2: M-Files

In this section we introduce M-files and use them to construct somewhat more elaborate plots.

Consider the following scenario. A curve in the plane is given by a set of *parametric equations*. For example, the equations $x = \cos t$ and $y = \sin t$, where $0 \le t \le 2\pi$, define a circle of radius one centered at the origin. In this case it is easy to identify the curve geometrically, since the points satisfy the relationship $x^2 + y^2 = 1$. In general, however, it is not so easy to identify in an abstract way the curve given by parametric equations of the form $x = f(t)$, $y = g(t)$. We can, however, obtain a plot of the curve, which is often a useful starting point for further study.

In the first lesson we developed plots directly from the command line. This is OK for simple one or two step procedures, but as soon as our construction begins to involve additional steps this approach reveals drawbacks. For one, if we want to modify our work we must typically repeat all the steps. Two, although we can save or print the output, we will have no record of the procedure once we have closed Matlab.

In an M-file, we type commands in a word processor ("editor") as if we were typing them at the command line. The file is then named and saved. After the file is saved, we "run" it by typing the name of the file at the command line. When we do so, all the commands in the file are executed, as if they had been typed directly at the command line. If the file does not work correctly at first,we go back, "debug" it to remove errors, save it and run it again. We continue with this cycle until the M-file does what we want.

We will illustrate this procedure by writing a simple M-file that will plot a parametric curve defined by the user's input.

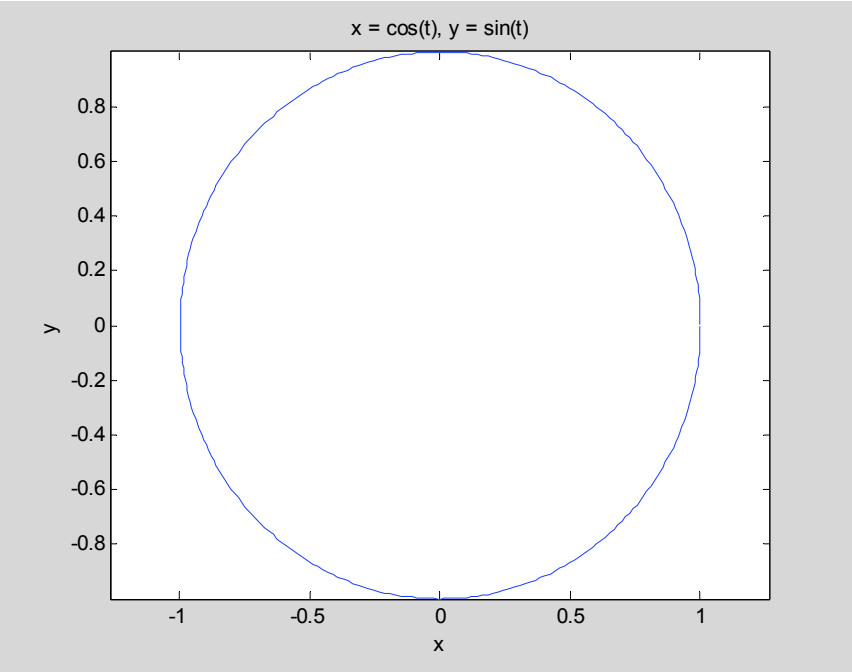
2.1 A Simple M-File

To construct an M-file, click on the New M-file icon, the first icon in the Matlab icon bar. This opens a window called the Editor. The Editor is just a simple word processor that has been customized particularly for Matlab. For example, it recognizes reserved terms in Matlab and displays those in color. Also, where appropriate, it automatically idents lines, a feature that is very useful in debugging certain types of programs.

Example 2.1: Develop an M-file to create a plot of a curve given by parametric equations.

Solution: Open the editor. We will set up our code so that the input supplied by the user comes at the beginning, followed by the code for plotting. To do this you might type the following code in the editor. Note that the text following a % sign consists of comments. These are ignored by Matlab, but are useful to readers who want to understand what the code is doing.

```
 % Parametric plotting script
syms t  % declare t a symbolic variable

% User Input section

x=cos(t); % the semi-colon suppresses display of the output
y=sin(t);
a=0; % smallest parameter value
b=2*pi; % largest parameter value

% End of user input. Now we write code to do the basic plotting

clf  % clears current figure
ezplot(x,y,[a,b])
```



Now save the file. We saved it in the course website under the name **param1.m**. You don't have to add the .m when you save it. The editor does it automatically. Please note that a file name cannot begin with a numeral and should not contain spaces. You should use an underscore as a replacement for a space, or use a mix of lower and upper case, such as MyFile. To run the file, type the file name (without the .m) at the command line. Or even simpler, click the icon for running an M-file on the Editor toolbar. If you made no errors in copying the code, all you should see is the plot shown above – a circle drawn in blue. The output of all other commands has been suppressed using the semicolon. Bravo! ■

We have written this M-file assuming that a potential user will interact with it directly. It is also possible to include the `input` command for each of the lines requesting input so that a user is prompted to enter the information and need not interact with the code at all.

In general, we will not produce code with this structure, but the reader is invited to experiment with how this works in the exercises.

2.2 Fancy Stuff

In Chapter 1 we learned how to change the characteristics of a graph interactively. It is useful however to be able to change such features programmatically. For example, we may want to change the line style of the graph, or add or change the title, add grid lines, etc. The point is, anything that can be changed interactively can be changed in a program as well – and it's not that hard to do so.

The key idea in manipulating graphics objects, as well as many other objects in the Matlab programming environment, is the notion of an **object handle**. Roughly speaking, an object handle is a name that we (or Matlab) attaches to an object. We manipulate the properties of the object by using the name to assign new values to a particular property. The easiest way to obtain a handle for an object is to assign a name to the object when it is created. If you don't attach a name, Matlab will assign one, but it can sometimes be hard to figure out exactly what it is.

Since there are literally hundreds of properties that are associated with the various components of a graph, it can be a daunting task to find out exactly how to refer to these in Matlab. The Plot Tools browser that we used in the first lesson can be used to find the exact name associated with each property. The list of such names, with their current values, can be found using the *Inspector*. More detailed information on a property can be found using Matlab's Help pages. For example, searching for *LineSpec* in the search box in Help, will bring up a page with the most commonly used references. For now, we will show how to

a) add grid lines

b) change the title

c) add a legend

d) change the line style, thickness, and color.

