quad('sin',0,pi/2)
```

or,

```
    integral=quad8('sin',0,pi/2)
```

3.3 Differential equations

Matlab has implemented, among other ones, the functions `ode23` and `ode45`, to solve initial values problems associated with differential equations.

Let us suppose that we want to solve an initial value problem associated with Van der Pol's oscillator

$$\frac{d^2x}{dt^2} - \nu(1-x^2)\frac{dx}{dt} + x = 0 \ .$$

First, we have to rewrite the differential equation into its normal form, that is, we introduce the auxiliary variables $y_1 = x$ and $y_2 = \frac{dx}{dt}$, obtaining the following system of differential equations

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \ , \\ \frac{dy_2}{dt} &= \nu(1-y_1^2)y_2 - y_1 \ . \end{aligned}$$

In an m-file we write a function with the following structure

```
function yprime=vdpol(t,y)
% devuelve las derivadas del oscilador de Van der  Pol
%  se escoge mu = 2
mu=2;% se suele elegir 0 < mu < 10
yprime = [ y(2);
  mu*(1-y(1)^2)*y(2) - y(1) ] ;
```

After this, we can calculate the initial value problem associated with Van der Pol's oscillator with initial conditions $y_1(0) = 1$, $y_2(0) = 0$ for $t \in [0, 30]$ in the following way

```
[t,y]=ode23('vdpol',0,30,[1;0]);
```

and to plot the solutions in the phase plane we write:

```
y1=y(:,1);
y2=y(:,2);
plot(y1,y2)
```

The function `ode45` works in a similar way.

3.4 Exercises

1. Obtain the roots of the equation

$$x = x^2 \operatorname{sen}(x) \ ,$$

placed in the interval $[0, 10]$.

2. Use the function `fslove` to obtain a root of the system of equations

$$x_1^2 - 10x_1 + x_2^2 + 8 = 0 \ ,$$
$$x_1 x_2^2 + x_1 - 10x_2 + 8 = 0 \ .$$

3. We have to build a roof using a machine that compresses a flat sheet of aluminium converting it into a sheet whose transversal section is a sinoidal wave. Let us suppose that we need a roof of 50 cm and that each undulation has a height with respect to the horizontal line of 1 cm and a period of 2π cm. The problem of finding the length of the original sheet is solved calculating

$$L = \int_0^{50} \sqrt{1+\cos^2(x)}\, dx \ .$$

Estimate this length using Matlab.

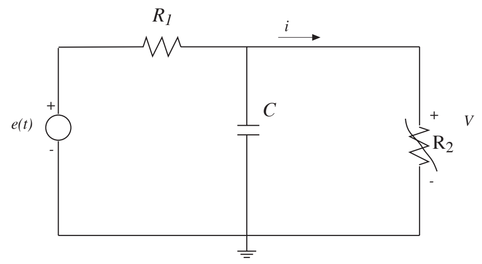
4. Estimate the value of
$$\int_0^{\infty} \frac{\cos(2x)}{\cosh(x)} \ ,$$
and compare it with the exact value, $\frac{\pi}{2}\,\mathrm{sech}(\pi)$.

5. The current through a RCL circuit which is driven by an alternating tension $E(t)$ is described by the equation

$$L\frac{dI}{dt} + RI + \frac{1}{C}Q = E(t) \ .$$

Let us suppose that we have a resistance of 2Ω, a capacitance of 1 F and an inductance 0.5 H. If $E(t) = 5\,\mathrm{sen}(t)$, knowing that the following relation holds
$$\frac{dQ}{dt} = I \ ,$$
that the condenser at $t = 0$ is uncharged, and that initially there is no current through the circuit, obtain a plot with the time evolution of the current through the circuit from $t = 0$ to $t = 1$ seconds, and the charge in the condenser at $t = 0.5$ seconds.

6. In the circuit we show in the figure



we have a linear resistance $R_1 = 1$ Ohm, a capacitance $C = \frac{1}{2}$ F, an alternating tension generator $e(t) = 18 + 5\cos(50t)$, and a non-linear resistor R_2, which follows a law $i = 2V^3$, where V is the tension drop through the element. The circuit is modeled using the equation

$$\frac{1}{2}\frac{dV}{dt} + V + 2V^3 = 18 + 5\cos(50t)$$

If $V(0) = 0$ V., obtain and plot the time evolution of $V(t)$ in the interval $[0, 1]$.

Chapter 4

Some topics on signal analysis

Correlation is an operation used in many applications in digital signal processing. It is a measure of the degree to which two sequences are similar. Given two real-valued sequences $x(n)$ and $y(n)$ the cross-correlation of $x(n)$ and $y(n)$ is a sequence $r_{xy}(n)$ defined as

$$r_{xy}(l) = \sum_{n=-\infty}^{+\infty} x(n)y(n-l) \ . \tag{4.1}$$

The index l is called the shift or lag parameter. The special case of (4.1) when $y(n) = x(n)$, is called autocorrelation and is defined by

$$r_{xx}(l) = \sum_{n=-\infty}^{+\infty} x(n)x(n-l) \ . \tag{4.2}$$

It provides a measure of self-similarity between different alignments of the sequence.

Given two sequences $x(n)$ and $h(n)$ the linear convolution of these sequences is defined as

$$y(n) = x(n) * y(n) = \sum_{k=-\infty}^{+\infty} x(k)h(n-k) \ . \tag{4.3}$$

1. In a certain concert hall, echoes od the original audio signal $x(n)$ are generated due to the reflections at the walls and ceiling. The audio signal experienced by the listener $y(n)$ is a combination of $x(n)$ and its echoes. let
$$y(n) = x(n) + \alpha x(n-k) \ ,$$

where k is the amount of delay in samples and α is its relative strength. We want to estimate the delay using the correlation analysis.

