# Chapter 1

# Introduction

## 1.1. Brief introduction to MATLAB

### 1.1.1. *MATLAB software presentation*

MATLAB (MATrix LABoratory) is an interactive software, developed by Math Works Inc. and intended especially for digital signal processing. It is particularly effective when the data format is vector or matrix.

MATLAB integrates digital calculus, data visualization and open environment programming. MATLAB exists under both Windows and UNIX. Many demonstrations are available using the command `demo`.

This digital simulation software enables a fast and simple visualization of the obtained results.

MATLAB was primarily written in FORTRAN and C. However, MATLAB knows to interpret commands, while a compilation of the source code is required by FORTRAN and C.

MATLAB is especially designed for digital signal processing and for complex digital system modeling and simulation. It is also suitable for processing data series, images or multidimensional data fields.

MATLAB software general structure is provided in Figure 1.1.

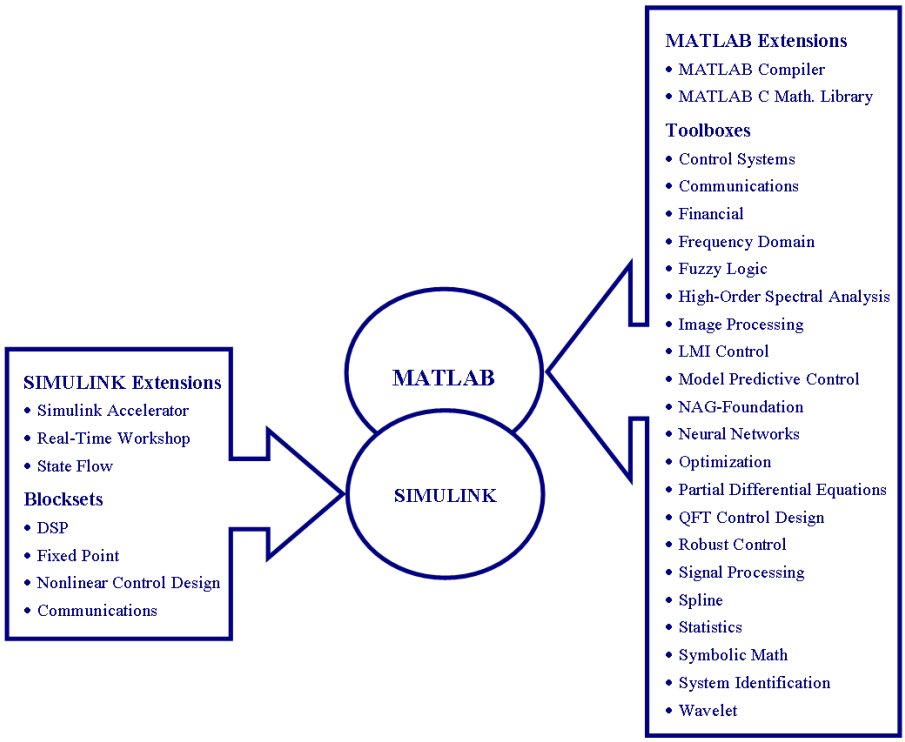


**Figure 1.1.** *MATLAB software general structure*

The toolboxes extend the basic MATLAB functions and perform specific tasks corresponding to different digital processing fields, such as image processing, optimization, statistics, system control and identification, neural networks, fuzzy systems, etc.

SIMULINK is an interactive software designed for modeling and simulating continuous-time or discrete-time dynamical systems or hybrid structures containing both analog and digital systems. It makes use of a mathematical equation set and provides a large variety of predefined or user-defined functional blocks.

MATLAB has been developed for several years, especially as a consequence of its use in the academic environment as an excellent education tool in mathematics, engineering and science. In addition, MATLAB has already proven its utility for scientific research and technological development.

In order to run MATLAB, type the command `matlab` with UNIX shell (if a MATLAB license under UNIX is available) or double click on the MATLAB icon if the operating system is Windows. To exit MATLAB, type `exit` or `quit`. If MATLAB is running under UNIX, you may have access to all UNIX commands using just before the symbol! (example: `!ls -l`).