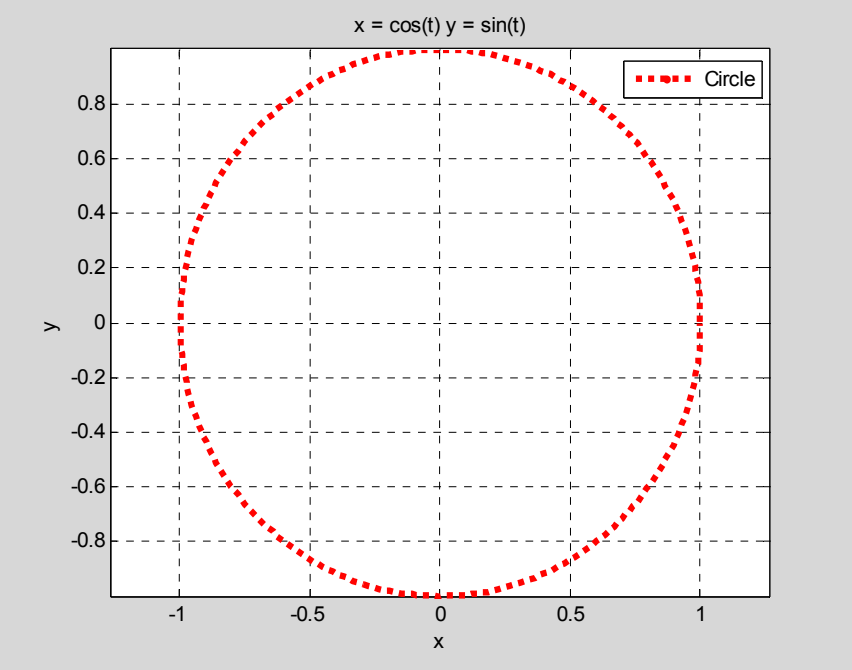
Here is the modified file which we save as **param2.m**:

Example 2.2: Create a parametric plot of the circle, adding items a) through d) above to the output.

Solution:

```
% Parametric plotting script
syms t  % declare t a symbolic variable
x=cos(t); % the semi-colon suppresses display of the output
y=sin(t);
a=0; % smallest parameter value
b=2*pi; % largest parameter value

% End of user input. Now we write code to do the basic plotting

clf  % clears current figure

h=ezplot(x,y,[a,b]); % this assigns the handle h to the graph.
```

```
% The value of the handle is not displayed, but the plot is.

%Now we change some of the characteristics of the graph window

grid on  % turns grid lines on

% Add a title by concatenating several strings.
% The command char( ) produces a string from the symbolic expression x.
title(['x = ',char(x),' y = ', char(y)]);

% Add a legend
legend('Circle'); % The legend text appears in quotes.

% To change properties of the graph we must use the handle
% h.  Each property name is listed in single quotes, followed
% by the value we want to set it to.

set(h,'Color','red', 'LineWidth', 3, 'LineStyle', ':');

axis equal; % equalizes scales on the axes.  Useful when non-distorting
of distances is important (drawing circles and squares, for example).
```



■

2.3 Publishing an M-file

When we ran the M-files above we produced a nice graph, which we could print using the Print button in the graphics window. However, the M-file code is sitting in the Editor window. We can print that as well, as a separate document. If we want to produce a

report showing the code and the output we can merge these in another word processor, say MS Word, but if we then decide to change our code we have to recreate the output document. This is quite cumbersome.

Matlab, however, provides a way to actually print your M-file code and its output in a single document. It's called publishing. You can publish your result as a webpage, a PowerPoint presentation, a Word document, or in several other formats. We will describe how to take your completed M-file and publish it to a webpage, which Matlab calls "Publish to HTML."

In this case, where the M-file produces only a single output at the end, all you do is go to the File menu in the editor and select "Publish to HTML." There is also a button at the left side of the icon bar that does the same thing. In very short order, a window will appear with your code and output (the graph). You can then print that window for a complete record of your work and what it produces. Very neat…

For M-files that produce more than one output we can divide the M-file into regions called "cells." Any output produced by a command in a cell appears right after the cell, rather than at the end of the document. This makes your document into something like a report, which is the basic idea of this feature. You can add additional "markup" to cells to give them a more distinctive appearance, as well as provide a hyperlinked structure for the webpage that is produced. We will make greater use of this as we proceed in the course and we produce work with more complex structure.

If you would like to learn more about the features of publishing, there is an excellent video demo that you can access from the Matlab Help menu. Click on the Demos tab and then open the folder named "Desktop Tools and Development." Look for the video called "Publishing M Code from the Editor." It will explain this feature far better than our simply writing about it.

That's it for this lesson. Let's summarize what we have learned.

2.4 Summary

M-file: A text file consisting of Matlab commands plus comments. When the file name is entered at the command line the Matlab commands are executed.

`semi-colon`: - suppresses display of output for a command

`% (percent sign)`: - allows comment to be inserted in an M-file

`char` – converts a symbolic expression into a string (text)

`clf` – clears current figure to prepare for new drawing

`handle` – a name assigned to a graphics figure that can be used to change the properties of the figure

`grid` – on/off – toggle command to display or remove grid lines from graph

`title` – add a title to current figure

`legend` – add a legend to current figure

`set`(*handle*, property, value, property, value, …) - set specified properties for the object named by the handle.

2.5 Exercises

Exercise 2.1 Write an M-file that displays the graphs of

a) the ellipse with parametric equations $x = 4\cos t$, $y = 3\sin t$, $0 \le t \le 2\pi$.

b) the largest circle that can be inscribed in the ellipse (Make sure the circle looks like a circle – use the *axis equal* command.)

c) the ellipse should be drawn in a **solid linestyle** and the circle in a **dotted style**

d) the graph should have a **title** that includes your name

e) the graph should display a **legend** that shows which graph is the ellipse and which is the circle.

Publish your output to HTML.

Exercise 2.2 Look up the `input` command in Help**.** Rewrite the M-file `param1.m` so that the input section prompts the user for the parametric equations and the range of the parameter. Print a copy of your code and the output generated from a typical session. (Note that you cannot use the option "publish to html" when the code uses the `input` command.)

Exercise 2.3 A polar curve $r = f(\theta)$ can be plotted by using the parametric representation $x = r\cos\theta = f(\theta)\cos\theta$ and $y = r\sin\theta = f(\theta)\sin\theta$, with the polar angle θ as the parameter. Write an M-file that displays the graphs of

a) the polar curve with equation $r = 10\cos 4\theta$, $0 \le \theta \le 2\pi$ (use t instead of θ in your Matlab code). This curve should be drawn with a **solid** linestyle.

b) the polar curve with equation $r = \theta$ with $0 \le \theta \le 4\pi$. This curve should be drawn in a **dashed** linestyle.

c) Include a **title** that contains your name.

d) Include a **legend** that identifies the two graphs.

Publish your output to HTML.

3: Data Plotting

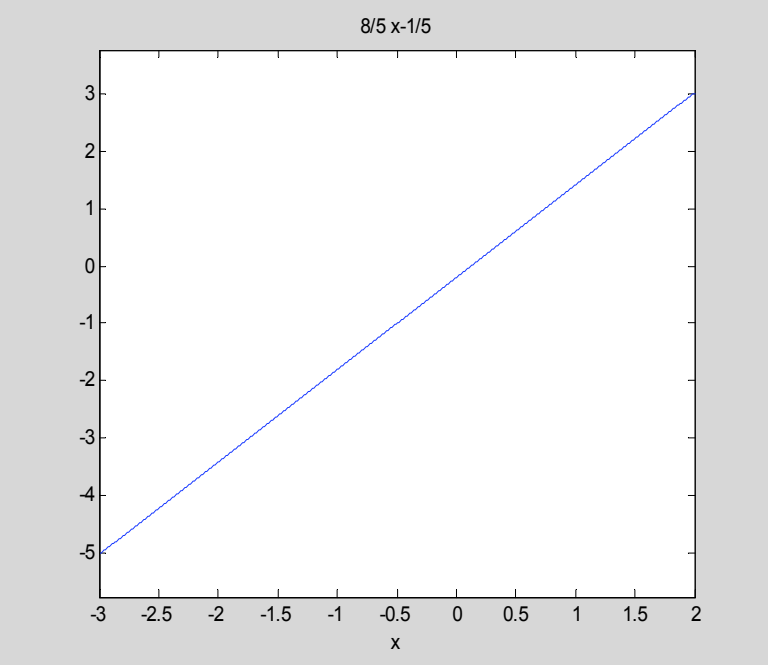
In this chapter we will learn an alternate approach to plotting in Matlab. This approach is more general than using the ez-commands. It will enable us to plot figures that can't be created using the ez-commands. It is particularly valuable for plotting in three dimensions.

