Chapter 3: The graphics

Chapter 3: The graphics

3.1– Introduction

A 2D curve is represented by a series of abscissas and a series of ordinates. The software usually draws segments between these points. The MATLAB function is called **plot**.

The "help graph2d" instruction provides an introduction to 2D graphics and the table of available functions.

Elementary X-Y graphs:

plot : Linear plot.

loglog : Log-log scale plot.

semilogx : Semi-log scale plot.

semilogy : Semi-log scale plot.

polar : Polar coordinate plot.

plotyy : Graphs with y tick labels on the left and right.

This chapter presents the most important commands for developing the most common graphic representations. There are, however, a large number of commands and scripts for creating more sophisticated graphics.

3.2– The plot instruction

The simplest use of the **plot** instruction is as follows:

plot (abscissa vector, ordinate vector) or, plot ([x1 x2 ... xn], [y1 y2 ... yn])

The vectors can be either row or column, provided they are both of the same type. In general they are lines because the generation of lists of values seen previously provides line vectors by default.

Example:

If we want to plot sin(x) on the interval [0; 2π], we start by defining a series (of reasonable size, say 100) of equidistant values on this interval:

>> x = 0: 2*pi/100: 2*pi;

then, as the sin function can be applied term by term to an array:

>> plot(x, sin(x))

which, provides the graph opposite in the graphics window:

We see that the axes automatically adapt to the extreme values of the abscissa and ordinate.

By the way, all plot requires is an abscissa vector and an ordinate vector. The x-coordinates can therefore be a function of x rather than x itself. In other words, it is therefore possible to draw parameterized curves:

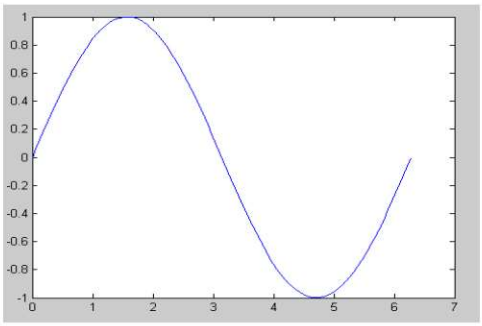
>> plot(cos(*x*), sin(*x*))

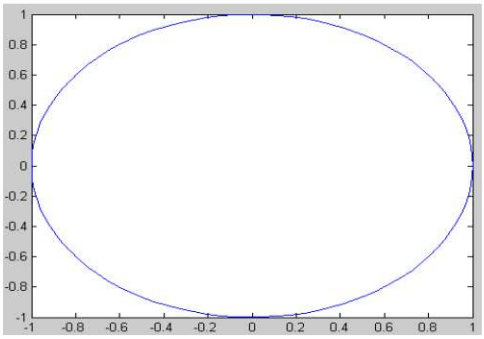
3.2.1– Superimpose several curves

To superimpose several curves, you can proceed in two ways:

-Specify as many pairs (abscissa, ordinate) as there are curves to trace in the plot command. For example to superimpose sin and cos:

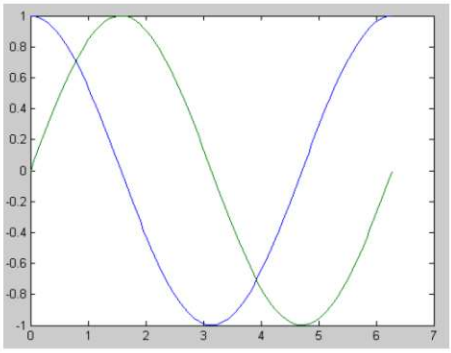
>> plot(*x*, cos(*x*),*x*, sin(*x*))





The two curves are actually in different colors. This method works even if the abscissa of the two curves are not the same.

-Or draw the first curve on a Plot figure (abscissa1, ordinate1, 'color1') and paste the other curves on it using the "Hold On" command, then draw one by one the Plot curves (abscissae2, ordinate2, 'color2'), Plot(abscissae3, ordinate3, 'color3'),… this technique requires specifying the color of the curve to be able to distinguish them.



3.2.2– Curve attributes

MATLAB assigns default colors to curves. It is possible to modify the color, the style of the line and that of the points, by specifying after each pair (abscissa, ordinate) a character string (between primes) which can contain the following codes (obtained by taping help plot) [8]:

Specifier	Line Style
-	solid line (default)
--	dashed line
:	dotted line
-.	dash-dot line

Specifier	Color
r	red
g	green
b	blue
c	cyan
m	magenta
y	yellow
k	black
w	white

Specifier	Marker Type
+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)

Codes can be combined with each other. For example:

>> plot(x,sin(x),':',x,cos(x),'r-.')

