Plotting
in
Matlab

Math 50C — Multivariable Calculus

David Arnold

David-Arnold@Eureka.redwoods.cc.ca.us

Abstract

In this activity you will learn how to plot lines and curves in both the plane and space. In addition, you will also be introduced to script files. *Prequisites: Some knowledge of how to enter vectors and matrices in* Matlab. *Some familiarity with* Matlab*'s array operations is useful.*

Table of Contents

Introduction

This is an interactive document designed for online viewing. We've constructed this onscreen documents because we want to make a conscientious effort to cut down on the amount of paper wasted at the College. Consequently, printing of the onscreen document has been purposefully disabled. However, if you are extremely uncomfortable viewing documents onscreen, we have provided a print version. If you click on the Print Doc button, you will be transferred to the print version of the document, which you can print from your browser or the Acrobat Reader. We respectfully request that you only use this feature when you are at home. Help us to cut down on paper use at the College.

Much effort has been put into the design of the onscreen version so that you can comfortably navigate through the document. Most of the navigation tools are evident, but one particular feature warrants a bit of explanation. The section and subsection headings in the onscreen and print documents are interactive. If you click on any section or subsection header in the onscreen document, you will be transferred to an identical location in the print version of the document. If you are in the print version, you can make a return journey to the onscreen document by clicking on any section or subsection header in the print document.

Finally, the table of contents is also interactive. Clicking on an entry in the table of contents takes you directly to that section or subsection in the document.

Working with Matlab

This document is a working document. It is expected that you are sitting in front of a computer terminal where the Matlab software is installed. You are not supposed to read this document as if it were a short story. Rather, each time your are presented with a Matlab command, it is expected that you will enter the command, then hit the Enter key to execute the command and view the result. Furthermore, it is expected that you will ponder the result. Make sure that you completely understand why you got the result you did before you continue with the reading.

Getting Started

MATLAB excels at plotting graphs of functions. Much of this manual is dedicated to drawing the graphs of the various functions encountered in a multivariable calculus course. However, you have to walk before

you can run, so we will begin with basic plotting techniques and move toward more advanced technique in subsequent chapters.

Functions of a Single Variable

In this section we will learn how to draw the graph of a single-valued function of one real variable; i.e., a function defined by a rule or equation such as $y = f(x)$. We begin with a definition.

Definition 1

Let $f : \mathbf{R} \to \mathbf{R}$ be defined by the rule $y = f(x)$. The **graph** of f is defined as

$$\text{Graph of } f = \{(x, f(x)) : x \text{ is in the domain of } f\}.$$

Alternatively, the graph of f is the set of all points (x, y), such that $y = f(x)$ for some x in the domain of f.

The idea is simple: the graph of f is the set of all points (x, y) that satisfy the equation $y = f(x)$. Consequently, a standard graphing technique is to create and plot table of points that satisfy the equation. An example should make this clear.

Example 1

Use MATLAB to sketch the graph of $f(x) = x^2$.

If we were plotting this graph by hand, we would start by selecting values for the independent variable x that lie in the domain of f. We would place these values in the first column of a table, similar to **Table 1**. Then, for each value of x, we would calculate the corresponding function value with equation $f(x) = x^2$, placing the results in the second column of the table as we work. Finally, we would set up a coordinate system on a sheet of graph paper and plot each of the points in **Table 1**.

x	-3	-2	-1	0	1	2	3
$f(x) = x^2$	9	4	1	0	1	4	9

Table 1

But how would one accomplish this task in MATLAB? We could begin by creating a list of x-values.

```
>> x=[-3 -2 -1 0 1 2 3]
x =
    -3    -2    -1     0     1     2     3
```

Next, we wish to square each value in this list. The natural thing to try is xˆ2, but that won't work.

```
>> xˆ2
```

Remember, in MATLAB, xˆ2 means x*x. You cannot multiply these lists because their dimensions won't accommodate matrix multiplication. However, we don't want to square the list; we want to *square each element in the list.* That's what *array exponentiation* is for!

```
>> y=x.ˆ2
y =
     9     4     1     0     1     4     9
```

There are two things to note: (1), array exponentiation squared each element of the list x, and (2), we've stored the result of this operation in the variable y for future use. This allows us to easily plot each of the data points provided in the lists x and y. The command

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

will plot the image shown in **Figure 1** (a).

