First part: Basic elements

1.3 Calculation on matrix

1.3 calculation on matrix

With Matlab, we essentially work with one type of object: matrices. A scalar variable is a matrix of dimension 1 x 1 and a vector is a matrix of dimension 1 x n or n x 1. It is essential to be comfortable with these notions to better understand the philosophy of Matlab and to exploit it effectively.

1.3.1 Definition of a vector

A vector is nothing else than a picture of stocks. There are several manners of creating a vector and the simplest of them is to write it expressly.

```
>> v = [1 3 3 4]
v =
1 3 3 4
```

All components are given in brackets and values are separated by a space (or a comma « , »). Here we have defined a line vector. A column vector is created using a semicolon « , » as delimiter.

```
>> v = [1 ; 3 ; 3 ; 4]
v =
1
3
3
4
```

Although simple, this method is not practical for defining large vectors. A second method uses the two-point operator « , » to discretize an interval with a constant step.

```
>> v = 0:0.2:1
v =
0 0.2 0.4 0.6 0.8 1
```

This statement creates a vector containing values ranging from 0 to 1 with a step of 0.2. The syntax is as follows: vector = initial_value: increment: final_value. By default, the step is equal to 1.

```
>> v = 0:6
v =
0 1 2 3 4 5 6
```

Finally, predefined functions make it possible to automatically generate vectors.

```
>> v = linspace(0,10,1000);
>> v = logspace(-1,2,1000);
```

The first function creates a vector of 1000 points with stocks going of 0 - 10 also spacing out. The second creates a vector of 1000 points on space from 10-1 to 102 with a logarithmic spacing out.

```
>> v = [9 4 -1 3 11 0.3];
>> v(4)
ans =
3
>> v (2:4)
ans =
4 -1 3
```

v(3) returns the 3rd element of vector v. Argument 2:4 selects a block of elements (here from the second to the fourth).

1.3.2 Some useful functions

We present in this paragraph a set of usual functions related to the use of tables.

+	addition of matrices
-	subtraction of matrices
*	die product
^	power
eye (n)	unit matrix (identity matrix) of size n x n
inv (X)	inverse of square matrix X
rank (X)	X matrix rank (number of independent columns or rows)
det (X)	determining the square matrix X
X '	transposed from matrix X
/	right division: A / B is equivalent to A * inv(B)
\	left division: A B is equivalent to inv(A) * B
length(v)	returns the size of the table.
max(v)	returns the maximum value of the array.
min(v)	returns the minimum value of the table.
mean(v)	returns the average value of the array elements.
sum(v)	calculate the sum of the elements in the table.
prod(v)	calculating the product of the elements in the table.
sort(v)	Ranks the elements in the table in ascending order.

All mathematical functions are applicable to vector-type variables.
In this case, the function is performed on each element of the vector.

```
>> v = [0 pi/4 pi/2 pi 2*pi]
v =
0 0.7854 1.5708 3.1416 6.2832
>> cos(v)
ans =
```

1.0 0.7071 0.0000 -1.0000 1.0000

1.3.3 Definition of a matrix

The definition of a matrix is delimited by square brackets «[ ]». The different elements of a line are separated by a space and the different lines are separated by semicolons «; ». Thus to define a matrix variable M=$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

We will write:

```
>> M = [1 2 3; 4 5 6; 7 8 9];
ou
>> M = [1, 2, 3; 4, 5, 6; 7, 8, 9];
```

Access to an element of a matrix is done by specifying indices in parentheses following its name. The element located $i^{\text{ième}}$ line and the $j^{\text{ième}}$j column is obtained by the command M (i,j). For example, the value is retrieved by typing M_{23}

```
>> M (2, 3)
ans =
6
```

You can also modify one of the elements directly by assigning it a new value.

```
>> M (2, 3) =11;
>> M
M =
1 2 3
4 5 11
7 8 9
```

1.3.4 Particular matrices

Some particular matrices, and very used, are more easily defined through functions. These functions take in argument the dimensions of the matrix which they like to construct. The first indicates the number of lines and second numbers it of columns.

The no matrix:

```
>> Z = zeros (2, 4)
Z =
0 0 0 0
0 0 0 0
A matrix full of 1:
>> U = ones (3, 3)
U =
1   1   1
1   1   1
1   1   1
The matrix identity:
>> I = eye (3)
I =
1 0 0
0 1 0
```

```
0 0 1
A random matrix (elements between 0 and 1):
1) :>> R = rand (2, 2)
R =
0.9575 0.1576
0.9649 0.9706
A diagonal matrix:
>> D = diag([42,33,0,71])
D =
42 0 0 0
0 33 0 0
0 0 0 0
0 0 0 71
```

Unlike the previous ones, this last function takes as an argument a vector. The size of the diagonal matrix is therefore determined by the size of the vector.

1.3.5 Extraction of sub-arrays

It is often useful to extract blocks from an existing table. For this we use the operator « : ». To do this, it is necessary to specify for each index the start value and the end value. The general syntax is therefore as follows (for a two-dimensional array): array (start:end, start:end).

Thus to extract the block $\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$ from the matrix M, we will type:

```
>> M(1:2,2:3)
ans =
2 3
5 6
```

The character « : » alone, means the entire length is extracted. In this way, one can isolate a complete row, or column.

Example

```
>> M(1:2,:)
ans =
1 2 3
4 5 6
>> M(1,:)
ans =
1 2 3
>> M(:,2)
ans =
2
5
8
```

1.3.6 Matrix construction by blocks

You know this principle in mathematics. For example, to start the previously defined matrices and vectors, we can define the matrix

$$N = \left[\begin{array}{c|c} M & V \\ \hline U & 0 \end{array}\right]$$

Which is a 4x4 matrix. To do ¸can under Matlab, we do as if the blocks were scalars, and we simply write:

```
>> N= [M V U 0]
N =
1    2 3  11
11 12 13 12
21 32 23 13
1    2 3    0
```

Or by using the character;

```
>> N= [M V; U 0]
```

This syntax is very used to lengthen vectors or matrices, for example if I want to add a column A M, made up by V:

```
>> M= [M V]
M =
1    2 3  11
11 12 13 12
21 32 23 13
```

If I want to add a line to him, made up of U:

```
>> M = [M; U]
M =
1    2    3
11 12   13
21 32   23
1    2    3
```

1.3.7 Operations on tables

We saw in the preamble that Matlab did not make a strong distinction between tables and matrices. In fact, for Matlab, everything is a picture, and a matrix is only a table which has a particular mathematical meaning.

So, if historically MATLAB offered functions in most cases for counting implicating matrices, today the functional ghost broadly stretched, and MATLAB offers functions for all counting on numerical data, tabulées in picture, or matrices in the mathematical sense of term. In this chapter, we review the arithmetic operations that we can perform with tabulated data or matrices [3].

- **Addition and subtraction**

Both operators are the same as for scalars, namely + and -. From the moment the two tables concerned have the same size, the resulting table is obtained by adding or subtracting the terms from each table.

1.3.8 Multiplication, division and power term to term

These operators are noted.*, / And.^ (Be careful not to forget the point).

They are planned to carry out term operations on two tables of the same size.

These operations are fundamental when we want to trace curves, and we will always see it again in this case of use, or more generally when we want to carry out these arithmetic operations on a set of tabulated data.

- **Multiplication**

Since you can manipulate matrices, Matlab also offers these matrix operations. The multiplication is noted simply * and should not be confused with the term term multiplication (which by definition does not give the same result).

It goes without saying that if we write a*b, the number of columns of A must be equal to the number of B lines for the multiplication to work.

- **Division**

The matrix division is defined as the multiplication by the reverse of the matrix. Thus A/B represents the matrix has multiplied (in the matrix sense) by the reverse matrix of B.

- **Complement**

There is also a division on the left which is noted \.Thus the syntax A \ B means the opposite of A multiplied by B. This symbol can also be used to resolve linear systems: if V is a vector, a \ v represents mathematically a^(- 1) to say the solution of the linear system AX = V.

- **Power**

The umpteenth power of a matrix represents this matrix multiplied, in the matrix sense, n times by itself.

1.3.9 Distinction between term operations and matrix operations

It is fundamental to clearly distinguish the multiplication operation (and by consequence of division and power) defined term term of that defined for matrices.

In many cases (and especially if the matrices are square and of the same dimensions), the two types of operations can be carried out, without producing a syntax error for the Matlab language, but will produce completely different digital results.

Example

To show the difference between operators. * And *, let's take a trivial example involving the Multipliée Identity matrix at the matrix (I*A). Here is the multiplication in the sense of matrices:

```
>> [1 0; 0 1] * [1 2; 3 4]
  ans =
       1    2
       3    4
```

In this case, we find the matrix $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$, by definition of what the identity matrix is.

And now let's look at the term -term multiplication:

```
>> [1 0; 0 1].* [1 2; 3 4]
ans =
        1    0
        0    4
```

- **Fundamental**

If you handle tabulated data, the operations you are going to carry out are necessarily term term; the operators *, / and ^ will then be preceded by a point.

If you perform matrix calculations, the operators have no point.

- **Complement**

For these three operations, for tables or matrices, there are also functions that can replace operators:

Operators and equivalent functions				
Term term operations			**matrix operations**	
A.*B	times(A,B)		A*B	mtimes(A,B)
A./B	rdivide(A,B)		A/B	mrdivide(A,B)
A.^B	power(A,B)		A^B	mpower(A,B)

First part: Basic elements

- **Transposition**
  The transposition operator is the character and is often used to transform line vectors into column vectors and vice versa.
- **Summary**
  The following table summarizes the various operators applicable to matrices.

The following table summarizes the different operators applicable to matrices or tables.