### 1.1.2. *Important MATLAB commands and functions*

| | |
|---|---|
| `who` | lists the variables in the current workspace |
| `whos` | the same as previous, but lists more information about each variable |
| `what` | lists MATLAB-specific files in directory |
| `size` | provides the size of a data array |
| `length` | provides the size of a data vector |
| `help` | displays help text in Command Window |
| `exit, quit` | exits from MATLAB |

**Table 1.1.** *General commands*

| | |
|---|---|
| `dir, chdir, delete, load, save, type` | similar to the corresponding DOS commands |
| `pack` | consolidates workspace memory |

**Table 1.2.** *Commands related to the workspace*

| | |
|---|---|
| `+, -, *, /, ^` | usual arithmetical operators |
| `.` | followed by an arithmetical operator for applying it to each array element |
| `'` | Hermitian operator |
| `.'` | transpose operator |

**Table 1.3.** *Arithmetical operators*

| | |
|---|---|
| `<, <=, >, >=` | usual relational operators |
| `==` | equality operator |
| `~=` | inequality operator |
| `&` | element-wise logical AND |
| `\|` | element-wise logical OR |
| `~` | logical complement (NOT) |

**Table 1.4.** *Relational and logical operators*

| | |
|---|---|
| `=` | variable assignment operator |
| `,` | used to separate the arguments of a function or the elements of a data array |
| `[]` | used to build data arrays |
| `()` | used in arithmetical expressions |
| `:` | used for indexing variables |
| `;` | used at the end of a statement to cancel displaying any output |
| `...` | used to continue a command on the next line |
| `%` | used to enter a comment |

**Table 1.5.** *Special characters*

| | |
|---|---|
| `ans` | default name of a variable or a result |
| `eps` | spacing of floating point numbers |
| `pi` | value of $\pi = 3.14159...$ |
| `i,j` | value of $\sqrt{-1}$ |
| `Inf` | IEEE arithmetic representation for positive infinity (1/0) |
| `NaN` | IEEE arithmetic representation for Not-a-Number (0/0) |
| `nargin` | returns the number of function input arguments |
| `nargout` | returns the number of function output arguments |

**Table 1.6.** *Special variables and constants*

| | |
|---|---|
| `abs` | absolute value function |
| `sqrt` | square root function |
| `real` | real part of a complex variable |
| `imag` | imaginary part of a complex variable |
| `angle` | returns the phase angles, in radians, of a complex variable |
| `conj` | complex conjugate operator |
| `sign` | signum function |
| `rem` | returns the remainder after division |
| `exp` | exponential function |
| `log` | natural logarithm function |
| `log10` | base 10 logarithm function |

**Table 1.7.** *Elementary mathematical functions*

| | |
|---|---|
| `sin, cos, tan, cot, sec` | usual trigonometric functions |
| `asin, acos, atan, acot, asec` | inverse trigonometric functions |
| `sinh, cosh, tanh, coth, sech` | hyperbolic functions |
| `asinh, acosh, atanh, acoth, asech` | inverse hyperbolic functions |

**Table 1.8.** *Trigonometric functions*

| | |
|---|---|
| `max` | largest component |
| `min` | smallest component |
| `mean` | average or mean value |
| `std` | standard deviation |
| `sum` | sum of elements |
| `cumsum` | cumulative sum of elements |
| `prod` | product of elements |
| `cumprod` | cumulative product of elements |

**Table 1.9.** *Data analysis functions*

| | |
|---|---|
| conv | convolution and polynomial multiplication |
| deconv | deconvolution and polynomial division |
| roots | finds polynomial roots |
| poly | converts roots to polynomial |
| polyval | evaluates polynomial |
| residue | partial-fraction expansion (residues) |

**Table 1.10.** *Polynomial related functions*

| | |
|---|---|
| zeros | enables generation of zero arrays |
| ones | enables generation of ones arrays |
| rand | enables generation of uniformly distributed random numbers |
| randn | enables generation of normally distributed random numbers |
| linspace | enables generation of linearly spaced vectors |
| logspace | enables generation of logarithmically spaced vector |
| det | calculates the determinant of a square matrix |
| norm | calculates matrix or vector norm |
| inv | calculates matrix inverse |
| eig | calculates matrix eigenvalues and eigenvectors |

**Table 1.11.** *Vector or matrix related functions*

| | |
|---|---|
| input | gives the user the prompt and then waits for input from the keyboard |
| ginput | gets an unlimited or a predefined number of points from the current axes and returns their coordinates |

**Table 1.12.** *Input functions*

| | |
|---|---|
| `plot` | plot vectors or matrices |
| `subplot` | create axes in tiled positions |
| `bar` | draws a bar graph |
| `hist` | draws a histogram graph |
| `polar` | makes a plot using polar coordinates |
| `stairs` | draws a stairstep graph |
| `stem` | plots the data sequence as stems |
| `semilogx`, `semilogy` | semi-log scale plot: a logarithmic (base 10) scale is used for the x-axis or y-axis |
| `loglog` | log-log scale plot: a logarithmic (base 10) scale is used for both the x-axis and y-axis |
| `xlabel`, `ylabel` | adds text beside the x-axis or y-axis |
| `title` | adds text at the top of the current axes |
| `grid` | adds grid lines to the current axes |
| `figure` | creates a new figure window |
| `clf` | clears current figure |
| `close all` | closes all the open figure windows |
| `hold on/off` | holds/discards the current plot and all axis properties |
| `axis` | controls axis scaling and appearance |
| `legend` | puts a legend on the current plot using the specified strings as labels |
| `gtext` | allows placing text with mouse |
| `image` | displays a matrix as an image |

