MATLAB Graphics - 2D Plotting

Leigh J. Little SUNY Brockport September 5, 2001

Contents

1 Introduction

MATLAB makes creating a large variety of 2D plots very simple. This is a brief introduction to the use of MATLAB's 2D plotting capabilities.

2 Basic Plotting

2.1 X-Y Plots

To create a basic plot, the data needs to be made available to MATLAB in some way. The data can be created artifically or it can be the result of a numerical simulation (in MATLAB or some other application). For most of this tutorial, the former approach will be used.

Create 2 vectors x and y from the MATLAB command prompt as follows:

```
>>  x = (-pi:2*pi/50:pi); y = sin(x);
```

x is a vector that divides the interval $[-\pi, \pi]$ into 51 equally spaced points and y represents the sine of the elements of x.

A simple graph of $y = sin(x)$ can be created in the following way:

```
>> plot(x,y)
```

The resulting plot is shown in Figure 1. Note that this command plots the graph with the points connected by a solid line. This behavior can be altered in many ways:

```
>> plot(x,y,':') - connects points with a dotted line
>> plot(x,y,'-.') - connects points with a dash-dot line
>> plot(x,y,'--') - connects points with a dashed line
```

These plots are shown in Figures 2,3 and 4 respectively.

The special characters at the end of the command are the `plot specification` (*plot spec*). In general, the plot spec consists of 3 characters (which can appear in any order) indicating the plot color, marker type and line type. See the command `help plot` for the details the available options.

Depending on the situation, you may want to create a graph that displays only the raw data. You can accomplish this by issuing the following commands:

```
>> plot(x,y,'.') - plot the points with a dot, not connected.
>> plot(x,y,'o') - save as above, but use circles.
>> plot(x,y,'x') - same as above, but use x's.
```

These are shown in Figures 5,6 and 7 respectively. There are many more marker types available. If the there are many points to be plotted, using markers may be problematic as they tend to obscure the data.

The color, marker and linetype options can be combined to create several graph types.

```
>> plot(x,y,'r*-') - connect the points with a red, solid line and mark
 the points with asterisks (*).
>> plot(x,y,'bs:') - connect the points with a blue dotted line and mark
 the points with a square
```

You can close the current figure by issuing the `close` command.

2.2 Lines

If you need to plot a region that consists primarily of straight line connections, you can use the line command. This command has the syntax:

```
>> line(x,y)
```

where `x` and `y` are vectors containing the x- and y-coordinates respectively. For example, to plot the square with vertices

$$S = \{(1,1),(1,2),(2,2),(2,1)\}$$

you can form the vectors `x` and `y` then issue a line command.

```
>> x = [1 1 2 2 1];
>> y = [1 2 2 1 1];
>> line(x,y)
```

The resulting plot is shown in Figure 8. Note that an extra set of coordinates is needed to 'close' the square. See `help line` for more information.

2.3 The Plot Window

Every plot you create will be encased in a plot window with a menubar and toolbar. The menubar allows you to do things such as add arrows and additional information to the figure as well as zoom and rotate the figure. The toolbar contains shortcuts to the most commonly used menubar items.

Note that this plot window has `Figure No. 1` in the title bar. Every figure receives a number that can be referred to from the command line (see Section 4.3).

2.4 Other Plots

MATLAB has the capability to generate a large number of 2D plots. These include pie and bar graphs, histograms and logarithmic plots.

2.5 Printing

There are 2 ways to print a graph in MATLAB. One way is to use the menu pull-down in the plot window and selecting print. You may also sometimes need to print the graph to a file instead (for example, if you are going to insert the plot into another document). You can print any graph to a file using the print command. For example:

```
>> print -deps file.ps
```

will output the graph (in postscript format) to the file `file.ps` while

```
>> print -djpeg75 file.jpg
```

will output the graph (in JPEG format with a quality level of 75) to the file `file.jpg`. There are many other graphics formats supported, but variations on the postscript format are usually the most convenient for course work.

2.6 Plot Background

The default background color for MATLAB plots is white. This can make it difficult to see some of the lighter colors. You can force the background color to be black by issuing the command

```
>> colordef none
```

3 Titles, Labels and Legends

The graphs created so far are interesting, but they need some comments (also called *annotation*) to help in their interpretation. With very few exceptions, graphs without annotation are meaningless. Once the basic plot has been created, annotations can be added.

3.1 Titles and Labels

To add a title to the plot, use the `title` command.

```
>> close
>> x = (-pi:2*pi/50;pi); y = sin(x);
>> plot(x,y)
>> title('This is a plot of y = sin(x)')
```

This is shown in Figure 9. Notice that the input to the `title` command is a string variable. The sequence of statements below would accomplish the same result.

```
>> t1 = 'This is a plot of y = sin(x)'
>> title(t1)
```

The latter version is helpful for complex titles. One case where you might want to use this is when you don't know the details of the title until the graph is actually created. For example, suppose you wanted the total number of data points included in the title. This can be accomplished as follows:

```
>> t1 = ['This is a plot of y = sin(x) with ' num2str(length(x)) ' points'];
>> title(t1)
```

This is shown in Figure 10. We have use the `num2str` function to convert the number of points (`length(x)`) into a string variable. Also note that the inserting of spaces around the number needs to be done manually.

To add labels on either the x- or y-axis, use the `label` commands.

```
>> xlabel('This is the x-axis')
>> ylabel('This is the y-axis')
```

This is shown in Figure 11.

Close this plot using the `close` command.