The different operators applicable to matrices or tables		
Matlab operator	**Mathematical writing**	**General term**
A	A	Aij
B	B	Bij
A+B	A+B	Aij+Bij
A-B	A−B	Aij−Bij
A.*B		AijBij
A./B		Aij/Bij
A.^B		AijBij
A.^s		Aijs
A*B	AB	∑kAikBkj
A/B	AB−1	
A\B	A−1B	
A^n	An	
A'	A^T	Aji

Example

Entering a square matrix of size 3 x 3:

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


     2     4     6
     1     9     7
    -3     1     1
>> A (2, 3)
ans =
7
>> A (2, 3) = 7
A =


     2     4     6
     1     9     7
    -3     1     1
 >> A'
```

```
ans =

     2     1    -3
     4     9     1
     6     7     1
>> inv(A)
ans =

    0.0238    0.0238   -0.3095
   -0.2619    0.2381   -0.0952
    0.3333   -0.1667    0.1667
 >> D = A * inv(A)
D =

    1.0000         0         0
    0.0000    1.0000    0.0000
         0   -0.0000    1.0000
 >> rank(A)
ans =
3
 >> det(A)
ans =

    84
>> eye(7)
ans =
ans =

     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     1     0     0     0
     0     0     0     0     1     0     0
     0     0     0     0     0     1     0
     0     0     0     0     0     0     1
 >> B = [1 9 0 ; 1 3 1 ; 0 -1 1]
B =

     1     9     0
     1     3     1
     0    -1     1
 >> A + B

ans =
     3    13     6
     2    12     8
    -3     0     2
 >> 11 + A

ans =

    13    15    17
    12    20    18
     8    12    12
 >> 11 * A
ans =

    22    44    66
    11    99    77
   -33    11    11
 >> A * B
ans =
     6    24    10
    10    29    16
    -2   -25     2
 >> B * A
ans =
    11    85    69
     2    32    28
    -4    -8    -6
 >> A*A*A*A
ans =
        -768        4008        2880
       -1428        7812        5772
         180        -564        -420
 >> A^6
ans =
      -55440      324576      240480
     -110016      635760      470304
        7200      -42048      -30384
 Entering a matrix with complex coefficients of size 2 x 3:
>> C = [2- i 0 0; 1 - i 2*i 2]
C =

   2.0000 - 1.0000i        0                  0
   1.0000 - 1.0000i        0 + 2.0000i   2.0000
 >> C * A
```

```
ans =

   4.0000 - 2.0000i   8.0000 - 4.0000i 12.0000 - 6.0000i
  -4.0000             6.0000 +14.0000i   8.0000 + 8.0000i
```

1.3.10 Relational or logical operations on tables

As with arithmetic operations, logical operations that exist for scalar numbers can also be applied to digital tables. Relational operators make it possible to make logical comparisons between digital values.

The logical value 'True' is called True, and 'False' is called False; These values are of the logical type.

All relational or logical operations refer a result equal to logical value 0 or 1.

- **Relational operators**

Relational operators allow the comparison of two values between them. The following table synthesizes the syntaxes of the different relational operators available:

Syntax of relational operators		
Relational operation	**Syntaxe MATLAB**	
	symbol	**function**
A equal to B	A == B	eq(A,B)
A different from B	A ~= B	ne(A,B)
A greater than B	A > B	gt(A,B)
A greater than or equal to B	A >= B	ge(A,B)
A less than B	A < B	lt(A,B)
A less than or equal to B	A <= B	le(A,B)

Example

Let's take some examples of relational operations on any numeric values:

```
>> 20 == 20
  ans =
    1
  >> 1.4 > 6.2
  ans =
    0
```

- **Comparison of a table and a scalar**

First part: Basic elements

Example

```
>> A=[2 4; 5 2]
A =
     2     4
     5     2

>> A > 3

ans =

     0     1

     1     0
```

Les termes de A supérieurs à 2 donnent 1 (true), les autres 0 (faux). La possibilité d'appliquer une opération relationnelle sur un tableau sera exploitée dans la suite pour calculer les valeurs d'une fonction définie par morceaux.

- **Comparison of two tables**

Example

Now let's take any two tables of the same dimension, and compare them:

```
>> A=[-1 6 ; 3 -4]
A =
    -1     6
     3    -4
  >> B=[-9 2.5 ; 1 -2]
B =
   -9.0000    2.5000
    1.0000   -2.0000

>> T = A > B

T =

     1     1

     1     0
```

- ***Complement***

It may be useful to know if the Table T Result of a test contains that non-zero values (or logical values true) or if there is at least one non-null value (or logical value True).

To do this, you can use the All (T) and Any (T) functions, respectively. By default, if T is not a vector, these functions test the presence of non -zero values according to the columns of T.

Example

Let's resume the previous example***:***

```
>> all(T)
```

```
ans =

     1     0
  >> any(T)
ans =
    1  1
  >> all(all(T))
  ans =
    0
  >> any(any(T))
  ans =
    1
```

- **Logical operators**

Logical operators are operators that apply exclusively to logical type values. They allow the combination of logical conditions. The following table gives the syntax of logical operators available in Matlab:

Syntax of logical operators		
Relational operation	**Syntaxe MATLAB**	
	symbol	**function**
A et B	A & B	and(A,B)
A ou B	A | B	or(A,B)
A or exclusive B		xor(A,B)
Negation of A	~A	not(A)

While this table recalls the result of these logical operations:

Logic table					
A	**B**	**and(A,B)**	**or(A,B)**	**xor(A,B)**	**not(A)**
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

A and B can be logical scalar values or logical values tables of the same dimensions. For tables, these operations apply to term.

- **Short-circuit logic operators**

We call logical operators short-circuit operators ** and ||. Unlike operators * and |, short-circuit logical operators do not assess the second opendial, if the value of the first already allows you to know the overall result.

- **Table lengths**

The size function applied to a matrix returns a table of two integers: the first is the number of lines, the second the number of columns. The command also works on the vectors and returns 1 for the number of lines (resp. Columns) of a line vector. For vectors, the Length command is more practical and returns the number of components of the vector, whether line or column.

1.3.11 Creation of the .m file of a function

To define a new function in Matlab, we write the definition of the function in a file with an extension .m (M-File function). The name of the file must be the name of the first defined function (the visible only one). It is performed by typing the name of the function with the list of arguments in parentheses.

The first line of the file of function must follow the following syntax:

Function arguments of exit = name (arguments of entrance)

They must therefore declare stocks calculated by function in arguments of exit, and parameters of function in arguments of entrance.

Example 1

Recursive function computing by dichotomy of the root of a function if f(x)=`x^3-3*x-7` and x on `[a,b]=[1,4]`.

```
function x=racinefun(a,b)%[a,b]=[1,4]
% calcul la racine de f(x) définie ci-après sur [a,b]
fa=f(a); fb=f(b); x=(a+b)/2;
if (fa*fb>0) x=-Inf; return; end;
while (b-a)>eps*x
  x=(a+b)/2; fx=f(x);
  if(sign(fx)==sign(fa))
   a=x; fa=fx;
  else
   b=x; fb=fx;
  end;
end;
% definition de f(x) (fonction locale)
function y=f(x)
y=x^3-3*x-7;
```

First part: Basic elements

The function root ci over is written in a racinefun.m file. To carry it out, they knock simply

```
>> racinefun(1,4)
ans =
   2.4260
```

Example 2

we want to create a function which calculates the cube of a number X then returns the result y. The program will be:

```
function  y = cub3(x)
% cette fonction calcule le cube d'un nombre x
      y = x^3 ;
end
>> cub3(5)
ans =
  125
```

The function will be recorded under the name "cub3.m". A function must always end with an end delimator ("end").

Example 3

```
function  y = foc(x)
 % cette fonction calcule le cube d'un nombre x
       y =cos( x.^2)+x.^3+10./(x-1) ;
end
```

Let be the function:

a) Start to open a text editor: In the Matlab command window:

File -> New -> M-file

With version 6.5. the default text editor is the 'M-File Editor' application.

b) Give this function a name (in this example foc) and enter its mathematical expression:

c) Safeguard the file in your working directory (for instance c:USERS)
Name:foc
Extension: .m
d) Add the way of the directory where is your foc.m file

```
>> path(path,'c:\USERS')
File -> Set Path -> Add Folder
-> Save -> Close
```

- **Valuation of a function**

Calculation of y(x=0):

```
>> foc(0)
```

First part: Basic elements

```
ans =
    -9
Calculation of y ( x = 10 ) :
>> foc(10)
ans =
  1.0020e+003
>> foc(1)
Warning: Divide by zero
ans =
Inf
```

With a vector in argument, the function returns a vector:

```
>> foc([0 1 2 3 4 5])
ans =
   -9.0000       Inf    17.3464    31.0889    66.3757   128.4912>> >>
x=0:6
x =
     0     1     2     3     4     5     6
>> y = foc(x)
y =
  -9.0000 Inf   17.3464   31.0889   66.3757  128.4912  217.8720
```

With a matrix in argument, the function returns a matrix:

```
>> foc( [ 0 1 2 3 ; 5 6 7 8] )
ans =
   -9.0000        Inf    17.3464    31.0889
  128.49127.8720  344.9673  513.8204
```

Exercise:

Be the matrix $A=\begin{pmatrix} 1 & 4 & 1 & 1 \\ 1 & 7 & 1 & 2 \\ 1 & 4 & 1 & 2 \\ 3 & 10 & 2 & 5 \end{pmatrix}$

a/ Create matrix A using MATLAB

b/ Extract the following blocks from the matrix A:

$b1=\begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 2 & 5 \end{pmatrix}, b2=\begin{pmatrix} 1 & 7 & 1 \\ 1 & 4 & 1 \\ 3 & 10 & 2 \end{pmatrix}.$

c/ Give the values of **A(3,2) , A(3 :4,3) , A(3, :), tril(A)**

d/ Write with MATLAB the matrix D defined by:

D = Id - A.*At where Id denotes the identity matrix and At the transpose matrix of A.

First part: Basic elements

e/ Define matrix **B = [0.5*ones(4,2) - 2*ones(4,2)]** and give acquired result. Is the product of A and B possible ? Justify your answer. If yes which is the MATLAB order which allows to make this product?

Solutions

```
A =

     1     4     1     1
     1     7     1     2
     1     4     1     2
     3    10     2     5
b1 =
     1     2
     1     2
     2     5
b2 =
     1     7     1
     1     4     1
     3    10     2
ans =
     4

ans =
     1
     2

ans =

     1     4     1     2
ans =
     1     0     0     0
     1     7     0     0
     1     4     1     0
     3    10     2     5
D =
     0    -4    -1    -3
    -4   -48    -4   -20
    -1    -4     0    -4
    -3   -20    -4   -24
B =
   -1.5000   -1.5000
   -1.5000   -1.5000
   -1.5000   -1.5000
   -1.5000   -1.5000
ans =
     4     4
ans =
     4     2
ans =
  -10.5000  -10.5000
  -16.5000  -16.5000
  -12.0000  -12.0000
  -30.0000  -30.0000
```

1.4 Graph in two dimensions and management of the graphic windows

1.4 Graph in two dimensions and management of the graphic windows

1.4.1 Draw the graph of a function

- **Functions**

fplot	trace point by point the graph of a function
grid	adds a grid
xlabel	adds a legend for the x-axis
ylabel	adds a legend for the y-axis
title	adds a title
axis	modifies the scales of the axes
zoom	zooms in
gtext	places a legend with the mouse
hold	adds a graph in the current window
figure	creates a new window

The fplot command draws the graph of a function over a given interval.
Syntax is: fplot('nomf', [xmin , xmax]) où

- nomf is either the name of an incorporated matlab function, or an expression defining a function of the variable X, or the name of a user function.
- [xmin , xmax] is the interval for which the function graph is drawn.

Let us illustrate by examples the three ways to use the command fplot.

Example 1

Draw the function graph `'x*cos(2*x)+x^2'in [-2*pi 2*pi]`