**Table 1.13.** *1D and 2D graphical commands*

| | |
|---|---|
| `plot3` | plot lines and points in 3-D space |
| `mesh/surf` | plots a 3-D mesh/colored surface |
| `contour` | plots a contour plot of a matrix treating its values as heights above a plane |

**Table 1.14.** *3D graphical commands*

| | |
|---|---|
| `if` | conditionally executes statements |
| `else, elseif` | used with `if` command |
| `end` | terminates scope of `for`, `while`, `switch`, `try` and `if` statements |
| `for` | repeats statements a specific number of times |
| `while` | repeats statements an indefinite number of times |
| `switch` | switches among several cases based on expression |
| `break` | terminates execution of `while` or `for` loop |
| `return` | causes a return to the invoking function or to the keyboard |
| `pause` | pauses and waits for the user response |

**Table 1.15.** *Control commands*

### 1.1.3. *Operating modes and programming with MATLAB*

The "online command" default operating mode is available after MATLAB gets started. It displays the prompt >> and then waits for an input command. Running a command usually results in creating one or several variables in the current workspace, displaying a message or plotting a graph. For instance, the following command:

```
v = 0:10
```

creates the variable v and displays its elements on screen. A semicolon has to be added at the end of the statement if it is not necessary to display the result.

The previously typed commands can be recalled with the key ↑, while a statement can be modified using the keys ← and →. You may also analyze the effects on the command lines of the following keys: ↓, *home, end, esc, del, backspace* and of the following key combinations: ctrl + →, ctrl + ←, ctrl + *k*.

Besides the "online command" operating mode, MATLAB can also create script files and function files. Both of these are saved with the extension .m, but the function files accept input arguments and return output arguments and operate on variables within their own workspace.

In order to create a script file you have to select the menu *File/New/M-file*, while to edit an existing file you have to first select *File/Open M-file* etc., and then choose the appropriate file. After these commands, an edition session will be open using the

chosen editor from *Edit/View/Edit Preference*. The edited file can be saved with the menu *File/Save As* etc., followed by the file name (with the extension .m).

In MATLAB, many functions are predefined and saved as m-files. Some of them are intrinsic, the others being provided by external libraries (*toolbox*): they cover specific domains such as mathematics, data analysis, signal processing, image processing, statistics, etc.

A function may use none, one or several input arguments and return none, one or several output values. These different cases for a MATLAB function are called:

– one output value and no input argument:

variable_name = function_name

– no output value and one input argument:

function_name (argument_name)

– several output values and several input arguments:

[var_1, var_2, ...,var_n] = function_name (arg_1, arg_2, ...., arg_m)

For the last case, the first line of the file `function_name.m` has the following form:

– function [var_1, var_2, ...,var_n] = function_name(arg_1, arg_2, ...., arg_m)

Usually, the input arguments are not modified, even if their values change during the function execution. In fact, all the variables are local by default. Nevertheless, this rule can be changed using the command: `global variable_name`.

In a MATLAB file, the comment lines have to begin with the symbol %.

The on-line help can be obtained using: `help` <*function_name*>. The first lines of the file <*function_name*>.*m* beginning with % are then displayed. It is also possible to search all the files containing a given keyword in their help using the command: `lookfor` <*keyword*>.

NOTE.– The user-defined MATLAB files are recognized only in the current directory, unlike the original MATLAB functions (*toolbox,* etc.). In order to make available a user-defined file <*file_name.m*> outside the current directory you have to type the command:

```
path(path,'<file_acces_path>/file_name>')
```

(see `help path, help addpath`).

The data from the current workspace can be saved in a *.mat file using the command `save`. They can be reloaded using the command `load`. (Type `help save` and `help load` for more information).

Another possibility is to use the same procedure to manage the files as in the C language:

```
fid = fopen('x.dat','wb'); fwrite(fid,x,'double'); fclose(fd);
```

MATLAB is also able to manage other file formats, such as postscript.

### 1.1.4. *Example of work session with MATLAB*

*Format*

All the calculations are performed in MATLAB using the format double, but the display format can be controlled using the function `format` (type `help format`). Some examples are provided here after:

- `format short`: scaled fixed point format with 5 digits (default);
- `format long`: scaled fixed point format with 15 (7) digits for double (simple);
- `format short e`: floating point format with 5 digits;
- `format long e`: floating point format with 15 (7) digits for double (simple).

