Chapter 3
Plotting Data[1]

MATLAB can display data in high-quality graphs. There are many built-in functions for creating scatter plots, 2D and 3D bar graphs, pies charts, line graphs, etc. MATLAB makes it possible to control each characteristic of a graphical object, so that the resulting graph shows exactly what you want to show in the way you want to present it.

Plot Data

Data can often be better understood if they are represented in a graphical format rather than in a numerical format. MATLAB is a powerful tool for plotting data, either in 2D or 3D form. Graphs can be created, edited, and saved for later modification or exported as graphics files.

The simplest way to draw a graph is to use the `plot` function. The following code creates a plot of the sine function:

```
>> x=[0:0.2:7];
>> y=sin(x);
>> plot(x,y);
```

If no other figure is open, the window is automatically named *Figure 1,* and in Fig. 3.1 it is reported how it looks like.

MATLAB assigns a progressive number to the figures, so that it is easy to refer to the different figures. The figure window contains buttons, such as the zoom in/zoom out buttons, and above all, the edit button. This button allows you to select the graph and edit its characteristics such as its color and width. The edit button provides a graphical and intuitive way to modify the graph. However, it is often better

[1] Note that, although the book figures are black and white, the commands reported in the current chapter generate color figures.

M. Borgo et al., *MATLAB for Psychologists*,
DOI 10.1007/978-1-4614-2197-9_3, © Springer Science+Business Media, LLC 2012

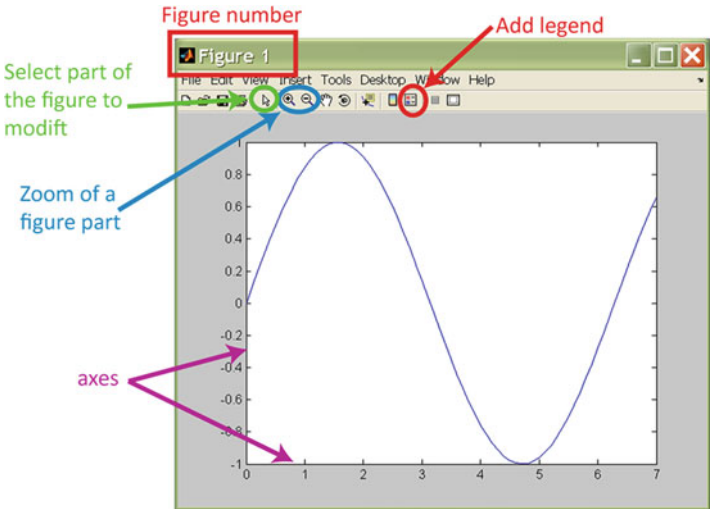
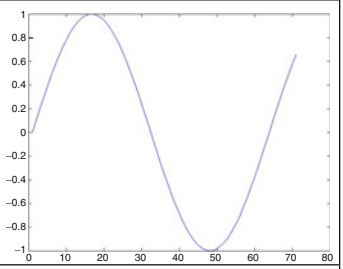
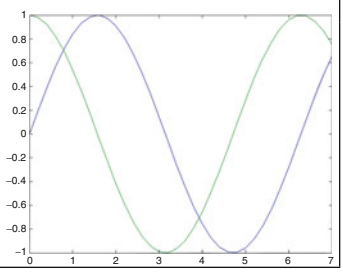


Fig. 3.1 Layout of the plot function

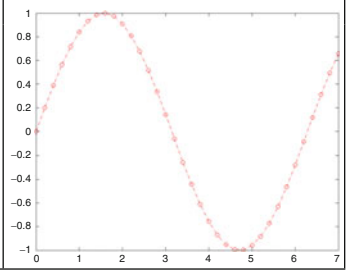
to modify a figure's appearance using MATLAB code. This is because once the code is saved, it can be used again to draw similar graphs with different data.

The main function for plotting 2D figures is `plot`, and it takes many arguments. The following table shows some examples. Before running the examples, make sure you have saved in the workspace the x and y variables as implemented above.

Syntax and description	Example	Graphical result
`plot(Y)` Plot the columns of Y versus the Y vector indexes	`Plot(y)`	
`plot(X1,Y1,…)` Plot all lines defined by the Xn-Yn pairs	`plot(x,y,x,cos(x))`	

(continued)

(continued)

Syntax and description	Example	Graphical result
`plot(X1,Y1,LineSpec,…)` Plot all lines defined by the Xn- Yn, LineSpec triplets. LineSpec is a string specifying the line type, marker symbol, and color of the plotted lines	`plot(x,y,'-.ro')`	

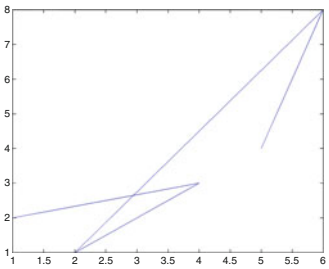
Let us see in more detail how to use the LineSpec string, which is the argument specifying the property of the line drawn in the plot. The following table lists all the LineSpec options.