The command plot(x,y,'o') warrants some explanation. The command plot(x,y,s) plots list y versus list x; i.e., MATLAB plots all ordered pairs (u, v), where u is taken from list x and v is taken from list y. The character string s determines the symbol used for plotting. In our case, s='o', which instructs MATLAB to use a small, unfilled circle to plot each point. Other plotting symbols are available. For example, try plot(x,y,'+'), plot(x,y,'*'), or plot(x,y,'s'). A full list of plotting symbols is given in the help file for the plot command.[1]

Thus far, we've only plotted seven points that satisfy the equation $f(x) = x^2$, but **Definition 1** requires that we plot *all* points satisfying the equation $y = f(x)$. We need to plot more points.

---

[1]For a full explanation of the plot command, type help plot at the MATLAB prompt and hit the **Enter** key.

MATLAB provides the construct `start:increment:stop` for creating lists of equally spaced values. The command[2]

```
>> x=-3:.5:3
x =
  Columns 1 through 7
   -3.0000   -2.5000   -2.0000   -1.5000   -1.0000   -0.5000        0
  Columns 8 through 13
    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
```

creates a list whose first entry is -3, the second entry is -2.5, and so on, up to and including (in this case) 3.[3] Square each element in the list x and store the result in the list y.

```
>> y=x.^2
y =
  Columns 1 through 7
    9.0000    6.2500    4.0000    2.2500    1.0000    0.2500        0
  Columns 8 through 13
    0.2500    1.0000    2.2500    4.0000    6.2500    9.0000
```

Now you can plot the graph, as shown in **Figure 1** (b), with

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

We still haven't plotted *all* points satisfying the equation $f(x) = x^2$, so we're not finished. The commands

```
>> x=-3:.1:3;
>> y=x.^2;
>> plot(x,y,'o')
```

increment entries in the vector x by a tenth and produce the plot in **Figure 1** (c).

---

[2]The output of this command warrants some explanation. The vector x has length 13 (check this with `length(x)`), but it is a *row* vector, one row and thirteen columns. Of course, there is not enough room to fit thirteen columns across the screen, so part of the list is put on a second row, with a heading to indicate that you are seeing columns 8 through 13. If you find this terribly distracting, try taking the transpose, x=x', then enter y=x.^2.

[3]Some times incrementing will fall short of the stop value, as in x=-3:.7:3. Try it!
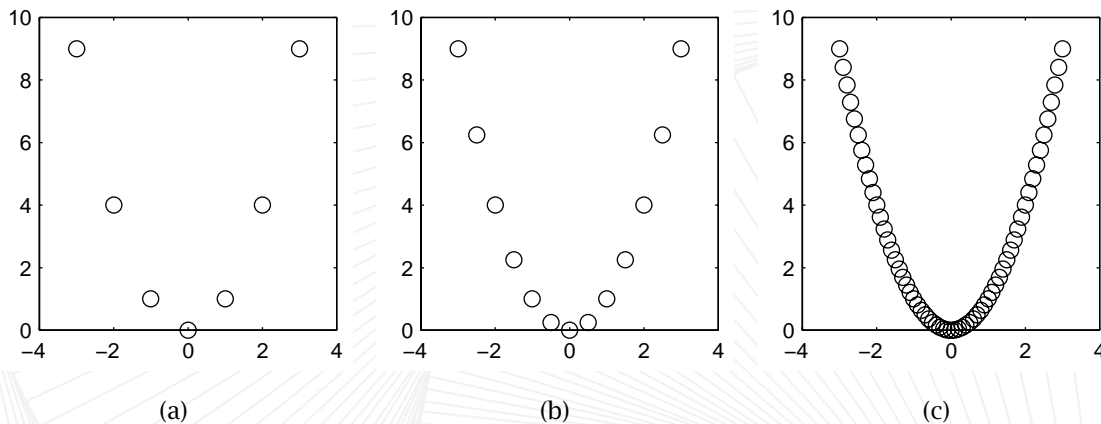
Figure 1 Plotting the graph of $f(x) = x^2$.

Clearly, we can't continue in this manner indefinitely. At some point we are going to have to make an assumption about where the rest of the points satisfying $y = f(x)$ will fall.