```
>> fplot('x*cos(2*x)+x^2',[-2*pi 2*pi])
```

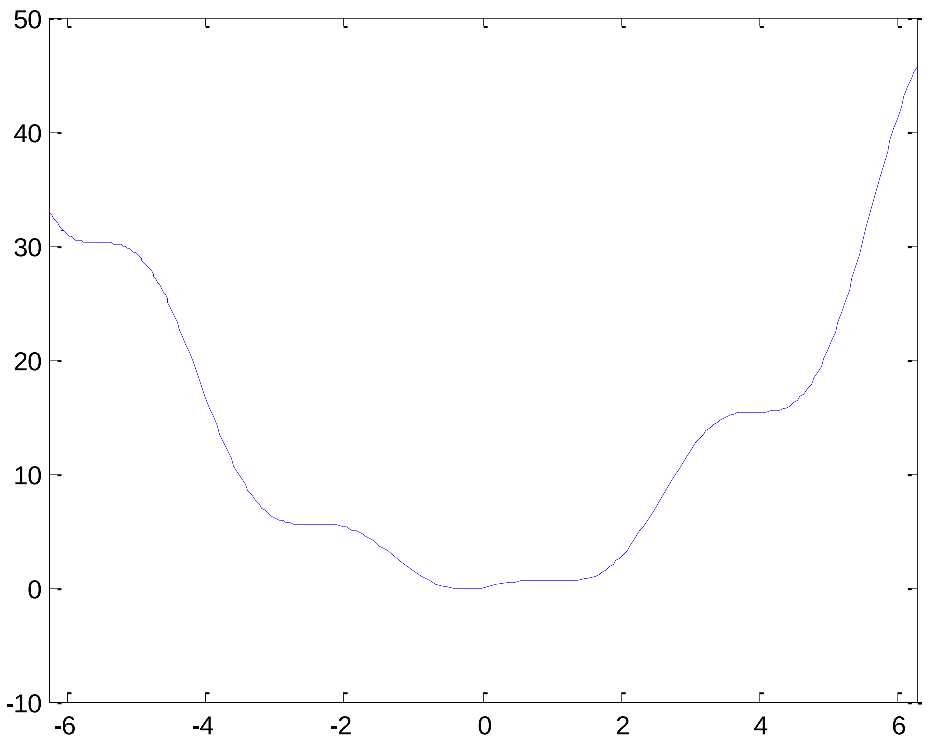


Fig. 1 – Function graph `'x*cos(2*x)+x^2'in [-2*pi 2*pi]).`

To plot the graph of the function h(x), we can define the user function h in the file h.m as follows:

```
      function y=h(x)
     y=…. ;
>>fplot('h', [-x x]).
```

To plot the graph of the function h(x), we can define the user function h in the file h.m as follows:

```
fplot('h', [-x x]).
```

Example 2

plot the graph of the function `2*x^2*sin(-2*x)+x in [-pi pi]`

```
function y=foc(x)
y='2*x^2*sin(-2*x)+x';
end
ans =

2*x^2*sin(-2*x)+x
>> fplot('2*x^2*sin(-2*x)+x', [-pi pi])
```

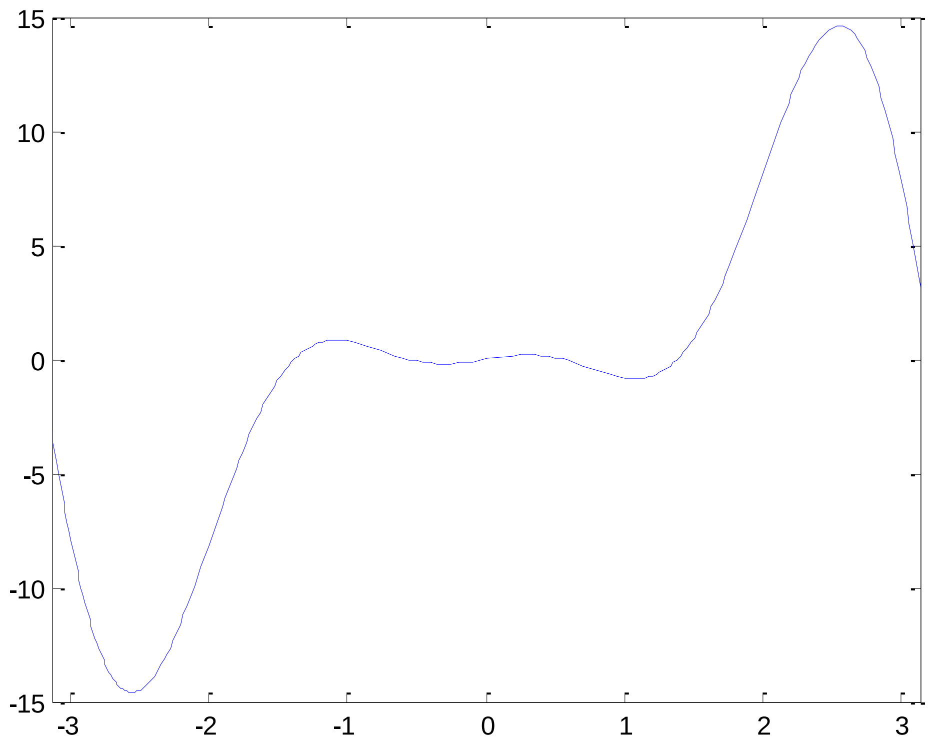


Fig. 2 – Function graph `2*x^2*sin(-2*x)+x in [-pi pi]).`

It is possible to draw several functions on the same figure. It is necessary to use for this the fplot command as follows: fplot('[nom_f1 , nom_f2 , nom_f3]', [x_min , x_max]) where nomf_f1, nom_f2, nom_f3 is either the name of an embedded matlab function, or an expression defining a function of variable x, or the name of a user function.

It is also possible to manage the bounds of the values in ordinates. To limit the graph to the ordinates between the values y_min and y_max we will pass as the second argument of the fplot command the array [x_min , x_max , y_min , y_max] ,[4].

Example 3

plot the graph of the function `-sin(-x) , x*cos(-x) in [-3, 2, -2, 1]`

```
fplot('[-sin(-x) , x*cos(-x)]', [-3 ,2, -2 ,1])
```

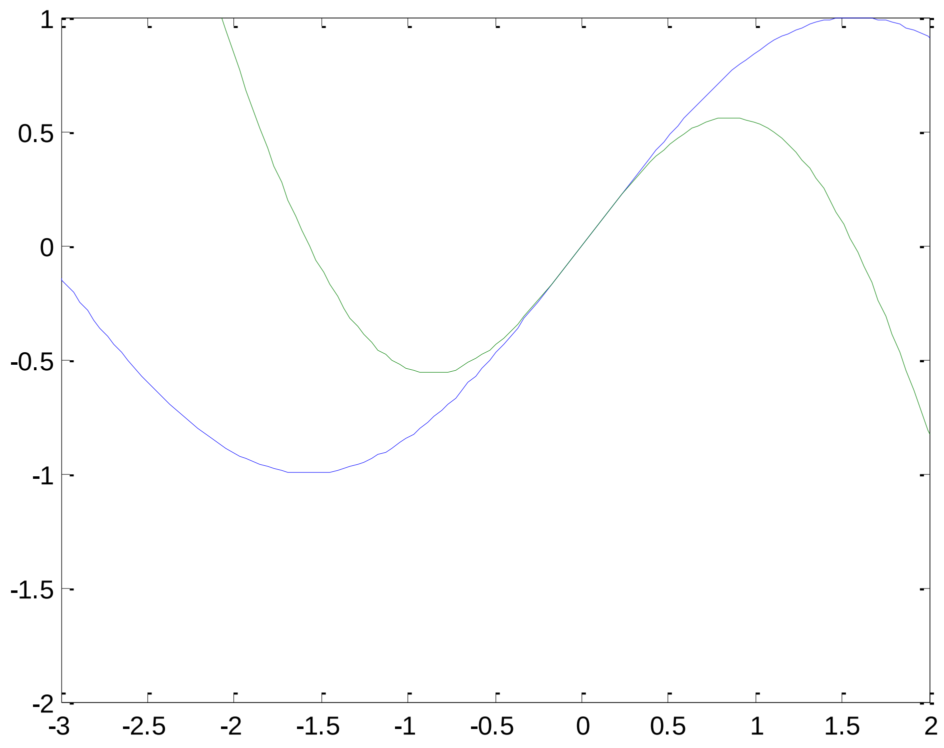


Fig. 3 – Function graph (`-sin(-x) , x*cos(-x)) in [-3, 2, -2, 1]`.

1.4 .2 The Plot command

The plot command allows you to draw a set of coordinate points (xi, yi)
i = 1, ... N. The syntax is plot(x,y) where x is the vector containing the x values on the abscissa and y is the vector containing the yi values on the ordinate. Of course the vectors x and y must be of the same dimension but they can be row or column vectors.
By default, the points (xi, yi) are connected to each other by straight segments.

Example 1

plot the graph of the function h(x) = 2x sin(2x)-1 in `[-pi pi]`