Color				Marker				Line style	
Charact.	Description	Charact.	Description	Charact.	Description	Charact.	Description	Charact.	Description
`B`	`Blue`	`m`	`Magenta`	`.`	`Point`	`*`	`Star`	`-`	`Solid`
`G`	`Green`	`y`	`Yellow`	`o`	`Circle`	`s`	`Square`	`:`	`Dotted`
`R`	`Red`	`k`	`Black`	`x`	`x-mark`	`d`	`Diamond`	`-.`	`Dashdot`

`C`	`Cyan`			`+`	`Plus`	`V`
						`^`
						`<`
						`>`
							`Triangle (down)`
							`Triangle (up)`
							`Triangle (left)`
							`Triangle (right)`
								`--`	`Dashed`

Note that `plot()` generates a graph that connects the x-y coordinates written in the input vectors with a line. Therefore, the graphic result is not necessarily a continuous graph. In the first example the sinusoidal curve appears continuous because the x points were very close to each other; however, the following example shows that the line can be fragmented if the coordinates are not contiguous:

Example	Graphical result	Description

`>> x=[1,4,2,6,5];`
`>> y=[2,3,1,8,4];`
`>> plot(x,y);`
		The first point coordinates are 1 and 2; it is connected to the second point, having as coordinates 4 and 3. Then this second point is connected to a third one having coordinates 2 and 1, and so on

It is possible to add curves to the same figure by freezing the figure. This can be done using the `hold on` command. Once `hold on` has been activated, the plot commands add curves to the same figure. For example, the following lines add one curve to another one:

```
>> plot(x,y)
>> hold on;
>> plot(x,cos(x));
```

The same result can be obtained by means of the following complex line:

```
plot(x,y,x,cos(x));
```

MATLAB also has a function enabling plotting two curves with different y-axes within the same figure. This function is called `plotyy`:

```
>> plotyy(x,sin(x),x, 2*cos(x));
```

Here, the cosine is multiplied by 2. If you plot the graph, you will see that the left y-axis ranges from −1 to +1, whereas the right y axis ranges from −2 to +2.

Control the Plot's Objects: Labels, Legend, Title…

When a `plot` is made, the axes are automatically scaled to include the minimum and the maximum values. However, it is possible to customize the axes limits using `axis([xmin, xmax, ymin, ymax])`. For example, type:

```
>> axis([1, 6, -1.5, 1.5]);
```

The command `axis` has other specifications, such as the command `axis equal`, which makes unit increments along the *x*- and *y*-axes of the same length. If you do not need the axes, you can turn them off using the command `axis off`. To turn the axis on again, use the command `axis on`.

There are other useful functions enabling the customization of the plots: it is possible to add labels, titles, legend, text, and so on. The most frequently used of these functions are listed in the following table:

Function	Description	Examples

`xlabel(STR)`
`ylabel(STR)`
`zlabel(STR)`
`xlabel(STR,'FontSize',fs)`
`ylabel(STR,'FontSize',fs)`
`zlabel(STR,'FontSize',fs)`
	Add the content of the string STR along the X, Y, Z axes. The font size is set by the `fs` value if the property `'FontSize'` is specified	`xlabel('time')`
		`ylabel('record level')`
		`xlabel('time','FontSize',16)`

`title(STR)` `title(STR,'FontSize',fs)`	Add the content of the string STR at the top of the graph. The font size is set by `fs` if the property `'FontSize'` is specified	`title('Sine experiment)` `title('Sine experiment' 'FontSize',18)`

(continued)

(continued)

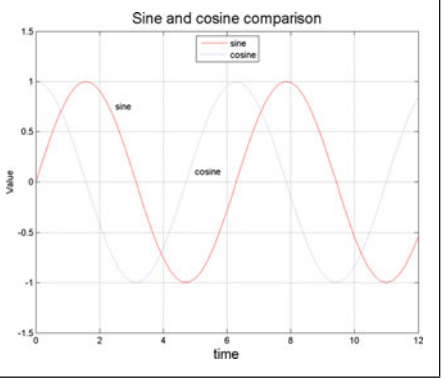
Function	Description	Examples
`text(posx,posy,STR)`	Write the content of the string STR at the point specified by `posx` and `posy`. Note that `posx` and `posy` relate to the current axis	`text(2,3,'very nice exp!');`
`gtext(STR)`	Write the content of the string STR in the graphics window at the position pointed to by the mouse	`gtext('A comparison');`

`legend(S1, S2,... SN)`
`legend(S1,S2,...`
`'location',LOC)`
	Add a legend to the graph using the specified strings as labels. The association of the strings with the curves follows the order in which the curves were plotted
	Sometimes it is useful to specify the legend position with respect to the axes. This can be done using the specification `'Location'` and then a string LOC such as `'Best'`
	For further details see the help
		`legend('sine','cosine')`
		`legend('sin','cos',...`
		`'location','Best')`

`Grid`	Add/remove the grid to/from the current graph	`grid;`

Let's plot a sine and a cosine function ranging from 0 to 12 with step 0.2; then we shall add a legend, a title, and labels:

Example	Graphical result

`>> x=[0:0.2:12]; y1=sin(x);`
`y2=cos(x);`
`>> plot(x,y1,'r',x,y2,'b:');`
`>> xlabel('time','FontSize',16);`
`>> ylabel('Value');`
`>> axis([0,12,-1.5,1.5]);`
`>> grid;`
`>> legend('sine','cosine',`
`'location',…`
`'North');`
`>> title('Sine and cosine`
`comparison',… 'FontSize',18);`
`>> text(2.5,0.75,'sine');`
`>> text(5,0.1,'cosine');`
	

If you want to close a figure, you use `close` followed by the figure number. Alternatively, if you want to close all the figures, you use `close all`.

To open an empty window in which to add graphical data, use the command `figure`. It is a good idea to open a new window before creating a plot. If you don't, every time you plot new data, the new data are plotted on the existing figure, overwriting the previous data.

Subplot: Multiple Plots in One Figure

It is possible to display multiple plots within the same figure using `subplot(Nrows,Ncolumns,CurSub)`. This function divides the figure into `Nrows` rows and `Ncolumns` columns. Each part of the figure is identified by the `CurSub` value; this value increases by row (top to bottom) and from left to right. For example, if your figure has four subplots, two at the top and two at the bottom, number 1 identifies the top-left graph, number 2 identifies the top-right one, number 3 the bottom-left, and number 4 the bottom-right graph, as in Fig. 3.2.

