Chapter 6

Visualization using MATLAB

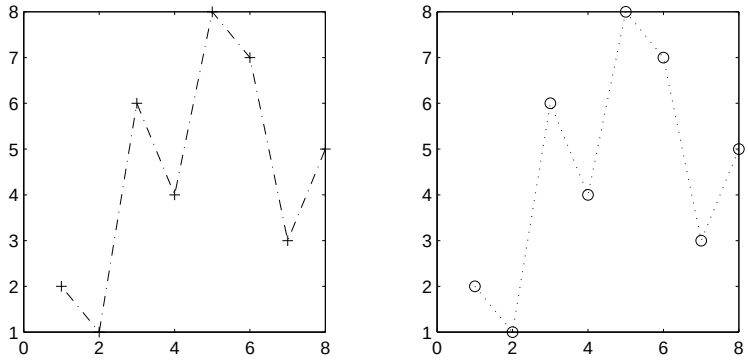
In this lecture, we will see how MATLAB can be used for visualizing mathematical concepts. Often we are interesting in producing the results of a calculation graphically, such as the *phase portraits* introduced in Chapter 4 and MATLAB has a number of powerful graphics capabilities. These can generate figures of many different types and feels, quickly and easily. We shall concentrate on graphing two and three dimensional data sets. We shall introduce only the main features in this class, `helpdesk` and the book MATLAB guide, provide further in-depth information about the many different `plot` commands in MATLAB.

6.1 Two dimensional plots

We have already introduced the basic `plot` and `plot3` commands for dot-to-dot plots of two and three dimensional vectors, however, you can change the style and feel of these, by specifying addition information. For example, given the following two vectors:

```
>> x = [1 2 3 4 5 6 7 8];
>> y = [2 1 6 4 8 7 3 5];
>> plot(x,y)
```

We could replace the `plot(x,y)` command with the more general `plot(x,y,`*`string`*`)`, where *string* combines elements to control the colour, marker and line style. For example, `plot(x,y,'g+-.')` specfies that a green plus sign be placed at each point, and that a *dash-dot* line join the plusses. Two examples of what is possible are displayed in the following figure:



The complete list of options involving plot are contained in the following tables:

Abbr.	Colour
b	blue
c	Cyan
g	Green
k	Black
m	Magenta
r	Red
w	White
y	Yellow

Abbr.	Marker
o	Circle
*	Asterisk
.	Point
+	Plus
x	Cross
s	Square
d	Diamond
∧	Upward triangle
v	Downward triangle
>	Right triangle
<	Left triangle
p	Five-point star
h	Six-point star

Abbr.	Line style
-	Solid line
- -	Dashed line
:	Dotted line
-.	Dash-dot line

You may have noticed that we have produced two graphs in one figure. To do this, we used the `subplot` command. The exact MATLAB commands we used were as follows:

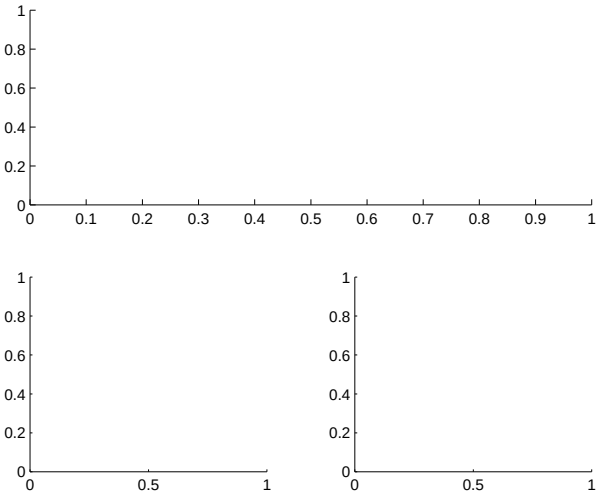
```
>> subplot(121)
>> plot(x,y,'g+-.')
>> subplot(122)
>> plot(x,y,'k0:')
```

The tt subplot command takes the form `subplot(abc)`, where a is the number of rows, b is the number of columns and c is the number of the plot. The numbering is in ascending order starting from the top left, along each row. You can create figures with different numbers of plots in each row or column. For example:

```
>> subplot(211)
```

```
>> subplot(223)
>> subplot(224)
```

produces the following figure:



6.1.1 Axes and Annotation

Various aspects of the axes of a plot can be controlled using the `axis` command. The axes MATLAB chooses for a plot are sometimes not those that we would choose ourselves, but fortunately MATLAB is very flexible and these can be tailored specifically to the situation at hand.