3.3– Decoration of graphics

3.3.1– Title

It is the "**title**" instruction to which you must provide a character string. The title appears at the top of the window chart:

>> plot(*x*,cos(*x*),*x*,sin(*x*))

>> title('Sin and cos functions')

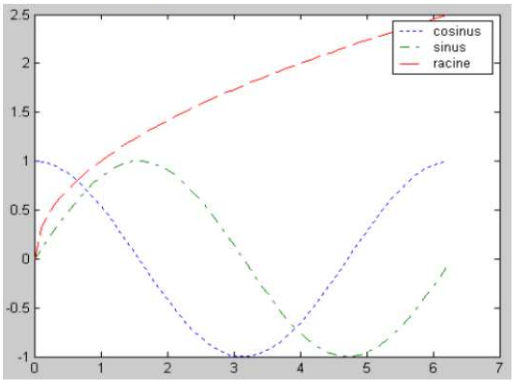
3.3.2– Labels

This involves displaying something below the x-axis and next to the y-axis:

>> plot(*x*,cos(*x*))

>> xlabel('Abscissa')

>> ylabel('Ordinated')



3.3.3– Legends

Using the legend statement. It is necessary to communicate as many character strings as curves drawn on the screen. A box is then drawn on the

graph, which displays opposite the style of each curve, the corresponding text. For example :

>> plot(x,cos(x), ':', x, sin(x), '-.', x, sqrt(x),'--')

>> legend('cosine', 'sine', 'root')

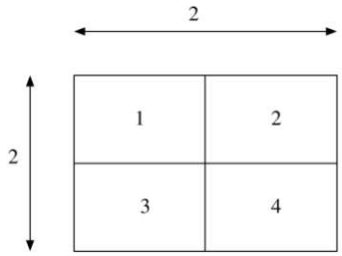
3.3.4– Drawing a grid

This is the "grid" instruction, which is used after a plot instruction displays a grid on the curve. If we type "grid" again, the grid disappears.

To have a fine grid, you must use the "grid minor" instruction.

3.4– Show multiple graphs with the (subplot) function

This is a very useful feature for presenting several results on the same graphic page. The general idea is to divide the graphics window into tiles of the same size, and to display a graph in each tile. We use the "subplot" instruction by specifying the number of tiles on the height, the number of tiles on the width, and the number of the tile in which we are going to plot:



subplot (Number of tiles on height, Number of tiles on width, Number of tiles)

The comma can be omitted. The blocks are numbered in the direction of reading a text:

from left to right and top to bottom:

Once a "subplot" command is typed, all subsequent graphics commands will be executed in the specified pad. Thus, the following graph was obtained from the sequence of instructions:

>> subplot(221)

>> plot(x,sin(x))

>> subplot(222)

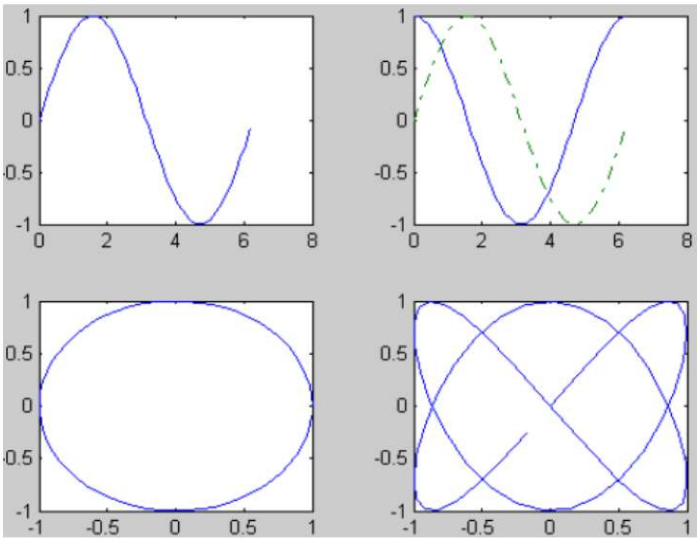
>> plot(x,cos(x),x,sin(x),'-.')