Let's suppose we want two rows and three columns of subplots. We want to plot in row 1, column 3 a sine wave ranging from 0 to 6. Moreover, we want to plot in row 2, column 2 a cosine wave ranging from 6 to 9. This is how it can be done:

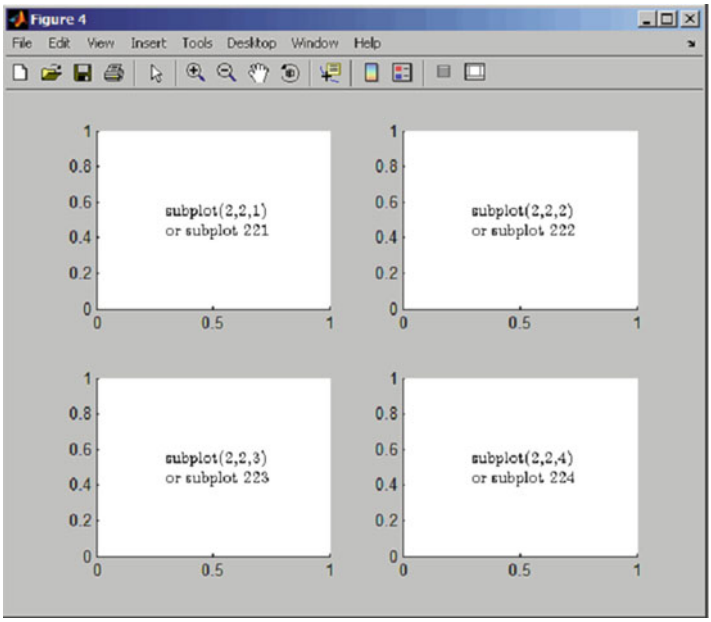
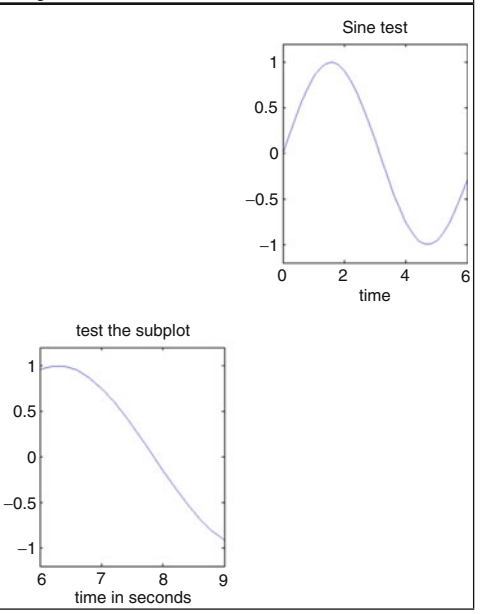


Fig. 3.2 Subplot layout

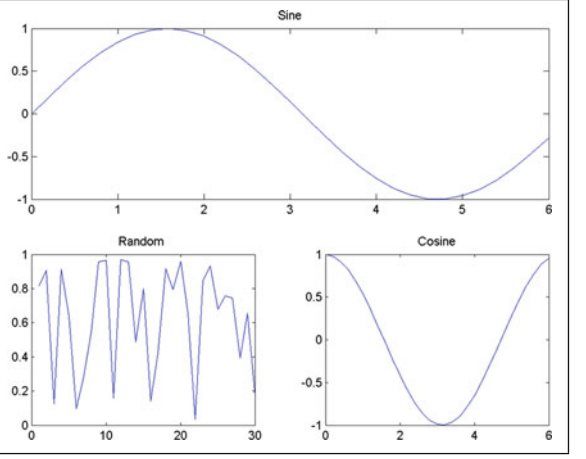
Example	Graphical result

`>> xs=[0:0.2:6];`
`>> xc=[6:0.2:9];`
`>> figure;`
`>> subplot(2,3,3);`
`>> plot(xs,sin(xs));`
`>> xlabel('time');`
`>> title('Sine test');`
`>> axis([0 6 -1.2 1.2]);`
`>> subplot(2,3,5);`
`>> plot(xc,cos(xc));`
`>> xlabel('time in seconds');`
`>> title('test the subplot');`
`>> axis([6 9 -1.2 1.2]);`
	

Each time you use `subplot`, all the commands you write refer to the part of figure identified by `CurSub`. A common mistake is to use two consecutive subplot commands with different numbers of rows and columns. If you do so, everything you have plotted after the first subplot command is overwritten by the second one.

The subplots within a figure do not need to be all of the same size. For example, you can have one plot that extends over the space of two (or more) subplots. To do this, pass a vector to `CurSub`. The following table shows how to divide the figure into three parts, in which the length of one graph occupies the same amount of space as the other two combined.

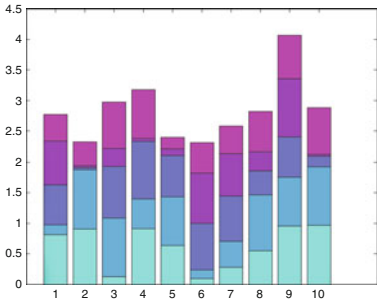
Example	Graphical result

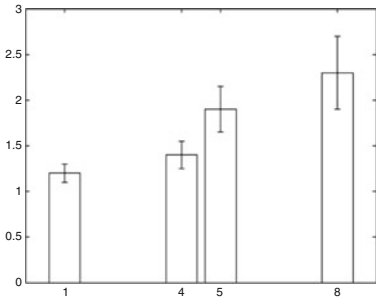
`>> xs=[0:0.2:6];`
`>> figure;`
`>> subplot(2,2,[1 2]);`
`>> plot(xs,sin(xs));`
`>> title('Sine');`
`>> subplot(2,2,4);`
`>> plot(xs,cos(xs));`
`>> title('Cosine');`
`>> subplot(2,2,3);`
`>> plot(rand(1,30));`
`>> title('Random ');`
	

As can be seen from the table, `CurSub` is a vector telling MATLAB that the first subplot spans from position 1 to position 2.