You can instruct MATLAB to connect points in your plot with line segments. The command plot(x,y,'o-') was used to produce the graph in **Figure 2** (a). MATLAB's default plotting command, plot(x,y), connects the points created from list x and y with line segments. If you plot enough points, you will get a smooth looking curve, as shown in **Figure 2** (b), but if you don't plot enough points, then plot(x,y) will produce a kinky plot like the one shown in **Figure 2** (c). It is important to note that only the points created by the list x and y satisfy the equation $y = f(x)$. In between points, MATLAB is approximating the actual graph with line segments.

There are a number of colors, plotting symbols, and line styles available for use with the plot command. The idea is simple: in the command plot(x,y,s), s is a character string used to select the color, plotting symbol, and line style. For example, plot(x,y,'gd-') will use green diamonds for the plotting symbols, then connect each pair of points with a green line segment. Order is irrelevant, so plot(x,y,'-gd') produces exactly the same image. If you don't select a plotting symbol, then one won't be used. For example, plot(x,y,'r--') will connect each pair of points with a red, dashed line segment. A full list of
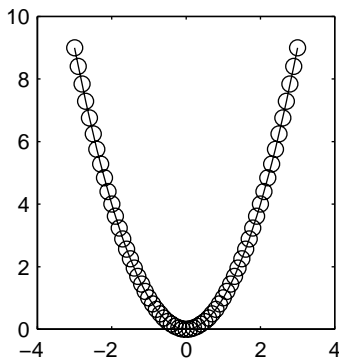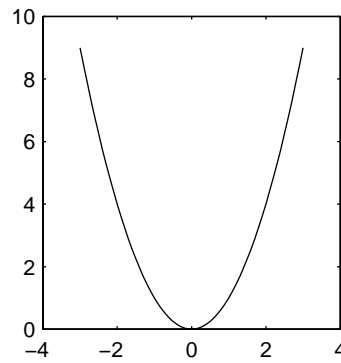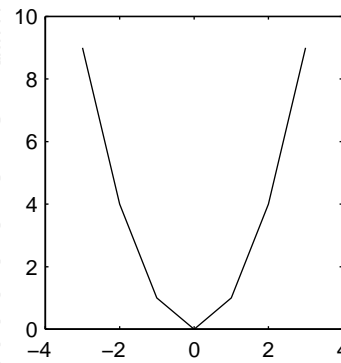
Figure 2 Plotting the graph of $f(x) = x^2$.

colors, plotting symbols, and line types is available in the help file for the `plot` command.[4]

Example 2

Use MATLAB to draw the graph of $f(x) = x^2 - 2x - 3$ on the interval $[-1, 3]$.

Users new to MATLAB always struggle with array operations at first. It can be discomforting to work with entire arrays of numbers rather than with the individual numbers themselves. Let's take a slightly different tack on this example in the hopes of adding some clarity to array operations in MATLAB.

Suppose that we enter the numbers $x_1, x_2, \ldots, x_n$ into the equation of the function f to produce the data in **Table 2**.

Now, let's look carefully at the vector `y=x.^2-2*x-3`. Technically, we immediately have a problem, because you cannot subtract the scalar number 3 from a vector. However, since this type of operation is so common, MATLAB is forgiving and allows it. In fact, if you subtract three from a vector in MATLAB, MATLAB will subtract 3 *from each entry of the vector.*[5] Consequently, the computation

---

[4]Type `help plot` at the MATLAB prompt.
[5]For example, try `[3,3,3]-3` and see what happens.

x	$f(x)$
x_1	$x_1^2 - 2x_1 - 3$
x_2	$x_2^2 - 2x_2 - 3$
$\vdots$	$\vdots$
x_n	$x_n^2 - 2x_n - 3$

Table 2

$$y = \text{x.}\hat{}2 - 2*\text{x} - 3 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} .\hat{}2 - 2* \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 3 = \begin{bmatrix} x_1^2 - 2x_1 - 3 \\ x_2^2 - 2x_2 - 3 \\ \vdots \\ x_n^2 - 2x_n - 3 \end{bmatrix} .$$

produces the second column of **Table 2** and stores it in the variable y. Hopefully, it should now be clear why the commands

```
>> x=-1:.1:3;y=x.^2-2*x-3;
>> plot(x,y)
```

produce the graph shown in **Figure 3**.

It is important to label the axes of your graph. You might also need a grid and a title. This is easily done in MATLAB. The commands

```
>> xlabel('x-axis')
>> ylabel('y-axis')
>> title('The graph of f(x) = x^2 - 2x - 3')
>> grid on
```

produce the axes labels, title,[6] and grid shown in **Figure 3**.