```
x=[-5*pi:0.001:5*pi];
 y = 2*x.*sin(2*x)-x;
 plot(x,y)
```

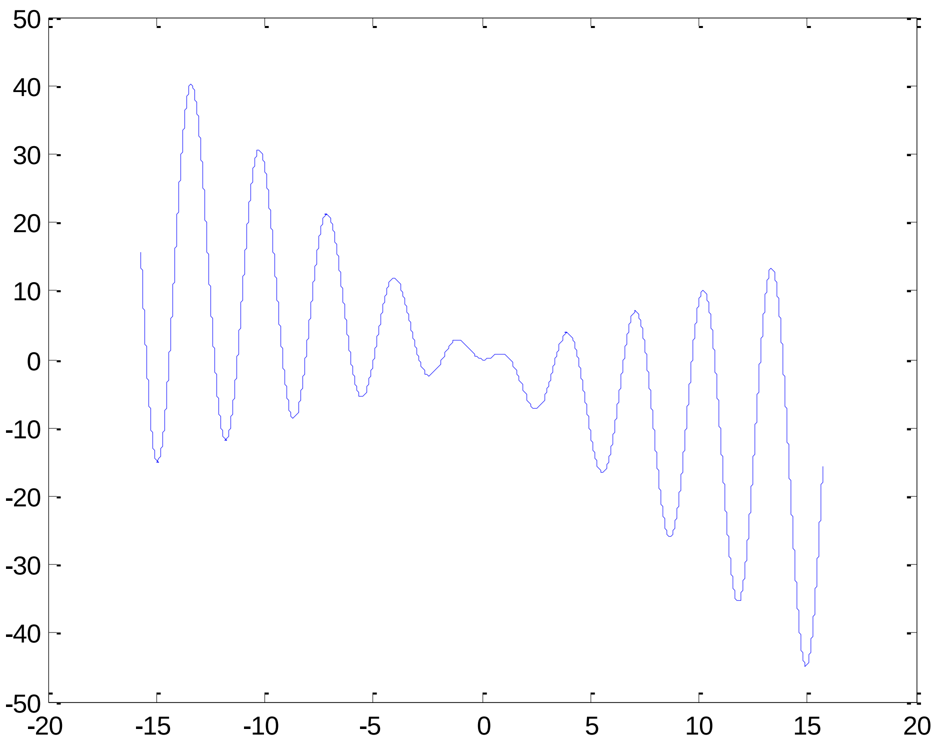


Fig. 4– Function graph 2`x*sin(2*x)-x in [-5pi 5pi]).`

We can specify to matlab what should be the color of a curve, what should be the line style and / or what should be the symbol at each point (xi, yi). For this we give a third input parameter to the plot command which is a string of 3 characters of the form 'cst' with c denoting the color of the line, s the symbol of the point and t the line style. The possibilities are as follows:

Color Name	Short Name
`'red'`	`'r'`
`'green'`	`'g'`
`'blue'`	`'b'`
`'cyan'`	`'c'`
`'magenta'`	`'m'`
`'yellow'`	`'y'`
`'black'`	`'k'`
`'white'`	`'w'`

: Point

o: Circle

First part: Basic elements

x: X-Mark

+: Plus

s: Square

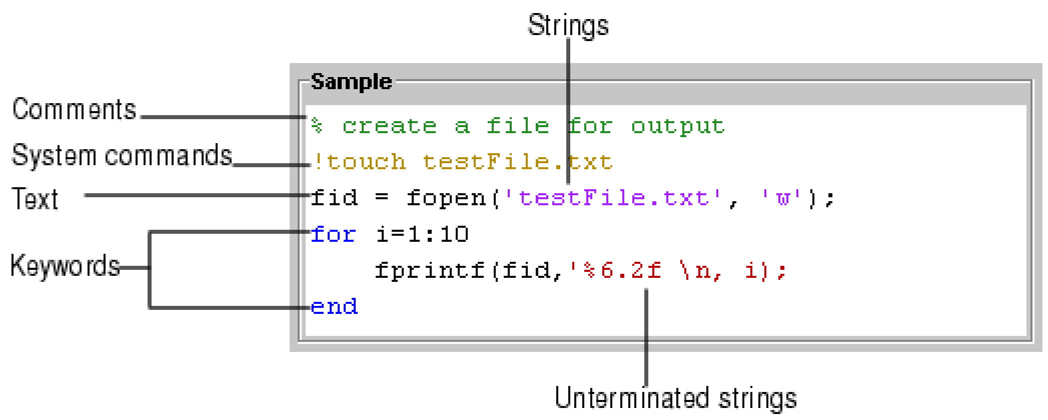
*: Star

d: Diamond

v: Triangle (down)

^: Triangle (up)

<: Triangle (left)



```
% create a file for output
!touch testFile.txt
fid = fopen('testFile.txt', 'w');
for i=1:10
    fprintf(fid,'%6.2f \n, i);
end
```

The default colors are listed here:

Keywords--Flow control functions, such as for and if, as well as the continuation ellipsis (...), are colored blue.

Comments--All lines beginning with a %, designating the lines as comments in MATLAB, are colored green. Similarly, the block comment symbols, %{ and %}, as well as the code in between, appear in green. Text following the continuation ellipsis on a line is also green because it is a comment.

Strings--Type a string and it is colored maroon. When you complete the string with the closing quotation mark ('), it becomes purple. Note that for functions you enter using command syntax instead of function syntax, the arguments are highlighted as strings. This is to alert you that in command notation, variables are passed as literal strings rather than as their values. For more information, see MATLAB Command Syntax in the MATLAB Programming documentation.

Unterminated strings--A single quote without a matching single quote, and whatever follows the quote, are colored maroon. This might alert you to a possible error.

System commands--Commands such as the ! (shell escape) are colored gold.

Errors--Error text, including any hyperlinks, is colored red.

Example 2

```
X = -10 : 0.05 : 10;
Y = x.^4 - x.^2;
plot (x, y, 'r')
```

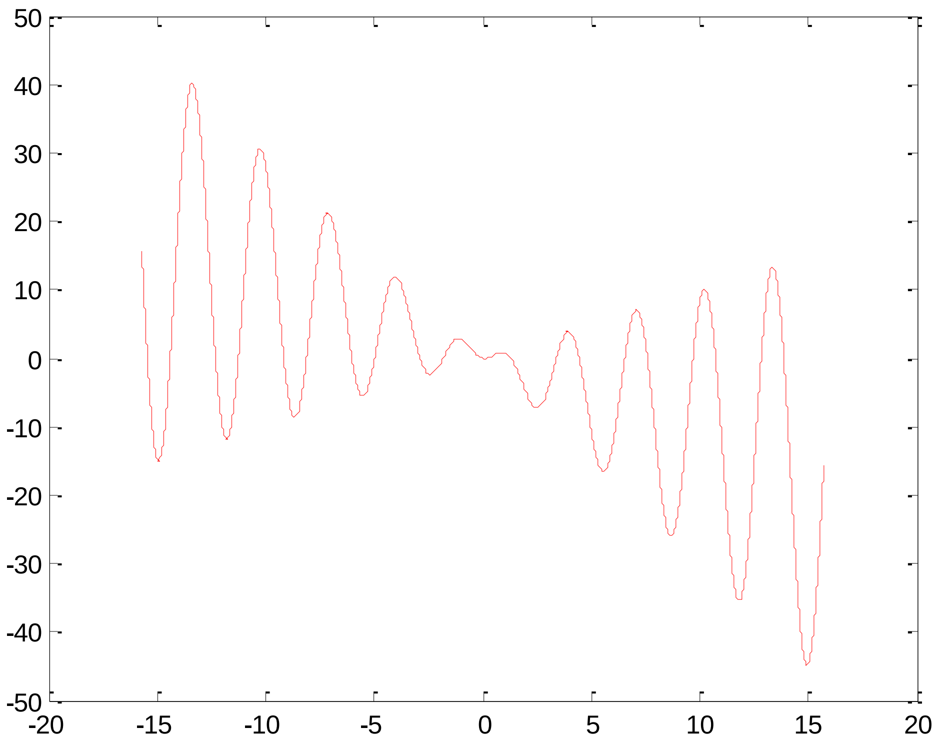


Fig. 5– Function graph `Y = x.^4 - x.^2;in [-10 10].`

```
X = -10 : 0.5 : 10;
Y = x.^4 - x.^2;
plot (x, y, 'dr')
```

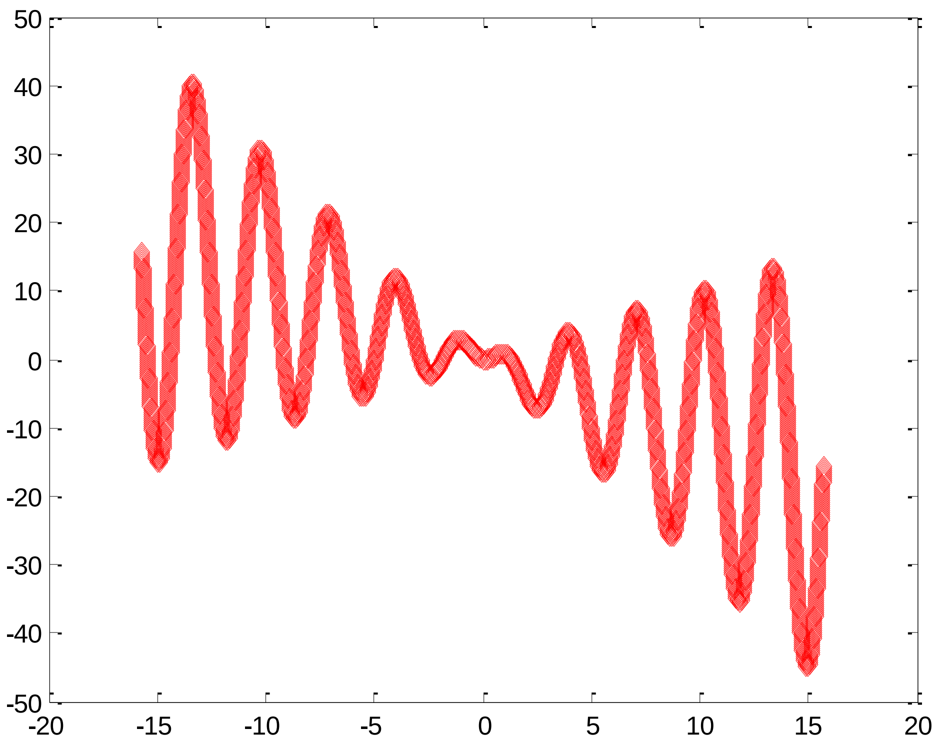


Fig. 6– Function graph `Y = x.^4 - x.^2;in [-10 10].`

Example 3

