MATLAB Graphics - 2D Plotting

Leigh J. Little SUNY Brockport September 13, 2021

Contents

1 Script Files

We have seen that you can type your MATLAB commands at the command line. This is very useful because you can immediately see the results of calculations (by leaving off the (;) at the end).

However, this is a problematic way to manage a very involved calculation. Suppose you did a calculation that involved 20 steps and discovered you made an error in Step 5. You can use the arrow key to bring up the previous calculations again, but this is tedious. It also doesn't help if you then discover an error with Step 11. It also does not help if you discover that you need to perform the calculation again a few days later.

We need a better way to handle situations where there are many steps involved in a calculation. We have a powerful computing tool in MATLAB and it is a good idea to let it handle the heavy lifting. The solution to this is a *script file*.

A script file is simply a set of MATLAB commands that you type into a file instead of the command line. The commands in the script file will be executed when you *run* the script file. Now if you discover an error at Step 5, you only need to modify the script file to correct the error, then re-run the script file. If you need to run the same set of commands a few days later, you only need to start MATLAB and run the script file again.

Here are the general guidelines for using script files

- A script file must end with a `.m` extension. Note that this is the same extension used by Mathematica so be careful about double clicking on `.m` files when you are using the campus computers.
- You need to choose a name for your script file. This must not be the same as an existing MATLAB command, variable name or another script file that is in your current working directory.

- The script file must be in the same working directory that MATLAB is in (technically this is not 100% correct, but we will do this for now).
- To run the script file, you type in the name of the script file at the command line (minus the `.m` extension). For example, if you created a script called `myscript.m`, you would type

  ```
  >> myscript
  ```

  to run the script.

2 Basic Plotting

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.

2.1 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 artificially 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 = linspace(-pi,pi,51)';
>>  y = sin(x);
```

x is a column 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 last 3 commands are the *plot specification* (or *plot spec*). In general, the plot spec consists of 1 to 4 characters indicating the plot color, point 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 data 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 active 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 (note, this is just a special case of the `plot command` and it not used that frequently). 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 triangle with vertices

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

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

```
>> x = [0 1 1 0]';
>> y = [0 0 1 0]';
>> line(x,y)
```

The resulting plot is shown in Figure 8. Note that an extra set of coordinates is needed to 'close' the triangle. Also, due to the auto scaling of the axis limits, you can only see the diagonal of the triangle. See `help line` for more information.

2.3 The Plot Window

Every plot you create will be encased in a plot window with a menu bar and toolbar. The menu bar 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 menu bar items. See Figure 9.

Note that this plot window has `Figure 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 either the menu pulldown or the print command. For example:

```
>> print -dpdf figurename.pdf
```

will output the active figure window as a .pdf to the file `figurename.pdf` while

```
>> print -djpeg figurename.jpg
```

will output the active figure window as a .jpg image to the file `figurename.jpg`. There are many other graphics formats supported, but variations on the .pdf, .jpg and .png formats 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 = linspace(-pi,pi,51)';
>> y = sin(x);
>> plot(x,y)
>> title('This is a plot of y = sin(x)')
```

This is shown in Figure 10. 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 11. 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 12.

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 Figure 13. 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 an identifier to each of 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 14. 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,w,'go-')
>>   title('w = sin(x)*cos(x)')
>>   xlabel('x')
>> subplot(2,2,4)
>>   plot(x,v,'md-')
>>   title('v = sin(2*x)')
>>   xlabel('x')
```

The plot is shown in Figure 15. 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 9, 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,w,'go-')
>>   title('w = sin(x)*cos(x)')
>>   xlabel('x')
>> figure(4)
>>   plot(x,v,'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 16. 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 limits to the interval $[-2, 2]$ and the y-axis limits to the interval $[-\frac{1}{2}, \frac{1}{2}]$.

Scaling of axes is sometime required. By default, MATLAB will auto scale 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 17. 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 equal
```