---

[6]MATLAB recognizes a small subset of TEX commands. So typing title('x^2') produces a superscript, while title('x_2') produces a subscript. If you need more than one character in your sup/subscript, enclose it in curly braces, as in \title('x^{2\alpha}').
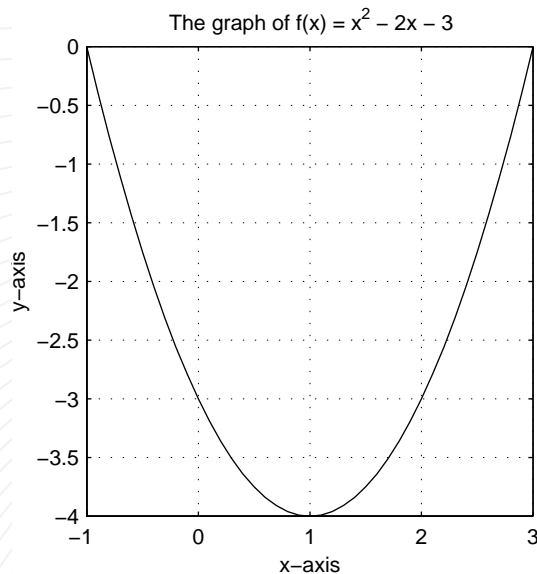
Figure 3 The graph of $f(x) = x^2 - 2x - 3$.

Two or More Plots

The `plot` command can draw more than one graph at a time. If you have data in `x1,y1,x2,y2,...,xn,yn`, then the command `plot(x1,y1,s1,x2,y2,s2,...,xn,yn,sn)` will plot the data using the line style `s1` for the data in `x1, y1`, etc.

Example 3

Sketch the graphs of $f(x) = 1/x$ and $g(x) = \ln(x - 1)$ on the interval $[2, 5]$.

Use the functions f and g to calculate data for the two plots.

```
>> x=2:.1:5;
>> f=1./x;
>> g=log(x-1);
```

Two comments are in order. First, MATLAB's command for the natural logarithm is `log`.[7] Secondly, it is highly illegal to divide a scalar by a vector, as any mathematician will tell you. So the command, `f=1./x` is perplexing, to say the least. Technically, we should enter `ones(size(x))./x`, which makes a lot more sense, but this is such a common operation that MATLAB makes a special interpretation in this case. That is, MATLAB takes `1./x` to mean: "form a vector whose first element is computed by dividing 1 by `x(1)`, whose second element is computed by dividing 1 by `x(2)`, etc., exactly what we want.

Now, the command

```
>> plot(x,f,x,g)
```

will produce a plot containing the graphs of both f and g on the desired interval. MATLAB also cycles through a default set of colors, automatically coloring each graph with a different color. The command

```
>> plot(x,f,'-',x,g,'--')
```

plots the graphs of f and g with different linestyles, f with a solid line, g with a dotted line, useful if you don't have a color printer. Finally, when plotting more than one graph, it is good form to create a legend to help distinguish the plots. The command

```
>> legend('y = f(x)','y = g(x)')
```

was used to create the image shown in **Figure 4**.

Formatting Plots

The latest version of MATLAB[8] offers interactive formatting designed to help with the annotation of MATLAB plots. For those of you who own the latest version of MATLAB, this feature is mostly self-explanatory and needs little explanation.

After plotting in a figure window, the figure toolbar is usually present by default. If not, you can activate it by selecting `Figure Toolbar` from the `View` menu in your figure window. Icons for adding text, arrows,

---

[7]Base two and base ten logarithms are also available as `log2` and `log10`.
[8]MATLAB 6.0 as of this writing. You can determine your version of MATLAB by typing `ver` at the MATLAB prompt.
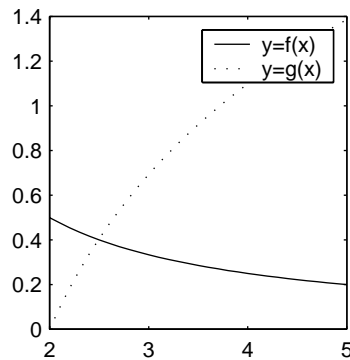
Figure 4 The graphs of $f(x) = 1/x$ and $g(x) = \ln(x - 1)$.

and line segments are available and clicking the arrow to the left of the "insert text" icon enables plot editing.[9] Once this arrow is depressed, double-clicking on an object in a figure window opens the property editor where changes can be made to the object's properties. You can also right-click an object and select `Properties` from the resulting popup menu. Popup menus usually also list a number of frequently used formatting features for the selected object.