3.1 Plotting A Line

Let's start with a simple problem. How do we get Matlab to draw a line segment joining two points in the plane? To be specific, suppose we want to draw the line segment connecting the points $(2,3)$ and $(-3,-5)$. We can use the techniques discussed in previous chapters to solve this problem in either of two ways. Let's review.

First, we can find the standard $y = mx + b$ equation of the line. This is $y = \frac{8}{5}x - \frac{1}{5}$. Now using `ezplot` we can plot the graph over the interval $x = -3$ to $x = 2$.

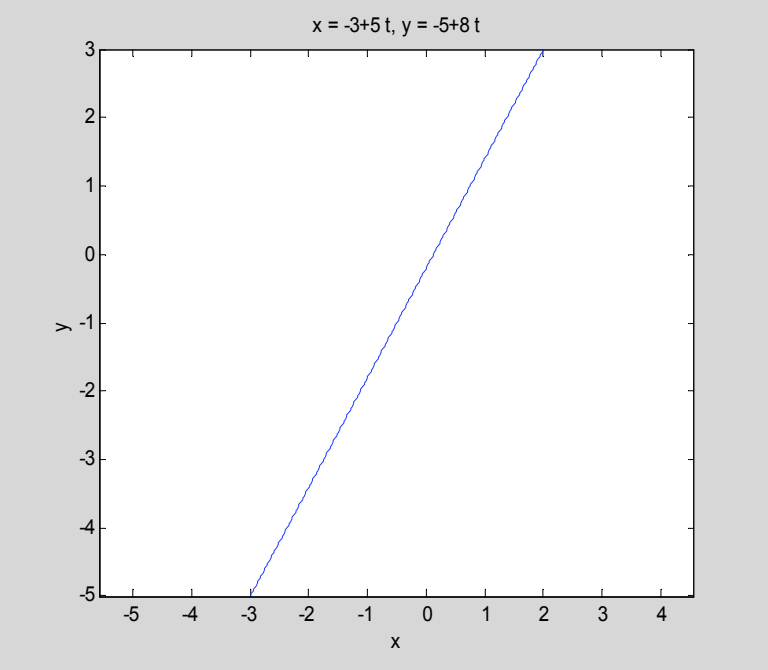
```
syms x
ezplot(8/5*x-1/5,[-3 2])
```



A second method is to find parametric equations for the line segment. The direction vector of the line is the vector joining the two given points. It works out to $\langle 5,8 \rangle$, so the

vector parametric equations are $\mathbf{r}(t) = \langle -3, -5 \rangle + t \langle 5, 8 \rangle$, for $0 \le t \le 1$. Splitting these into component functions and introducing t as a symbolic variable we can develop the plot through

```
syms t
ezplot(-3+5*t, -5+8*t, [0 1])
```



Notice that in the second picture, Matlab chose to plot a larger portion of the x axis than in the first figure. Visually the result is a more accurate representation of the steepness than in the first picture. We will see later how to control this effect by varying the size of the plotting window.

The third, and preferable approach to this problem, draws the line without converting the data to any functional form. We simply give Matlab a list, or array, of the x coordinates of the two points defining the line, in the order of smallest to largest. Then we give it a list of the corresponding y coordinates associated with these points (these will not necessarily be in increasing order, but they are in this case). Here's how it's done.

Example 3.1: Draw the line segment from $(-3, -5)$ to $(2, 3)$ by connecting data points. Modify the resulting plot by adjusting the aspect ratio, line style, etc.

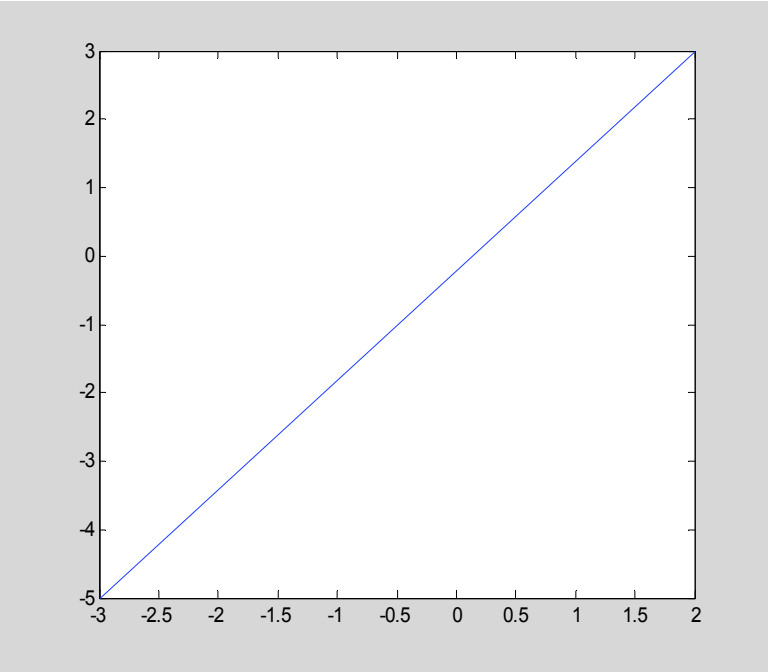
Solution:

Define two arrays `x` and `y` that hold the x coordinates, respectively the y coordinates, of the points to be plotted.

```
x=[-3 2]; y=[-5 3];
```
Note that x and y will now no longer be symbolic variables.

Then we use the command `plot`:

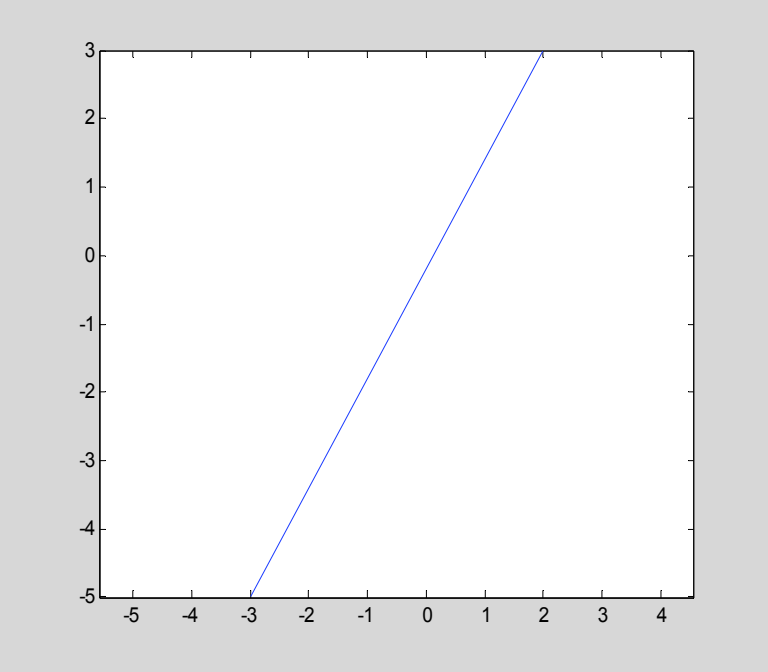
```
plot(x,y)
```



The `plot` command takes every value in the *x* list and pairs it with the corresponding entry in the *y* list to create an ordered pair. It then plots the ordered pair and connects the plotted points with a solid line.