   (a) Determine analytically the autocorrelation $r_{yy}(l)$ in terms of the autocorrelation $r_{xx}(l)$.

   (b) Let $x(n) = \cos(0.2\pi n) + 0.5\cos(0.6\pi n)$, $\alpha = 0.1$, and $k = 50$. Generate 200 samples of $y(n)$ and determine its autocorrelation (use `xcorr( )`). Can you obtain α and k by observing $r_{yy}(l)$?.

2. When we have two finite sequences $x(n)$ and $h(n)$ of lengths N_x and N_y, respectively, then their linear convolution (4.3) can also be implemented using matrix-vector multiplication. If elements of $y(n)$ and $x(n)$ are arranged in column vectors x and y, respectively, then (4.3) can be written
$$y = Hx \ ,$$
where the linear shifts $h(n-k)$, $n = 0, \ldots, N_x - 1$ are arranged as rows in the matrix H. This matris has an interesting structure and is called a Toeplitz matrix. To investigate this matrix, consider the sequences
$$x(n) = \{1, 2, 3, 4\} \ , \quad h(n) = \{3, 2, 1\} \ .$$

   (a) Determine the convolution $y(n) = h(n) * x(n)$.

   (b) Express $x(n)$ as a 4×1 column vector x and $y(n)$ as a 6×1 vector y. Now determine the 6×4 matrix H so that $y = Hx$.

   (c) What can you say about the first row and the first column of H?

3. Matlab provides a function called `toeplitz` to generate a Toeplitz matrix, given the first row and the first column.

   (a) Using this function develop an alternate Matlab function to implement linear convolution. The format of the function should be

```
function [y,H]=conv_tp(h,x)
% Linear convolution using Toeplitz matrix
%---------------------------------------
% [y,H] = conv_tp(h,x)
% y = output sequence in column vector form
% H = Toeplitz matrix corresponding to sequence h
% h = Impulse response in column vector form
% x = Input sequence in column vector form
```

(b) Verify your function on the sequences given in the exercise exposed above.

For a given finite sequence, $x(n)$, $n = 0, \dots, N-1$ it is possible to define its discrete Fourier transform as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i\frac{2\pi}{N}nk} , \tag{4.4}$$

that can be viewed as an approximation to the Fourier Transform of the function $x(t)$ such that $x(nT) \equiv x(n)$, where T is the sampling time. This Fourier transform is evaluated at frequencies $w_k = \frac{2\pi k}{NT}$, $k = 0, 1 \dots N-1$.

The inverse discrete Fourier transform is defined as

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{i\frac{2\pi}{N}nk} . \tag{4.5}$$

The following Matlab functions implement the discrete Fourier transform of a sequence and its inverse.

```
function [Xk]=dft(xn,N)
% Computes the Discrete Fourier Transform
% ----------------------------------------
% [Xk] = dft(xn,N)
% Xk = DFT coeff. array
% xn = N-point finite-duration sequence
% N = Length of DFT
%
n=[0:1:N-1];   % row vector for n
k=[0:1:N-1];   % row vector for k
WN=exp(-i*2*pi/N);   % Wn factor
nk=n'*k;              % creates a NxN values of nk values
WNnk=WN.^nk;   % DFT matrix
Xk=xn*WNnk;

function [xn]=idft(Xk,N)
% Computes the Inverse Discrete Fourier Transform
% ----------------------------------------
% [Xk] = idft(Xk,N)
% xn = N-point sequence
```

```
% Xk = DFT coeff.
% N = Length of DFT
%
n=[0:1:N-1];   % row vector for n
k=[0:1:N-1];   % row vector for k
WN=exp(-i*2*pi/N);   % Wn factor
nk=n'*k;              % creates a NxN values of nk values
WNnk=WN.^(-nk);   % IDFT matrix
xn=(Xk*WNnk)/N;
```

Compare the performance of these functions with the Fast Fourier Transform algorithm implemented in the Matlab functions `fft` and `ifft`. To do this comparison use the function `etime`.

The Fourier transform of a given signal provides information about its frequency contents. In this way, for example, if we want to know if two signal have a common frequency we can calculate the Fourier transform of the cross-correlation of both signals. The magnitude of the Fourier transforms give us the main frequency, and the phase of the Fourier transform indicates the phase shift between the two signals. Check these assertions using the sequences $x(n) = \sin(0.1n)$ and $y(n) = \cos(0.1n)$.

4.0.1 Filtering signals using the DFT

Generally the experimental studies are carried out performing a series of measurements. In this measurements we obtain, together with the information associated with the system under study, some other unwanted contributions known as *noise*. The set of measurements obtained are called signal, and a process to eliminate the noise is called filtering.

Now we will show that the DFT can be used to filter noise in some cases. Let us start from a function

$$x(t) = \sin(0.1t) + 3\sin(0.3t) ,$$

which plays the role of process under study. The function is sampled using a sample time of $T = 0.05$ seconds and $N = 1024$ samples are considered, obtaining

$$x(n) = \sin(0.1nT) + 3\sin(0.3nT) .$$

To simulate the noise we add to the signal a uniform random process, ε_n, obtaining the noisy signal

$$x_r(n) = \sin(0.1nT) + 3\sin(0.3nT) + \varepsilon_n .$$

In the figure 4.1 we show the noisy signal, $x_r(n)$ and the clean signal $x(n)$.

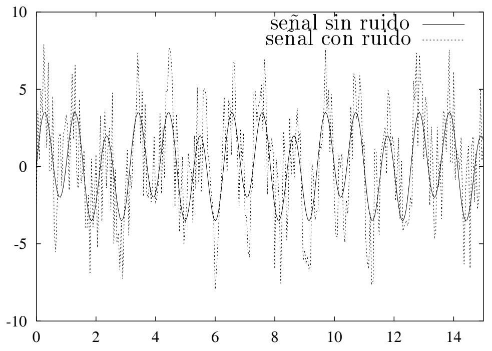


Figure 4.1: Noisy signal and clean signal.