*Scalars, vectors, matrices*

MATLAB handles only one data type, because all the variables are considered as floating point complex matrices. It is not necessary to declare or to size these matrices before using them. In fact, when a variable is assigned a value, MATLAB replaces the previous value if this variable exists in the work space; otherwise the variable is created and sized properly.

A vector is a one row or a one column matrix, while a scalar is a 1×1 matrix. MATLAB is optimized for matrix calculations. You should try to use matrix operation as much as possible instead of loops in order to save execution time and memory space.

The effectiveness of an algorithm can be measured using the functions `flops` (*number of floating point operations*) and `etime` (*elapsed time*). Thus, the couple of commands `flops(0)` and `flops` inserted just before and after an algorithm code line returns the number of operations required. The function `clock` yields the present time, while `etime(t1,t2)` provides the time elapsed between t1 and t2.

EXAMPLE

```
t = clock;
%Algorithm;
time = etime(clock,t)
```

`etime` is not an accurate measure of the algorithm effectiveness because the execution speed depends on the CPU.

EXERCISE 1.1.

Type `a = 3` and then `a = 3;`

What is the signification of the symbol ";"?

There are some predefined variables. For instance `pi` = π, while *i* and *j* are defined as the square root of –1. Type `a = 1+2*i.`

Pay attention to the use of these keywords for defining new variables: any assignment replaces the predefined value by the new input (for instance the assignment `pi = 3` replaces the value π). Type `clear pi` to recover the initial value of this variable.

You should avoid assigning *i* and *j* other values in a MATLAB program which handles complex numbers.

EXERCISE 1.2.

Type `i = 2`, then `a = 1+2*i` and finally `clear i`.

`clear` command allows one or several variables to be removed.

*Elementary operations*

An operation involving 2 variables is possible only if the corresponding matrix sizes match.

EXERCISE 1.3.

Type `v = [1 2 3]` then `v = [1; 2; 3]` and `v(1)`.

As opposed to the case of C language, where the array index begins with 0, in MATLAB it begins with 1: see the effect of `v(0)`.

A vector filled with equally spaced values is defined in the following manner: initial_value:increment:final value (for example `v = 4:-0.1:3.2`).

A matrix can be defined as indicated below:

- `M = [1 2; 3 4];`
- `N(1,:) = [1 2]` and `N(2,:) = [3 4]`.

Type `M(:,1)`,`M(:,2)`,`N(:,1)` and `M(:,2)`.

The pointwise operators: `".*"`, `"./"` or `".^"` are useful for performing matrix operations.

EXERCISE 1.4.

Define the following matrix: `A = [exp(j) 1; 2 j]` and see `A'`, `A.'`, `A^2`, `A.^2`.

The relational operators: `<`, `<=`, `>`, `>=`, `~=` and `==` compare couples of elements belonging to equal size matrices and return a matrix filled with 1 (true) and 0 (false).

The logical operators such as: `&`, `|`, `~`, `any` or `all` consider all the non-zero elements as true and return a matrix filled with 0 and 1, according to the logical operation result.

MATLAB has no pointer structures, but it automatically allocates (when using =) and recovers (when using `clear`) memory space. For example, for solving `A*x=y`, MATLAB automatically creates a vector for `x`.

Notice the difference between matrix right division and matrix left division: `X=A\B` (equivalent to $A^{-1}*B$) is the solution to `A*X=B` while `X=A/B` (equivalent to $A*B^{-1}$) is the solution to `X*B=A`.

EXERCISE 1.5.

```
A = [1 2 1; 2 1 3; 4 0 5];
y = [3; 2; 1];
x = A\y
z = A/y
```

The matrices can be concatenated either line by line or column by column.

`N = [1 2]; P = [M; N];` then `Q = [M'; N'];`

The inverse submatrix extraction can be performed using brackets as indicated below:

Type `B=A(1:3,:)` and `C=A([1 3],:)`.

### 1.1.5. *MATLAB language*

MATLAB is a true programming language. However, it is an uncompiled language and thus is not particularly suitable for developing very complex applications. However, it is provided with all the necessary algorithmic structures for rigorous programming.

*The "for" loops*

```
for (expression)
     code lines;
end
```

*The "while" loops*

```
while (condition)
     code lines;
end
```

*The "if ... then" loops*

```
if (condition1)
     code lines;
else if (condition2)
     code lines;
else
     code lines;
end
```

## 1.2. Solved exercises

EXERCISE 1.6.

Define a 4×3 matrix zero everywhere excepting the first line that is filled with 1.

```
b = ones (1,3); m = zeros (4,3); m(1,:) = b

m =
     1     1     1
     0     0     0
     0     0     0
     0     0     0
```

EXERCISE 1.7.