Although `plot` is a useful function, MATLAB is provided with additional functions to represent data. The following table lists the most common types of graphs together with the functions used to draw them. Since we do not have data here, we have used the useful `rand` function to create them. The command `rand(r,c)` creates a, r by c matrix with uniformly distributed random numbers. There is also `randn(r,c)`, which creates an r by c matrix with normal Gaussian distributed random numbers.[2]

Function	Description	Examples

`bar(X,Y,W)`
`barh(X,Y,W)`
	Draw the columns of the M by N matrix Y as M groups of N vertical bars. X gives the position of the values in Y. If X is omitted, the default value of X=1:M is used. W specifies the width of the bars (0.8 by default)
	`bar(…,'stacked')` produces a vertical stacked bar chart
	`barh` is the same as bar but plots the bars horizontally
		`>> bar(rand(10,5),'stacked');`
		`>> colormap(cool);`
		 

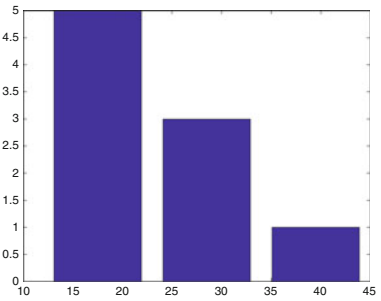
`errorbar(X,Y,Err,'LSp')`	Plot Y versus X with error bars [Y−Err Y+Err]. Here `'LSp'` is a string specifying the line style as for the `plot` function and can be omitted	`x=[1,4,5,8];`
		`RT=[1.2,1.4,1.9,2.3];`
		`SD=[0.1,0.15,0.25,0.4];`
		`bar(x,RT,'w'); hold on;`
		`errorbar(x,RT,SD,'.k');`
		 

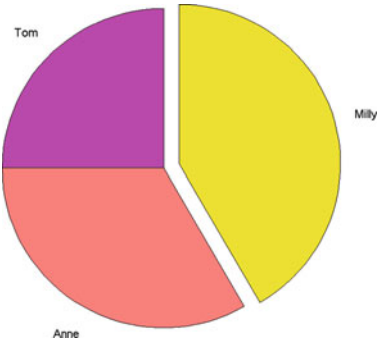
(continued)

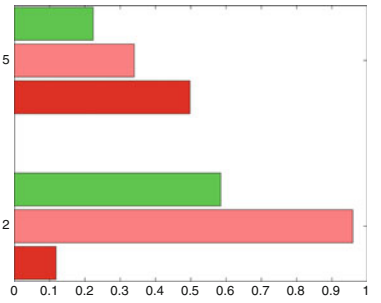
[2]The functions `rand` and `randn` will be documented more fully in the chapter on statistical analysis.

(continued)

Function	Description	Examples

`[Nel,xc]=hist(y);`
`[Nel,xc]=hist(y,Ndiv);`
	`[N,xc]=hist(y)` places the elements of `y` into ten equally spaced containers and returns in `Nel` the number of elements in each container. In `xc` can be found the positions of the bin centers
	Once we obtain the number of events of each bin and the bin center using hist, the distibution of events can be plotted using `bar` or `plot`
		`Ages=[22,25,23,22,45,12,34,33,21];`
		`[N,xc]=hist(Ages,3);`
		`bar(xc,N);`
		 

`pie(X)`
`pie(X,explode)`
`pie(…,labels)`
	PIE draws a pie chart of the normalized data in the vector X. `explode` is a (logical) vector of the same size as X, specifying which slices have to be pulled out from the pie. The cell array `labels` contains strings. The number of cells must be equal to the size of X
	`explode` and `labels` can be omitted
		`x=[2 4 6 3 5];`
		`pie(x,[0 0 1],…`
		`{'Tom','Anne','Milly'});`
		`colormap(spring);`
		 

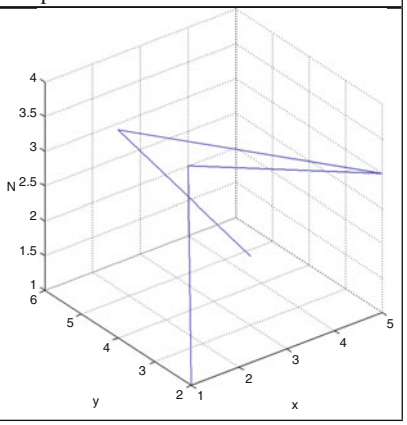
`colormap(CM);`
`colormap(srt);`
	Set the color lookup table. You can set up a color map matrix CM on your own. CM may have any number of rows, but it must have exactly three columns (the RGB color combination; see Chap. 5). In any case, you can use the predefined `colormap` (e.g., `str='cool'`) or create one using the function `colormapeditor`
	For more details on `colormap` refer to Chap. 5
		`x=[2,5]; y= rand(2,3);`
		`barh(x,y, 0.9);`
		`colormap([1 0 0; 1 0.5 0.5; 0,1,0]);`
		 

3-D Plots

MATLAB can also handle 3-D plots, including lines and various types or surfaces. For a 3-D plot you need three dimensions; therefore, you need to pass to the function data points with three coordinates. The basic command is `plot3`. It works like `plot`, except that it takes three vectors instead of two, one for the *x*-coordinate, one for the *y*-coordinate, and one for the *z*-coordinate.