The next step consists of calculating the discrete Fourier transform of the signals $x(n)$ y $x_r(n)$, called $X(k)$ and $X_r(k)$, respectively,

$$X(k) = \mathcal{D}[x(n)]_k \ , \quad X_r(k) = \mathcal{D}[x_r(n)]_k \ , \quad k = 0, \dots N-1 \ .$$

In figure 4.2 the magnitude of these transforms is shown, $\mid X(k) \mid$ and $\mid X_r(k) \mid$. In this figure we can distinguish two fundamental harmonics associated with the clean signal and some secondary harmonics, associated with noise with lower amplitude. To filter the signal from a plot similar to 4.2 a threshold, U, is selected and the filtered sequence in the frequency domain is of the form

$$X_f(k) = X_r(k)H(k) \ , \quad k = 0, \dots, N-1 \ ,$$

where $H(k)$ is the filter

$$H(k) = \begin{cases} 1 & \text{si} \ \mid X_r(k) \mid \geq U \ , \\ 0 & \text{si} \ \mid X_r(k) \mid < U \ . \end{cases}$$

To obtain the filtered signal it is enough to calculate the inverse discrete Fourier transform of $X_f(k)$,

$$x_f(n) = \mathcal{D}^{-1}[X_f(k)]_n \ .$$

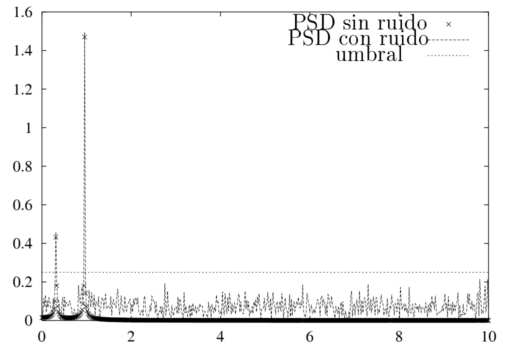


Figure 4.2: Magnitudes of $X(k)$ and $X_r(k)$.

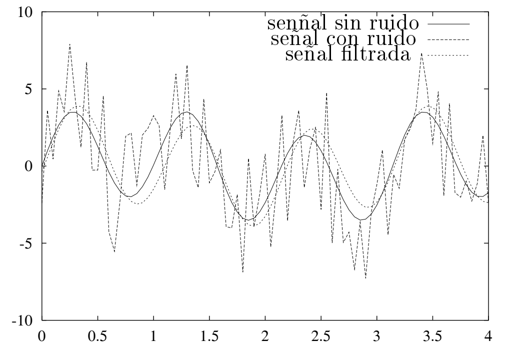


Figure 4.3: Filtered signal, clean signal and noisy signal.

In figure 4.3, the filtered signal, the noisy signal and the clean signal are shown

We observe that in this simple case it is possible to design a filtering process using the DFT.

Design a Matlab function to filter a given signal using the DFT. This function should select an adequate threshold and produce a plot for comparison and the filtered signal as outputs.

Chapter 5

Filters

5.1 Analog filters

Here we will present a brief introduction [1] to two popular analog kind of filters, the Butterworth filters and the Chebyshev filters.

5.2 Butterworth filters

We will represent an analog filter by its transfer function $H(s)$, which is related to the frequency response of the filter, $T(w)$, by the relation $T(w) = H(iw)$. In this way the magnitude of the frequency response of the filter is given by

$$|T(w)|^2 = H(s)H(-s)|_{s=iw} \ .$$

The Butterworth response is characterized by

$$|T_n(w)|^2 = \frac{1}{1+w^{2n}} \ , \qquad (5.1)$$

which for large n constitutes a good approximation for the ideal response, as can be seen in figure 1.

[1]Mainly from the book: M.E. Valkenburg, 'Analog Filter Design'. Harcourt Brace Jovanovich College Publishers. (19982).

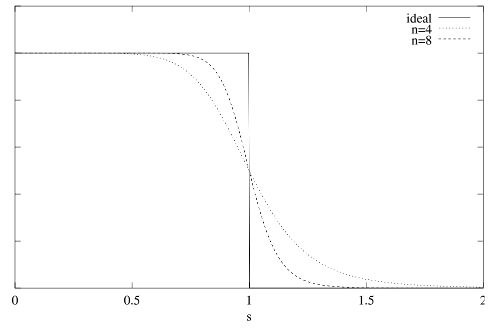


Figure 1.- Butterworth response.

To obtain the transfer function of the filter, we rewrite (5.1) in terms of s

$$H(s)H(-s) = \frac{1}{1+\left(\frac{s}{i}\right)^{2n}} \ .$$

Thus, the poles are given by

$$s^{2n} = -i^{2n} = e^{i\pi}e^{i\pi n} \ .$$

That is,

$$s = e^{i\pi(n+1)+2k\pi} \ , \quad k = 0, 1, \ldots 2n-1.$$

For example, if we select $n = 3$, we have the poles

$$\begin{aligned}
s_1 &= e^{i\frac{2\pi}{3}} = -\frac{1}{2} + i\frac{\sqrt{3}}{2} \ , \\
s_2 &= e^{i\pi} = -1 \ , \\
s_3 &= e^{i\frac{4\pi}{3}} = -\frac{1}{2} - i\frac{\sqrt{3}}{2} \ , \\
s_4 &= e^{i\frac{5\pi}{3}} = \frac{1}{2} - i\frac{\sqrt{3}}{2} \ , \\
s_5 &= e^{i2\pi} = 1 \ , \\
s_6 &= e^{i\frac{7\pi}{3}} = \frac{1}{2} + i\frac{\sqrt{3}}{2} \ .