This is shown in Figure 18. 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(0,0,'*')
>> text(-0.3,0.1,'Center of Circle')
```

This is shown in Figure 19. This plots a `*` at the coordinates $(0, 0)$ and puts the text message at the coordinates $(-0.3, 0.1)$. 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.

7 External Data and the Load Command

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 can be a very complex process depending on how the data you need to import is stored.

In many situations, you can use the `load` command. This command will load the numbers from an external data file into a MATLAB variable. In order to use the load command:

a) The data file should be a plain text file.

b) The data file can contain only numerical values.

c) Each row of data needs to have the same number of values.

d) There should not be any blank lines between rows of data.

. A sample use of the `load` command would look like

```
load('testdata.dat');
```

If the data file satisfies the criteria, a MATLAB variable called `testdata` will be created. This variable will contain the values in the file.

An alternative way to use the `load` command is to do

```
A = load('testdata.dat');
```

In this case, the values in the file would be stored in the variable `A`.

Create a file called `testdata.dat` (use the MATLAB editor to do this) in your working directory that contains the following data:

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

Be sure to save the file after entering the values.

Load this into the MATLAB environment by entering:

```
>> load('testdata.dat')
```

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 (recall that `testdata(:,1)` is column 1 of `testdata` and `testdata(:,2)` is column 2).

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

The resulting plot is shown in Figure 20.

8 Problems for MATLAB 2D Graphics

1) a) Plot the parametric curve

$$x(t) = 2\cos(t),\ y(t) = 3\sin(t),\ t \in [0, 2\pi].$$

Do this by dividing the t interval into an equally spaced array, computing x and y at each value of t, then plotting x versus y.