3.2 Legends

To illustrate the `legend` command, create 2 more vectors as follows:

```
>> z = cos(x);
>> w = cos(x).*sin(x);
```

Plot these vectors all on the same graph. This can be done in several different ways. The simplest is to do:

```
>> plot(x,y,x,z,x,w)
```

which is demonstrated in 12. This is not very useful however. All the graphs are plotted with solid connecting lines and you cannot distinguish from the different graphs. This can be remedied by using a different plot spec for each line.

```
>> close
>> plot(x,y,'r*-',x,z,'b.-',x,w,'go-');
```

This plots y with red stars, z with blue dots and w with green circles. The `legend` command is used to add a description to each the plots on a multi-line plot. You can see this effect by entering:

```
>> legend('sin(x)','cos(x)','sin(x)*cos(x)')
```

The final version of this graph is shown in Figure 13. The order of titles in the legend command must be the same as the order in the plot command. This is a much better graph since the individual plots can now be distinguished. You can control the placement of the legend on the graph (see `help legend`).

Finish up this graph by adding some titles and axis labels.

```
>> title('Simple trigonometric graphs')
>> xlabel('x')
```

Notice that the legend serves as the y-axis label.

4 Multiple Plots

The previous example was just one of many ways to create a figure having multiple components. This section describes several other ways of viewing multiple figures.

4.1 The hold command

A convenient way to place several graphs on the same set of axes is to use the `hold` command. This command acts like a switch. When you turn it on, all future plot commands will appear in the same plot window and will overlay each other. The previous plot could have also been created by entering:

```
>> close
>> hold on
>> plot(x,y,'r*-')
>> plot(x,z,'b.-')
>> plot(x,w,'go-')
>> hold off
```

The advantage of using `hold` is that you have more control over the overlaying. Also, it makes a long plot command easier to decipher if you encounter errors or unexpected plot results.

4.2 Subplots

The `subplot` command is used to display several different graphs in the same plot window. This is useful when each of the graphs has it's own scaling. The `subplot` command takes a standard graph and subdivides it into a rectangular array of smaller plots. As an example, consider x, y, x, w as defined above and define

```
>> v = sin(2*x)
```

We can create a 2×2 grid of smaller plots, each containing a single plot by entering:

```
>> close
>> subplot(2,2,1)
>>   plot(x,y,'r*-')
>>   title('y = sin(x)')
>>   xlabel('x')
>> subplot(2,2,2)
>>   plot(x,z,'b+-')
>>   title('z = cos(x)')
>>   xlabel('x')
>> subplot(2,2,3)
>>   plot(x,y,'go-')
>>   title('y = sin(x)*cos(x)')
>>   xlabel('x')
>> subplot(2,2,4)
>>   plot(x,y,'md-')
>>   title('v = sin(2*x)')
>>   xlabel('x')
```

The plot is shown in Figure 14. In the above command sequence, the `subplot(2,2,1)` command means that the small plots will be arranged in 2 rows and 2 columns. The last number indicates which subplot is active. The subplots are numbered row by row from left to right.

4.3 Multiple Plot Windows

When you are examining several complex graphs, it is helpful to have each one in it's own plot window. This can be done using the `figure` command. As mentioned in Section 2.3, each figure the MATLAB creates receives a number. You can specify the figure number yourself, but MATLAB will chose the lowest available figure number (starting from one) if you do not. Repeat the previous sequence, but create each plot in its own window.

```
>> close
>> figure(1)
>>   plot(x,y,'r*-')
>>   title('y = sin(x)')
>>   xlabel('x')
>> figure(2)
>>   plot(x,z,'b+-')
>>   title('z = cos(x)')
>>   xlabel('x')
>> figure(3)
>>   plot(x,y,'go-')
>>   title('y = sin(x)*cos(x)')
>>   xlabel('x')
>> figure(4)
>>   plot(x,y,'md-')
>>   title('v = sin(2*x)')
>>   xlabel('x')
```

This will create 4 separate plot windows, each containing the indicated graph. If you need to change the properties of a figure, you can bring it to the foreground (make it active) by entering:

```
>> figure(1) (makes figure 1 active)
>> figure(4) (makes figure 4 active)
```

All future plot commands you issue will apply only to the active figure (for example, the `close` command closes only the active figure). If you want to close all open windows at once, enter:

```
>> close all
```

5 Axis properties

It is frequently necessary to change the properties of the axis. With the `axis` command you can change the axis limits, control how the axis looks and alter the axis scaling.

The simplest axis command is

```
>> close
>> plot(x,y)
>> axis off
```

This is shown in Figure 15. This removes both the x- and y-axes from the plot. To get them back, enter

```
>> axis on
```

Sometimes you want to change what portion of a graph is displayed. You can do this by setting up a vector containing the desired axis limits.

```
>> axlim = [-2 2 -0.5 0.5];
>> axis(axlim)
```

This sets the x-axis window to the interval $[-2, 2]$ and the y-axis limits to the interval $[-\frac{1}{2}, \frac{1}{2}]$. For this command to work properly, the axis limits must be entered in increasing order.

Scaling of axes is sometime required. By default, MATLAB will autoscale a plot so that the entire graph fits into the window. This is helpful, but it can cause distortion of the data. For example, enter:

```
>> close
>> plot(z,y)
>> title(The unit circle in the z-y plane')
>> xlabel('z')
>> ylabel('y')
```

This is shown in Figure 16. Notice that the plot looks like an ellipse, not a circle. This is because by default, MATLAB uses different scales for the x- and y-axes. To make the graph look like a circle, enter:

```
>> axis square
```

This is shown in Figure 17. This forces the same scale factors to be used for both axes. The graph should now look correct.

6 Placing Other Text

Occasionally, you want to place text on the graph in a location that is not controlled by the `label` or `title` commands. You can do this using the `text` and `gtext` commands.

As an example of the `text` command, regenerate the previous figure and enter the commands:

```
>> hold on
>> plot(-0.25,0.10,'*')
>> text('Center of Circle')
```

This is shown in Figure 18. This plots a `*` at the coordinates $(0, 0)$ and puts the text message at the coordinates $(-0.25, 0.10)$. You can alter the appearance of the text (font, size, *etc.*,). See `help text` for more information.