\end{aligned}$$

There is symmetry with respect to the imaginary axis in the poles. To obtain a stable transfer function we associate to this transfer function the poles with real part negative. That is,

$$H_3(s) = \frac{1}{(s-s_1)(s-s_2)(s-s_3)} = \frac{1}{(s+1)(1+s+s^2)} \ .$$

In the following table we present the polynomial constituting the denominator of the transfer function of Butterworth filters for different orders n.

Table 1.- Butterworth polynomials.

n	$B_n(s)$
1	$(s+1)$
2	$(s^2+1.414s+1)$
3	$(s+1)\,(s^2+s1)$
4	$(s^2+0.765s+1)\,(s^2+1.848s+1)$
5	$(s+1)\,(s^2+0.618s+1)\,(s^2+1.618s+1)$
6	$(s^2+0.518s+1)\,(s^2+1.414s+1)\,(s^2+1.932s+1)$
7	$(s+1)\,(s^2+0.445s+1)\,(s^2+1.247s+1)\,(s^2+1.802s+1)$
8	$(s^2+0.390s+1)\,(s^2+1.111s+1)\,(s^2+1.663s+1)\,(s^2+1.962s+1)$
9	$(s+1)\,(s^2+0.347s+1)\,(s^2+s+1)\,(s^2+1.532s+1)\,(s^2+1.879s+1)$

5.2.1 Chebyshev filters

A generalization of Butterworth frequency response can be such that

$$|T_n(w)|^2 = \frac{1}{1+(F_n(w))^2} \ ,$$

where $F_n(w)$ is a given function.

The Chebyshev magnitude response is defined as

$$|T_n(w)|^2 = \frac{1}{1+\varepsilon^2 C_n^2(w)} \ , \qquad (5.2)$$

where $C_n(w)$ are the Chebyshev polynomials, which can be defined as

$$C_n(w) = \cos\left(n \arccos\left(w\right)\right) \ , \quad |w| \leq 1 \ , \qquad (5.3)$$
$$C_n(w) = \cosh\left(n \operatorname{argcosh}\left(w\right)\right) \ , \quad |w| > 1 \ . \qquad (5.4)$$

These polynomials satisfy the recurrence relation

$$C_{n+1}(w) = 2wC_n(w) - C_{n-1}(w), \quad C_0(w)=1, \quad C_1(w) = 1 \ . \qquad (5.5)$$

This relation allows to obtain the different polynomials, showed in the following table

Table 2.- Chebyshev polynomials.

n	$C_n(w)$
0	1
1	w
2	$2w^2-1$
3	$4w^3-3w$
4	$8w^4-8w^2+1$
5	$16w^5-20w^3+5w$
6	$32w^6-48w^4+18w^2-1$
7	$64w^7-112w^5+56w^3-7w$
8	$128w^8-256w^6+160w^4-32w^2+1$
9	$256w^9-576w^7+432w^5-129w^3+9w$

Choosing, for example, $\varepsilon = 0.2$, in figure 2, we compare the magnitude of the frequency response of Butterworth and Chebyshev filters for $n=4$.

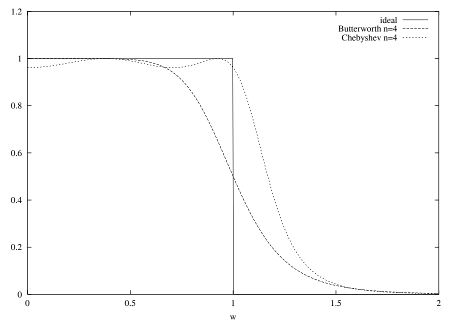


Figure 2.- Chebyshev and Butterworth responses.

To obtain the transfer function associated to the Chebyshev filter we calculate the poles of the magnitude of the response, taking into account the equation

$$1+\varepsilon^2 C_n^2(w) = 0 \ ,$$

that is,

$$1+\varepsilon^2 C_n^2\left(\frac{s}{i}\right) = 0 \ .$$

Assuming that $\mid s \mid \leq 1$, and using the definition (5.3), we have

$$\cos\left(n \arccos\left(\frac{s}{i}\right)\right) = \pm\frac{i}{\varepsilon} \ .$$

We call

$$w = \arccos\left(\frac{s}{i}\right) = u + iv \ ,$$

thus,

$$\cos(nw) = \cos(nu)\cosh(nv) - i\sin(nu)\sinh(v) \ ,$$

and we obtain the equations

$$\cos(nu)\cosh(nv) = 0 \ ,$$
$$\sin(nu)\sinh(nv) = \pm\frac{1}{\varepsilon} \ .$$

The solutions are

$$u = \frac{\pi}{2n}(2k+1) \ ,$$
$$v = \pm(-1)^k \frac{1}{n} \operatorname{argsinh}\left(\frac{1}{\varepsilon}\right) \ ,$$

and the poles

$$s = i\cos\left(\frac{\pi}{2n}(2k+1) \pm i(-1)^k \frac{1}{n}\operatorname{argsinh}\left(\frac{1}{\varepsilon}\right)\right) \ . \qquad (5.6)$$

To obtain a stable transfer function for the Chebyshev filter only the poles with negative real part are selected.

5.3 Filters from transfer functions

The filters exposed above are given by generic transfer functions. To obtain low-pass filters, high-pass filters, band-pass filters and band-stop filters from this kind of transfer functions we make use of the following practical substitutions

Low-pass	$s \to \frac{s}{w_c}$
High-pass	$s \to \frac{w_c}{s}$
Band-pass	$s \to \frac{w_0}{B_w}\left(\frac{s}{w_0} + \frac{w_0}{s}\right)$
Band-stop	$s \to \frac{B_w}{w_0}\left(\frac{s}{w_0} + \frac{w_0}{s}\right)^{-1}$

where w_c is the cut-off frequency, $w_0 = \sqrt{w_{c2}w_{c1}}$, $B_w = w_{c2} - w_{c1}$.