b) What is this figure usually called? Make sure that each axis has the same scaling.

c) Now plot the parametric curve

$$v(t) = 3\cos(t),\ w(t) = 2\sin(t),\ t \in [0, 2\pi]$$

on the same set of axes as the curve from part a).

d) The 2 curves should intersect at 4 points. Find the coordinates of these points accurate to 2 significant digits (can the symmetry of the problem be exploited?). To do this, click on the + magnifying glass (on the plot window toolbar) and zoom in on an intersection point. How can you be sure that you have found the required accuracy?

e) Use your answer from d) to plot a '*' at each of the intersection points.

f) Save this plot as a jpeg image file and print it out.

2) Plot the following sequences of curves on the intervals indicated. All the graphs for each subpart 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) $x^2-2,\ x^2-1,\ x^2+1,\ x^2+2,\ \ x \in [-2, 2]$.

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

9 Figures

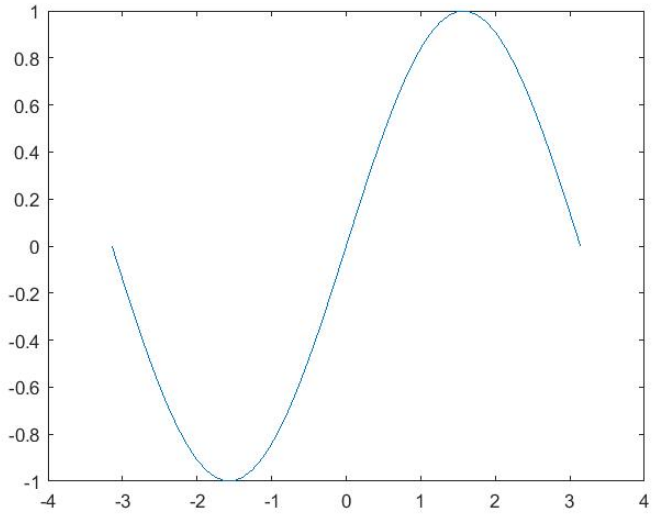

Figure 1: Simple plot of $y = \sin(x)$.

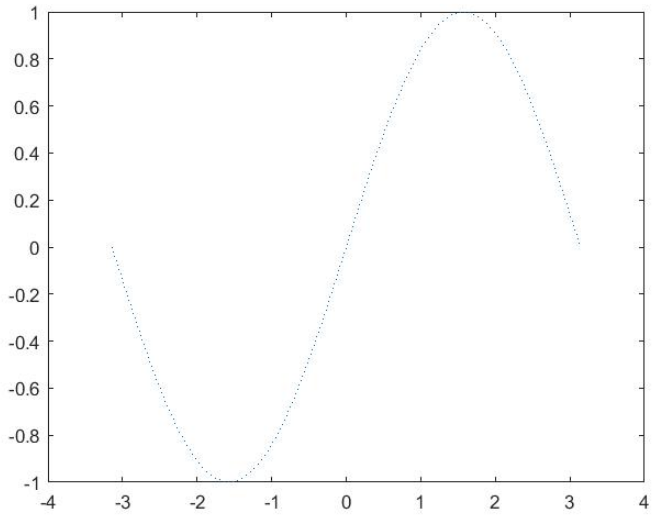

Figure 2: Simple plot of $y = \sin(x)$, dotted line.

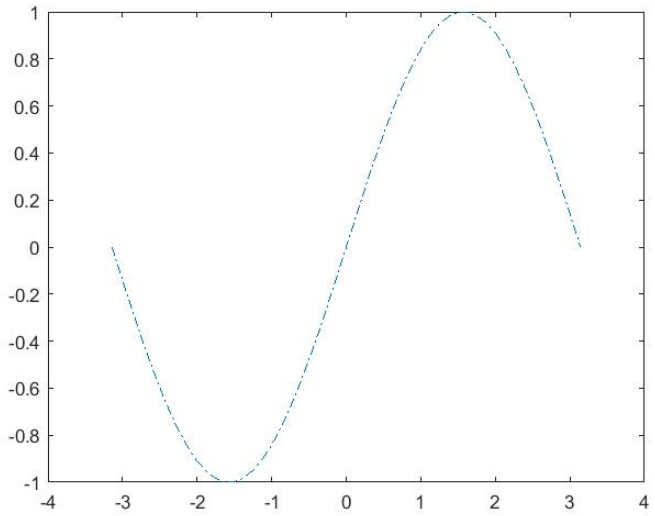

Figure 3: Simple plot of $y = \sin(x)$, dashed line.

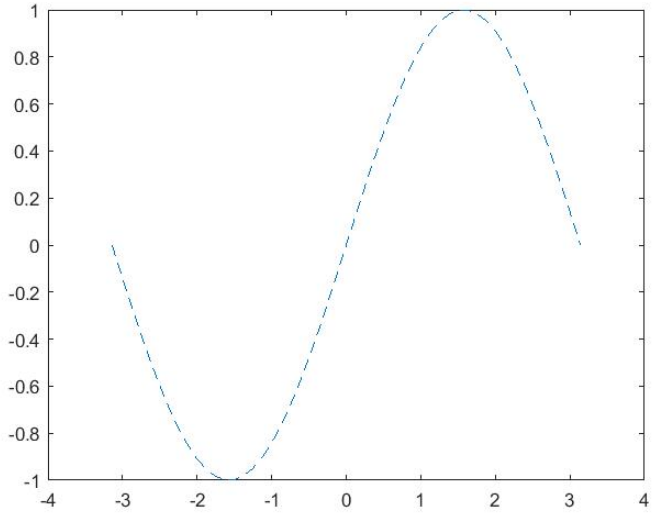

Figure 4: Simple plot of $y = \sin(x)$, dash-dot line.

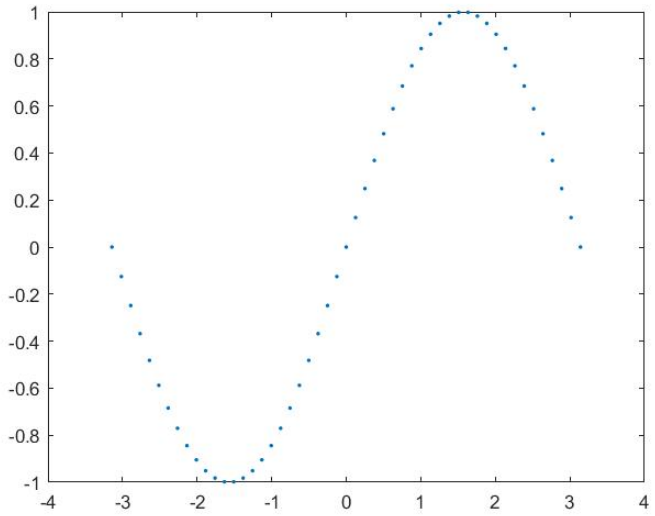