Here we report an example of a 3-D a line connecting five points:

Example	Graphical result

`>> x=[1,4,5,1,3];`
`>> y=[2,6,2,4,3];`
`>> z=[1,2,3,4,2];`
`>> plot3(x,y,n);`
`>> axis square;`
`>> grid on;`
`>> xlabel('X');`
`>> ylabel('Y');`
`>> zlabel('N');`
	

In the example, the first point has coordinate $x = 1$, $y = 2$, and $z = 1$. Note that you can add a label for the *z*-axes using the command `zlabel`.

MATLAB can also display 3-D surfaces using the commands `mesh` and `surf`: the `mesh` function gives a transparent "mesh" surface, whereas `surf` gives an opaque shaded surface.

Usually a 3-D surface is a set of `z` values associated to a set of `(x,y)` coordinates. For example if we have a set of coordinates

$$(x,y) = \begin{pmatrix} 1,1 & 1,2 & 1,3 & 1,4 \\ 2,1 & 2,2 & 2,3 & 2,4 \\ 3,1 & 3,2 & 3,3 & 3,4 \\ 4,1 & 4,2 & 4,3 & 4,4 \end{pmatrix},$$

the (x, y) pairs can be split into two matrices:

$$x = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix} \qquad y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

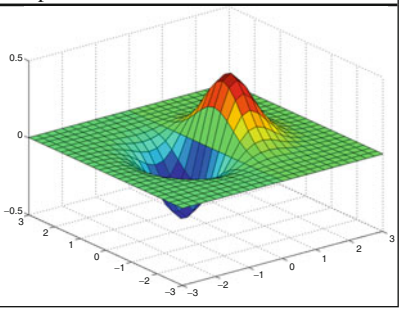
MATLAB provides a function called `meshgrid` that can be used to simplify the generation of `x` and `y` matrix arrays used in 3-D plots. It is invoked using the form

`[X,Y]=meshgrid(a,b)`, where `a` and `b` are vectors that specify the region in which the coordinates, defined by element pairs of the matrices `x` and `y`, will lie. To obtain the `x`, `y` matrix you can do as follows:

```
>> a=[1:4];
>> b=[1:4];
>> [x,y]=meshgrid(a,b)
x =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
y =
     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4
```

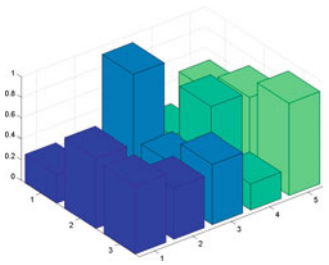
Now it is simple to obtain the `z=f(x,y)` values as a function of each `(x,y)` pair as shown in the following example:

Example	Graphical result

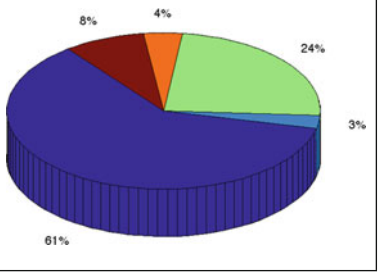
`>> a=[-3:0.25:3];`
`>> b=[-3:0.25:3];`
`>> [X,Y]=meshgrid(a,b);`
`>> Z= X.*exp(-X.^2-Y.^2);`
`>> surf(X,Y,Z);`
	

MATLAB has equivalent 3-D functions to obtain 3-D bar graphs and 3-D pie charts. Here we show an example of these functions. For further details, please refer to online help.

```
>> y=rand(3,5);
>> bar3(y);
>> colormap(winter);
```



```
>> y=rand(5,1);
>> pie3(y);
>> axis square; grid off;
```



Printing and Saving Images

Figures can be saved or printed by means of the `print` function. The structure of the function is `print -dformat fileName -options`, where `-dformat` stands for the specified graphics format (such as JPEG) and `filename` is the filename.

The following table lists some of the formats that can be used to print and save figures.

Example	Description
`print -dmeta`	Saves the active figure in the clipboard. This command is equivalent to the copy command that can be used via the mouse or keyboard
`print -dpng pippo`	Saves the active figure in the file named pippo.png The file is saved in the Portable Network Graphics (png) format
`print -depsc pluto`	Saves the active figure in the file named pluto.eps. The file is saved in the Encapsulated Postscript Color (epc) format
`print -djpeg83 Minnie`	Save the active figure in the file `Minnie.jpg`. The file is saved in jpeg format with quality 83%. To obtain a different quality (=compression), the last number can be changed: e.g., print `-djpeg25 Minnie2` saves a jpeg image named `Minnie2.jpg` with quality 25%

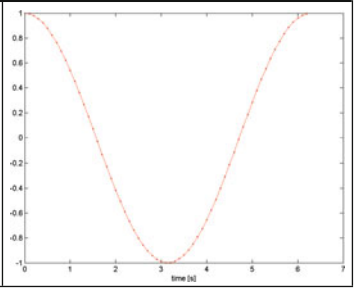
Handle Graphics

In this chapter we have seen the MATLAB functions that enable the production of simple graphs and how to set many parameters such as the color of the axes, the line thickness, the position of the plot, the font size, and so on. However, MATLAB allows us to control many of the graph's characteristics by getting and setting some of the properties for each object in the figure window (lines, axes, text, surfaces, etc.).

Each object in the figure window has a unique identifier (a number) called a *handle*. The handle is used with the commands `get` and `set` to read the current properties of the object and to change and set these properties according to your needs.