5.4 Discretización de sistemas continuos

Seguidamente pasaremos a estudiar, mediante un ejemplo concreto, tres posiblidades de discretizar un sitema dinámico asociado a una ecuación diferencial lineal de segundo orden con coeficientes constantes. Estudiaremos la solución de las ecuaciones en diferencias mediante la Transformada $\mathcal{Z}$ y compararemos estas soluciones con la solución del sistema continuo.

Partimos del problema de valores iniciales

$$\frac{d^2y}{dt^2} + \frac{dy}{dt} + 1.25y = 1 \ \ ; \quad y(0) = y'(0) = 0 \quad . \qquad (5.7)$$

Para obtener su solución, tomamos la Transformada de Laplace y hacemos uso de las condiciones iniciales, con lo que se obtiene

$$s^2 L[y] + sL[y] + 1.25L[y] = L[1] \quad ,$$

que da lugar a la relación

$$L[y] = \frac{1}{s^2 + s + 1.25} L[1] \quad , \qquad (5.8)$$

y, por tanto, la solución será el producto de convolución

$$y = L^{-1}\left[\frac{1}{s^2+s+1.25}\right] * 1 \quad .$$

Sin más que descomponer en fracciones simples, obtenemos

$$L^{-1}\left[\frac{1}{s^2+s+1.25}\right] = \frac{1}{2i}(e^{\alpha_1 t} - e^{\alpha_2 t}) \quad ,$$

donde $\alpha_{1,2}$ son las soluciones de la ecuación

$$s^2 + s + 1.25 = 0 \quad ,$$

que son

$$\alpha_{1,2} = -0.5 \pm i \quad , \qquad (5.9)$$

así,

$$y = \int_0^t e^{-0.5x}\operatorname{sen}(x)\,dx = \frac{1}{1.25}(1 - (\cos(t) + 0.5\operatorname{sen}(t))e^{-0.5t}) \quad . \qquad (5.10)$$

El comportamiento asintótico de la solución es

$$\lim_{t\to+\infty} = \frac{1}{1.25} = 0.8 \quad .$$

Pasaremos a ver ahora, posibles sistemas discretos que se pueden obtener a partir de (5.7). Para pasar de un sistema continuo a uno discreto hay que aproximar las derivadas de la función. La utilización de una u otra aproximación nos caracterizará las distintas discretizaciones de un sistema continuo.

5.4.1 Discretización 'backward'

Si en el sistema (5.7) utilizamos la aproximación

$$\begin{aligned} \frac{dy}{dt} &\approx \frac{1}{T}(y(kT) - y((k-1)T)) \quad , \\ \frac{d^2y}{dt^2} &\approx \frac{1}{T^2}(y(kT) - 2y((k-1)T) + y((k-2)T)) \quad , \end{aligned} \tag{5.11}$$

obtenemos el sistema discreto

$$\begin{aligned} &\frac{1}{T^2}(y(kT) - 2y((k-1)T) + y((k-2)T)) + \\ &+\frac{1}{T}(y(kT) - y((k-1)T)) + 1.25y(kT) = 1 \ , \\ &y(0) = 0 \ . \end{aligned} \tag{5.12}$$

Para obtener la solución de este sistema, se toma la Transformada $\mathcal{Z}$ de la ecuación

$$\frac{1}{T^2}(1 - 2z^{-1} + z^{-2})\mathcal{Z}[y] + \frac{1}{T}(1 - z^{-1})\mathcal{Z}[y] + 1.25\mathcal{Z}[y] = \mathcal{Z}[1] \quad ,$$

con lo que

$$\mathcal{Z}[y] = \frac{1}{(\frac{1}{T^2}(1 - z^{-1})^2 + \frac{1}{T}(1 - z^{-1}) + 1.25)}\,\mathcal{Z}[1] \quad . \tag{5.13}$$

Si comparamos esta expresión con la expresión (5.8), correspondiente al caso continuo, observamos que basta hacer la sustitución

$$s \longleftrightarrow \frac{1}{T}(1 - z^{-1}) \tag{5.14}$$

y cambiar la transformada de Laplace por la transformada $\mathcal{Z}$, para pasar de una expresión a la otra. Esta relación es general para los sistemas lineales de coeficientes constantes cuando se realiza una discretización del tipo (5.11).

La solución de la ecuación (5.12) viene dada por

$$y = \mathcal{Z}^{-1}\left[\frac{1}{(\frac{1}{T^2}(1 - z^{-1})^2 + \frac{1}{T}(1 - z^{-1}) + 1.25)(1 - z^{-1})}\right] \quad .$$

Utilizando la fórmula de inversión

$$\begin{aligned} y(kT) &= \operatorname{Res}\left(\frac{T^2 z^{k+2}}{((z-1)^2 + Tz(z-1) + 1.25T^2z^2)(z-1)},\ z = \beta_1\right) \\ &+ \operatorname{Res}\left(\frac{T^2 z^{k+2}}{((z-1)^2 + Tz(z-1) + 1.25T^2z^2)(z-1)},\ z = \beta_2\right) \\ &+ \operatorname{Res}\left(\frac{T^2 z^{k+2}}{((z-1)^2 + Tz(z-1) + 1.25T^2z^2)(z-1)},\ z = 1\right) \\ k &= 1, 2, \ldots \quad , \end{aligned} \tag{5.15}$$

donde $\beta_{1,2}$ son las raíces de la ecuación

$$(1.25T^2 + T + 1)z^2 - (T + 2)z + 1 = 0 \ ,$$

o sea,

$$\beta_{1,2} = \frac{T + 2 \pm i2T}{2(1.25T^2 + T + 1)} \quad .$$

Como es conocido, para que la solución de un sistema continuo sea asitóticamente estable, o sea, exista el límite de la solución cuando la variable tiende a $+\infty$, se ha de satisfacer que la parte real de las raíces del polinomio característico sea menor que cero. La relación existente entre las raíces (5.4.1) y las correspondientes del caso continuo, (5.9), viene dada por la expresión (5.14). Así, $\operatorname{Re}(s) < 0$ implica que $\operatorname{Re}(1 - z^{-1}) < 0$ y tomando $z = x + iy$, tenemos

$$\operatorname{Re}(1 - z^{-1}) = \operatorname{Re}\left(\frac{x + iy - 1}{x + iy}\right) = \frac{x^2 + y^2 - x}{x^2 + y^2} < 0 \rightarrow (x - \frac{1}{2})^2 + y^2 < (\frac{1}{2})^2$$

y, por tanto, esta transformación lleva el semiplano real negativo a la circunferencia de centro $(1/2, 0)$ y radio $1/2$.