Figure 5: Simple plot of $y = \sin(x)$, mark points with dots.

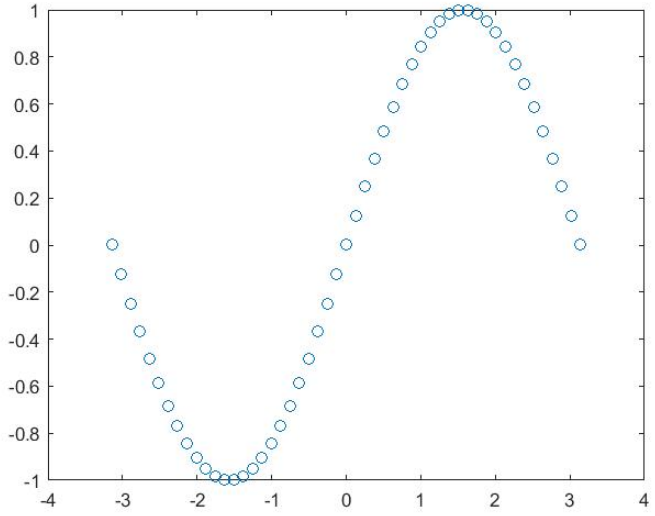

Figure 6: Simple plot of $y = \sin(x)$, mark points with o's.

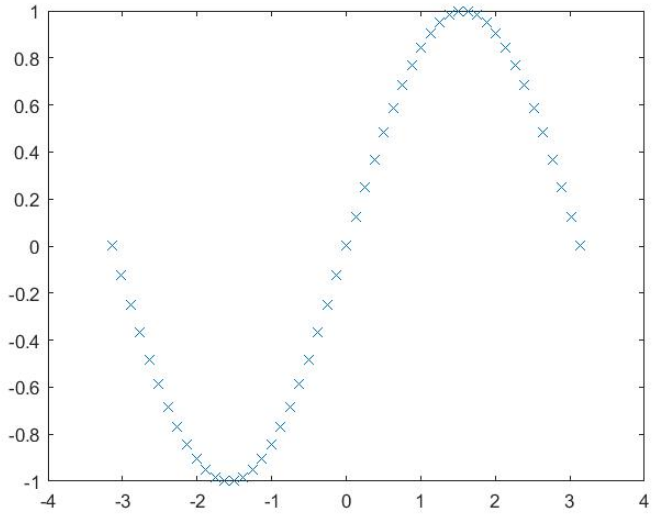

Figure 7: Simple plot of $y = \sin(x)$, mark points with x's.

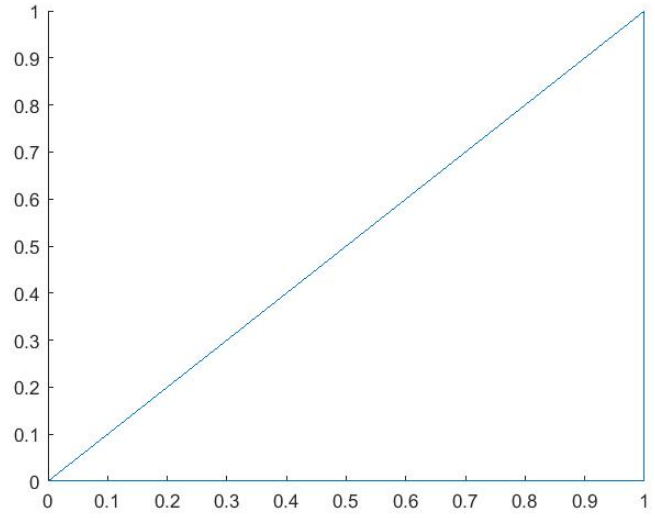

Figure 8: Plotting a triangle with the line command.

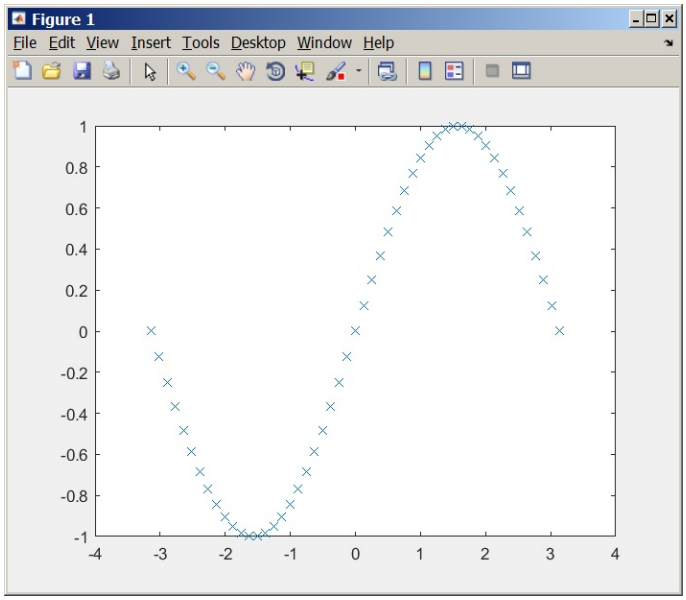

Figure 9: The entire plot window for a figure.

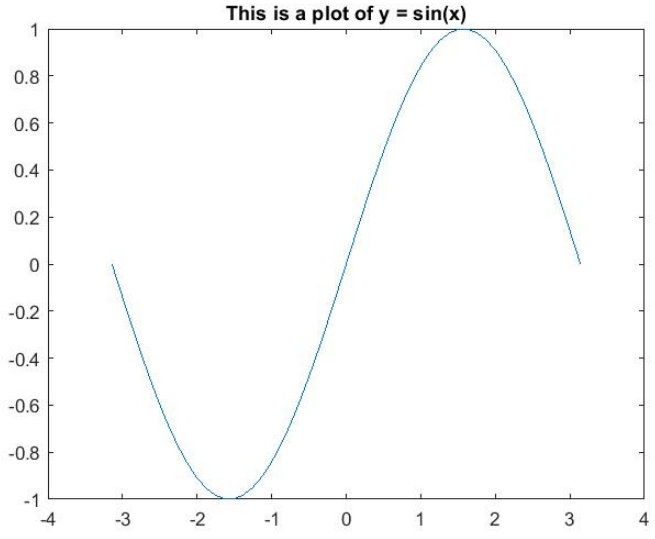

Figure 10: Simple plot with a title.

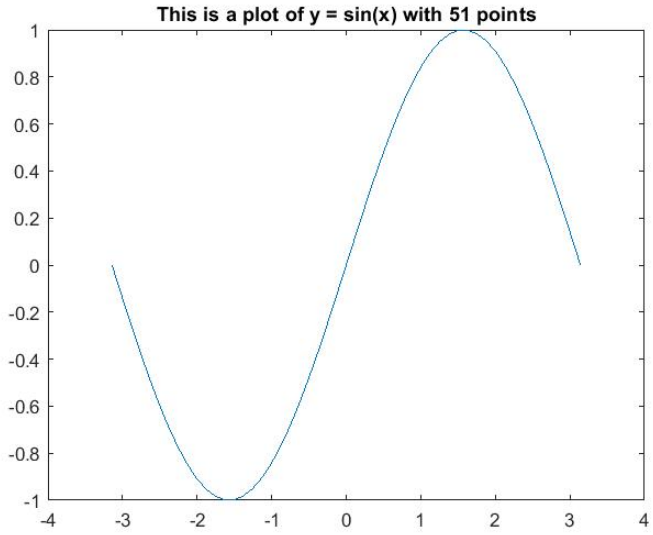

Figure 11: Simple plot with a more complex title.

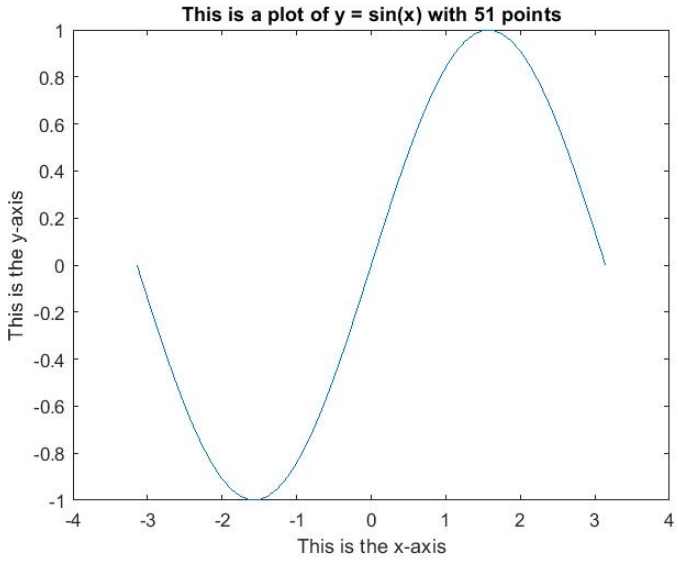

Figure 12: Simple plot with a more complex title and axis labels.