The `text` command is not always convenient to use. You can select the location of the text with the mouse using the `gtext` command. Issue the sequence:

```
>> close
>> plot(z,y)
>> title('The unit circle in the z-y plane')
>> xlabel('z')
>> ylabel('y')
>> axis square
>> hold on
>> plot(-0.25,0.10,'*')
>> gtext('Center of Circle')
```

The system will pause and bring the plot window to the foreground. Place the mouse where you want the left edge of the text to go and press the left button. Note that you have to allow for the width of the text you are placing.

A final note on using text. It can be difficult to undo a text placement and if you make a mistake, you may have to redo the entire plot.

7 External Data

Frequently you will need to plot data that was generated from an application (*i.e.*, a C or FORTRAN program) outside of MATLAB. Plotting the data is a simple matter, but you also need to import your data into the MATLAB environment so it can be accessed. This is done with the `load` command.

Suppose you have a text file called `testdata.dat` in your working directory that contains the following data:

```
1 2
3 4
5 6
7 8
9 10
11 12
```

This can be loaded into the MATLAB environment by entering:

```
>> load testdata
```

The data will be stored in a variable called `testdata`. This data can now be accessed. Note that the data is imported as a 6×2 matrix. As such, when it is plotted, you must use the MATLAB matrix notation to refer to each column individually (see the handout on MATLAB Linear Algebra for more information on this).

```
>> plot(testdata(:,1),testdata(:,2),'r*-')
>> xlabel('x')
>> ylabel('y')
>> title('Reading in Data from File')
```

The resulting plot is shown in Figure 19.

The ability to plot data from other applications is very convenient. The only thing you need to remember is that the data in the external file must be 'rectangular' (*i.e.*, it must have the same number of rows and columns).

8 Lab for MATLAB Graphics - 2D Plotting

For each of the problems below, be sure to add appropriate annotation (labels, legends, *etc.*) to your plots.

1) Plot the following sequences of curves on the intervals indicated. All the graphs should be on the same set of axes. Select a set of x points that is fine enough to capture all the features of the curves.

   a) $\sin(x)$, $\sin(x+\frac{\pi}{4})$, $\sin(x+\frac{\pi}{2})$, $\sin(x+\frac{3\pi}{4})$, $\sin(x+\pi)$, $x \in [-2\pi, 2\pi]$.

   b) $(x-2)^2$, $(x-1)^2$, x^2, $(x+1)^2$, $(x+2)^2$, $x \in [-3,3]$.

   c) x^2-2, x^2-1, x^2+1, x^2+2, $x \in [-2,2]$.

2) Repeat Problem 1c), but use the `subplot` command to put each figure in its own subplot window.

3) Use the `line` command to connect the points

$$S = \{(0,0),(1,1),(1,0)\}.$$

4) This example uses the zoom utility.

   a) Use the `line` command to plot lines determined by the 2 set of points

$$S_1 = \{(0,1),(4,3)\},$$

$$S_2 = \{(0,5),(4,-1)\}.$$

   (You should have 2 separate lines).

   b) Find the magnifying glass on the plot window toolbar, and click on the one with a + inside.

   c) Zoom in on the intersection point of the two lines by placing the pointer near the intersection point and clicking the left mouse button.

   d) Repeat until you are certain of the coordinates of the intersection point. What are these coordinates?

5) Use the `pie` function to create a pie graph of the following table.

Item	Percent
Fixed Costs	50
Variable Costs	20
Taxes	10
Advertising	20

Table 1: Simple expenditures for fictional company.

9 Figures

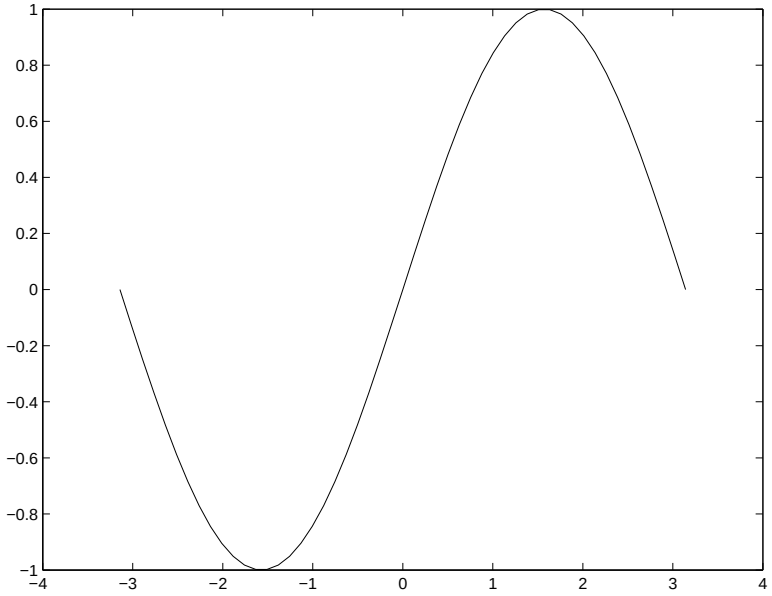

Figure 1: Simple plot of $y = \sin(x)$.

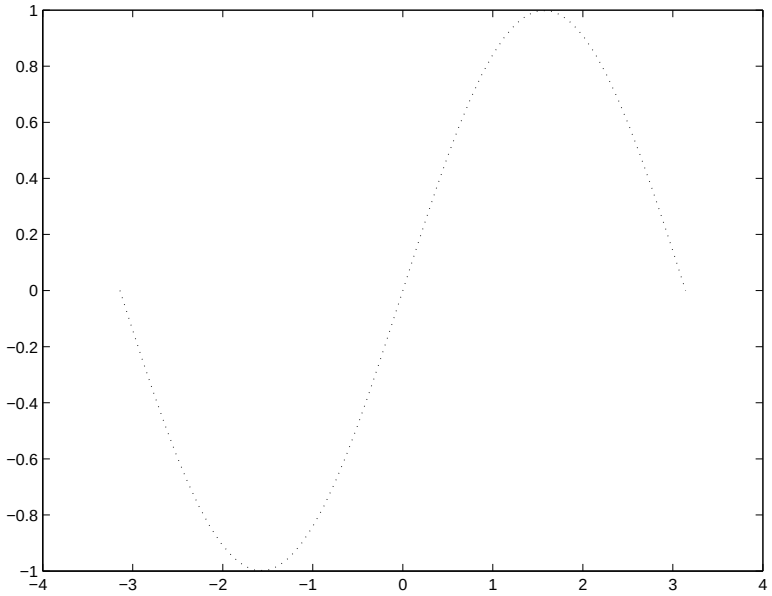

Figure 2: Simple plot of $y = \sin(x)$, dotted line.