Most users find it a simple matter to teach themselves about the plot editing through trial and discovery, but you can also select `Formatting Graphs` from the `Help` menu in your figure window to get a full tutorial on the use of this wonderful formatting tool.

When you finish with your plot annotation, disable the Plot Editing tool by single-clicking the plot editing arrow on the toolbar (this icon acts as a toggle, click once to enable, click again to disable) or unchecking the `Edit Plot` item on the `Tools` menu.

Parametric Equations

Rather than express y as a function of x, it is often advantageous to express x and y in terms of another variable such as t, called a *parameter*. What follows is called a set of *parametric equations*.

---

[9]Select `Edit Plot` from the `Tools` menu will also enable plot editing.

$$x = |t|$$
$$y = |1 - t|$$

First, arbitrarily choose some t-values in a domain common to each function. Place your values of t in a table, then calculate the value of x and y for each value of the parameter t, placing your results in the appropriate columns of your table as your work. For example, if $t = -5$, then

$$x = |-5| = 5$$
$$y = |1 - (-5)| = 6$$

which we've recorded in **Table 3**, along with the corresponding x and y-values when $t = -4, -3, \ldots, 5$.

t	x	y	t	x	y
-5	5	6	1	1	0
-4	4	5	2	2	1
-3	3	4	3	3	2
-2	2	3	4	4	3
-1	1	2	5	5	4
0	0	1			

Table 3 Points satisfying $x = |t|$, $y = |1 - t|$.

This sort of work is tediously done by hand, but easily carried out in MATLAB. First, note that if no increment is provided in MATLAB's start:increment:stop structure, then MATLAB assumes an increment of 1.

```
>> t=-5:5
t =
    -5    -4    -3    -2    -1     0     1     2     3     4     5
```

We next calculate the corresponding values of x and y.

```
>> x=abs(t),y=abs(1-t)
x =
     5     4     3     2     1     0     1     2     3     4     5
y =
     6     5     4     3     2     1     0     1     2     3     4
```

Note that these values are identical to those found in **Table 3**.

Of course, we usually want to plot a lot more points, especially since MATLAB handles this chore with ease. The command

```
>> t=linspace(-5,5);
```

produces 100 equally spaced points from -5 to 5, including both -5 and 5 in the list. We've suppressed the output because of the large size of this list, but you should probably remove the semi-colons and examine the output carefully. The general form of the `linspace` command is `linspace(a,b,N)`, which produces N equally spaced points from a to b, including a and b in its output. If you don't provide a value for N, as in our `linspace(-5,5)`, MATLAB defaults to 100 equally spaced points. Calculate x and y with

```
>> x=abs(t);y=abs(1-t);
```

Again, you might want to remove the semi-colons to examine the output.

At this point, you have *three* choices of plots: x versus t, y versus t, and y versus x. The commands

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

produce the plots shown in **Figure 5** (a), (b), and (c), respectively.[10]
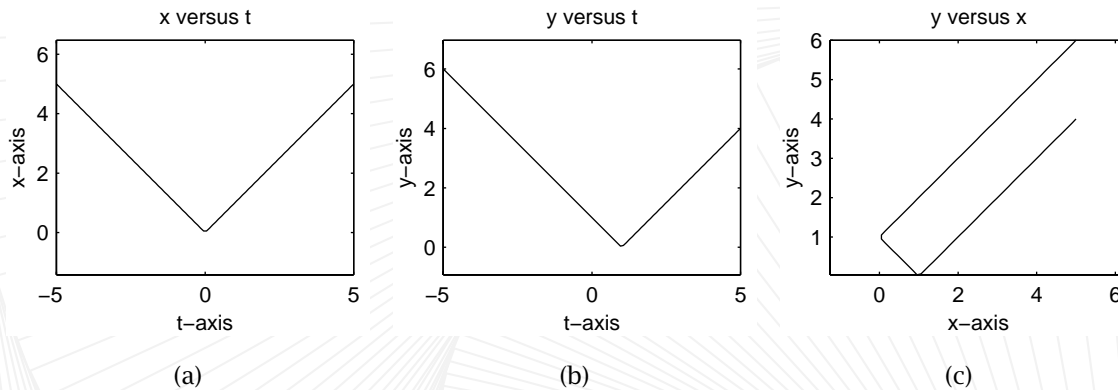
---

[10] If your instructor requires a data table, then you'll be happy with the output of the command [t',x',y'] Try it, then think about why it works! MATLAB's ability to build matrices out of matrices is one of its most powerful features.