Everything will be clarified by the following example:

Example	Graphical result

`>> figure;`
`>> x=[0:0.1:2*pi];`
`>> h=plot(x,cos(x),'r.-');`
`>> hl=xlabel('time [s]');`
	

Here `h` is the handle of the line object, while `h1` is the handle of the axes object. Now, if you type the following `get` commands, you obtain a list of the line properties and a list of the axes properties:

Example 1

```
>> get(h)
                Color: [1 0 0]
            EraseMode: 'normal'
            LineStyle: '-'
            LineWidth: 0.5000
               Marker: '.'
           MarkerSize: 6
      MarkerEdgeColor: 'auto'
      MarkerFaceColor: 'none'
                XData: [1x63 double]
                YData: [1x63 double]
                ZData: [1x0 double]
         BeingDeleted: 'off'
        ButtonDownFcn: []
             Children: [0x1 double]
             Clipping: 'on'
           CreateFcn: []
           DeleteFcn: []
          BusyAction: 'queue'
     HandleVisibility: 'on'
              HitTest: 'on'
        Interruptible: 'on'
             Selected: 'off'
   SelectionHighlight: 'on'
                  Tag: ''
                 Type: 'line'
       UIContextMenu: []
             UserData: []
              Visible: 'on'
               Parent: 158.0052
          DisplayName: ''
            XDataMode: 'manual'
          XDataSource: ''
          YDataSource: ''
          ZDataSource: ''
```

Example 2

```
>> get(hl)
      BackgroundColor = none
      Color = [0 0 0]
      EdgeColor = none
      EraseMode = normal
      Editing = off
      Extent = [3.12 -1.21 0.69 0.09]
      FontAngle = normal
      FontName = Helvetica
      FontSize = [10]
      FontUnits = points
      FontWeight = normal
      HorizontalAlignment = center
      LineStyle = -
      LineWidth = [0.5]
      Margin = [2]
      Position = [3.48 -1.13 1.00]
      Rotation = [0]
      String = time [s]
      Units = data
      Interpreter = tex
      VerticalAlignment = cap

      BeingDeleted = off
      ButtonDownFcn =
      Children = []
      Clipping = off
      CreateFcn =
      DeleteFcn =
      BusyAction = queue
      HandleVisibility = off
      HitTest = on
      Interruptible = on
      Parent = [158.005]
      Selected = off
      SelectionHighlight = on
      Tag =
      Type = text
      UIContextMenu = []
      UserData = []
      Visible = on
```

You can change each property using the command `set`. For example, let's try to change the font size of the `xlabel` object and the size of the `line` object by typing the following commands:

```
>> set(hl,'FontSize',18);
>> set(h, 'LineWidth',3);
```

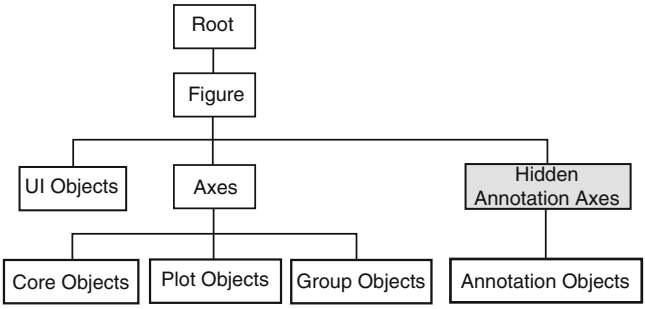


Fig. 3.3 MATLAB graphical object hierarchy

The way to use the set function is the following: the first argument of the function is the handle (a number that refers to the specific object), and the second argument is the property name (in our case the "FontSize"),[3] and finally the third argument is the value that is assigned to the property.

To really understand how to manage the MATLAB graphical objects, you must know that they are arranged according to the hierarchy shown in Fig. 3.3.

The object immediately above another is called a parent, and the objects below another are called children. In general, children inherit their handle graphics properties from their parent. For example, the position of a line on a plot depends on the position of the axes, which, in turn, depends on the position of the figure window. The `Root` object is the computer screen, and there can only be one `Root` object.

The UI objects are graphical user interface elements that are discussed in Chap. 8 of this book.

A parent can have any number of children. For example, the *Root* can have many *Figures*, a *Figure* can have many *Axes*, and a set of *Axes* can have many plot objects, such as *lines*, *surfaces*, and so on. If a parent has many children, one of them is designated to be the current one. For example, the current set of axes is the one that will be updated the next time you run a command. You can also make an object current by clicking on it with the mouse. The following functions return the handles of the current object:

Function	Description
`Gcf`	Return the handle of the current figure. The current figure is the last figure created, modified, selected or clicked on
`Gca`	Get the handle of the current *axes*. The current *axes* is typically the last *axes* used for plotting or the last *axes* clicked on by the mouse. Pay attention: do not confuse axes with the command `axis`
`Gco`	Get the handle of the current graphics object, which is the last graphics object created, modified or clicked on

(continued)

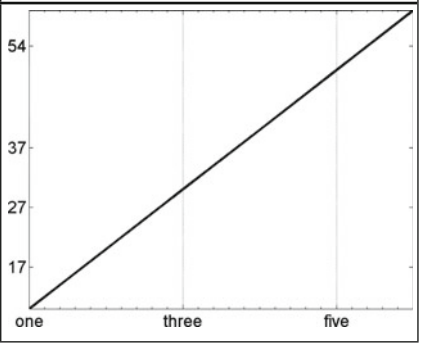
[3] This property and the associated value are listed by MATLAB using the command `get`.