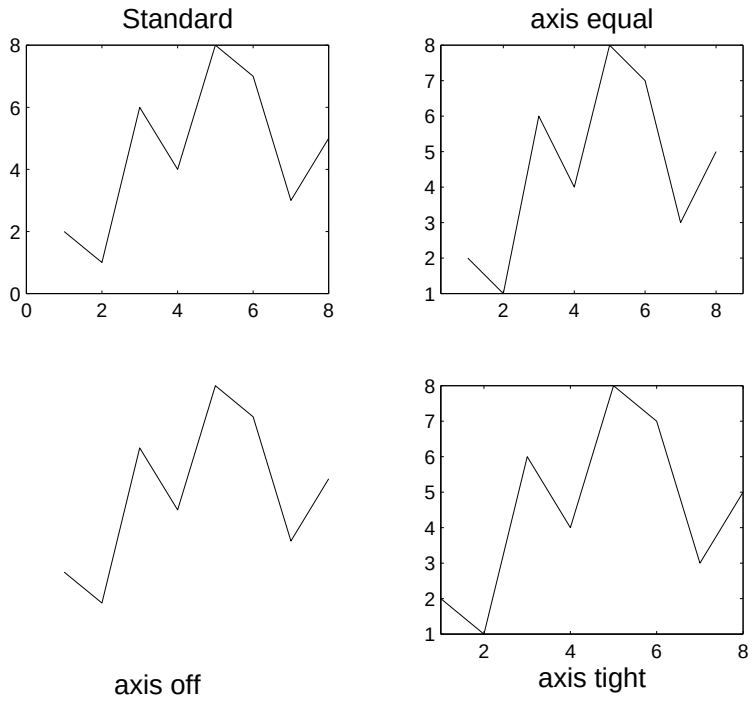
For example, the aspect ratio of a plot can be set to unity by typing `axis equal`. We can remove the axes altogether using `axis off` or we could make limit the axes to the range of the data using `axis tight`.

You can manually specify the range of the axes for x and y by using `axis([`x_{min} x_{max} y_{min} y_{max}`])`. Finally if you decide that MATLAB's choice was the best after all, you can revert to the default by typing `axis auto`.

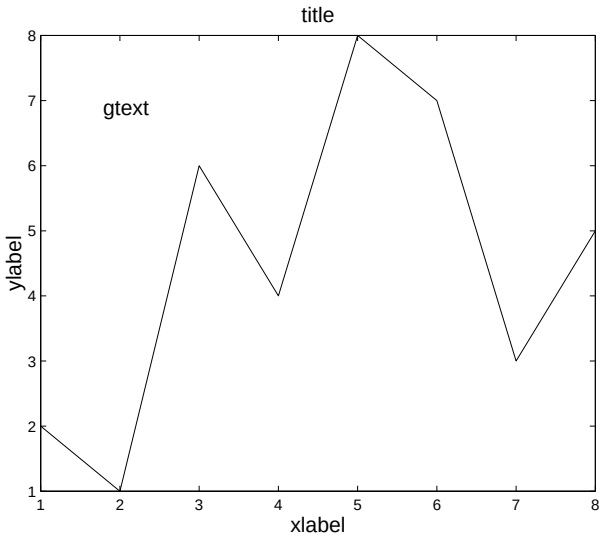
6.1.2 Labelling

You may have noticed in the previous figure that all of the plots had some form of labelling. There are four main ways of labelling a `plot` or `subplot`:

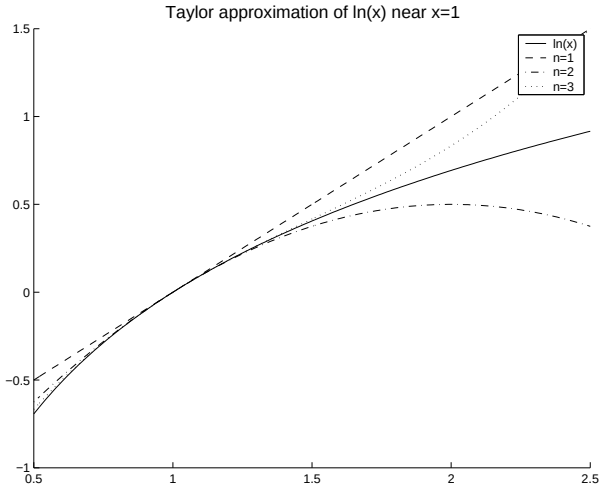
`title`	Graph title
`xlabel`	x-axis (horizontal)
`ylabel`	y-axis (vertical)
`gtext`	Manual choice with mouse



We illustrate these choices on the following figure:



In addition to this type of labelling we may also create a *legend*, whereby MATLAB will place a box in a specified location on the graph in order to explain the different line styles. The command is simply `legend('string1','string2',...,'stringn')`, where each *string* is the description associated with each line style. For example:



The MATLAB code required to generate the above figure is as follows:

```
>> x=0.5:.01:2.5;
>> p1=log(x);
>> p2=log(1)+(x-1);
>> p3=(x-1)-((x-1).^2)./2;
>> p4=(x-1)-((x-1).^2)./2+2*((x-1).^3)./6;
>> plot(x,p1,'k-',x,p2,'k--',x,p3,'k-.',x,p4,'k:')
>> legend('ln(x)','n=1','n=2','n=3')
>> title('Taylor approximation of ln(x) near x=1','Fontsize',14)
```

6.1.3 Plotting symbolic expressions in two dimensions

Notice that in the previous figure, we had to specify explictly the range of values for the vector x in order to produce other vectors containing the polynomial approximations we were considering. There is a more straightforward technique than this using MATLAB faciliities for the plotting of symbolic functions.