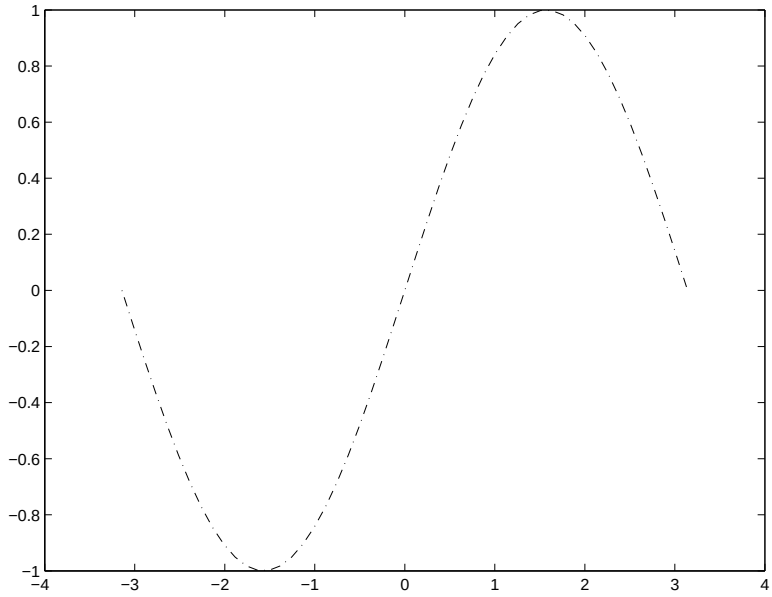

Figure 3: Simple plot of $y = \sin(x)$, dashed line.

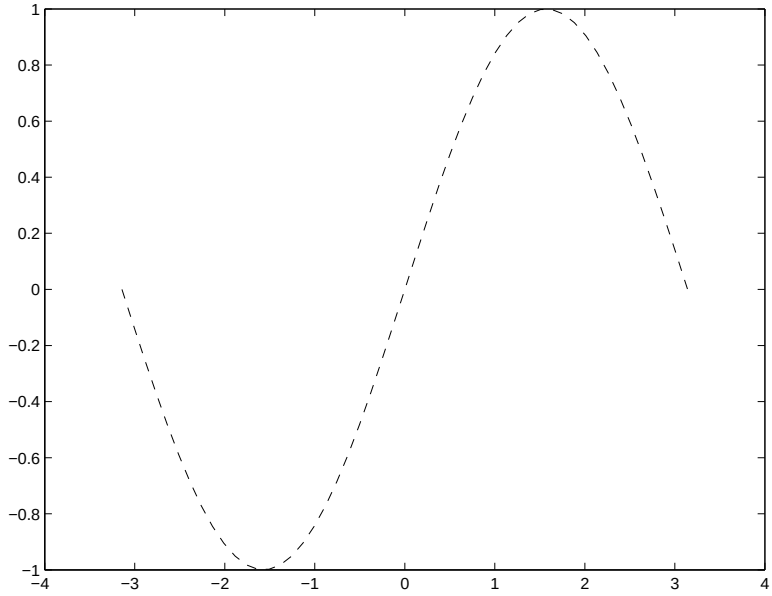

Figure 4: Simple plot of $y = \sin(x)$, dash-dot line.

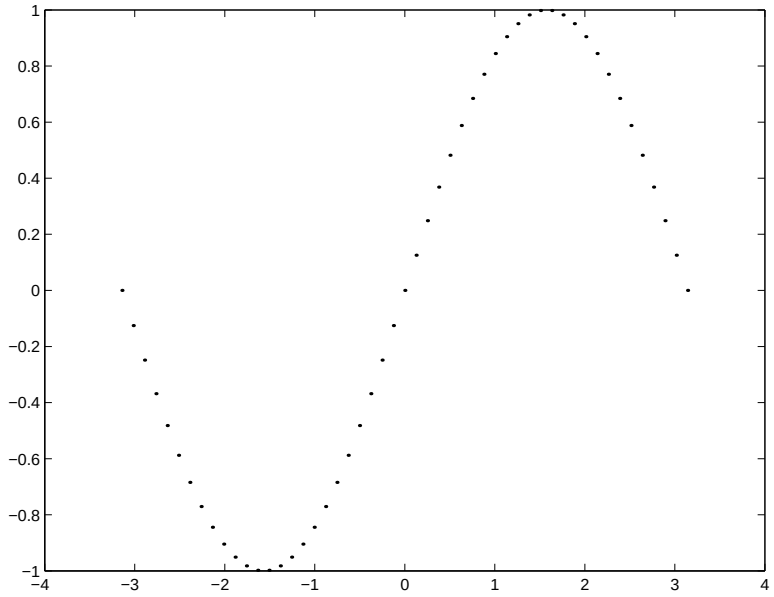

Figure 5: Simple plot of $y = \sin(x)$, mark points with dots.