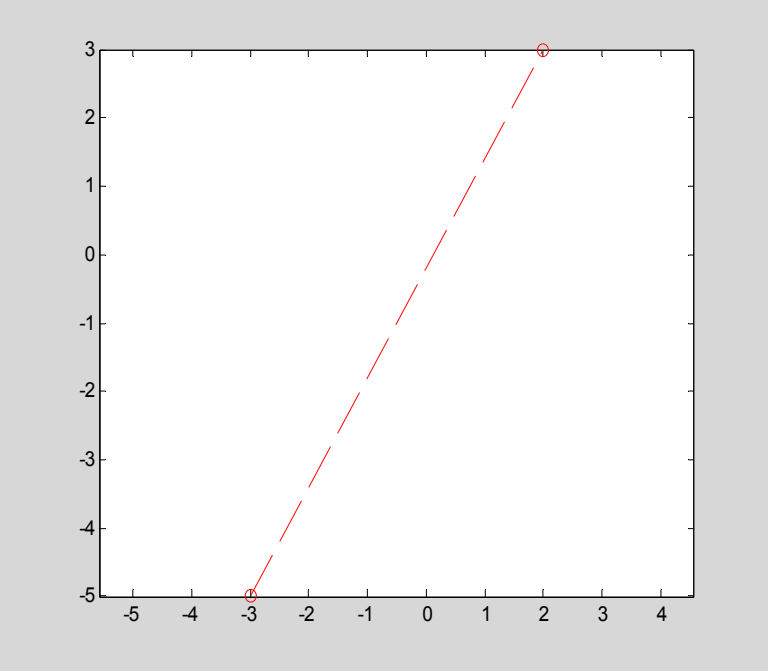
You may not like that the graph makes the line appear to have slope 1. We can get Matlab to display a visually correct representation by modifying the aspect ratio, which relates the unit of length displayed in the *x* direction to that displayed in the *y* direction. We can make these distances equal, by issuing the command

`axis equal` , which we used in drawing circles in Chapter 2. This produces the following figure.



Further, it is very easy once we generate the data points to create a plot with different line styles, color and whether and how the data points themselves are displayed. For example, consider

```
plot(x,y, 'or--'); axis equal
```



The string 'or- -' indicates that we want the data points displayed as small circles (o), drawn in red (r), with a line style dashed (- -). You can find a complete list of these line specification ('spec', for short) symbols by searching for **LineSpec** in Matlab's Help.

Note that is not necessary to include all options. If you only wish to change the line style to dashed, then use the string '- -' and omit the others. ■

3.2 Plotting Graphs

The `plot` command can be used to create graphs of functions. Basically, the procedure imitates what you learned to do in school – make a table of values, plot the associated ordered pairs, and connect the points with line segments. With Matlab all we have to do is create the table of values; Matlab does everything else. Matlab has two features that make it very easy to generate the table of values. Let's start with a rather standard example.

Example 3.2: Use the `plot` command to create the graph of $y = x^2$ on the interval $[-2, 2]$.

Solution:

a) The first step is to generate your x values. In order for the graph to be smooth looking we usually need to select at least 25 points to plot. When a graph is turning quickly, we will need even more points. However, we will first construct a more primitive graph so you can see how everything works. The `linspace` command is used to generate x values.

```
x=linspace(-2,2,5)    % generates  5 equally spaced points in the interval [-2, 2]
x =
    -2    -1     0     1     2
```

The variable x has been assigned as the name of the array [-2 -1 0 1 2] generated by the command. Usually, we don't bother to display the array.

b) For each value in the array x we want to compute the y value defined by $y = x^2$. You might think that you have to write a program to get Matlab to do this, but you don't. Namely, Matlab has a special operator, the dot operator. Look what happens if you type

```
y=x.^2
y =
     4     1     0     1     4
```

The dot operator "vectorizes" the operation that follows it, in this case the squaring procedure. This tells Matlab to carry out the squaring separately on each entry of the input x, exactly what we needed to do. We name the resulting array as y.

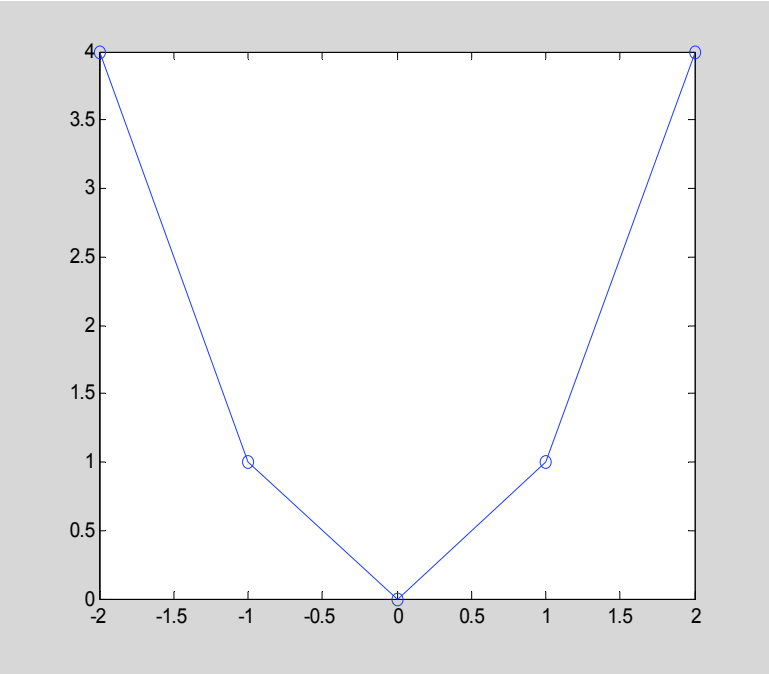
(Note: If you omitted the dot Matlab will complain.

```
y=x^2
??? Error using ==> mpower
Matrix must be square.
```

Matlab thinks you are trying to square the array *x* using matrix multiplication. But that procedure is only defined for square arrays, hence, the complaint. We won't deal with matrix multiplication in this course. You'll see it when you study linear algebra.)