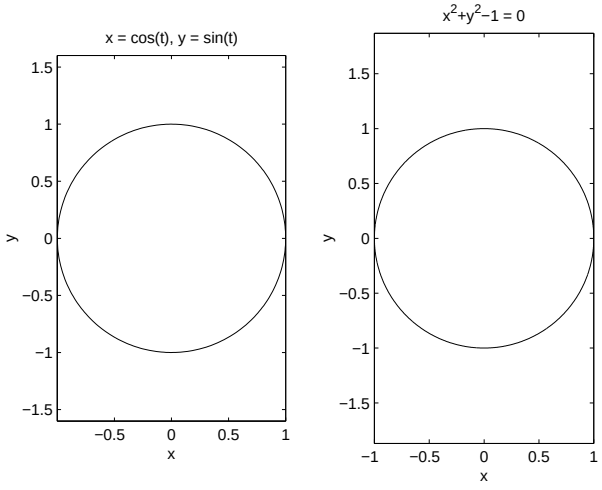
As before, we must start by specifying to MATLAB which variables we would like it to consider as symbolic.

```
>> clear
>> syms x y t s theta
```

The command `ezplot` can plot either parametrically or implicitly defined curves in two dimensions. We plot both $(x, y) = (\cos t, \sin t)$ and $x^2 + y^2 = 1$.

```
>> subplot(121)
>> ezplot(cos(t),sin(t),[0, 2*pi])
>> subplot(122)
>> ezplot(x^2 + y^2 - 1, [-1, 1, -1, 1])
```

Notice that `ezplot` offers less flexibility than the standard `plot` command, for example, the size of the figure is of a fixed aspect and cannot be adjusted by altering the size of figure on the screen.



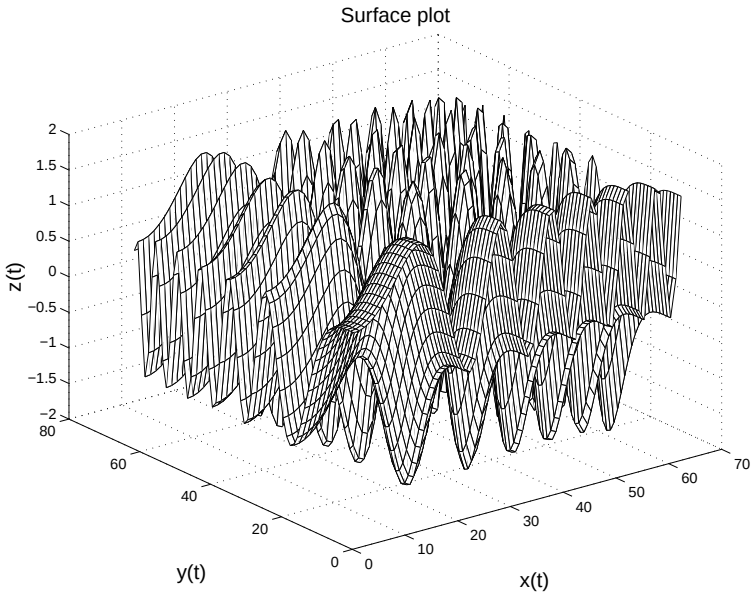
A summary of the main two dimensional plotting commands is given in the following table:

`plot`	Simple x-y plot
`loglog`	Plot with logarithmically scaled axes
`semilogx/semilogy`	Plot with logarithmically scaled x or y-axis
`ezplot/ezpolar`	Plotting symbolically defined functions
`polar`	Polar co-ordinate plot
`fplot`	Autmoatic function plot; similar to `ezplot`
`bar/barh`	Bar chart of given data or horizontal version
`hist`	Histogram of given data
`pie`	Pie chart of given data
`quiver`	Plots a vector field using arrows (see Chapter 4)

6.2 Three dimensional plots

As you may recall the function `plot3` is the same as the `plot` function, but in three dimensions. In addition to this standard three dimensional plot, we can also create *surfaces* and *contour* plots. For example:

```
>> x=0:.1:2*pi;
>> y=0:.1:2*pi;
>> [X,Y]= meshgrid(x,y);
>> Z=sin(X.^2)+cos(Y.^2);
>> mesh(Z)
```