Por tanto, como las raíces del caso continuo tienen su parte real negativa, podemos asegurar que $\mid \beta_{1,2} \mid < 1$, independientemente del valor de T.

Calculando los residuos obtenemos,

$$\begin{aligned} y(kT) &= 0.8 + \frac{T^2\beta_1^{k+2}}{(1.25T^2 + T + 1)(\beta_1 - \beta_2)(\beta_1 - 1)} + \\ &+ \frac{T^2\beta_2^{k+2}}{(1.25T^2 + T + 1)(\beta_2 - \beta_1)(\beta_2 - 1)} \quad , \end{aligned}$$

y si tomamos el límite cuando k tiende a $+\infty$, obtenemos como valor asintótico de la solución

$$\lim_{k\to+\infty} y(kT) = 0.8 \quad ,$$

independientemente del valor de T. Con esta discretización, recuperamos pues, el comportamiento asintótico de la solución del sistema continuo.

A continuación, presentamos en un gráfica la solución del sistema discreto para los valores del periodo $T = 0.5$ y $T = 1$, observándose que en ambos casos se alcanza el valor asintótico.

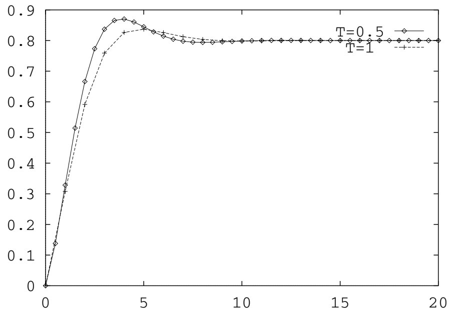


Fig. 5.1 Comparación de las soluciones 'backward' para $T = 0.5$ y $T = 1$.

5.4.2 Discretización 'forward'

Si en el sistema (5.7) utilizamos la discretización

$$\begin{aligned} \frac{dy}{dt} &\approx \frac{1}{T}(y((k+1)T) - y(kT)) \quad , \\ \frac{d^2y}{dt^2} &\approx \frac{1}{T^2}(y((k+2)T) - 2y((k+1)T) + y(kT)) \quad , \end{aligned} \tag{5.16}$$

obtenemos el sistema discreto

$$\begin{aligned} &\frac{1}{T^2}(y((k+2)T) - 2y((k+1)T) + y(kT)) + \\ &\frac{1}{T}(y((k+1)T) - y(kT)) + 1.25y(kT) = 1 \ , \\ &y(0) = y(1) = 0 \ . \end{aligned} \tag{5.17}$$

Para obtener la solución, tomamos la Transformada $\mathcal{Z}$ de la ecuación

$$\frac{1}{T^2}(z^2 - 2z + 1)\mathcal{Z}[y] + \frac{1}{T}(z-1)\mathcal{Z}[y] + 1.25\mathcal{Z}[y] = \mathcal{Z}[1]$$

con lo que

$$\mathcal{Z}[y] = \frac{1}{(\frac{1}{T^2}(z-1)^2 + \frac{1}{T}(z-1) + 1.25)}\mathcal{Z}[1] \quad . \tag{5.18}$$

Si comparamos esta expresión con la expresión (5.8), correspondiente al caso continuo, observamos que al utilizar esta discretización, la relación entre el 'plano s' y el 'plano z' viene dada por

$$s \leftrightarrow \frac{1}{T}(z-1) \quad . \tag{5.19}$$

La solución de la ecuación (5.17) viene dada por

$$y = \mathcal{Z}^{-1}\left[\frac{z}{(\frac{1}{T^2}(z-1)^2 + \frac{1}{T}(z-1) + 1.25)(z-1)}\right] \quad ,$$

y utilizando la fórmula de inversión

$$\begin{aligned} y(kT) &= \text{Res}\left(\frac{T^2z^k}{((z-1)^2 + T(z-1) + 1.25T^2)(z-1)},\ z = \gamma_1\right) + \\ &+ \text{Res}\left(\frac{T^2z^k}{((z-1)^2 + T(z-1) + 1.25T^2)(z-1)},\ z = \gamma_2\right) + \\ &+ \text{Res}\left(\frac{T^2z^k}{((z-1)^2 + T(z-1) + 1.25T^2)(z-1)},\ z = 1\right) \ . \end{aligned}$$

$k \geq 1$,

donde $\gamma_{1,2}$ son las raíces de la ecuación

$$z^2 + (T-2)z + (1 - T + 1.25T^2) = 0 \ ,$$

o sea,

$$\gamma_{1,2} = \frac{2-T}{2} \pm iT \quad .$$

Calculando los residuos, tenemos

$$\begin{aligned} y(kT) &= 0.8 + \frac{T^2\gamma_1^k}{(\gamma_1 - \gamma_2)(\gamma_1 - 1)} + \frac{T^2\gamma_2^k}{(\gamma_2 - \gamma_1)(\gamma_2 - 1)} = \\ &= 0.8 + \text{Im}\left(\frac{(1 - \frac{T}{2} + iT)^k}{-\frac{1}{2} + iT}\right) \quad . \end{aligned} \tag{5.20}$$

A diferencia de lo que ocurría anteriormente, si tomamos el límite cuando k tiende a $+\infty$, la solución convergerá, o no, dependiendo del valor de T, como se pone de manifiesto en la siguiente gráfica, donde se representa la solución de la ecuación en diferencias para los valores del periodo $T = 0.5$ (estable) y $T = 1$ (inestable).