Figure 5 Various plots of $x = |t|$, $y = |1 - t|$.

Script Files

It won't be long until you find yourself typing dozens of commands at the MATLAB prompt to produce a single plot. It is not uncommon in this situation to make a mistake, necessitating that you retype a sequence of commands. Even with the use of the up-arrow and MATLAB's command history window to replay commands, you'll soon find this is an inefficient way to work with MATLAB.

The best way to work with MATLAB is with script files and functions. A script file is simply a text file that contains a complete list of the commands that you intend to type at the MATLAB prompt. Perhaps an example will make this clear.

You can use any editor you wish to create your script file. Just remember to save your file in ASCII format with extension .m. However, the PC version of MATLAB contains a built-in editor with a number of features dear to the hearts of programmers. The easiest way to open this editor is to type edit at the MATLAB prompt, but you can also: (1), select New->M-file from the File menu, or (2), click the "New M-file" icon on the MATLAB toolbar. Once the editor opens, type the following lines in the editor, exactly as they appear.

```
close all
t=linspace(-5,5);
```

```
x=abs(t); y=abs(1-t);
subplot(131)
plot(t,x)
xlabel('t-axis')
ylabel('x-axis')
title('xversus t')
axis square
subplot(132)
plot(t,y)
xlabel('t-axis')
ylabel('y-axis')
title('y versus t')
axis square
subplot(133)
plot(x,y)
xlabel('x-axis')
ylabel('y-axis')
title('y versus x')
axis square
```

Select Save from the File menu of the editor and save the file as myfile.m. Return to the MATLAB prompt and type

```
>> myfile
```

and all of the commands in the file myfile.m are executed in sequence, just as if you typed them sequentially at the MATLAB command prompt. The script file myfile.m produces the image in **Figure 6**.

A few new commands in the script file myfile.m warrant some explanation. First of all, the command close all closes all open figure windows. Also new is the subplot command, which allows the user to draw multiple plots in a single figure window. In general, the command subplot(m,n,p) creates a matrix of plots, m rows and n columns, and makes the pth axis active, counting from left to right by rows. For example, the command subplot(132)[11] divides the current figure window into one row and three
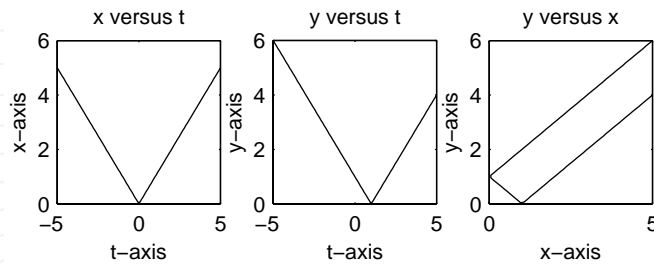
Figure 6 Script files are the way to go.

columns, placing an axes at each position, and dictates that the second axes (the one in row one, column two) is the current axes. All subsequent drawing commands are made on this axes.[12]

The Current Directory and Path

Once you start saving script files, then you have to understand how Matlab searches for files on your hard disk or network. Two phrases are key to this understanding: (1), the Matlab path, and (2), the current working directory. We assume that you have some familiarity with the directory structure on your hard disk and/or network.

When you enter a script file name at the Matlab prompt, the first place Matlab searches for this file is the "current working directory." You can determine the current working directory by entering[13]

```
>> pwd
ans = f:\MultCalcUsingMatlab\chap3
```

In this case, Matlab will first search in the directory \MultCalcUsingMatlab\chap3 on the f: partition of the computer's hard drive. If you did not save the file myfile.m in this directory, then Matlab will probably be unable to find your file, responding with

---

[11]Note that commas are optional.

[12]For a full explanation of the subplot command, type help subplot at the Matlab prompt and read the resulting help file.