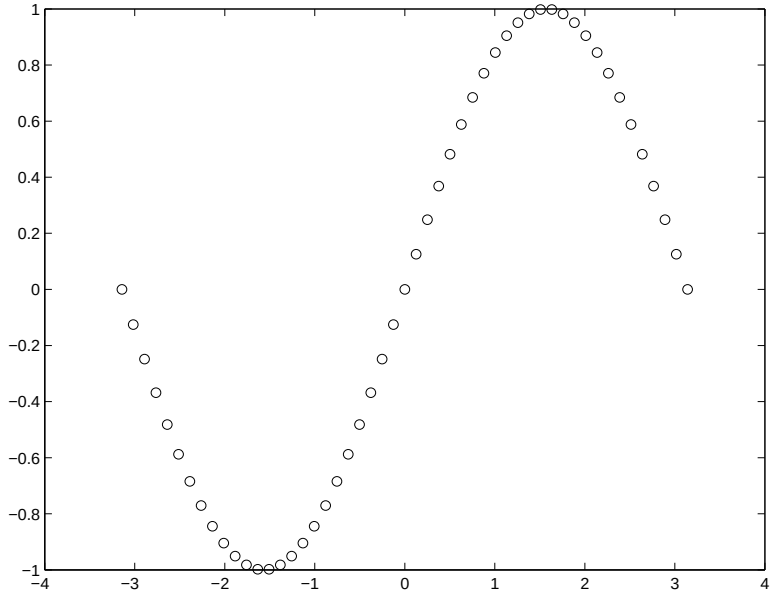

Figure 6: Simple plot of $y = \sin(x)$, mark points with o's.

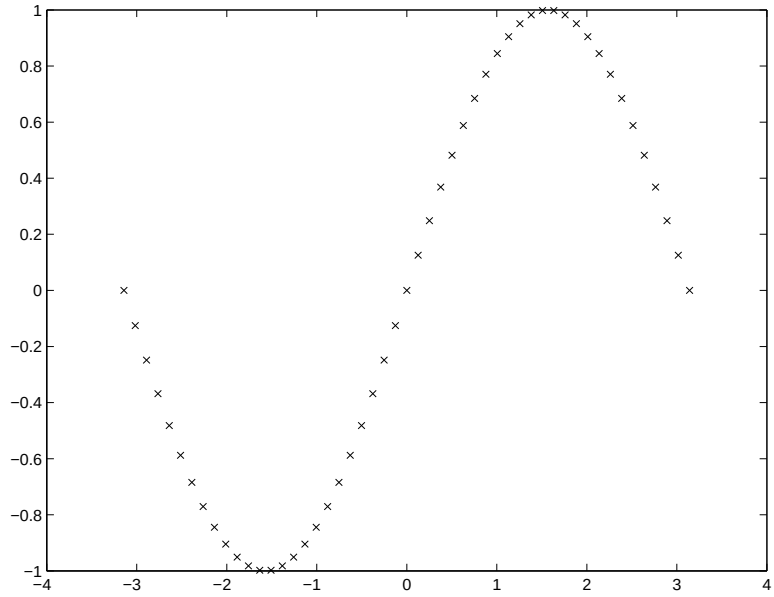

Figure 7: Simple plot of $y = \sin(x)$, mark points with x's.

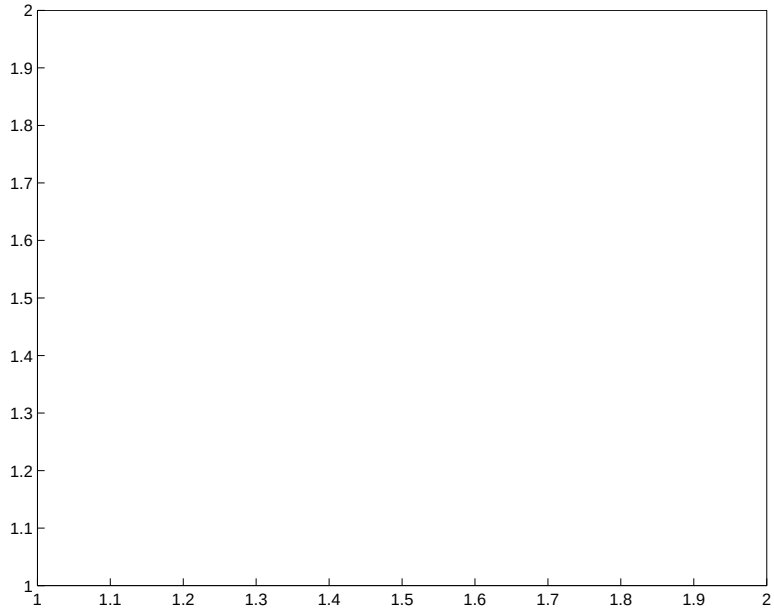

Figure 8: Plotting a box with the line command.

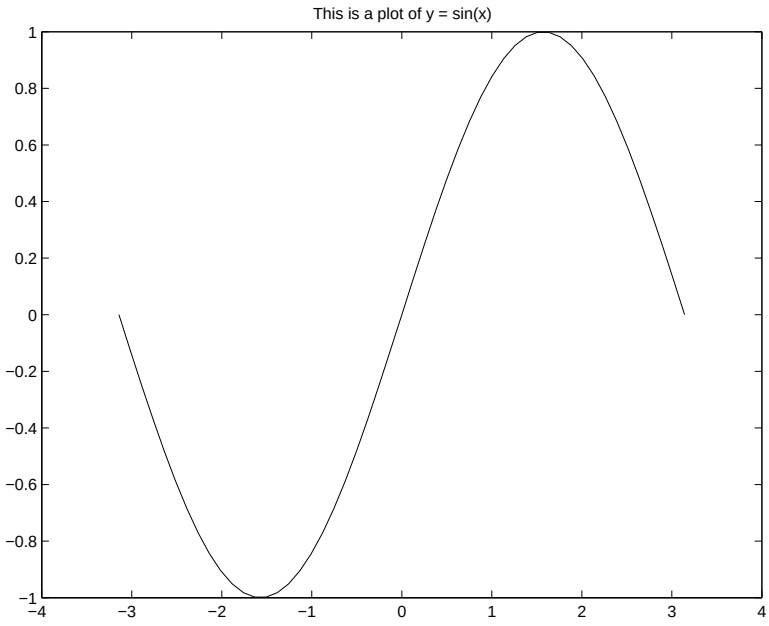

Figure 9: Simple plot with a title.