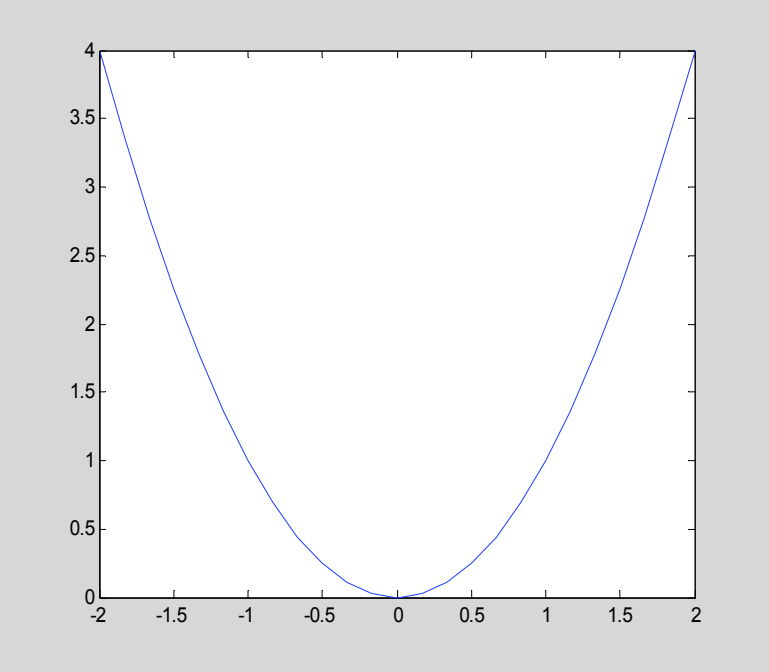
c) Now we can let Matlab plot the points whose *x* coordinates are taken from the *x* list and whose *y* coordinates are taken from the *y* list.

```
plot(x,y, 'o-')   % show plotted points with an o and use a straight line to connect points
```



It's not too pretty, but it does show what is going on. The `plot` command does not try to smooth things. It just draws lines between the points that you give it, in the order in which they appear in the arrays. To get a better picture, you need to use more plotting points. Let's repeat the construction using more points, suppressing unnecessary output.

```
x=linspace(-2,2,25); y=x.^2; plot(x,y)
```

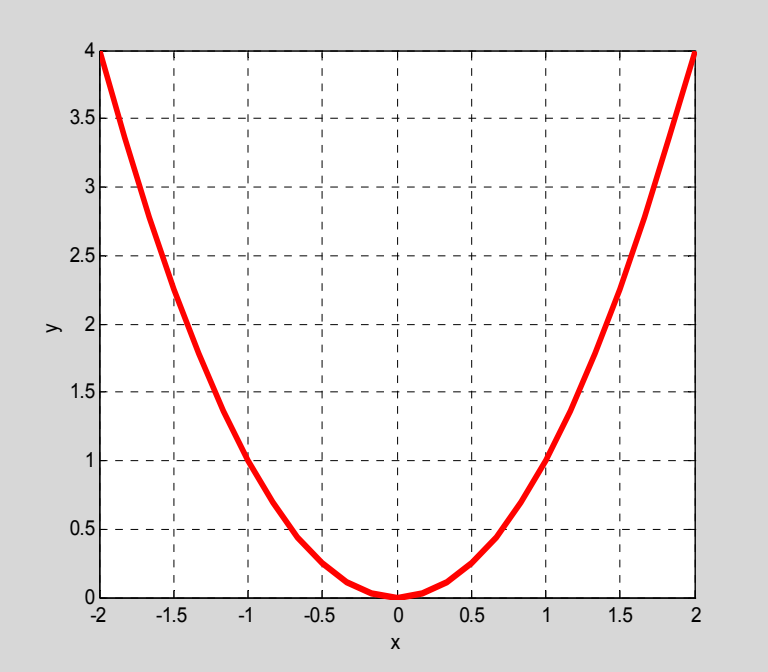


Much better. ■

OK – we are about ready to end this lesson, but first we want to indicate how one can add annotation to the graph produced by `plot`. Notice that the graph above has no title, no axis labels, and no grid lines. It's pretty barebones. We add those features to the figure using the `title`, `xlabel` [resp. `ylabel`] and `grid` commands. We haven't discussed `xlabel`. Look it up in Help to see how it works. We give an example below. If you want to change the thickness of the plotting line, you can do that in the same way we did in Chapter 2, by constructing a handle to the graph and using the `set` command to change the **LineWidth** property. However, with `plot` there is an easier way. You can include the new spec right in the `plot` command. For example,

```
plot(x,y, 'r', 'LineWidth',3); xlabel('x');ylabel('y');grid on
```

produces



Let's summarize what we have covered:

3.3 Summary

arrays: - these are just lists of numbers enclosed in brackets. Arrays are created either using Matlab commands (like `linspace`) or can be entered from the command line directly as [*value value value* …], where the *values* are separated by spaces.

`linspace`: - command to generate an array of a specific number of equally spaced points in an interval.

dot operator: - When preceding an ordinary operation, this vectorizes the operation so that when the variables are arrays, the operation is carried out on each component. The dot operator is only needed for multiplication, division, and exponentiation. Component addition of arrays and addition/multiplication by a scalar do not require the dot. Study the following examples to understand the behavior of the dot operator.

Example 3.3: Using the "dot" operator.

```
x=[1 2 3 4]
x =
     1     2     3     4
y=[2 4 6 8]
y =
     2     4     6     8
x+y   % add corresponding components of the two arrays.  No dot needed.
```

```
ans =
     3     6     9    12
```

`x.*y` % multiply arrays componentwise.

```
ans =
     2     8    18    32
```

`x./y` % divide arrays term-by-term; make sure no entry in *y* is zero.

```
ans =
    0.5000    0.5000    0.5000    0.5000
```

`2*y` % multiply each entry of y by 2. No dot needed when multiplying by a scalar.

```
ans =
     4     8    12    16
```

`x+1` % add 1 to each entry of x. No dot needed when adding a scalar to each component.

```
ans =
     2     3     4     5
```

■

`plot`: - plots pairs of points taken successively from input arrays *x* and *y*. Note that the input arrays *x* and *y* must have the same number of elements and must have the same shape in terms of row, column structure. The plotted points are connected in the order in which they appear in the arrays.

`axis`: - this can be used to set the plotting window to a specific rectangular region in the *x*,*y* plane. Typing `axis`([a b c d]) creates a plot window over the rectangle $[a,b]\times[c,d]$. The command can also be used to have Matlab adjust the window to achieve a desired effect (e.g. `axis` *equal* equalizes the aspect ratio, i.e. the physical length displayed on the screen corresponding to a unit of length in direction of the coordinate axes.)

LineSpecs:- These are string notational abbreviations for various options of color, line style, and plotting point marker. These specs can be changed directly when the graph is generated by `plot`, unlike when using `ezplot`, where we have to access the graph handle using `set` to make such changes. Also included in LineSpecs, are properties such as `LineWidth`, `MarkerSize`, and `MarkerFaceColor`.

`xlabel, ylabel`:- commands to add text labels to the coordinate axes.

3.4 Exercises

The first four problems are for practice and comprehension. Problem three is *especially important* because it introduces an alternate way of generating arrays.

Exercise 3.1 In this exercise use *pi* to enter the mathematical constant π in Matlab. Enter a list x whose entries are the values $0, \frac{\pi}{4}, \frac{\pi}{2}, \pi, \frac{3\pi}{4}, \pi$. You can do this using `linspace` (how?) or by just typing the values directly. Now compute $\sin(x)$ and $\cos(x)$. Observe that Matlab automatically vectorizes the computation for built-in functions. Compare Matlab's response when you enter each of the following commands:

```
>> x.*sin(x)
>> x*sin(x)
```

Exercise 3.2 Using `linspace`, generate an array of the integers 1 to 10. Call that array *k*. Using that array generate an array whose entries are the numbers $1/1^2, 1/2^2, 1/3^2, \ldots, 1/10^2$.