[13]You surely will get a different response on your computer, namely, your current directory, not mine.

```
>> myfile
??? Undefined function or variable 'myfile'.
```

Cryptic messages like this can cause quite a bit of panic on the part of the user; but, if you think about it, the file is truly undefined if MATLAB cannot find it.

So, how do you get yourself out of this mess? That is, what can you do so that MATLAB knows where to find your file? You have a lot of options, but we will share only two.

If you saved `myfile.m` in the directory `c:\mywork`, one obvious option is to change the current directory to `c:\mywork`. You can do this at MATLAB's command line.

```
>> cd c:\mywork
>> pwd
```

The command `cd c:\mywork` changes the current directory to `c:\mywork`.[14] The second command, pwd, stands for "present working directory," which responds with the current directory. It's always good to use pwd to check the current directory.

If you plan on placing most of your MATLAB files in one or two directories, then it's probably best to add these directories to MATLAB's "path." After searching the current directory, MATLAB begins searching directories in the order listed on MATLAB's path. You can type `path` at the MATLAB prompt and MATLAB will list the all of the directories on its path. Order is important. If you have two files named `myfile.m`, then the one executed is the one in the directory that occurs earliest on the path.[15]

So, how do you add your working directories to the path? Let's say that you have a directory, `c:\mywork`, which you want to add to MATLAB's path. There are a number of ways to do this, but the easiest is to use the `pathtool`. Open the `pathtool` by (1), typing `pathtool` at the MATLAB prompt, or (2), selecting Set Path from MATLAB's File menu. In the Set Path dialog box, click the Add to Path icon and browse to the folder you wish to add to the path. When first learning the intricacies of MATLAB's path, it's probably best to click "Move to Top", which moves your folder to the front of the path (first to be searched).[16] If you wish to make your path changes permanent, click Save before closing the Set Path dialog box.

---

[14] MATLAB 6 offers a current directory navigation window on its standard toolbar. You can select from a drop down list or click the "browse icon" to switch to another directory.

[15] One extremely useful debugging tool is MATLAB's `which` command. If you type `which myfile`, MATLAB will respond with the directory of the `myfile.m` it is about to execute.

Comet Plots

MATLAB has animation capability that enables you to examine the plot of a set of parametric equations as it is drawn in real time. Let's animate the plot of a *Lissajous* curve, shown in **Figure 7**.
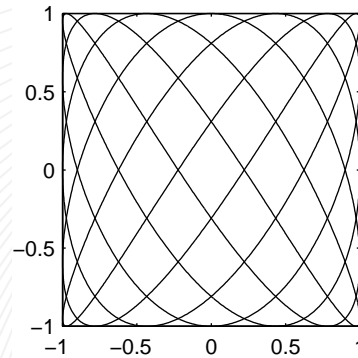
Figure 7   Lissajous curve

Save the following sequence of commands in a script file named `lissajous.m`.

```
close all
t=linspace(0,2*pi,4000);
x=cos(5*t);
y=sin(7*t);
comet(x,y)
```

Execute the script file by typing `lissajous` at the MATLAB prompt.

Clearly, the command `comet(x,y)` is similar to the `plot` command, but it animates the plot in real time. Two commands in the script file `lissajous` warrant further comment. The `close all` command close all open figure windows, clearing the desktop for "fresh start" before opening a new figure window

---

[16]Later you can open the `Set Path` dialog box, select your folder, then click the icons to change the position of the folder in the path or even remove the folder from the path.

with the `comet` command. Secondly, we've created 4,000 equally spaced time points with the command `t=linspace(0,2*pi,4000)`. This is just about right on a 300 MHz Pentium machine, but you can slow the animation further by adding more time points (or speed things up by adding fewer time points). Experiment with the `linspace` command to find the optimum setting for your personal workstation.

If you intend to do anything with the final image, it's best to redraw it with the `plot` command as the `comet` command saves only the last point plotted in its routine for further use. Try resizing the figure window containing the Lissajous plot with your mouse to see what we mean.

Multigraf

It won't take long for you to realize that we are wasting a lot of paper printing dozens of MATLAB plots. Dr. John Polking of Rice University has written a nice routine called `multigraf` which helps to save a lot of paper. This routine is already present on the campus server, so typing `multigraf` at the MATLAB prompt is all you need to do to start the routine. Once the `multigraf` figure window opens, its use is pretty self explanatory.

If you would like to use this routine at home, vistit Dr. Polking's site at Rice and download `multigraf`.