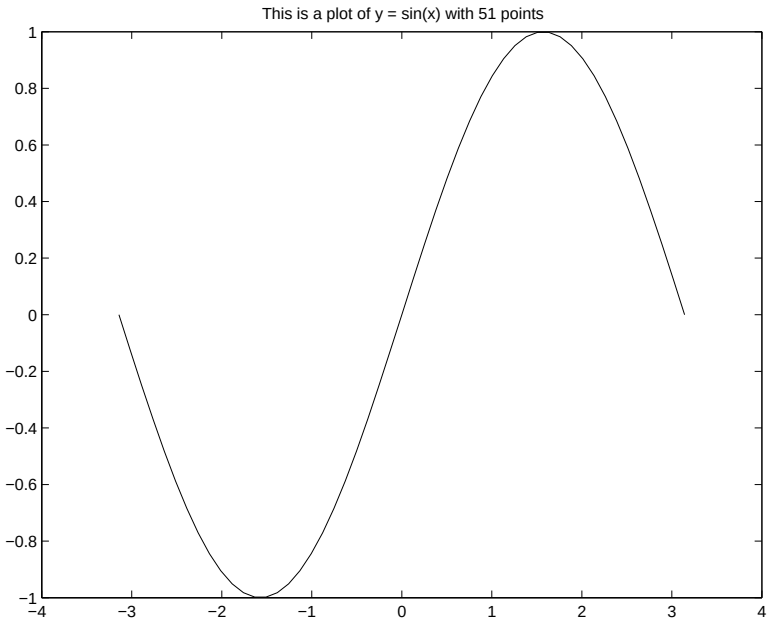

Figure 10: Simple plot with a more complex title.

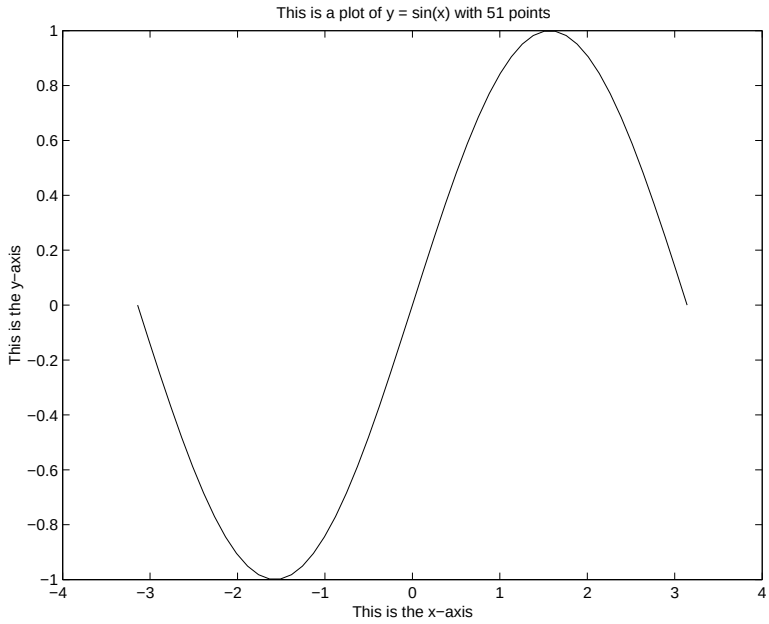

Figure 11: Simple plot with a more complex title and axis labels.

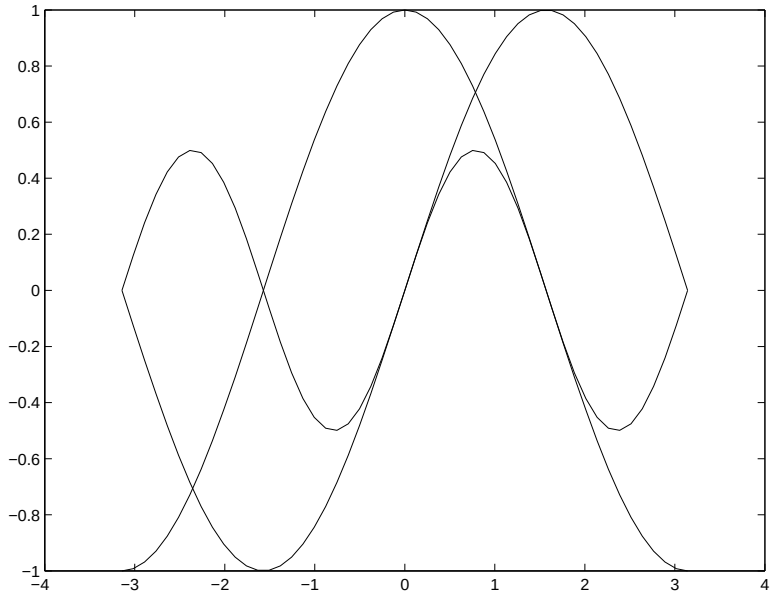

Figure 12: Multiple graphs on one set of axes.