Consider the couples of vectors $(x_1, y_1)$ and $(x_2, y_2)$. Define the vector $x$ so that:

$x(j) = 0$ if $y_1(j) < y_2(j)$;
$x(j) = x_1(j)$ if $y_1(j) = y_2(j)$;
$x(j) = x_2(j)$ if $y_1(j) > y_2(j)$.

```
function x = vectors(x1,y1,x2,y2)
x = x1.*[y1 == y2] + x2.*[y1 > y2];

vectors([0 1],[4 3],[-2 4],[2 0])

ans =
    -2     4
```

EXERCISE 1.8.

Generate and plot the signal: $y(t) = \sin(2\pi t)$ for $0 \le t \le 2$, with an increment of 0.01, then undersample it (using the function `decimate`) with the factors 2 and 16.

```
t = 0:0.01:2;
y = sin(2*pi*t);
subplot(311)
plot(t,y) ;
ylabel('sin(2.pi.t)');
title('Original signal');
t2 = decimate(t,2);
t16 = decimate(t2,8);
y2 = decimate(y,2);
y16 = decimate(y2,8);
subplot(312)
plot(t2,y2);
ylabel('sin(2.pi.t)')
title('Undersampled signal with a factor 2');
subplot(313);
plot(t16,y16);
ylabel('sin(2.pi.t)');
xlabel('Time t');
title('Undersampled signal with a factor 16');
```

You can save the figures in *eps* (Encapsulated PostScript) format, which is recognized by many software programs. The command `print -eps file_name` creates the file *file_name.eps*.

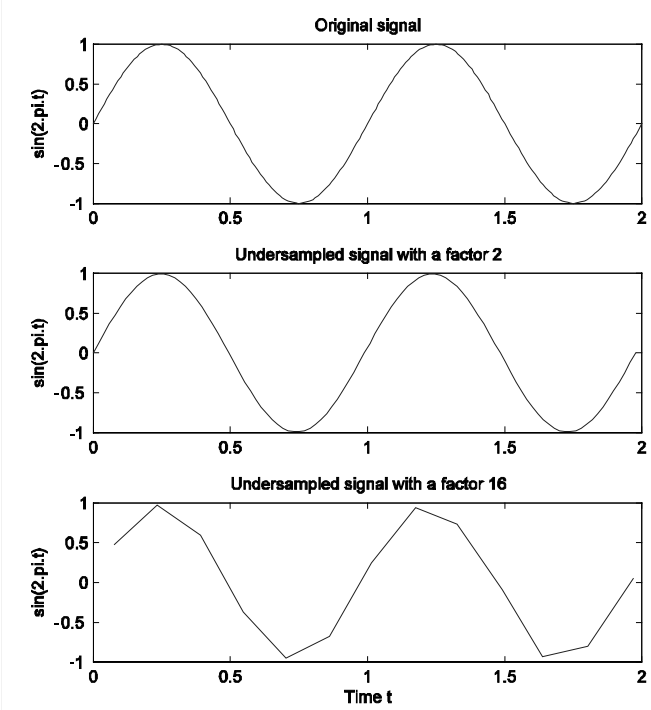


**Figure 1.2.** *Sinusoid waveform corresponding to different sample frequencies*

EXERCISE 1.9.

Plot the paraboloid defined by the equation: $z^2 = x^2 + y^2$ for $-50 \leq x, y \leq 50$.

```
N = 50; x = -N:N; y = -N:N;
% first solution (to avoid): two nested loops
%-------------------------------------------------------
for k = 1: 2*N+1
    for l = 1: 2*N+1
        z1(k,l) = sqrt(x(k)^2 + y(l)^2);
    end;
end;
figure; meshc(x,y,z1);
xlabel('x'); ylabel('y'); zlabel('z');

fprintf('Type a key to plot the paraboloid using another
method\n'); pause;
% second solution: one loop
%--------------------------------------
z2 = zeros(2*N+1,2*N+1);
```

```
for k = 1: 2*N+1
    z2(k,:) = sqrt(x(k)^2 + y.^2); % pointwise multiplication
for y
end;
figure; meshc(x,y,z2);
xlabel('x'); ylabel('y'); zlabel('z');
fprintf('Type a key to plot the paraboloid using another
method\n'); pause;
% third solution (the best): no loop
%-------------------------------------------
xc = x.^2; yc = y.^2;
mx=xc.'*ones(1,2*N+1); % line k of mx filled with the value
xc[k]
my=ones(1,2*N+1).'*yc; % column l of my filled with the value
yc[l]
z3 = sqrt(mx + my);
figure; meshc(x,y,z3);
xlabel('x'); ylabel('y'); zlabel('z');
```