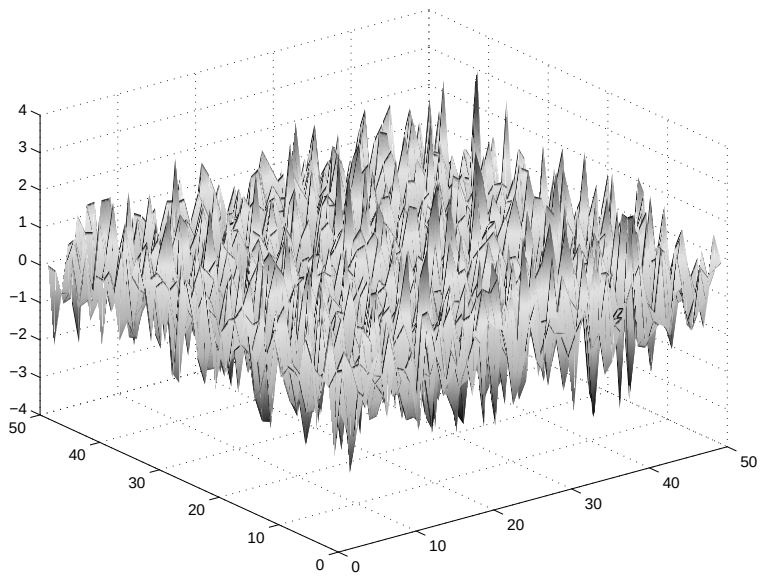
In addition to the above commands we have used `title`, `xlabel`, `ylabel` and the equivalent `zlabel` to produce the captions. If we wish to add a contour plot

underneath the three dimensional plot, we would have used the `meshc` command instead of `mesh`.

Another useful command is `surf`, whereby you can visually interpret a matrix. For example:

```
>>A=randn(50);
>>surf(A)
>>shading interp
```

produces the graph on the following page, the x and y axes are the grid points as determined by the entries in the matrix. The z axis represents the actual values taken by the individual entries.



A summary of the main three dimensional plotting commands is given in the following table; note many of them are similar to the two dimensional commands we considered earlier:

`plot3`	Simple x-y-z plot
`contour`	Contour plot
`contourf`	Filled contour plot
`contour3`	3D contour plot
`mesh`	Wireframe surface plot
`meshc`	Wireframe surface plot plus contours
`meshz`	Wireframe surface plot with curtain
`surf`	Solid surface plot
`surfc`	Solid surface plot plus contours
`waterfall`	Unidirectional wireframe plot
`bar3`	3D bar graph
`bar3h`	3D horizontal bar graph
`pie3`	3D pie chart
`fill3`	Polygon fill
`comet3`	3D animated, comet-like plot
`scatter3`	3D scatter plot
`stem3`	Stem plot

6.3 Plotting symbolic functions in three dimensions

MATLAB also has a number of commands for plotting symbolic expressions of the type introduced in the previous lecture. Specifically the commands we consider are `ezplot` for curves in 2D, `ezpolar` for plotting in polar coordinates, `ezplot3` for curves in 3D, `ezsurf` for 3D surfaces and `ezcontour` for contour plots.

6.3.1 Curves in 3D

To begin with we must again notify MATLAB of the variables which we would like it to consider as symbolic:

```
>> clear
>> syms xy t s theta
```

As with `ezplot`, we can use `ezplot3` to plot parametric curves in 3D. We can tell it to trace out the curve by giving it the string `'animate'` as an argument.

```
>> ezplot3(t*cos(t),t*sin(t),t,[0 50],'animate')
>> close
```

We can also plot in polar coordinates using `ezpolar`

```
>> ezpolar(1 + cos(theta))
```

6.3.2 Surfaces in 3D

`ezsurf` is for plotting surfaces, either parametrically defined or giving the z-coordinate in terms of x and y. The argument `'circ'` tells `ezsurf` to plot the function over a circular domain. `ezmesh` can also be used to plot $z = f(x, y)$, but it is almost the same as `ezsurf`. We plot the two surfaces $(x, y, z) = (s\cos t, s\sin t, s)$ and $z^2 = x^2 + y^2$.

```
>> ezsurf(s*cos(t),s*sin(t),s);
>> ezsurf(sqrt(x^2 + y^2), 'circ')
>> hold on
>> ezsurf(-sqrt(x^2 + y^2), 'circ')
>> hold off
```

`ezsurfc` is the same as `ezsurf` but adds a contour plot of the function below the surface.

```
>> ezsurf(sin(x)*cos(y))
>> ezsurfc(sin(x)*cos(y))
```

In today's lecture we have learnt about a number of different MATLAB functions for visualizing data in 2 and 3 dimensions. Next week we shall examine how MATLAB can be used to solve systems of linear equations and its use in eigenvalue problems.