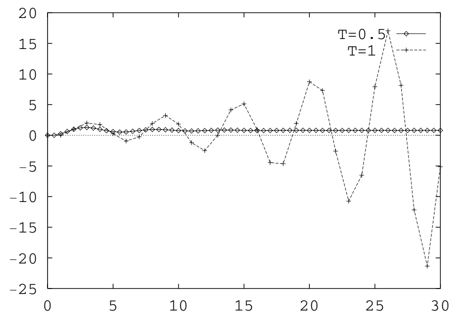


Fig.- 5.2 Comparación de las soluciones 'forward' para $T = 0.5$ y $T = 1$.

Esta dependencia en el valor del perodo de muestreo hace que la discretización 'forward' no se utilice en la práctica a la hora de construir sistemas discretos a partir de un sistema continuo.

5.4.3 Discretización por el Método de los Trapecios

Otro método para obtener una discretización del sistema continuo, consiste en lo siguiente. Se parte de la ecuación diferencial de segundo orden

$$\frac{d^2y}{dt^2} + \frac{dy}{dt} + 1.25y = 1 \quad ,$$

y se reduce de orden introduciendo una nueva variable $u = y'$, obteniendo

$$\frac{du}{dt} = -u - 1.25y + 1 \quad ,$$

$$\frac{dy}{dt} = u \quad .$$

Integrando estas ecuaciones entre $(k-1)T$ y T, obtenemos

$$\int_{(k-1)T}^{kT} \frac{du}{dt}\,dt = \int_{(k-1)T}^{kT} (-u - 1.25y + 1)\,dt \quad ,$$

$$\int_{(k-1)T}^{kT} \frac{dy}{dt}\,dt = \int_{(k-1)T}^{kT} u\,dt \quad ,$$

y aproximando el valor de las integrales por la Regla de los Trapecios

$$\begin{aligned}
\int_{(k-1)T}^{kT} (-u - 1.25y + 1)\,dt &= -\frac{T}{2}(u(kT) + u((k-1)T)) - \\
&- 1.25\frac{T}{2}(y(kT) + y((k-1)T)) + \\
&+ \frac{T}{2}(1(kT) + 1((k-1)T)) \ , \\
\int_{(k-1)T}^{kT} u\,dt &= \frac{T}{2}(u(kT) + u((k-1)T)) \quad ,
\end{aligned}$$

obtenemos el sistema de ecuaciones en diferencias

$$\begin{aligned}
u(kT) - u((k-1)T) &= -\frac{T}{2}(u(kT) + u((k-1)T)) - \\
&- 1.25\frac{T}{2}(y(kT) + y((k-1)T)) + \\
&+ \frac{T}{2}(1(kT) + 1((k-1)T)) \quad , \\
y(kT) - y((k-1)T) &= \frac{T}{2}(u(kT) + u((k-1)T)) \quad ,
\end{aligned} \tag{5.21}$$

con las condiciones iniciales $u(0) = y(0) = 0$.

Tomando la Transformada $\mathcal{Z}$ de la segunda igualdad de (5.21) y sustituyendo en la primera, obtenemos

$$\begin{aligned}
\mathcal{Z}[u] &= \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)\mathcal{Z}[y] \quad , \\
\frac{2}{T}\frac{(1 - z^{-1})^2}{(1 + z^{-1})}\mathcal{Z}[y] &= -(1 - z^{-1})\mathcal{Z}[y] - 1.25\frac{T}{2}(1 - z^{-1})\mathcal{Z}[y] + \\
&+ \frac{T}{2}(1 - z^{-1})\mathcal{Z}[1] \ ,
\end{aligned}$$

y, por tanto,

$$\mathcal{Z}[y] = \frac{1}{\frac{4}{T^2}(\frac{z-1}{z+1})^2 + \frac{2}{T}(\frac{z-1}{z+1}) + 1.25}\,\mathcal{Z}[1] \quad .$$

Comparando esta expresión con la expresión (5.8), correspondiente al caso continuo, observamos que al utilizar la discretización por el Método de los Trapecios la relación entre el 'plano s' y el 'plano z' viene dada por

$$s \leftrightarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) \quad . \tag{5.22}$$

Si $\mathrm{Re}(s) < 0$ se ha de cumplir que $\mathrm{Re}\left(\frac{z-1}{z+1}\right) < 0$, y si tomamos $z = x + iy$,

$$\mathrm{Re}\left(\frac{z-1}{z+1}\right) = \frac{x^2 - 1 + y^2}{(x+1)^2 + y^2} < 0 \to x^2 + y^2 < 1 \quad .$$

así pues, esta transformación nos lleva el semiplano real negativo a la circunferencia de centro el origen y radio uno.