Exercise 3.3 The `linspace` command is useful for creating equally spaced arrays in which your concern is on the number of points created. Sometimes it is more useful to focus on the spacing between the points. In that case, Matlab provides another mechanism, the **colon** operator. Enter the following commands at the command line (without the comments).

```
>> 1 : 10  % generate integers from 1 to 10.  You do not need the spaces.
>> 1 : .5 : 10  % Generate numbers from 1 to 10 with spacing of .5 . (You can read the symbol as "from 1 by .5 to 10." When the middle number is omitted, it defaults to 1.)
>> 10 : -1 : 1  % You can generate numbers in decreasing order
>> 1 : 1.5 : 9  % The increment does not have to fill out the range exactly.  Matlab stops at the largest value in the list before the endpoint.
```

a) Generate the sequence in Exercise 3.1 using the : operator.

Exercise 3.4 Here are some short answer questions. Make sure you first do Exercise 3.3. You should be able to answer these without using the computer, but feel free to check your work by actually using Matlab.

a) State a single Matlab command that will assign to the variable x the vector $[5,10,15,\ldots,990,995,1000]$, **without displaying the result**.

b) Generate the sequence `0:.01:1` using the `linspace` command, **without displaying the result**. (Hint: How many points are generated by the colon command?)

c) How would you create a vector whose entries are the numbers of the interval $[0, 2\pi]$ which subdivide the interval into 20 intervals of equal length?

d) If `x=[1 2 3 4]`, what Matlab expression involving x will produce the output $[1 \quad 2^2 \quad 3^3 \quad 4^4]$, i.e. [1 4 27 256]?

e) If `x=[1 2 4 5]`, what Matlab expression involving x will produce the output $[1 \quad 1/2 \quad 1/4 \quad 1/5]$, i.e. [1 .5 .25 .2]?

Exercise 3.5 Write an M-file that does all of the following:

a) Uses the `plot` command to draw the square with vertices $(1,2)$, $(3,2)$, $(3,4)$, and $(1,4)$. The square should be drawn in **red** (at least your M-file should show that).

b) Uses the `plot` command to draw the two diagonals of the square in blue with a dotted line.

c) Uses the `axis` command to change the plotting window to the region $[0,5]\times[0,5]$ and then to adjust the axes so that the square looks like a square and not a rectangle.

d) Finally, add a title: "A Square by *your name*"

Publish your M-file (put your name in a comment at the beginning of the M-file) and the graph.

Exercise 3.6 Let $y = \sin(x^2)\cos(x)$. (This exercise partially duplicates Exercise 1.1, but you are expected to generate the output using an M-file.)

Write an M-file that uses the `plot` command to draw graphs of y and y' over the interval $[-\pi,\pi]$. In this problem, you should enter the derivative formula by hand when you need numerical values of y'. Later in the course we will discuss how to mix symbolic and numerical calculations. The plot should have the following features:

a) a legend identifying which graph is which, for example one graph identified as y and one as y' or dy. (See help on the `legend` command.)

b) distinct line styles for the two graphs (one dotted, one solid) and different colors if you are using a color printer.

c) both graphs displayed without chopping any values in the range

d) labels on both axes **and** x-tick marks at the points $-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi$. See Matlab's Help on the commands `Xtick` and `Xticklabel` to find out how to change the labeling of the axis tick marks in an M-file. You cannot actually get the labels displayed with the Greek font. You can use pi/2, pi, etc.

e) A title: "Graph of $y = \sin(x^2)\cos(x)$ and y'". The title should also contain YOUR NAME .

f) Grid displayed.

Publish your M-file.

Exercise 3.7 The `plot` command can also be used to plot curves defined parametrically. One simply uses the parametric equations $x = x(t)$, $y = y(t)$ to generate x and y values from a subdivision of the parameter interval. Using this idea, generate a plot of the circle $x^2 + y^2 = 1$ from the parametric equations $x = \cos t$, $y = \sin t$, $0 \le t \le 2\pi$. Use the `axis` command to ensure that the graph looks circular. Label the axes and add a title. Publish your M-file.

4: Three Dimensional Plotting

In this section we will learn the basics of 3D plotting. We'll cover both the ez-mode and the not-so-ez mode. The latter is particularly needed when you wish to show more than one object in the same picture, for example, a curve and a tangent line, or two surfaces.

4.1 EZ-Plotting

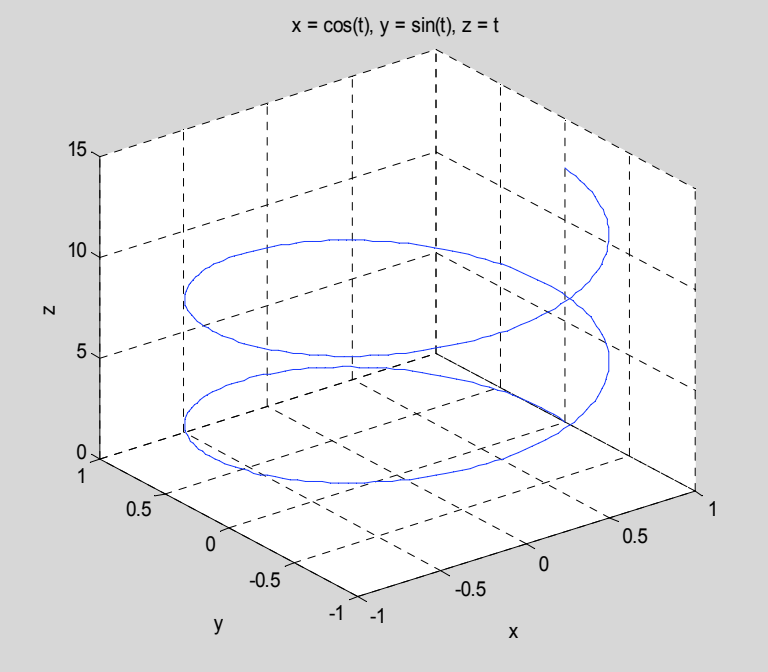
4.1.1 Curves

If we have parametric equations for a curve in space then it is very easy to obtain a graph using the command `ezplot3`.

Example 4.1: Draw the helix given by the parametric equations $x = \cos t$, $y = \sin t$, $z = t$ when t varies from 0 to 4π.