>> subplot(223)

>> plot(cos(x),sin(x))

>> subplot(224)

>> plot(sin(2*x),sin(3*x))



3.5– Axes and zoom

There are two ways to modify the extreme values on the axes, in other words to zoom on the curves. The simplest is to use the graphics window button. You can then:

- Frame an area to zoom with the left mouse button, or
- Click on a point with the left button. The clicked point will be the center of the zoom, the latter being carried out with an arbitrary factor,
- Click on a point with the right button to zoom out

To zoom more precisely, use the axis command (see the integrated help).

3.6– Clearing the graphics window

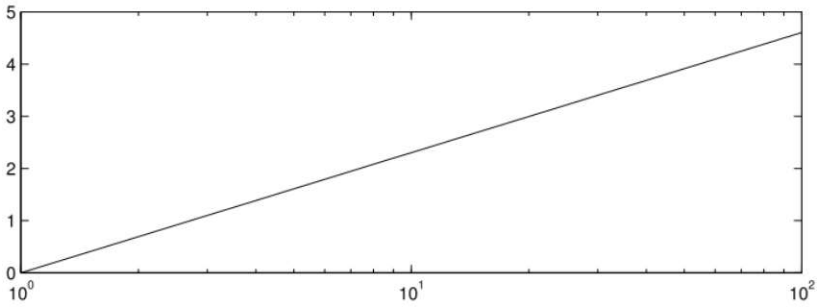
Just tap **clf**. This command also cancels any subplot and hold commands placed.

3.7– Entering a point with the mouse

The **ginput**(N) command allows you to click N points in the graphics window. The command then returns two vectors, one containing the abscissa, the other the ordinates. Used without the N parameter, the command loops until the "Enter" key is typed.

3.8– Logarithmic scales

We can plot log scales on the abscissa, on the ordinate or both. The corresponding functions are called **semilog*x*, semilog*y*** and **loglog**, respectively. They are used in exactly the same way as plot. For example:



>> *x*=1:100;

>> semilog*x*(*x*,log(*x*))

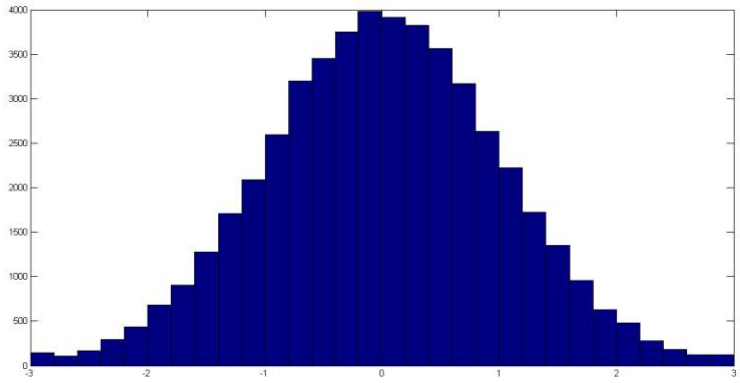
gives the curve opposite.

3.9– histograms: hist

>>*x*=-2.9: 0.2: 2.9; % Sets the interval and width of the histogram channels.

>>*y*=**randn**(50000,1); %Generates random numbers distributed according to a normal distribution

>> **hist**(*y,x*) % Draw the histogram of the figure below



3.10– Summary quote of some other graphic functions

- axis([xmin xmax ymin ymax],'option'); option: square, fill, tight, image...

xlim([xmin xmax]);

xlim('option'); option = mode, auto, manual

- **grid on**: allows you to add a grid

- **grid off**: hide the grid in the graph

- **clf** : clear all graphs in the current window or clear the graph (fig) mentioned clf('reset') or clf(fig)

-**close**: delete the figure(s) close fig or close all

-**refresh** : redraws the current figure refresh or refresh(fig)

-**cla** : delete current axes all its graphic objects **cla reset** or **cla(ax)**

3.11– 3D graphics and animations

3.11.1– 3D curves

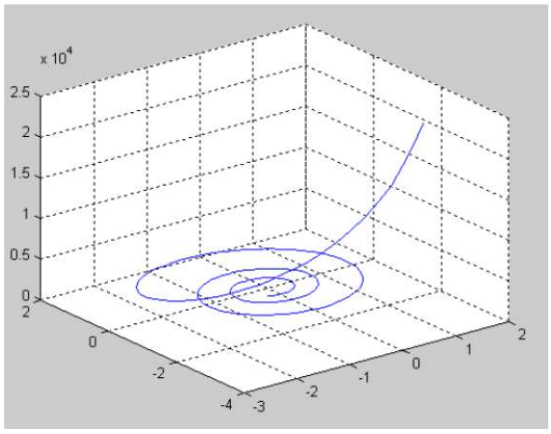
A 2D curve is defined by a list of doublets (*x; y*) and a 3D curve by a list of triplets (*x; y; z*). Since the plot instruction expected two arguments, one for *x*, one for *y*, the **plot3** instruction expects three: *x, y and z*.