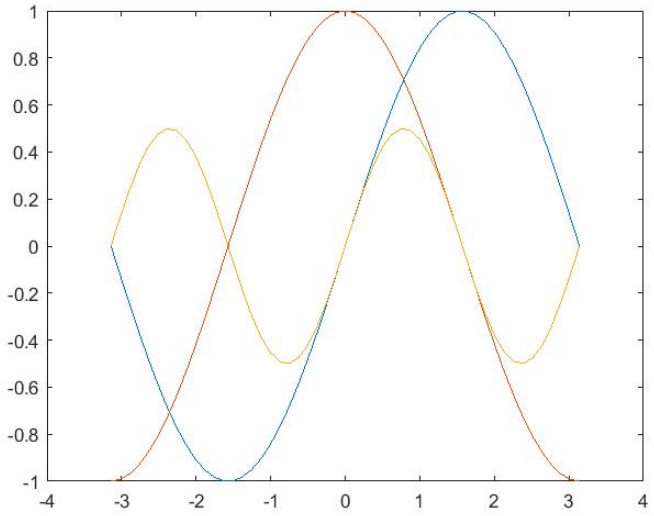

Figure 13: Multiple graphs on one set of axes.

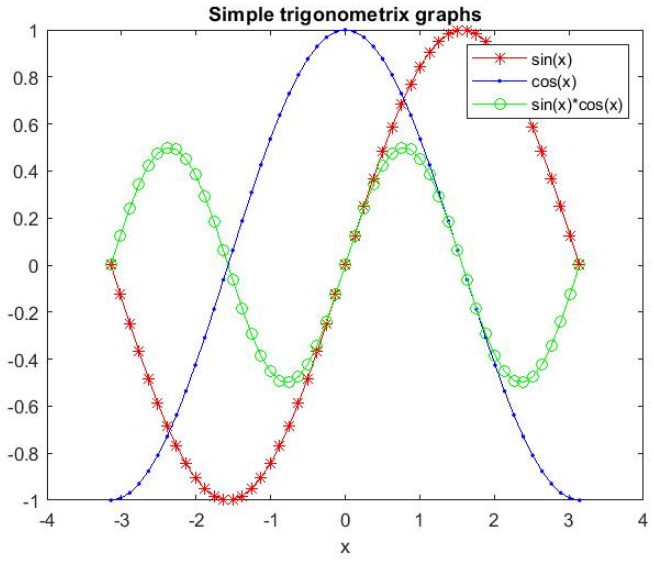

Figure 14: Better version of multiple graphs on one set of axes.

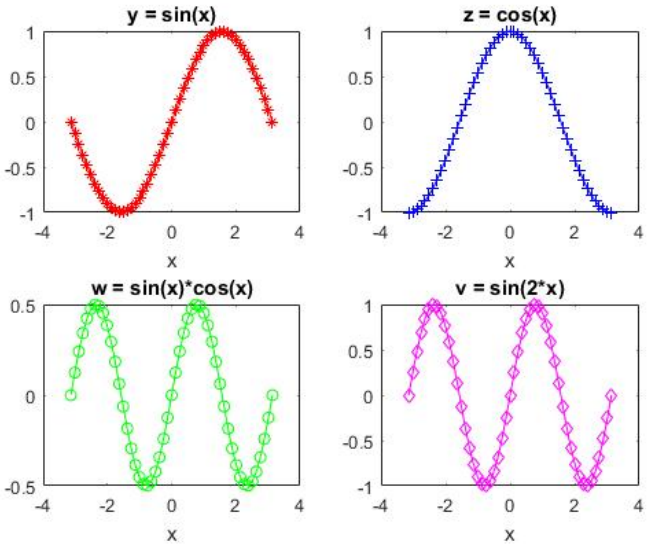

Figure 15: Example of the subplot command.

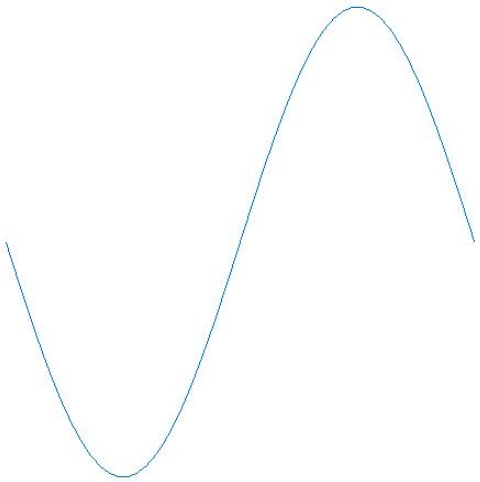

Figure 16: Turning the axis off.

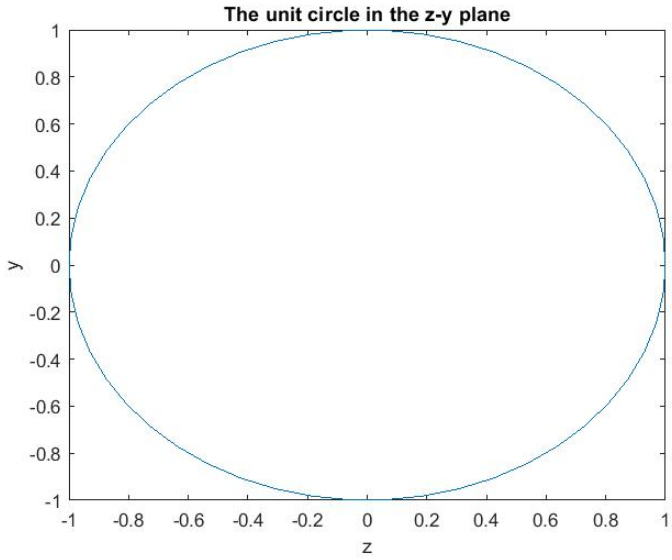

Figure 17: A skewed circle.

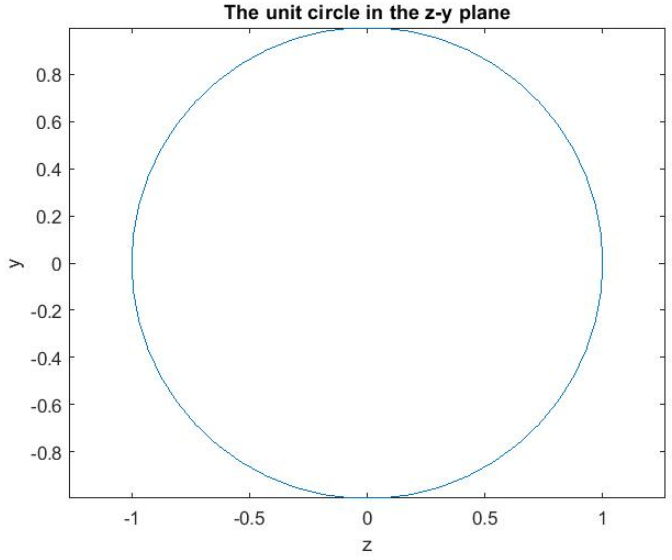

Figure 18: A circle with the proper aspect ratio.

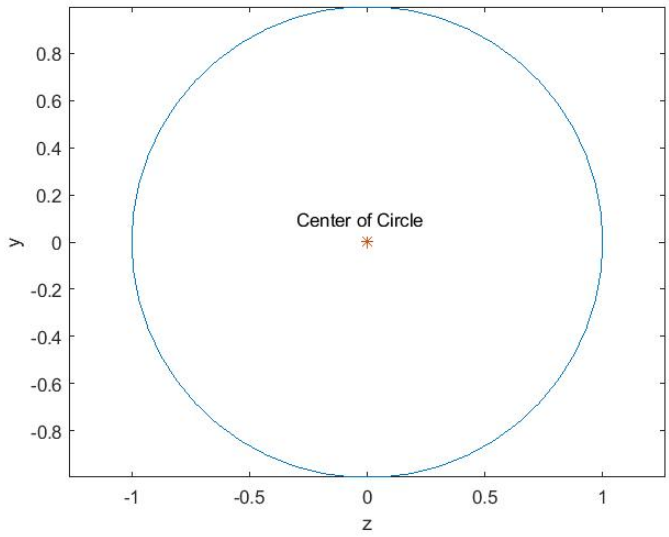

Figure 19: Placing additional text on the graph.

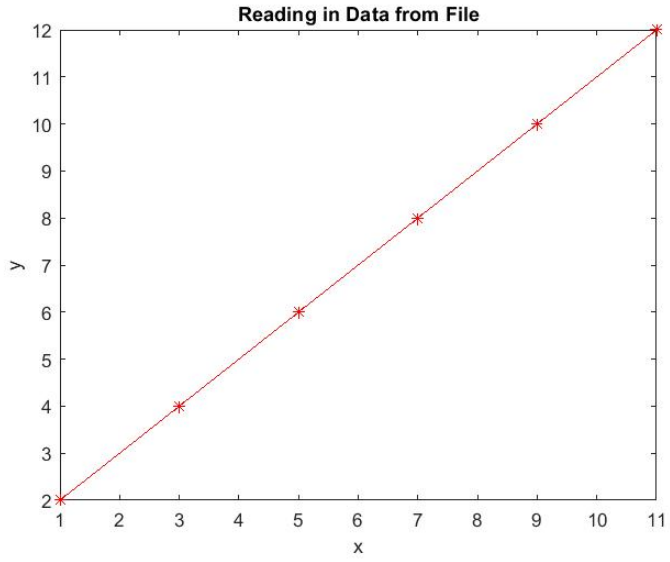

Figure 20: Plotting data from an external file.