<div align="center">

http://math.rice.edu/ dfield

</div>

Exercises

Use MATLAB's `plot` command to draw the graphs of the functions in **exercises 1-4** on the given interval. Label each axis and provide a title for your plot.

1. $f(x) = 5 - 4x - x^2$, $[-6, 2]$

2. $g(x) = 2x^2 - 8x - 11$, $[-1, 5]$

3. $f(t) = te^{-2t}$, $[-1, 5]$

4. $h(t) = e^{-0.1t}\sin(2t)$, $[0, 24\pi]$

Hint: In **exercise 4**, *enter* h=exp(-0.1*t).*sin(2*t). *Remember, you cannot multiply vectors, so you need to use array multiplication in this exercise.*

In each of **exercises 5-8**, use MATLAB's `plot` command to draw the graph of both given functions on the same axes, drawn on the given interval. Label each axis. Provide a title and legend for your plot.

5. $s(x) = \cos 2x + \sin 3x$, $\quad v(x) = -2\sin 2x + 3\cos 3x$, $\quad [0, 4\pi]$

6. $f(x) = xe^{-3x}$, $\quad g(x) = e^{-3x}(1 - 3x)$, $\quad [0, 2]$

7. $f(t) = \sin 3t \cos 2t$, $\quad g(t) = \frac{1}{2}\cos t + \frac{5}{2}\cos 5t$, $\quad [0, 4\pi]$

8. $x(t) = (1 + 2\sin t)\cos t$, $\quad y(t) = (1 + 2\sin t)\sin t$, $\quad [0, 2\pi]$

*Hint: In **exercise 8**, enter* x=(1+2*sin(t)).*cos(t) *and* (1+2*sin(t)).*sin(t). *Remember, you cannot multiply vectors, so you need to use array multiplication in this exercise.*

In each of **exercises 9-14**, x and y are defined in terms of a parameter t on a given interval. Use MATLAB's plot command to draw a plot of y versus x. Label each axis and provide a title for your plot.

9. $x = t - \sin t$, $\quad y = 1 - \cos t$, $\quad [0, 6\pi]$

10. $x = \cos^3 t$, $\quad y = \sin^3 t$, $\quad [0, 2\pi]$ $\qquad$ *Hint:* x=(cos(t)).^3

11. $x = 2\cos t + \cos 2t$, $\quad y = 2\sin t - \sin 2t$, $\quad [0, 2\pi]$

12. $x = t + 2\sin 2t$, $\quad y = t + 2\cos 5t$, $\quad [-2\pi, 2\pi]$

13. $x = 9\cos t - \cos 9t$, $\quad y = 9\sin t - \sin 9t$, $\quad [0, 2\pi]$

14. $x = \sin 3t$, $\quad y = \sin 4t$, $\quad [0, 2\pi]$

15. Use the comet command to animate the plot in **exercise 9**.

16. Use the comet command to animate the plot in **exercise 10**.

17. Use the comet command to animate the plot in **exercise 11**.

18. Use the comet command to animate the plot in **exercise 12**.

19. Use the comet command to animate the plot in **exercise 13**.

20. Use the comet command to animate the plot in **exercise 14**.

21. The `comet` command can be used to check your parametrizations. For example, a particle moves about the unit circle, starting at $(0, 1)$ at time $t = 0$ and proceeding in a counter-clockwise direction. You believe a valid parametrization of this path is given by

$$x = \cos t,$$
$$y = \sin t,$$

where $0 \le t \le 2\pi$. Check this supposition with

```
t=linspace(0,2*pi,1000);
x=cos(t);
y=sin(t);
comet(x,y)
```

22. Make the particle in **exercise 21** move about the unit circle, starting at $(1, 0)$ at $t = 0$, but traveling in a clockwise direction. Do this by adjusting the parametric equations, not the time span. Check your result with the technique shown in **exercise 21**.

23. Make the particle in **exercise 21** move about the unit circle, starting at $(1, 0)$ at $t = 0$, still traveling in a counterclockwise direction, but moving twice as fast. Do this by adjusting the parametric equations, not the time span. Check your result with the technique shown in **exercise 21**.

24. Make the particle in **exercise 21** move about the unit circle, but let it travel in a clockwise direction, starting at the point $(0, -1)$ at time $t = 0$. Do this by adjusting the parametric equations, not the time span. Check your result with the technique shown in **exercise 21**.