3.11.1.1– Parameterized curve (figure opposite)

```
>> t=-10:0.1:10;
>> plot3(exp(-t/10).*sin(t), exp(-t/10).*cos(t), exp(-t))
>> grid
```

For more details:

>>help graph3d % introduction to 3D graphics and table of available functions



3.11.1.2– Generation of points (meshgrid)

To define a surface, you need a set of triples (x; y; z). In general the points (x; y) trace a regular mesh in the plane but this is not an obligation. The only constraint is that the number of points is the product of two integers m×n (we will understand why very soon).

If we have a total of m×n points, this means that we have m×n values of x, m×n values of *y* and m×n values of *z*. It therefore appears that abscissa, ordinate and dimensions of the points on the surface can be stored in tables of size m×n.

All surface drawing instructions, for example surf, will therefore have the following syntax:

surf (abscissa table, ordinate table, dimensions table)

$$\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \vdots & x_{2n} \\ \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} \quad \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ y_{21} & \vdots & y_{2n} \\ \vdots & \vdots & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix} \quad \begin{bmatrix} z_{11} & \cdots & z_{1n} \\ z_{21} & \vdots & z_{2n} \\ \vdots & \vdots & \vdots \\ z_{m1} & \cdots & z_{mn} \end{bmatrix}$$

It now remains to construct these tables. Let us take the example of the surface defined by $z = x2 + y2$ for which we want to trace the representative surface on the plane [-1; 1] × [-2; 2].

To define a grid of this rectangle, you must first define a series of values x1; ... ; xm for x and a sequence of values y1; ... ; yn for y, for example:

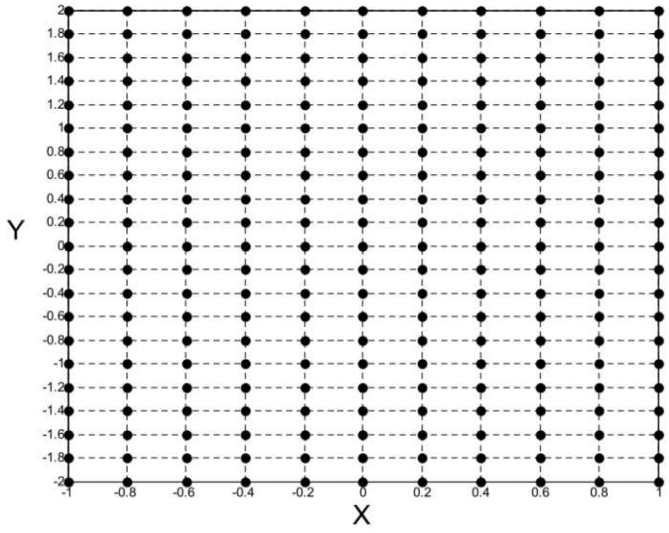
>> *x* = -1:0.2:1

>> *y* = -2:0.2:2

Combining all these values of x and y, we obtain m×n points in the (x; y) plane. We must now construct two tables, one containing the 11×21 abscissa of these points and the other the 11×21 ordinates, i.e.:

$$X = \begin{bmatrix} -1 & -0.8 & \cdots & 1 \\ -1 & -0.8 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -1 & -0.8 & \cdots & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} -2 & -2 & \cdots & -2 \\ -1.8 & -1.8 & \cdots & -1.8 \\ \vdots & \vdots & \vdots & \vdots \\ 2 & 2 & \cdots & 2 \end{bmatrix}$$

Thus, if we consider for example the first column of these matrices, we obtain all the pairs (*x, y*) for *x*=-1: (-1, -2), (-1, -1.8), (-1, -1.6), (-1, -1.4), ..., (-1, 1.8) and (-1, 2). The other columns cover the rest of the portion of the *xy* plane. Below, the corresponding mesh:

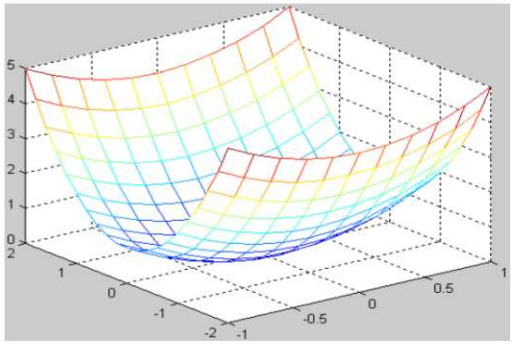


Don't worry, no need for loops, the meshgrid function is responsible for generating these 2 matrices:

>> [X,Y] = meshgrid(x, y);

It now remains to calculate the corresponding z = f(x, y). This is where term-to-term calculations on matrices still show their effectiveness: we apply the formula directly to tables X and Y, without forgetting to put a point in front of the operators *, / and ^

>> Z = X.^2 + Y.^2;



Then we can use all the surface drawing functions, for example mesh:

>> mesh(X,Y,Z)

The most common instructions are:

- mesh which draws a series of lines between points on the surface
- meshc which works like mesh but adds the contour lines in the plane (x; y)
- **surf** which "paints" the surface with a color depending on the coast
- **surfl** which "paints" the surface as if it were illuminated.
- **surfc** which works like mesh but adds the contour lines in the (x, y) plane

Finally, note that there are conversion functions between Cartesian, cylindrical and spherical coordinates, making it possible to easily draw curves defined in one of these coordinate systems. For example, we will look at the cart2pol documentation.

Example :

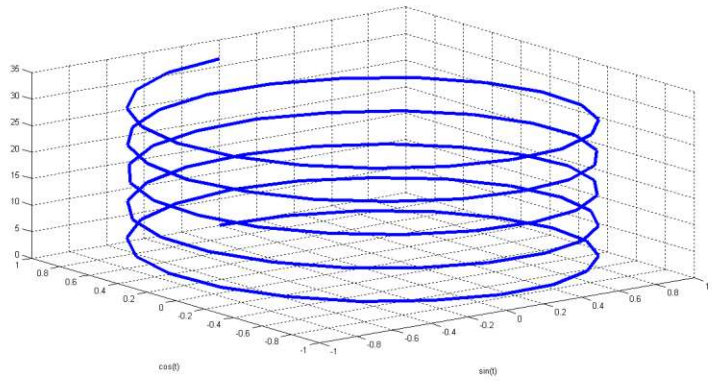
has. line in space:

>> t = linspace(0, 10*pi);

>> plot3(sin(t), cos(t), t)

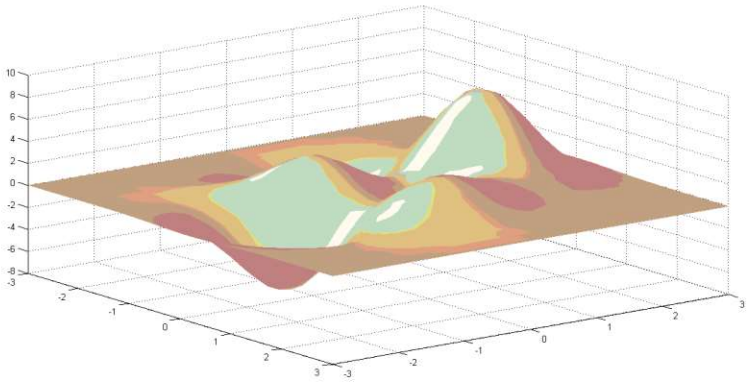
>> xlabel('sin(t)'), ylabel('cos(t)'), zlabel('t')