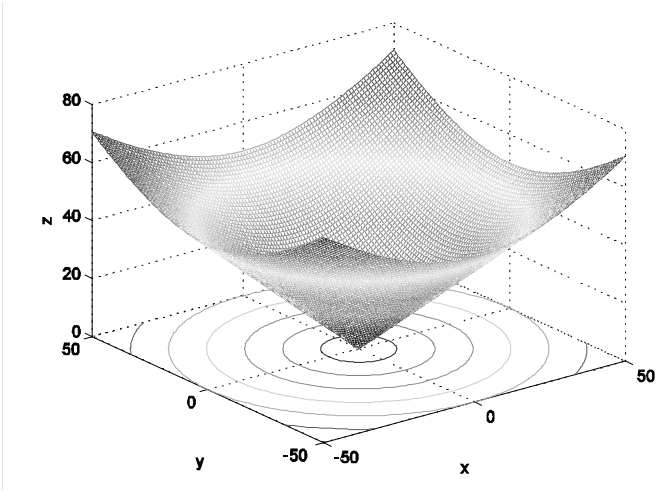


**Figure 1.3.** *Paraboloid plot*

EXERCISE 1.10.

1. Generate 1,000 independent values $x_1,...,x_{1,000}$ of a zero-mean random Gaussian variable with variance 4 using the function `randn`.

Plot the corresponding histogram and calculate the mean and the standard deviation of the generated series using the functions `hist`, `mean` and `std`.

Find out the mean and the standard deviation of the random series $x^2_1, \ldots, x^2_{1,000}$. Then compare the obtained results with the theoretical results.

```
clear all
n = 1000;

% If X ~ N(m,sigma^2) then Y = (X-m)/sigma ~ N(0,1)
Y=randn(1,n); X=2*Y;
[histoX,bins]=hist(X);
plot(bins,histoX);
xlabel('Bins');
ylabel('Number of values belonging to each bin');
title('Histogram of X using 10 bins');

% Find below 2 ways for displaying the results:

% 1) Character chain concatenation:
moyX=num2str(mean(X));
ecartX=num2str(std(X));
varX=num2str(var(X))
fprintf(strcat('\nMean of X          = ',moyX, '\n'));
fprintf(strcat('St. dev. of X      = ',ecartX, '\n'));
fprintf(strcat('Variance of X      = ',varX, '\n\n'));

% 2) Use of formats, just like in C:
% (type "help format" for more explanations)
fprintf('Mean of X          = %2.5f\n',mean(X));
fprintf('St. dev. of X      = %2.5f\n',std(X));
fprintf('Variance of X      = %2.5f\n',std(X)^2);
Z = X.*X;
fprintf('\nMean of Z          = %2.5f\n',mean(Z));
fprintf('St. dev. of Z      = %2.5f\n',std(Z));
fprintf('Variance of Z      = %2.5f\n',std(Z)^2);
fprintf('Var Z - 2*(Var X)^2 = %2.5f\n\n',std(Z)^2-2*std(X)^4);
```

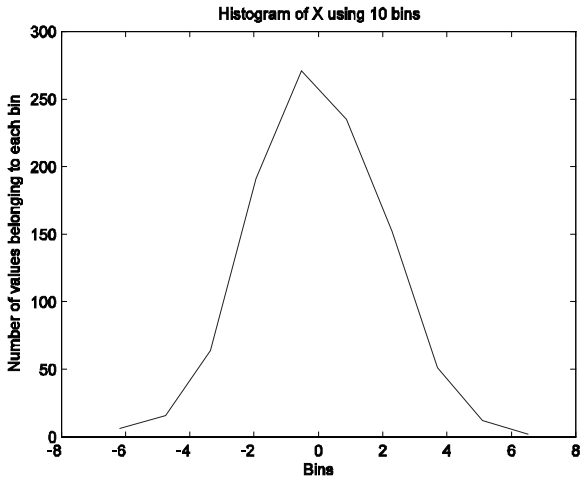


**Figure 1.4.** *Histogram of a Gaussian random variable*

```
Mean of X           = 0.0085986
St. dev. of X       = 1.963
Variance of X       = 3.8533

Mean of X           = 0.00860
St. dev. of X       = 1.96298
Variance of X       = 3.85328

Mean of Z           = 3.84950
St. dev. of Z       = 5.55695
Variance of Z       = 30.87972
Var Z - 2*(Var X)^2 = 1.18418
```