```
 x = [-4:0.01:4];
 y = x.^3.*sin(3*x)-x; z = 2*x.*sin(2*x)-x;
plot(x,y,'b-',x,z,'r:');
```

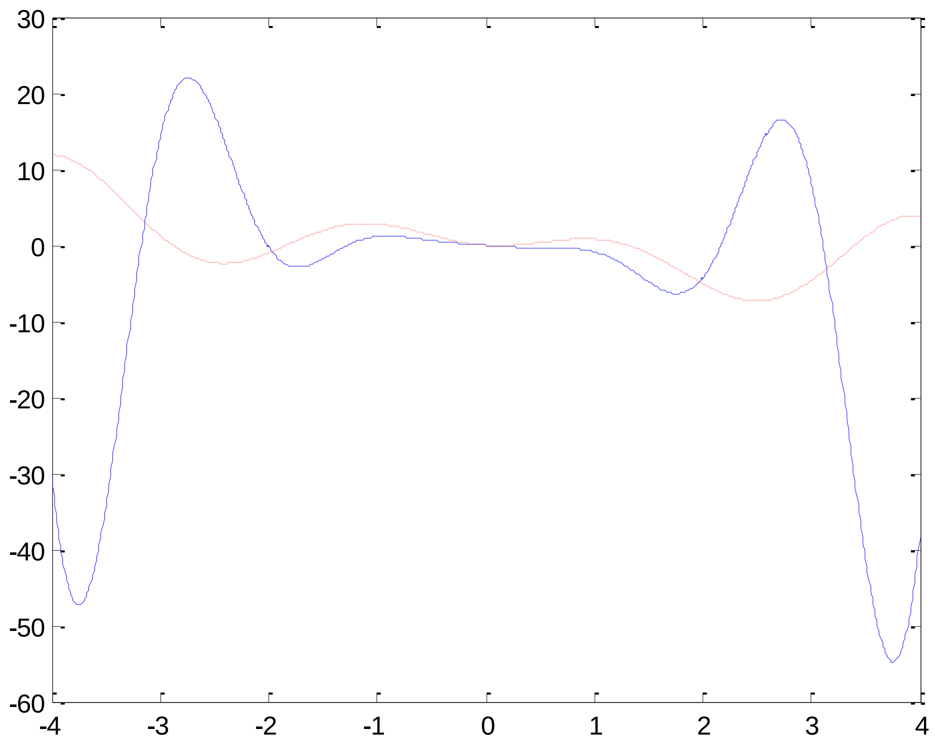


Fig. 7– Function graph y and z.

Example 4

```
N=100;
 x = rand(1,N); y = rand(1,N);
plot(x,y,'bd')
```

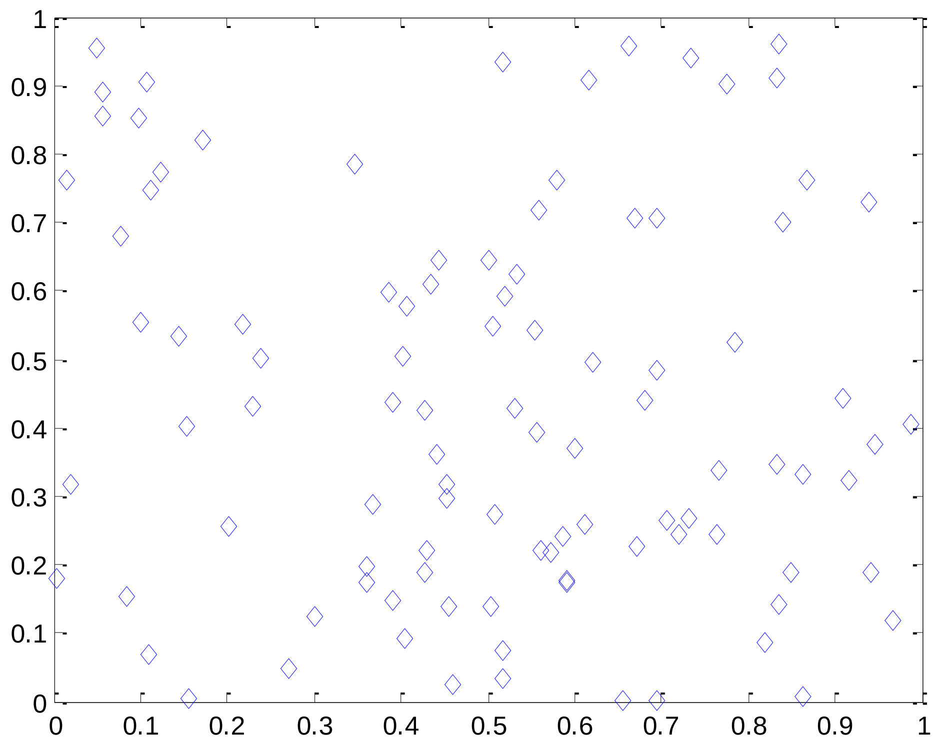


Fig. 8 – Function graph x and y.

1.4.3 The command loglog

If x and y are two vectors of the same dimension, the loglog(x,y) command displays the log(x) vector against the log(y) vector. The loglog command is used in the same way as the plot command

Example

```
x = [1:40:3000];
y = x.^3;
loglog(x,y)
```

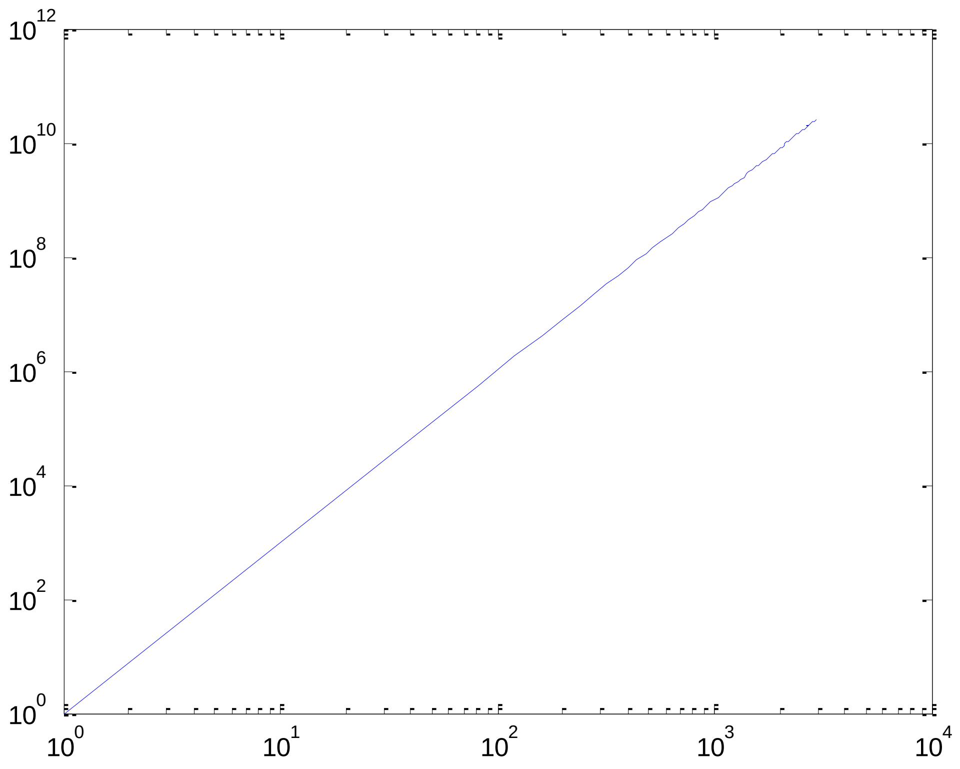


Fig. 9 – Result of the order loglog(x,y).

1.4.4 Legend of a figure

It is recommended to put a legend to a figure. The xlabel command is used to caption text below the x-axis. The syntax is xlabel(' legend ') to get the word legend in legend. The ylabel command does the same for the ylabel axis. The title command is used to give a title to the figure. The syntax is title(' the title ') to get the title as the title.

You can also write a given text at a specific position on the figure thanks to the text command. The syntax is text(posx,posy,'a text') where posx and posy are the coordinates of the point where a text is to begin writing.

The gtext command allows you to place the text at a chosen position on the figure using the mouse. The syntax is gtext(' a text '). A sight, which is moved using the mouse, appears. All it takes is a "mouse click" for the text to appear at the selected position.

It is possible with these commands to display a value contained in a variable in the middle of the text. To do this, we build an array of type character string by converting the value contained in the variable into a character string using the num2str command.

For example, suppose the numex variable contains the number of the sample being processed, say 5. The title of the figure Example number 5 is obtained by the statement: title(['Exemple numero ', num2str(numex)]).

Example

```
t = [0:0.001:8];
 h = 15*exp(-t) - 8*exp(-5*t);
 plot(t,h);
 grid
 xlabel('temps en minutes')
 ylabel('concentation en gramme par litre')
 title(['evolution de la concentration du produit ,  num2str(P),
... au cours du temps'])
 gtext('concentration maximale')
```

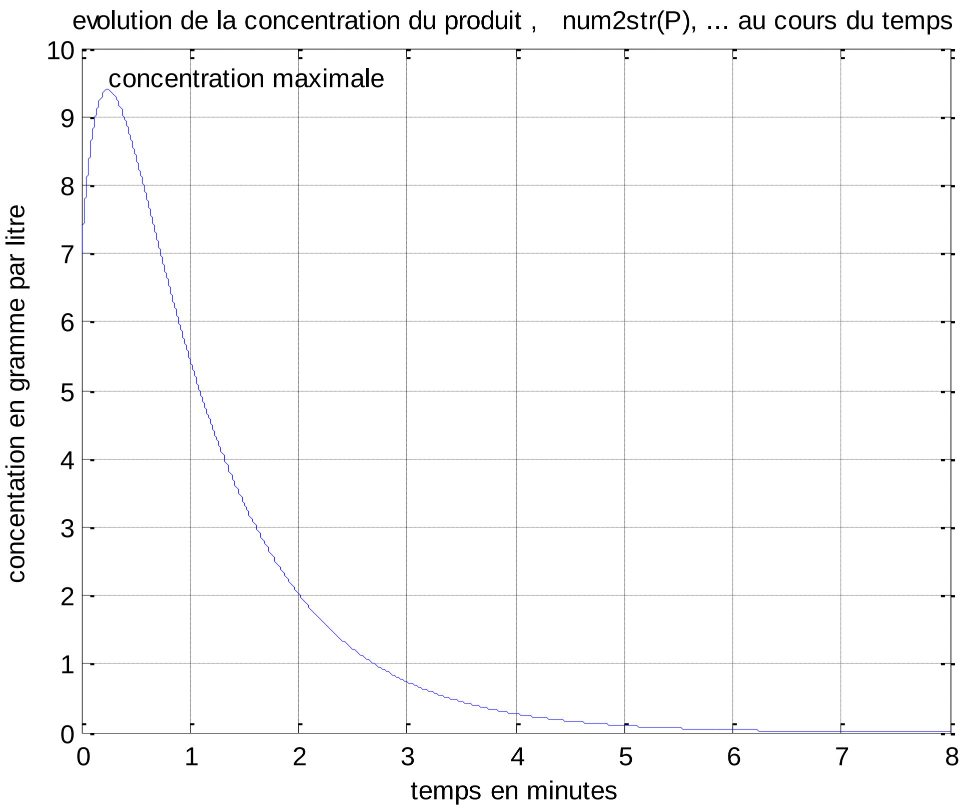


Fig. 10 – Graph of the function h.

1.4.5 Display multiple curves in a single window

It is possible to display several curves in the same graphics window thanks to the hold on command. The results of all graphics statements executed after calling the hold on command will be overlaid on the active graphics window. To restore the previous situation (the result of a new graphic instruction replaces the previous drawing in the graphics window) we will type hold off.

Example 1