Solution:

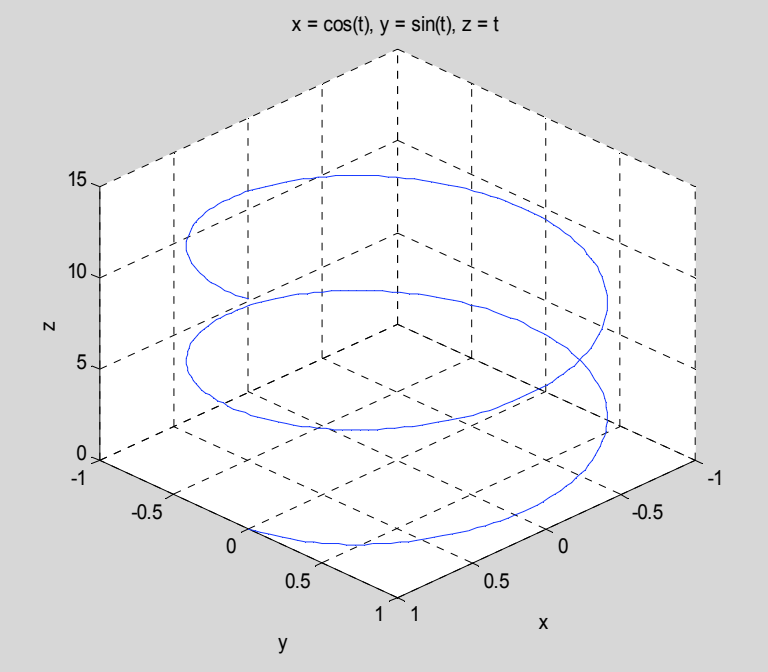
```
syms t;
x=cos(t); y=sin(t); z=t;
ezplot3(x,y,z, [0, 4*pi])    % The list [0, 4*pi] gives the range of the parameter t.
```



The picture has one small defect – the scene is not viewed from the perspective of the first octant, as we customarily do. You can see this from the tick marks on the x and y axes. If this is disturbing, you can correct it in two ways. First, you can manually rotate the image in the figure window by clicking on the Rotate3D icon and then clicking and holding the cursor on the figure while you drag the cursor on the screen (easier to do than

describe). You do this until the tick marks arrive in their standard configuration. Alternatively, you can change the viewpoint using the `view` command.

```
view([1 1 1])   % the argument [1 1 1] defines the point from which we view the curve.
```



A rather neat feature of the `ezplot3` command is the option 'animate'. This generates a point that moves along the curve in the direction it is traversed as the parameter increases. We can't show this is in the text, so enter

```
ezplot3(x,y,z, [0,4*pi]), 'animate'); view([1 1 1]);
```

and see what happens. If you miss the show, there is a repeat button in the window that lets you replay it. ■

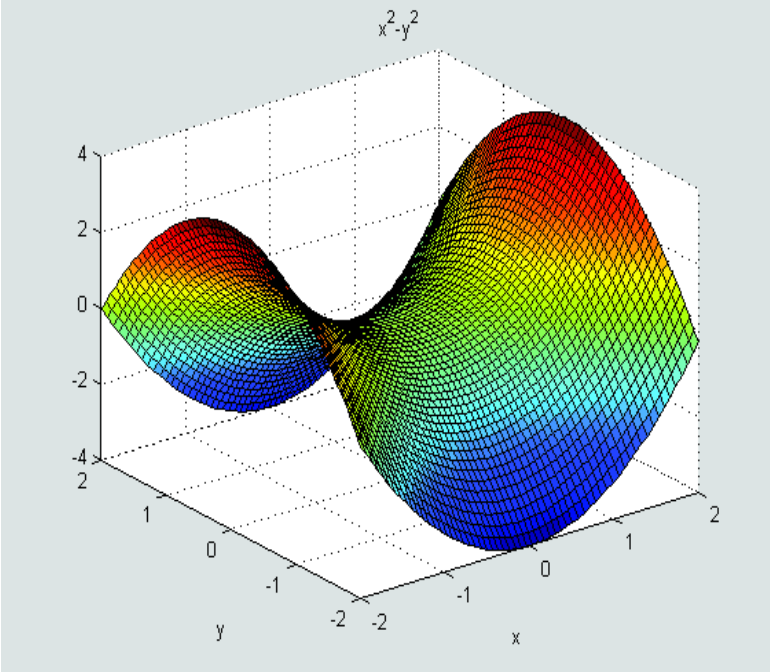
4.1.2 Surfaces

Constructing a 3D surface via ez-plotting is also very straightforward.

Example 4.2: Plot the saddle-shaped surface whose equation is $z = x^2 - y^2$, over the rectangle $[-2,2]\times[-2,2]$.

Solution: Simply enter

```
syms x y;
z=x^2-y^2;
ezsurf(z, [-2 2 -2 2])  % we could have specified the plotting domain simply as [-2 2], since the same interval is used for both x and y.
```



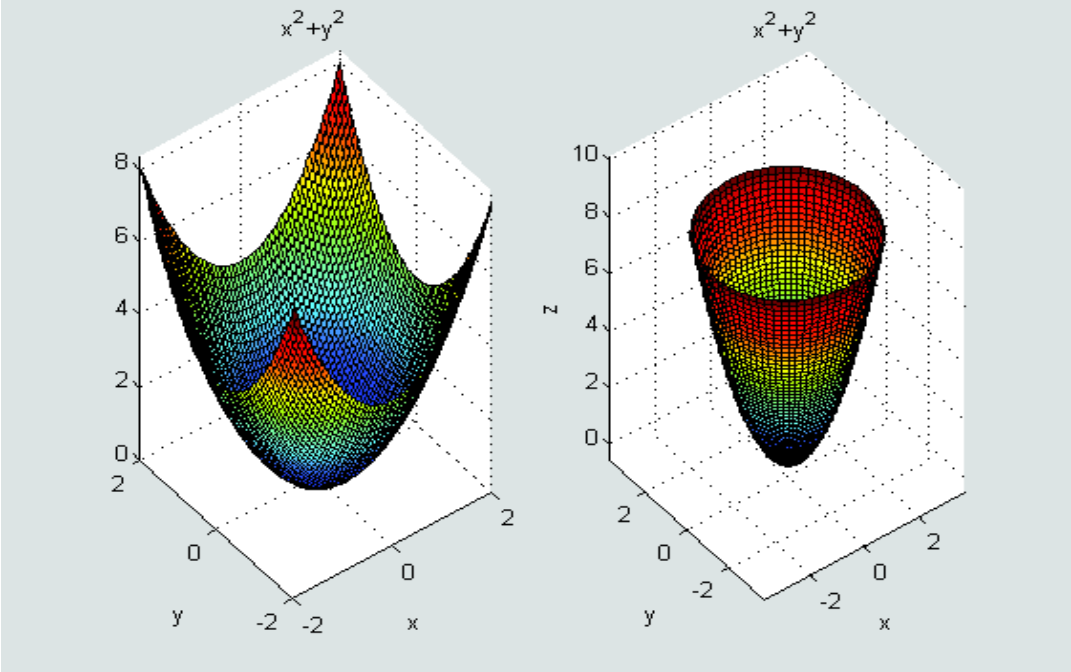
Again, the viewpoint is not quite standard, but the basic shape of the graph is immediately apparent. If it isn't, you can select the Rotate3D button and give yourself a tour of the surface until you throroughly understand it. Notice that the graph appears to be covered with a fine grid, as if it were woven. This is not simply an artistic effect. We will discuss the mathematical basis for the picture later when we consider the not-so-ez graphing method. ■

Most of the time, when plotting a surface given by $z = f(x, y)$, a rectangular domain for the independent variables is adequate. However, when dealing with a surface of revolution obtained, for example, by rotating a curve in the y,z plane around the z axis, it is more instructive to have a picture over a circular domain. We can achieve this using `ezsurf` by adding an optional argument, 'circ'. This produces a plot over the largest circle inside the rectangular domain we specify.