2. Use the function rand to generate 1,000 independent values of the random variable X defined by:

$$P(X = -1) = p_0; \quad P(X = 0) = p_1; \quad P(X = 1) = 1 - p_0 - p_1;$$

where $p_0$ and $p_1$ are the probabilities to be entered by the user.

```
function va_gen(n,po,p1)
help va_gen; Y = rand(1,n);
X = -1*[Y< po*ones(1,n)] + 1*[Y>((po+p1)*ones(1,n))];
% If Y is a uniformly distributed variable between 0 and 1, then the X
current value is obtained from the combination of 2 tests, so that X = -
1*(Y<p0) + 1*(Y>p0+p1):
% - if Y < p0 (this case occurs with the probability p0) then
%   the first test is true and the second is false, so X = -1
% - if Y > p0+p1 (this case occurs with the probability 1-p0-p1)
%   then the first test is false and the second is true, so X = 1
% - if p0 < Y < p0+p1 (this case occurs with the probability p1)
%   then the two tests are false, so X = 0
prob = hist(X,3)/n;
fprintf('\np [X = -1] = %1.4f\n', prob(1));
fprintf('p [X = 0]  = %1.4f\n', prob(2));
fprintf('p [X = 1]  = %1.4f\n\n', prob(3));
```

When the function va_gen is called:

```
va_gen(1000,0.1,0.5)
```

it provides the following result:

```
If Y is a uniformly distributed variable between 0 and 1, then the X current
value is obtained from the combination of 2 tests, so that X = -1*(Y<p0) +
1*(Y>p0+p1):
   - if Y < p0 (this case occurs with the probability p0) then
     the first test is true and the second is false, so X = -1
   - if Y > p0+p1 (this case occurs with the probability 1-p0-p1)
     then the first test is false and the second is true, so X = 1
```

```
   - if p0 < Y < p0+p1 (this case occurs with the probability p1)
     then the two tests are false, so X = 0

p [X = -1] = 0.1000
p [X = 0]  = 0.4840
p [X = 1]  = 0.4160
```

EXERCISE 1.11.

Plot in polar coordinates the poles of the filter having the following transfer function:

$$H(z) = \frac{1}{1 + az^{-1} + bz^{-2}}$$

The values of *a* and *b* are entered by the user and the function returns the poles. (use the commands `roots` and `polar`).

```
function c = filter_bis(a,b)
c = roots([1 a b]); % Comment: H(z) = poly(c)
fprintf('The poles are:\n'); z1 = c(1,:)
z2 = c(2,:)
if (abs(z1)> 1 | abs(z2) > 1)
     fprintf ('There is at least an instable pole\n');
else
     clf; figure; polar(angle(z1),abs(z1),'r+');
     % Second solution: use zplane
     hold on; polar(angle(z2),abs(z2),'r+');
     legend('Polar plot of H(z) poles',0);
end
```

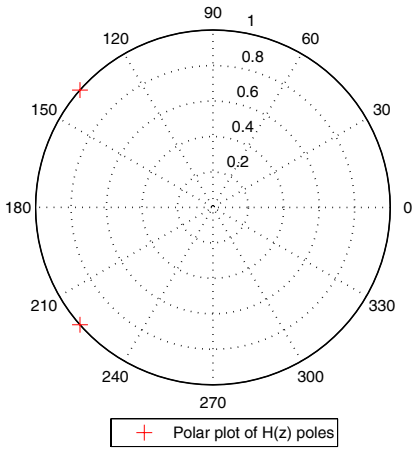


**Figure 1.5.** *Function call example – filter_bis (1.5,1)*

```
The poles are:

z1 =

   -0.7500 + 0.6614i

z2 =

   -0.7500 - 0.6614i
```

EXERCISE 1.12.

Generate the signal: $x(t) = A \cdot \sin(2\pi ft + \phi) + b(t)$, $t = 0..1024$, where $\phi$ is a uniformly distributed random variable on $[0, 2\pi]$ and $b(t)$ is a white Gaussian noise with mean zero and variance one (use `rand` and `randn`). $A$ and $f$ are chosen by the user.