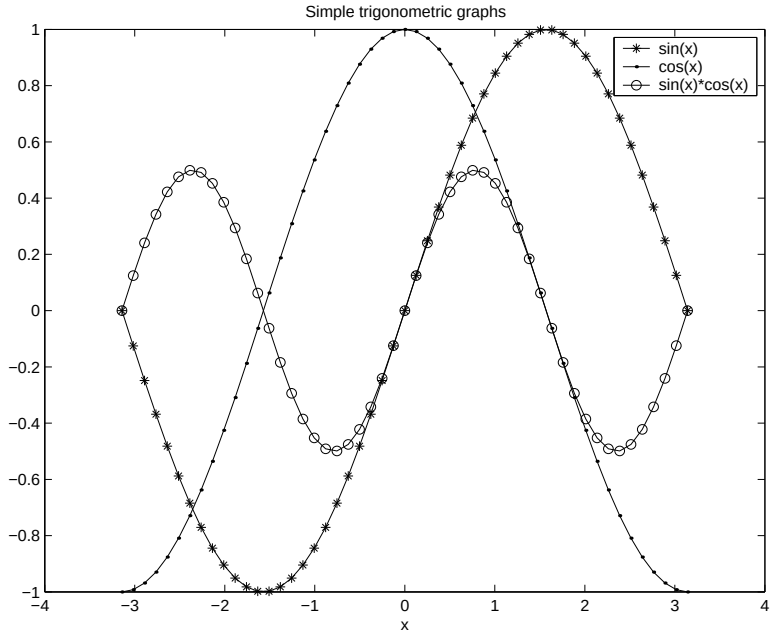

Figure 13: Better version of multiple graphs on one set of axes.

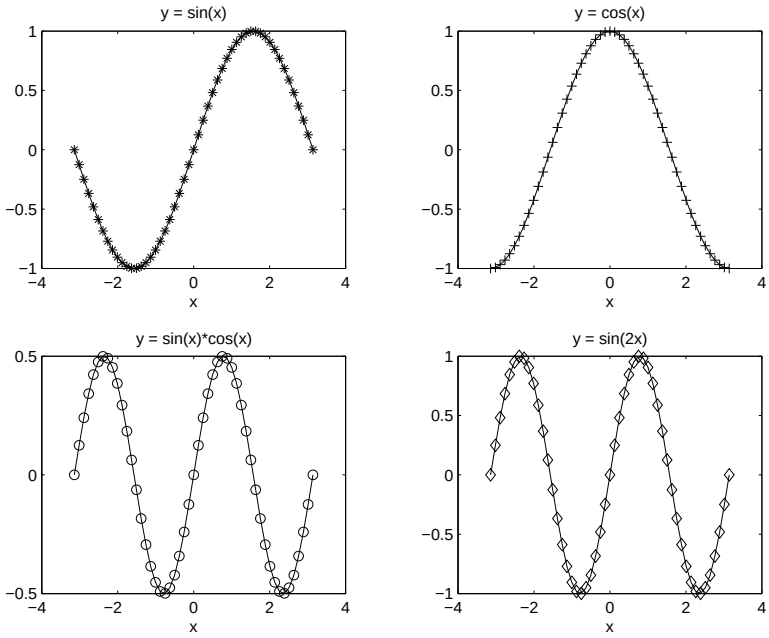

Figure 14: Example of the subplot command.

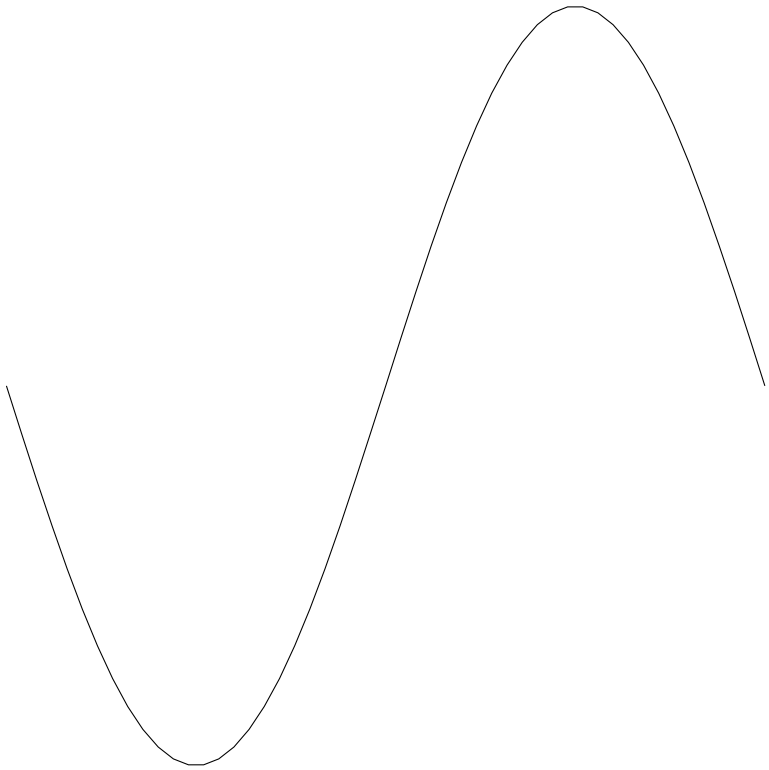

Figure 15: Turning the axis off.

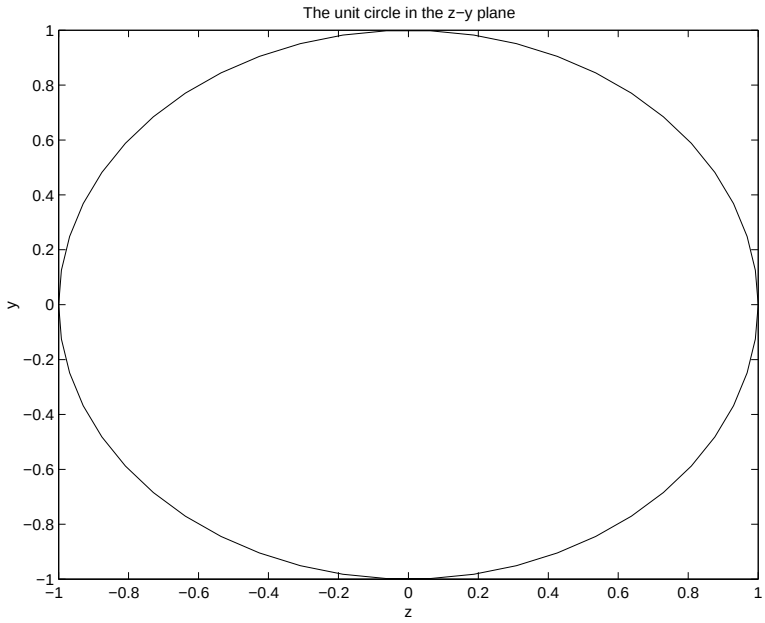

Figure 16: A skewed circle.