>> grid on % Figure below

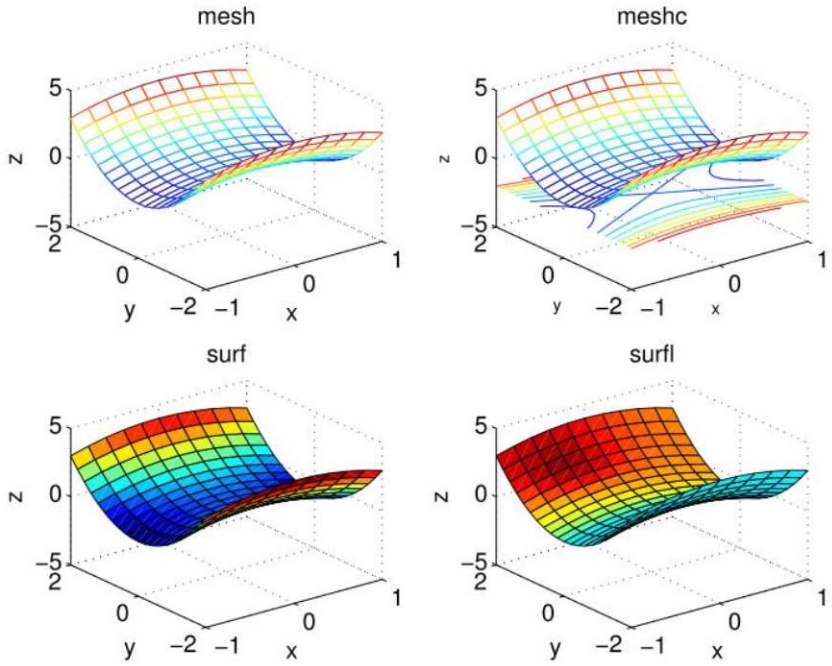


3.11.1.3– Surface with illumination: surfl

>> clf
>> [X,Y,Z] = peaks(30);
>> surfl(X,Y,Z) % graph with illumination
>> shading interp % best interpolation
>> colormap pink %choice of a predefined color palette
>> view(-37.5+90, 30) % change point of view: view(azimuth, elevation)



3.11.1.4– Differences between mesh, meshc, surf and surfl instructions



3.11.1.5– Colormaps

The colormap instruction defines the sequence of colors, or “palette”, for painting surfaces. You can give color tables directly or use predefined palettes (for example 'bone', 'winter', 'hot', etc.). Refer to the documentation for more information.

3.11.1.6– Interpolation

You will have noticed that MATLAB plots surfaces with facets. It is possible to smooth these surfaces with the shading instruction. We can test for example, following an instruction on fl (this is the most spectacular) see the figure in example 4.b:

>> shading interp

3.11.1.7– Function Usage

plot3: Plotting a parametric line in 3D

mesh: Trace of a surface in 3D, from mesh matrices

meshgrid: Defining mesh matrices from two vectors

surf: Drawing of a 3D surface with color gradient, from mesh matrices

surfc: Trace of a 3D surface with color gradient and iso-value lines

ezmesh ,ezmeshc: Easy surface plotting (mesh matrices set by default)

ezsurf, ezsurfc: Easy surface drawing with color gradient (mesh matrices defined by default)

sphere: Definition of mesh matrices for drawing a sphere

cylinder: Definition of mesh matrices for drawing a cylinder

3.12– Animations in Matlab

MATLAB treats an animation as a succession of images whose scrolling speed can be controlled.

Creating an animation begins by reserving the memory space necessary for the different images or frames.

The moviein command does this.

The command syntax is:

Anim = moviein(n)

This command initializes the memory for recording an animation. An animation matrix is created to be able to store n images. Each column of the matrix corresponds to an image of the animation.

To generate an animation whose images contain the entire graphics window and not just a graphic, we will use the following syntax:

Anim = moviein(n,gcf)

An image is an instantaneous photo, the animation is therefore a series of instantaneous images. Images are captured using the commands below:

getframe(h): allows you to capture an image of the graphic object whose pointer or handle is h (root, figure or axis)

getframe(h,rect): allows you to capture an image with specification of a rectangle as capture area, the rectangle is indicated by its coordinates in units and in relation to the lower left corner of the graphics window.

Movie(n): allows the display of the animation,

More generally, you can create an animation in two different ways:

• Either by saving a certain number of figures and scrolling through them later like a film,

• Or by erasing and redrawing the objects on the screen as you go.

Create a movie using the movie and getframe commands

Example:

```
>>for i=1:5
>>plot(sin(i*pi*[0:0.025:2]));
>>M(:,i) = getframe;
>>end;
>>movie(M);
```

Create an animation using the **hold on** and **pause** commands

• When two plot commands follow one another, the second erases the first on the figure. The hold on command keeps the result of the two plot commands in the figure.

• The **pause** command allows you to stop temporarily a program, in particular between two plot commands. To restart the program, simply press any key on the keyboard.

• The pause(x) command allows you to pause for a duration fixed by x in a program. The bigger x is, the longer the pause!

Example:

```
>>y=0:0.1:2;
>>plot(y,sin(y*pi),'-r');
>>hold on;
>>for x=0:0.1:2
plot(x,sin(x*pi),'*');
break(1);
end;
```

```
>>y=0:0.1:2;
>>for x=0:0.1:2
plot(y,sin(y*pi),'-r',x,sin(x*pi),'*');
break(0.5);
end;
```

3.12.1– Import/export animations

It is possible to export a Matlab movie to an MPEG file using mpgwrite, or movie2avi, commands found on the Mathworks website. Likewise, it is possible to load an MPEG file into a Matlab movie with mpgread.