```
clc
 figure
 hold on
 fplot('-2*exp(x)',[-1 1],'k')
 fplot('log(x)',[1/3 4],'g')
 plot([-1:0.01:3],[-1:0.01:3],'r')
 grid on
 hold off
```

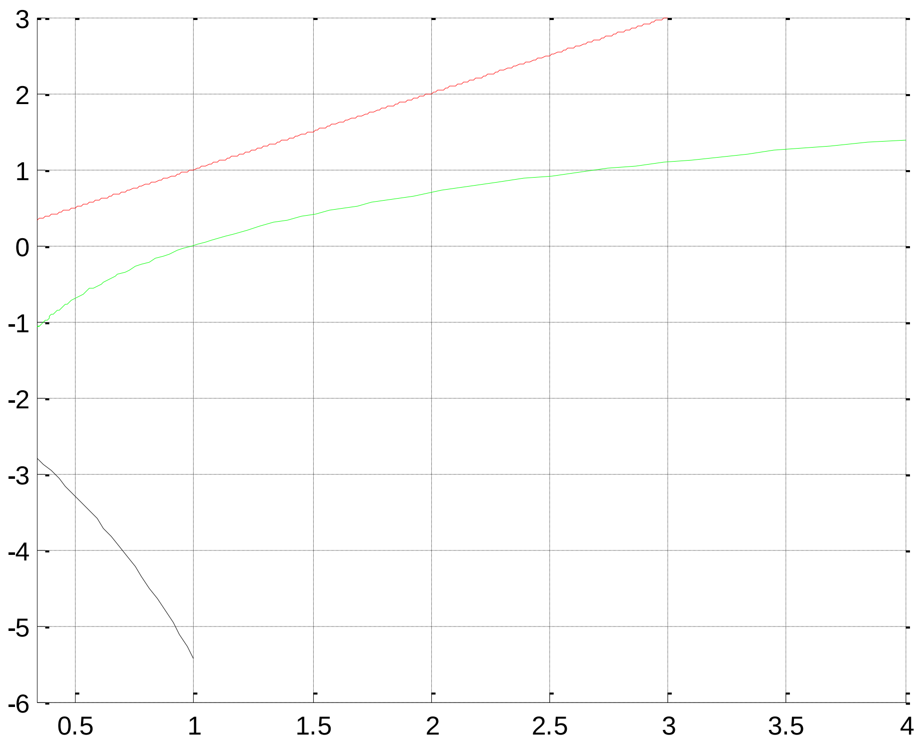


Fig. 11 – Multiple curves in the same graphics window.

It is possible to break down a window into panes and display a different figure on each of these subwindows using the subplot command. The syntax is subplot(m,n,i) where

- m is the number of sub-windows vertically;
- n is the number of subwindows horizontally;
- i is used to specify in which pane the display should be made.

Example 2

```
clc
figure
subplot(2,3,1), fplot('cos(-2*x)',[1 4*pi]), title('cosinus(-2*x)'), grid
subplot(2,3,2), fplot('sin(-5*x)',[1 4*pi]), title('sinus(-5*x)'), grid
subplot(2,3,3), fplot('tan(x^2+1)',[-pi pi]), title('tangente(x^2+1)'), grid
subplot(2,3,4), fplot('acos(x^2-1)',[-2 2]), title('arc-cosinus(x^2-1)'), grid
subplot(2,3,5), fplot('asin(x^2)',[-2 2]), title('arc-sinus(x^2)'), grid
subplot(2,3,6), fplot('atan(x^2+x)',[-sqrt(9) sqrt(9)]), title('arc-tangente(x^2+x)'), grid
```

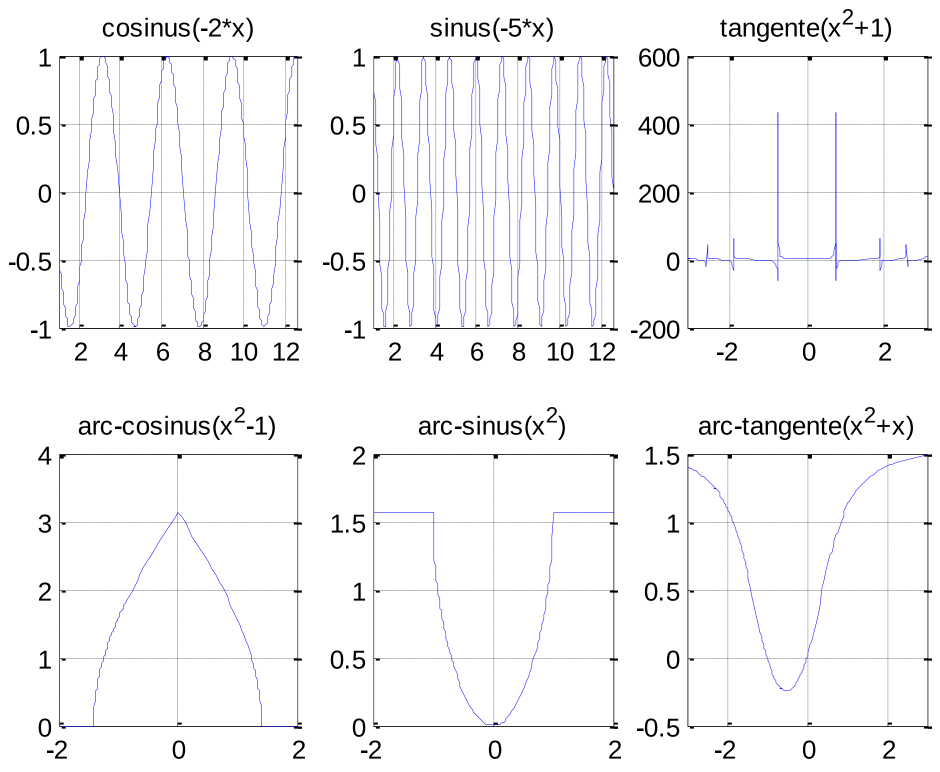


Fig. 12 – Graphic window broken down into sub-windows.

Example 3

Time (hours)	**0**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
Temperature (°C)	20	23	30	33	32	37	34	39	36

```
 clc
 Time = [0 2 4 6 8 10 12 14 16];
Temperature = [20 23 30 33 32 37 34 39 36];
 plot(Time , Temperature)
 grid on
 xlabel ( ' Time ( hours)' )
 ylabel ( ' Temperature  ( °C)' )
 title ( ' Monitoring of temperature ')
 axis ( [ 0 18 10 40 ] )
```

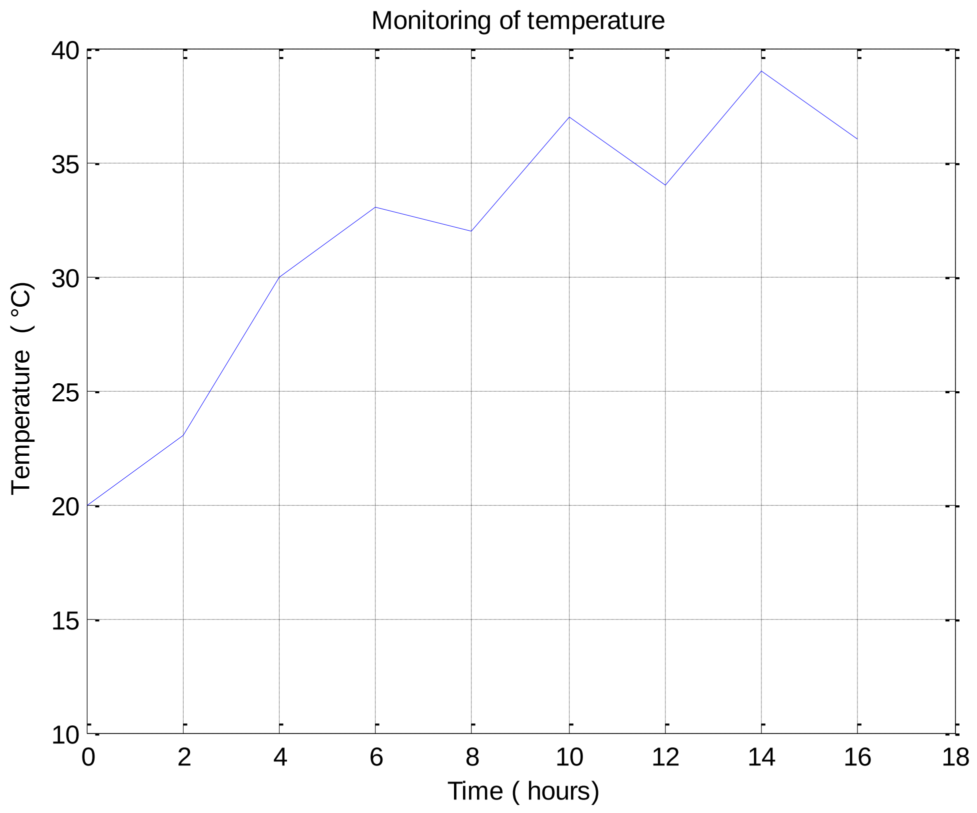


Fig. 13 – Graph of the Monitoring of temperature.

Example 4

$$y = \frac{t(t^2+1)}{10}$$

```
t = 0 : 0.001 : 2
y = t.*(1 + t.^3)./5
plot ( t , y ) ; grid on
```

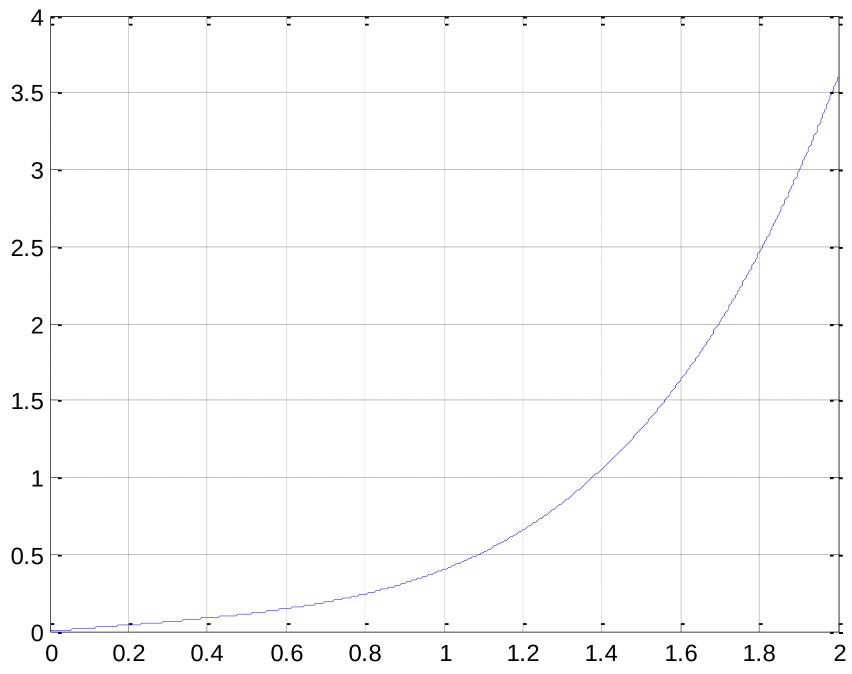


Fig. 14 – Graph of the function y.

Example 5