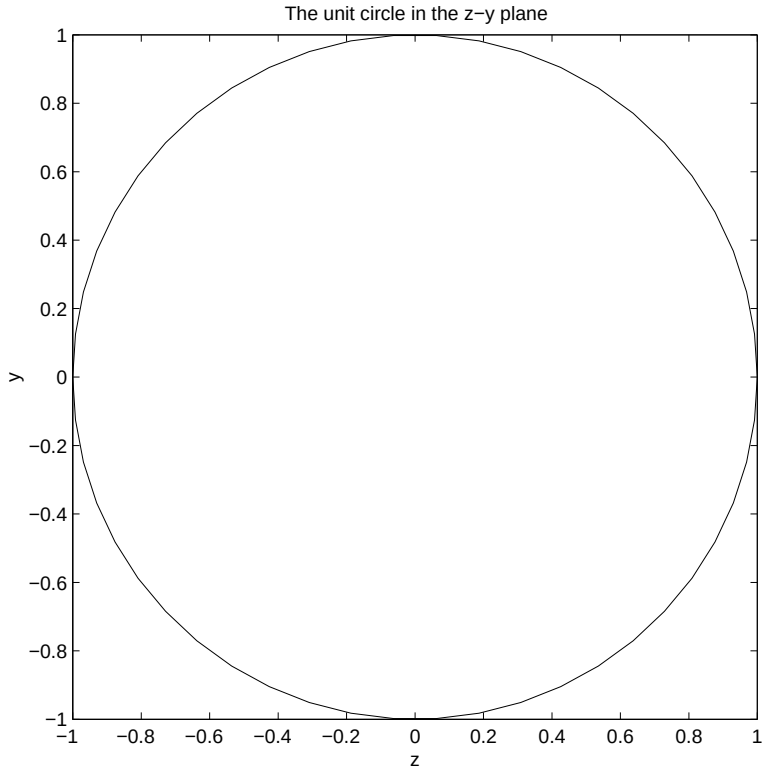

Figure 17: A circle with the proper aspect ratio.

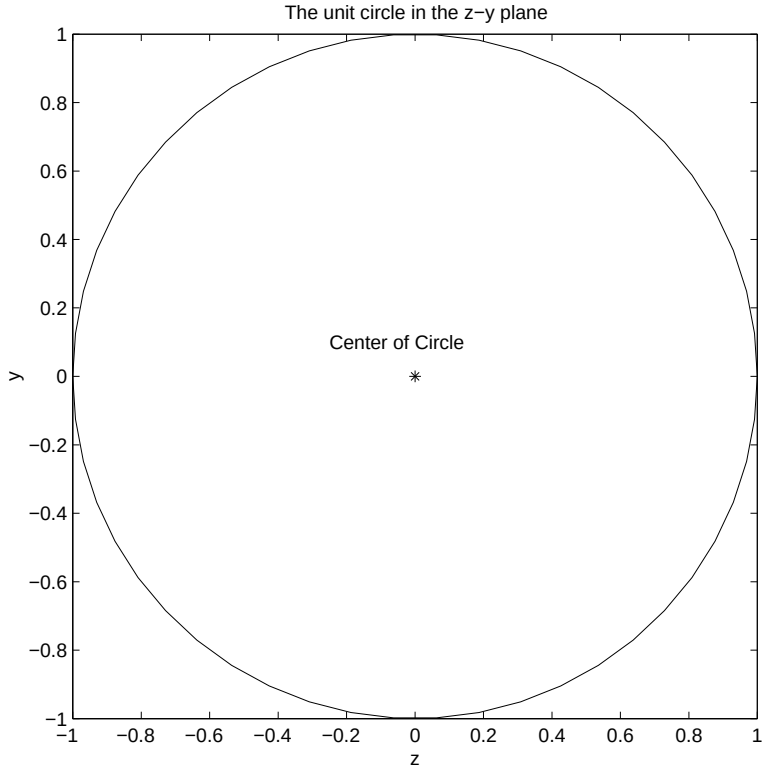

Figure 18: Placing additional text on the graph.

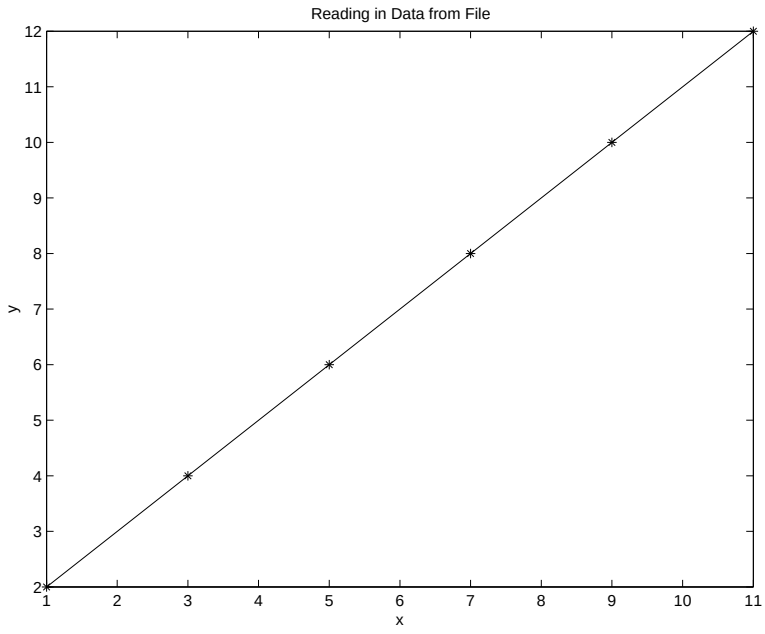

Figure 19: Plotting data from an external file.