(continued)

Function	Description
`h=findobj('PropName',PropVal)`	Return the handles of all graphics objects having the property *PropName*, set to the value *PropValue*. You can specify more than one property/value pair, in which case, `findobj` returns only those objects having all specified values

We have no wish to describe every MATLAB object's property here. The properties can be viewed using the online MATLAB help. We want to call your attention to the fact that every object's properties can be changed at will. The following example shows a few of the possible changes.

Example	Graphical result

`>> figure;`
`>> plot(1:6,10:10:60)`
`>> set(gca,'XTick',[1,3,5])`
`>> set(gca,'XTickLabel',…`
`{'one' ,'three','five'})`
`>> set(gca,'XMinorTick','on')`
`>> set(gca,'Xgrid','on')`
`>> set(gca,'YTick',[17,27,37,54])`
`>> h=findobj('Type','line');`
`>> set(h,'color','k')`
`>> set(h, 'LineWidth',3);`
`>> set(gca,'FontSize', 20);`
	

As you can see, there are many ways to change the appearance of a figure. As we wrote at the beginning of the chapter, it is possible also to change all a figure's properties by clicking "edit" in the figure menu bar and then by selecting one of the alternatives (Figure Properties, Axes Properties, etc.), or clicking on the arrow button in the figure menu and selecting the desired item (line, axis, title, etc.). However, in the long run, the possibility of writing a short code to create a figure and edit the figure's characteristics turns to be useful for researchers.

Summary

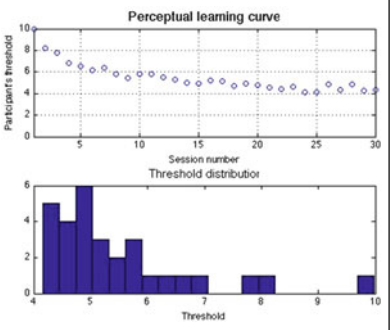
- `plot` is the basic function for 2-D plots.
- Graphs can be customized with `text, title, xlabel, ylabel, grid`, etc.
- Axes limits are implicitly calculated. However, they can be modified using the `axis` function.
- Multiple graphs can be obtained using `subplot`.
- `hist, bar, errorbar, pie`, are other functions to plot 2-D graphs.
- `plot3, bar3, surf, surfc, mesh, meshc`, are the functions to plot 3-D graphs.
- `meshgrid` is useful to define the x-y points for 3-D plots.

- Figures can be printed in files or directly to printer output using the command `print`.
- A handle is a number associated to a graphical object. It is used with the `set` and `get` commands to obtain or change an object's properties.
- The handles can be obtained when an object is created or by using one of the following commands: `gcf` (gets the handle of the current picture), `gca` (gets the handle of the current axes), `gco` (gets the handle of the current object).

Exercises

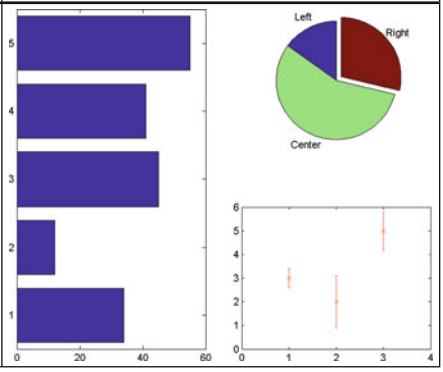
1. Create a vector x of values from 1 to 10. Then create a vector y containing the squares of the elements of x. In vector z put the values of x multiplied by 9.
   (a) In Figure 1 plot the vector y versus x using a red line with squares as markers.
   (b) In Figure 5 plot the vector z versus x using a black dash-dot line having triangles as markers.
   (c) In Figure 3, plot the vector y versus x using a green line and the vector z using a magenta line. Provide a title and a legend.
2. Create the figure of a hypothetical perceptual learning experiment divided in two graphs. At the top, plot the performance (i.e., the threshold) vs. the session number using the circle symbol. The session number goes from 1 to 100. The threshold is given by the following command: `th=1000*[1:100].^(-1/4)+randn(1,100)/5;` Set the y-axis to display values from 0 to 10. On the bottom part, plot the histogram of thresholds, subdividing the data intp 40 bins. Add legends, a title, and grids.

Solution	Graphical result

`th=10*[1:30].^(-1/4)+randn(1,30)/5;`
`figure;`
`subplot(2,1,1);`
`plot(th,'o');`
`axis([1 30 0 10]);`
`ylabel('Participant''s threshold');`
`xlabel('Session number');`
`grid;`
`title('Perceptual learning curve','FontSize',14);`
`subplot(2,1,2);`
`hist(th,20)`
`xlabel('Threshold');`
`title('Threshold distribution',…`
`'FontSize',12);`
	

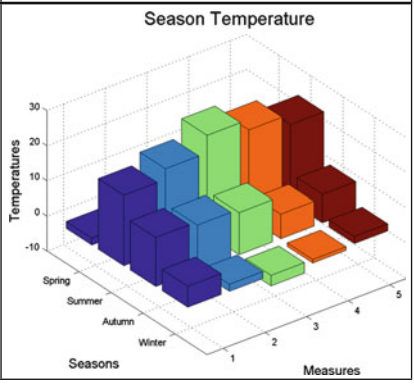
3. Create a figure divided into two parts. On the left side, display five horizontal bars with the following values: `horbarValue=[34,12,45,41,55]`. On the right side, display two graphs. The upper part will contain a 3-D pie, with three pieces named "left," "center," "right," with values `val=[12, 45, 23]`. In the bottom part place an error bar. The mean is equal to `mv=[3,2,5]`, and the standard deviation is equal to `stdval=[0.4, 1.1, 0.8]`.