```
x = 4cos(2*t),y = sin(3*t)

 t = 0 : pi/200 : 2*pi
 x = 4*cos(2*t)
 y = sin(3*t)
 plot ( x , y )
 grid on
```

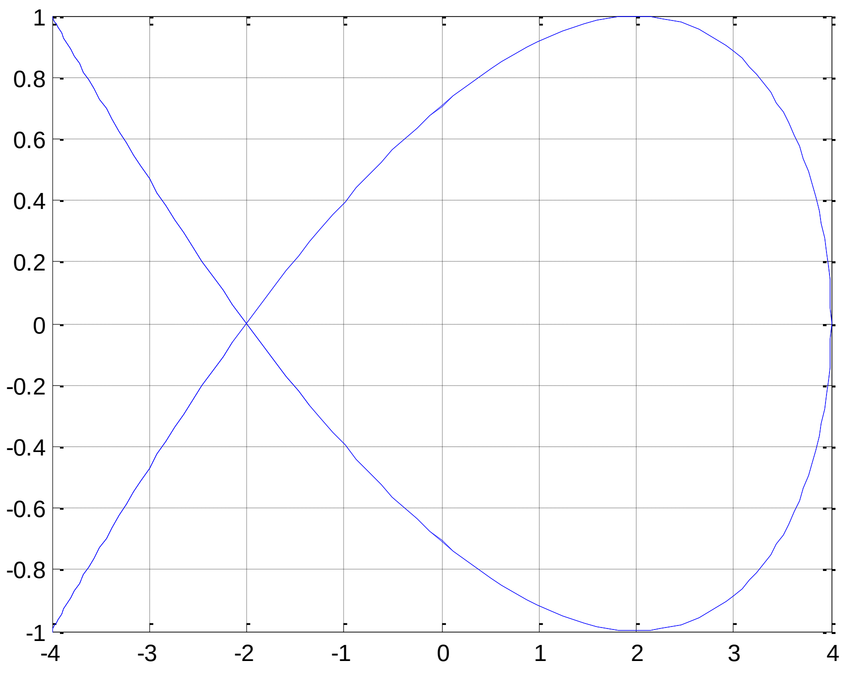


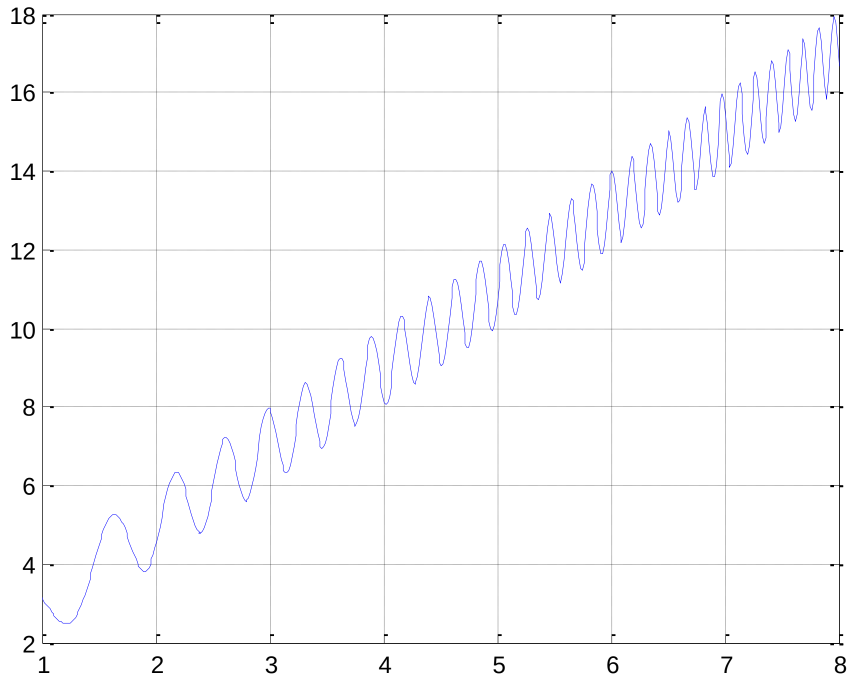
Fig. 15 – Graph of the function x and y.

Example 6

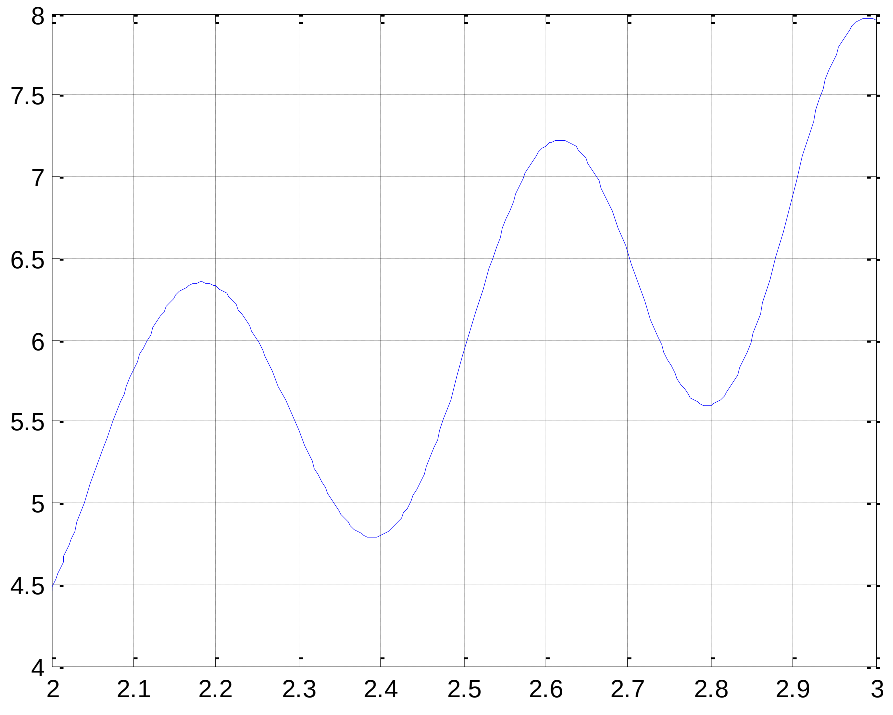
$$y = f(x) = 1 + 2x + \sin(3x^2)$$

• First method:

```
fplot('1+ 2*x + sin(3*x*x)', [ 1 8 ]),grid on
```



```
fplot('1+ 2*x + sin(3*x*x)', [ 2 3 4 9 11]),grid on
```



```
 grid off
 xlabel(axis of abscissa ')
 ylabel('axis of ordinates')
 title('y=f(x)')
```

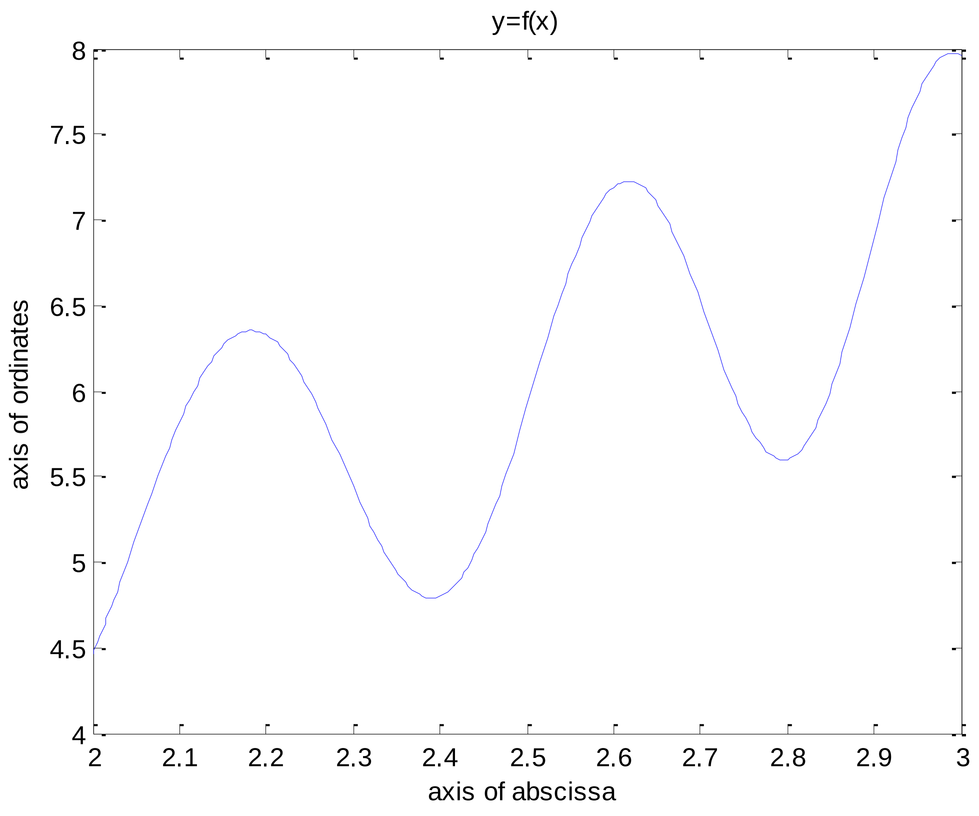
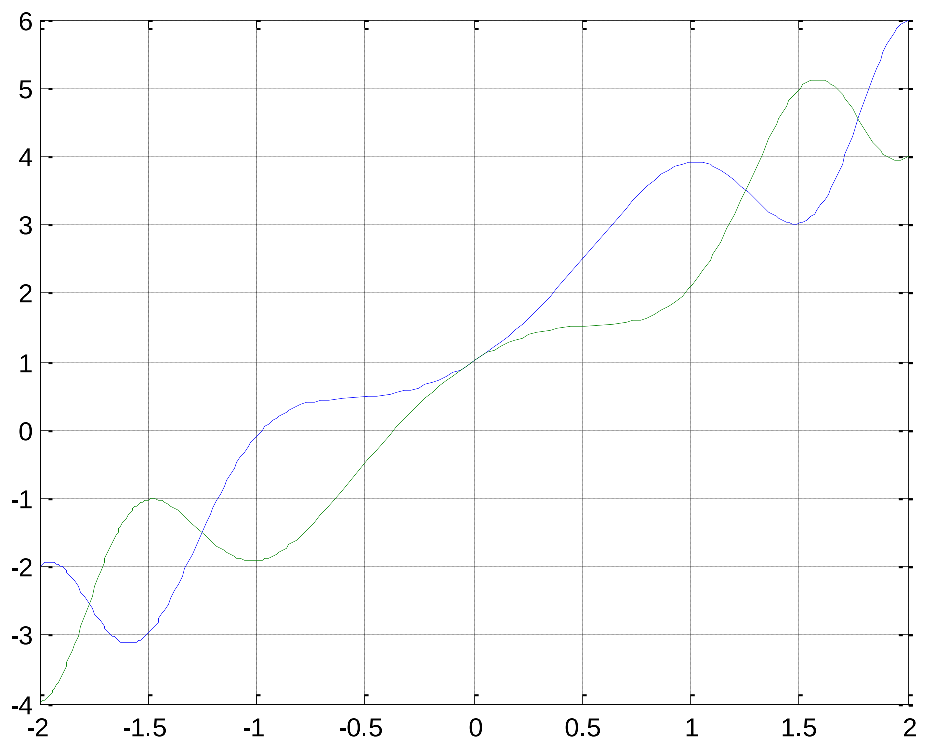


Fig. 16 – Graph of the function f(x).