La solución para y de la ecuación (5.21) es de la forma

$$y = \mathcal{Z}^{-1}\left[\frac{z}{(\frac{4}{T^2}(\frac{z-1}{z+1})^2 + \frac{2}{T}(\frac{z-1}{z+1}) + 1.25)(z-1)}\right] \quad .$$

Utilizando la fórmula de inversión, obtenemos

$$\begin{aligned} y(0) &= 0 \\ y(kT) &= 0.8 + \frac{T^2\delta_1^k(\delta_1+1)^2}{(\delta_1-\delta_2)(\delta_1-1)(4+2T+1.25T^2)} + \\ &+ \frac{T^2\delta_2^k(\delta_2+1)^2}{(\delta_2-\delta_1)(\delta_2-1)(4+2T+1.25T^2)} \\ & \quad k \geq , \end{aligned}$$

donde $\delta_{1,2}$ son las races de la ecuación

$$(4 + 2T + 1.25T^2)z^2 + (-8 + 2.5T^2)z + 4 - 2T + 1.25T^2 = 0$$

cuya relación con las del caso continuo viene dadas por (5.22). Podemos afirmar pues que $\mid \delta_{1,2} \mid < 1$ y, por tanto, al tomar el límite cuando k tiende a $+\infty$ obtenemos el valor asintótico

$$\lim_{k\to+\infty} y(kT) = 0.8 \quad ,$$

lo que se pone de manifiesto en la siguiente gráfica, donde se compara la solución para esta discretización con los valores del periodo de muestreo $T = 0.5$ y $T = 1$.

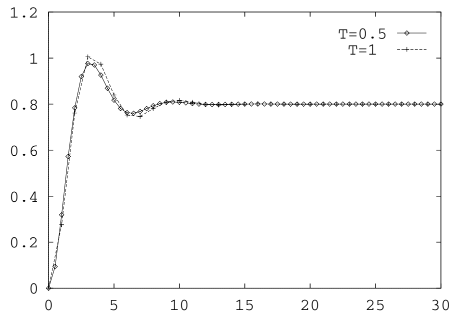


Fig.- 5.3 Comparación de las soluciones 'trapecios' para $T = 0.5$ y $T = 1$.

A continuación, presentamos la comparación de las soluciones del sistema continuo con la solución obtenida para cada uno de los sistemas discretos, con un valor del periodo $T = 1$.

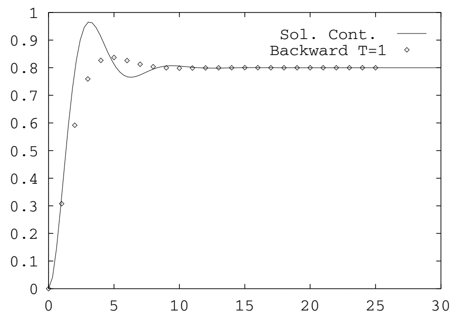


Fig.- 5.4 Solución del sistema continuo y solución del sistema 'backward'

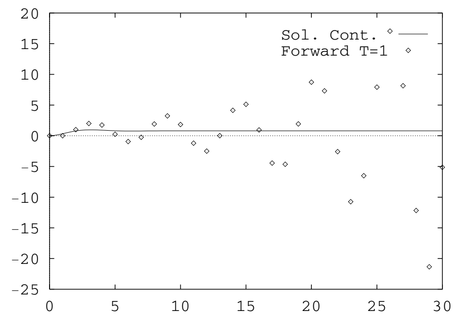


Fig.- 5.5 Solución del sistema continuo y solución del sistema 'forward'

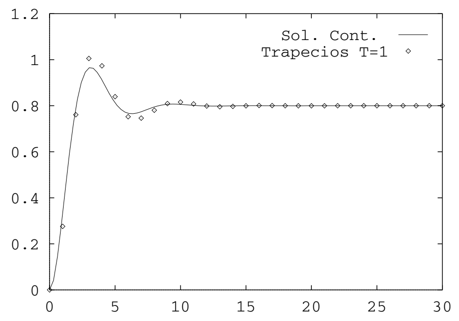


Fig.- 5.6 Solución del sistema continuo y solución del sistema 'trapecios'

Podemos observar que el sistema discreto que mejor se comporta es el que se obtiene mediante la discretización por el Método de los Trapecios. Esta es una caracterstica que se da en general para todos los sistemas.

5.4.4 Discretization of the impluse response

Let us suppose now that we start from the transfer function of a system of the form

$$H(s) = \sum_{k=1}^{N} \frac{A_k}{s - s_k} \ . \tag{5.23}$$

We assume that the multiplicity of the poles of the transfer function is 1, but the discretization method presented here can be generalized for poles with multiplicity larger than 1.

The impulse response of the system can be obtained computing the inverse Laplace transform of (5.23)

$$h(t) = \sum_{k=1}^{N} A_k e^{s_k t} \ . \tag{5.24}$$

On the other hand,

$$H(s) = \int_0^\infty h(t) e^{-st}\, dt \ . \tag{5.25}$$

If we assume that the only information we have of the system are the samples $h(nT)$, $n = 0, 1, \ldots$, and T is the sampling time, we can approximate (5.25) by

$$H(s) \approx \sum_{n=0}^{\infty} h(nT) e^{-snT} T \ ,$$

and using the variable $z = e^{sT}$, we have

$$H(z) = \sum_{n=0}^{\infty} T h(nT) z^{-n} \ .$$

Making use of (5.24), we have the transfer function of the discrete system

$$\begin{aligned} H(z) &= \sum_{n=0}^{\infty} T \left(\sum_{k=1}^{N} A_k e^{s_k nT} \right) z^{-n} = \sum_{k=1}^{N} T A_k \sum_{n=0}^{\infty} \left(e^{s_k T} z^{-1} \right)^n = \\ &= \sum_{k=1}^{N} \frac{A_k T}{1 - e^{s_k T} z^{-1}} \ . \end{aligned}$$

5.5 Ejercicio

Compute the low-pass filters of orders 4, 5 and 6 derived from the analog filters of Butterworth and Chebyshev, using the backward discretization, the trapezoidal rule, and the discretization of the impulse response. Compare their performance to eliminate the noise of a signal of the form

$$x(t) = \sin(0.2t) + \varepsilon \ ,$$

where ε is a Gaussian noise process of mean 0 and variance 0.2.