Estimate the mean value, the autocorrelation function (`xcorr`) and the spectrum of $x(t)$ using the periodogram and the correlogram (use `fft` and `fftshift`). Compare the obtained results to the theoretical results. Change $A$ in order to control the SNR.

```
function noisy_sin(A,f)
N = 1024;    % Number of calculated frequencies
nech = 1024; % Number of samples
t = 0:nech;
b   = randn(1,nech+1);
phi = 2*pi*rand(1);
x = A*sin(2*pi*f*t+phi)+b;
fprintf('\nMean of x(t) = %2.4f\n',mean(x));
fprintf('Mean of b(t) = %2.4f\n',mean(b));
cx = xcorr(x);
% plot(cx);
% Correlogram based spectrum estimation:
sx_correlo = (abs(fft(cx,N))).^2;
sx_correlo = sx_correlo / max(sx_correlo);
% the first N/2 values correspond to 0<=f<0.5
% the last N/2 values correspond to 0.5<=f<1 (or -0.5<=f< 1)
sx_correlo = fftshift(sx_correlo);
% The spectrum is centred around 0:
% the first N/2 values correspond to -0.5<=f<0
% the last N/2 values correspond to 0<=f<0.5

% Periodogram based spectrum estimation:
%---------------------------------------
sx_periodo = (abs(fft(x,N))).^2;
sx_periodo = sx_periodo / max(sx_periodo);
sx_periodo = fftshift(sx_periodo);

% SNR estimation for a noisy sinusoid
%--------------------------------------------
```

```
vector(1:N) = sx_periodo;
vector(N+1:2*N) = sx_correlo;

plot(-0.5:1/N:0.5-1/N,10*log10(sx_correlo(1:N)),'r-',-0.5:1/N:0.5-
1/N,10*log10(sx_periodo(1:N)),'b:');
legend('Correlogram','Periodogram',0);
xlabel('Normalized frequency');
ylabel('Magnitude spectrum [dB]');
axis([-0.5 0.5 min(10*log(vector)) 0]);

% 0 dB <=> 10.log10 (Psignal + Pnoise)
% background_noise <=> 10.log10 (Pnoise) < 0
% Psignal = A^2/(2.N) (periodogram)
% Pnoise = sigma^2 = 1

SNRth = A^2/2;

fprintf('\nSignal SNR = %2.4f dB \n',10*log10(SNRth));
fprintf('\t=> Theoretical mean background noise corresponding to the
periodogram estimated spectrum\n');
fprintf('\t    in the range [-0.5:%1.2f] & [%1.2f:0.5] =  %2.4f dB\n\n',-f-
0.05,0.05+f,-10*log10(N*SNRth/2+1));

background_noise1 = mean(10*log10(sx_periodo(1:round(N*(0.45-f)))));
background_noise2 = mean(10*log10(sx_periodo(round(N*(0.65+f)):N)));

mean_background_noise=mean([background_noise1,background_noise2]);

fprintf('Mean background noise corresponding to the periodogram estimated
spectrum \n');
fprintf('in the range [-0.5:%1.2f] & [%1.2f:0.5] = %2.4f dB\n',-f-0.05,
0.05+f, mean_background_noise);
fprintf('\t=> Estimated SNR = %2.4f dB \n',10*log10((2/N)*(-1+exp(-
noise_moy*log(10)/10))));
```

Function call example: noisy_sin(2,0.15)

```
Mean of x(t) = 0.0529
Mean of b(t) = 0.0499

Signal SNR = 3.0103 dB
      => Theoretical mean background noise corresponding to the periodogram
estimated spectrum in the range
[-0.5:-0.20] & [0.20:0.5] =  -30.1072 dB

Mean background noise corresponding to the periodogram estimated spectrum in
the range [-0.5:-0.20] & [0.20:0.5] = -29.7643 dB
      => Estimated SNR = 2.6670 dB
```

The SNR estimation error is related to several odd spectrum values, which lead to a biased mean background noise level.

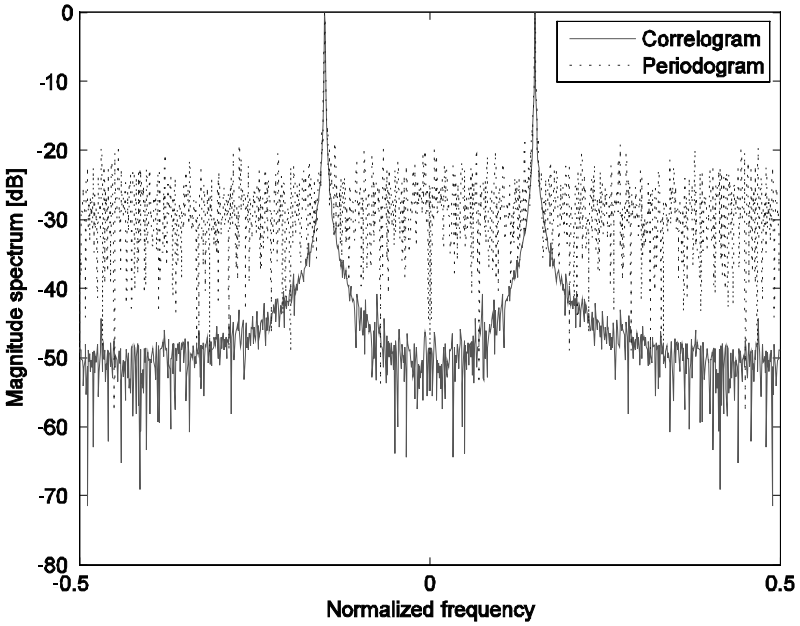


**Figure 1.6.** *Spectral representation of a noisy sinusoidal signal*