```
zoom on

right click: zoom out

left click: zoom in

left click and drag: zoom of an area

zoom off
```

To draw several graphs in the same window:

```
fplot('[1+ 2*x + sin(2*x*x) , 1+ 2*x - sin(2*x*x) ]', [ -2 2 6
]),grid on
```



```
gtext('fonction 1')
gtext('fonction 2')
```

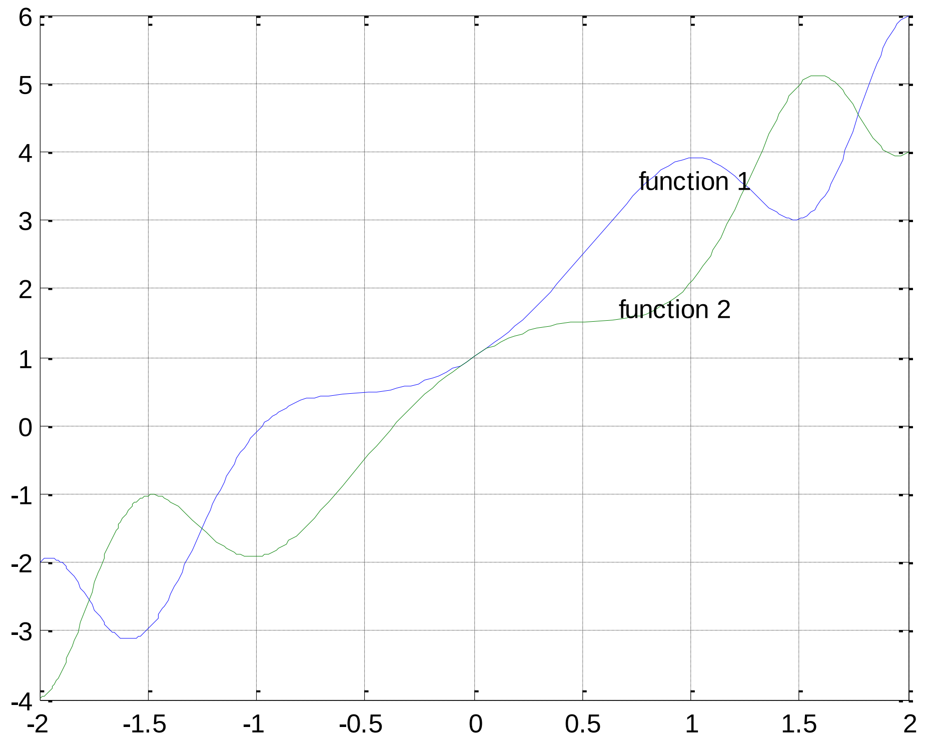


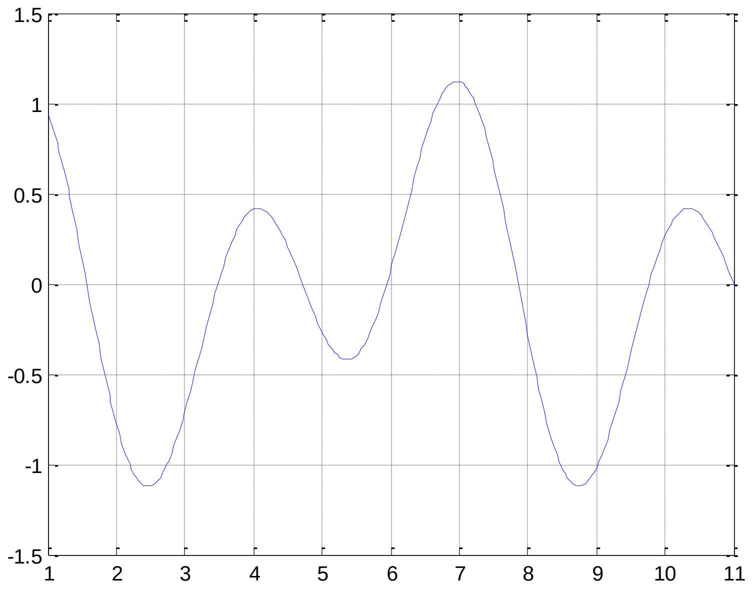
Fig. 16 – Graph of the function f1(x) end f2(x).

- **Second method**

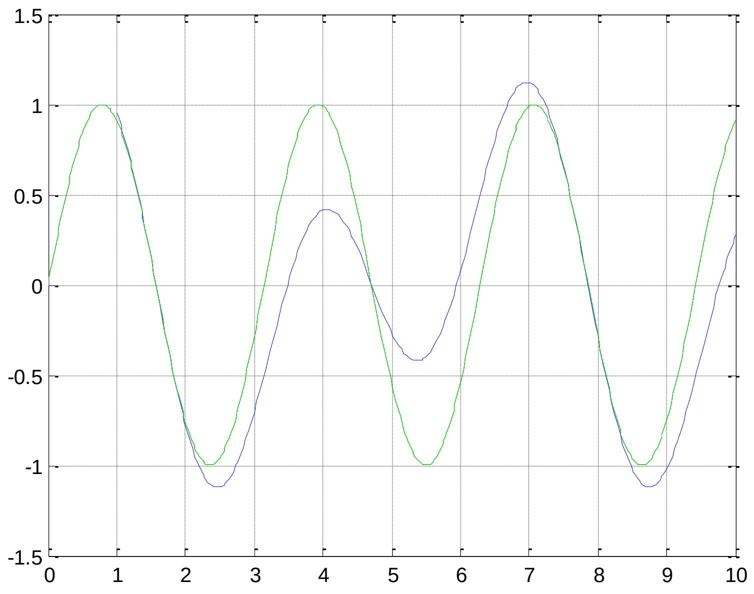
$$y = f(x) = \sin(2x) + \frac{\cos(x)}{2} - \frac{\sin(x)}{4}$$

We are going to create the function's .m file:

```
function y=f2(x)
y=sin(2*x)+0.5.*cos(x)-0.25.*sin(2*x);
>> fplot('f2', [ 1 11]),grid on
```

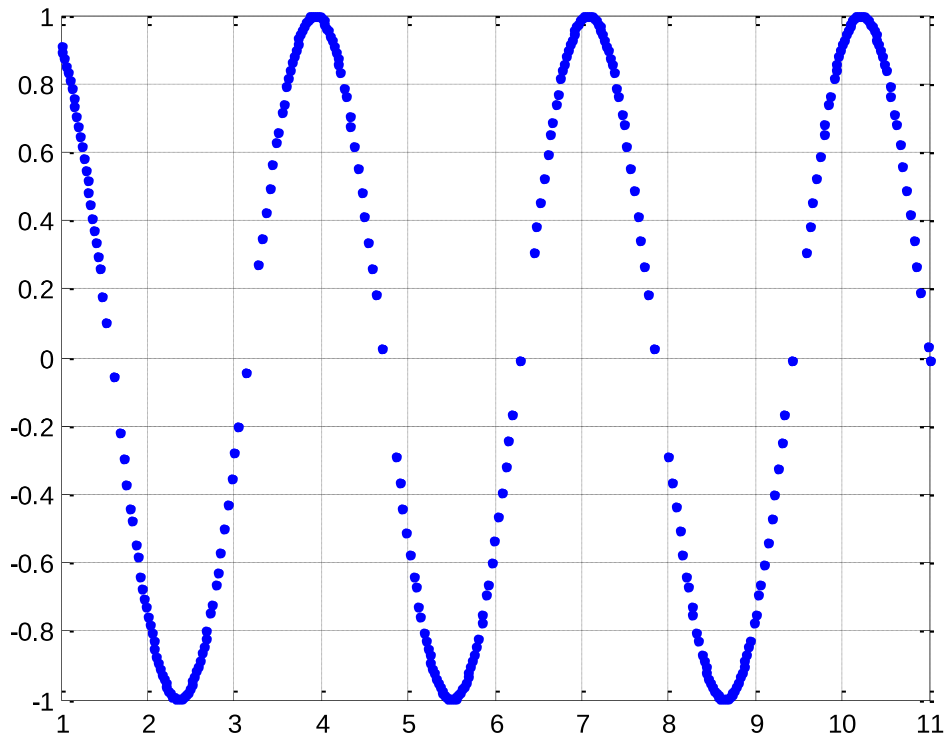


```
>> hold on
>> fplot('sin(2*x) ', [ 0 10 ] ,'g')
```



First part: Basic elements

```
>> [X Y] = fplot ( 'sin(2*x) ' , [ 1 11 ] )

X =
    1.0000
    1.0200
    1.0400
    1.0600
    .................
    10.9000
    10.9800
    11.0000
Y =
    0.9093
    0.8919
    0.8731
    .................
    0.1900
    0.0311
   -0.0089
>> fplot ( 'sin(2*x) ' , [ 1 11 ] , '.' ),grid on
```



1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1
2
3
4
5
6
7
8
9
10
11