Solution	Graphical result

`>> horbarValue = [34,12,45,41,55];`
`>> val=[12, 45, 23];`
`>> mv=[3,2,5];`
`>> stdval=[0.4, 1.1, 0.8];`
`>> figure;`
`>> subplot(2,2,[1 3]);`
`>> barh(horbarValue);`
`>> subplot(2,2,2)`
`>> pie(val,[0 0 1],`
`{'Left','Center','Right'});`
`>> subplot(2,2,4);`
`>> errorbar(mv,stdval,'xr');`
	

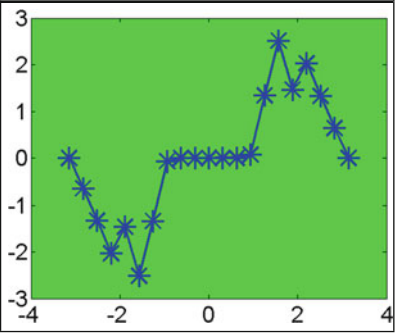
4. Given the 4×5 matrix `Temp=[2 3 5 7 8; 20 23 28 25; 14, 13, 12, 7; 6 2 −3 2]`, display its values using a 3-D bar graph. Title the figure "Season Temperature." Place on the y-axis the labels 'Spring', 'Summer', 'Autumn', 'Winter'. Label the x-, y-, and z-axes "Season," "Measures," and "Temperature."

Solution	Graphical result

`>> Temp = [2, 3, 5, 7, 8; …`
`20, 23, 28, 25, 22; …`
`14, 13, 12, 7, 8; …`
`6, 2, -3, -1, 2];`
`>> bar3(Temp)`
`>> set(gca,'yTickLabel',…`
`{'Spring','Summer','Autumn','Winter'});`
`>> ylabel('Seasons','FontSize',15)`
`>> xlabel('Measures','FontSize',15)`
`>> zlabel('Temperatures',`
`'FontSize',15)`
`>> title('Season Temperature',`
`'FontSize',20)`
	

5. Display the function `y=tan(sin(x))-sin(tan(x))`, where `x=-pi:pi/10:pi`. Change the color line to red, use stars (*) as a marker, with a marker size equal to 10. Set the graph background color to green. Set the axis font size to 20.

Solution	Graphical result

`>> x=-pi:pi/10:pi;`
`>> y=tan(sin(x))-sin(tan(x));`
`>> figure;`
`>> H=plot(x,y);`
`>> set(H, 'linewidth',5);`
`>> set(H,'markersize',20);`
`>> set(gca,'fontsize', 20);`
`>> set(gca, 'color', 'green');`
	

A Brick for an Experiment

Plot the Results

MATLAB is a powerful tool for graphics. However, this brick requires a relatively simple graph. Usually, the results of experiments like that of Sekuler et al. (1997) are represented with bar graps. Here too we will represent the results with a bar graph. We will draw a plot where the discs' motion (continuous versus with stop) is represented along the x-axis and bars are grouped by presence (or absence) of sound.

First, we need to get the means and the standard errors of the data we want to represent. We proceed as in the previous chapter. But first, we store the number of subjects we have run within the variable N.

```
>> N = 10;
>> m=zeros(2, 2);
>> m(1, 1) = mean(data(data(:, 5)==1 & data(:, 6)==1, 7)); %
continuous motion no sound
>> m(1, 2) = mean(data(data(:, 5)==1 & data(:, 6)==2, 7)); %
motion with stop
>> m(2, 1) = mean(data(data(:, 5)==2 & data(:, 6)==1, 7)); % sound
absent
>> m(2, 2) = mean(data(data(:, 5)==2 & data(:, 6)==2, 7)); % sound
present
>> err=zeros(2, 2);
>> err(1, 1) = std(data(data(:, 5)==1 & data(:, 6)==1, 7))/
sqrt(N); % continuous motion
>> err(1, 2) = std(data(data(:, 5)==1 & data(:, 6)==2, 7))/
sqrt(N); % motion with stop
```

```
>> err(2, 1) = std(data(data(:, 5)==2 & data(:, 6)==1, 7))/
sqrt(N); % sound absent
>> err(2, 2) = std(data(data(:, 5)==2 & data(:, 6)==2, 7))/
sqrt(N); % sound present
```

The brick plot will be made using a function that can be freely downloaded from MATLAB central. The function is called barweb.[4] The reason for using barweb is the following: this function combines within the same function two different functions: bar and errorbar. In other words, barweb simplifies the creation of a bar graph with error bars. We can now fill all the necessary fields and have a final look at the data:

```
>> barweb(m, err, [], {'without stop', 'with stop'}, [], {'kind
of motion'}, {'percent bouncing'}, [], [], {'no sound',
'sound'})
```

Finally, you may want to export the figure as a graphic file. This can be done as follows. In the file menu of the plot figure you can "save as" the figure as a jpg, tif, gif, PostScript, or one of various other graphics formats. Alternatively, you can use the print option at the MATLAB prompt. For example:

```
>> figure(1);
>> print -depsc finalresult.eps
```

Reference

Sekuler R, Sekuler AB, Lau R (1997) Sound alters visual motion perception. Nature 385:308

Suggested Readings

Some of the concepts illustrated in this chapter can be found, in an extended way, in the following book:

Marchand P, Holland OT (2003) Graphics and GUIs with MATLAB. CRC press. Boca Raton, FL
Siciliano A (2008) MATLAB: data analysis and visualization. World Scientific, Singapore
Wallisch P, Lusignan M, Benayoun M, Baker TI, Dickey AS, Hatsopoulos NG (2009) MATLAB for neuroscientists: an introduction to scientific computing in MATLAB. Elsevier/Academic Press, Amsterdam

[4] You can also download the file from the book website.