Example 4.3: Compare the graphs of $z = x^2 + y^2$ over the rectangle $[-2,2]\times[-2,2]$ with the plot drawn over the inscribed circle centered at the origin lying inside this square.

Solution: We use the `subplot` command to place these pictures side by side.

```
z=x^2+y^2;
subplot(1,2,1); ezsurf(z, [-2 2]);
subplot(1,2,2); ezsurf(z,[-2 2],'circ')
```



In this case, the `subplot` command sets up a 1x2 array to hold the graphical output. The left most panel is panel number 1 and the right most is panel number 2. The command `subplot(1,2,1)` directs the next graphical output to the first panel and `subplot(1,2,2)` directs the next output to the second panel.

Both surfaces are covered by a rectangular-like net, but the second shows the circular symmetry much more clearly. ■

4.2 Data Plotting

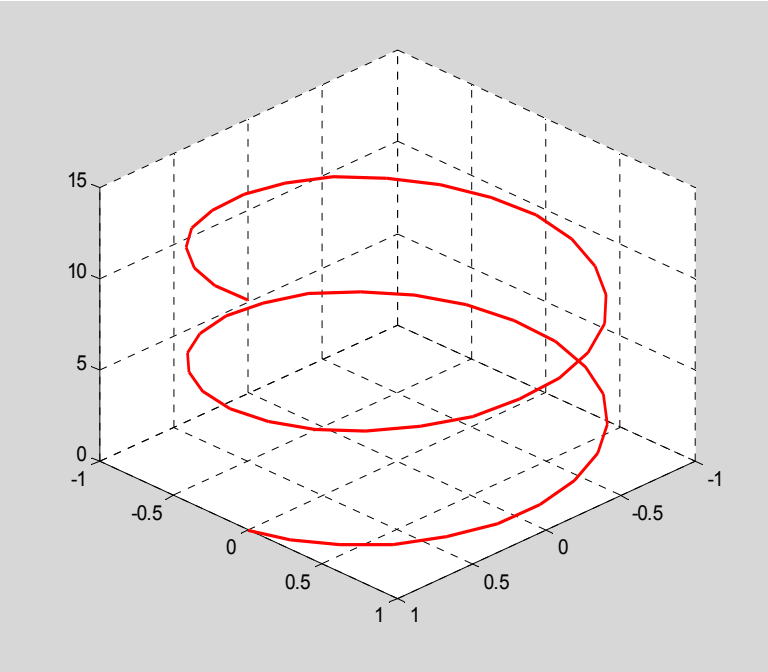
4.2.1 Curves

Plotting space curves is similar to plotting plane curves of the form $y = f(x)$. The principal difference is that the parameter plays the role of the independent variable and we must divide the parameter interval into small pieces to generate points on the curve, which Matlab then connects with line segments. Plotting options can be added to the plotting command, as we did for 2D plots.

Example 4.4: Use data plotting to plot the helix in Example 4.1**.** Add a second concentric helix to the figure.

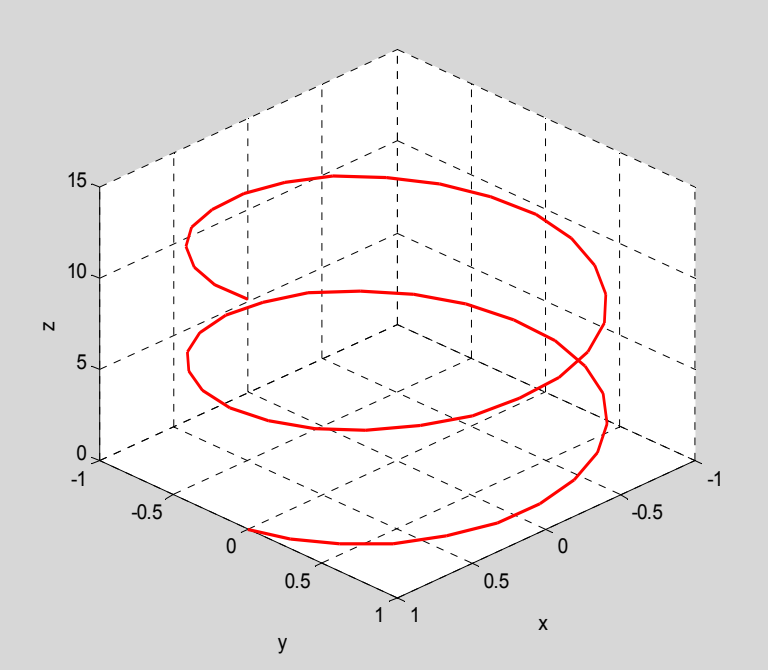
Solution: The appropriate command is now `plot3`.

```
t=linspace(0,4*pi,50);
x=cos(t); y=sin(t);z=t;
plot3(x,y,z, 'r', 'LineWidth', 2);
view([1 1 1]); grid on
```



Notice that we had to turn the grid on. The graph has no labels on the axes, or title. These must be added. For example, to add axis labels, which is quite important in parametric plotting, type

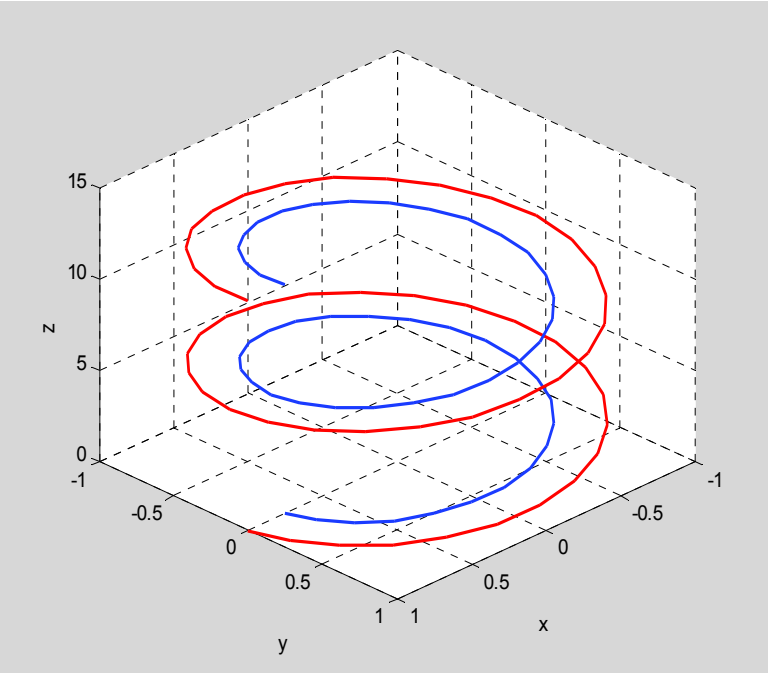
```
xlabel('x');ylabel('y');zlabel('z');
```



Naturally, you would want to do this in an M-file so as not to have to repeat all the lines each time you wished to change something.

We can add another curve to the picture. This can be done directly in the original `plot3` command or we can add it afterwards using `hold on`. For example, working from the graph already generated we can add another concentric helix via,

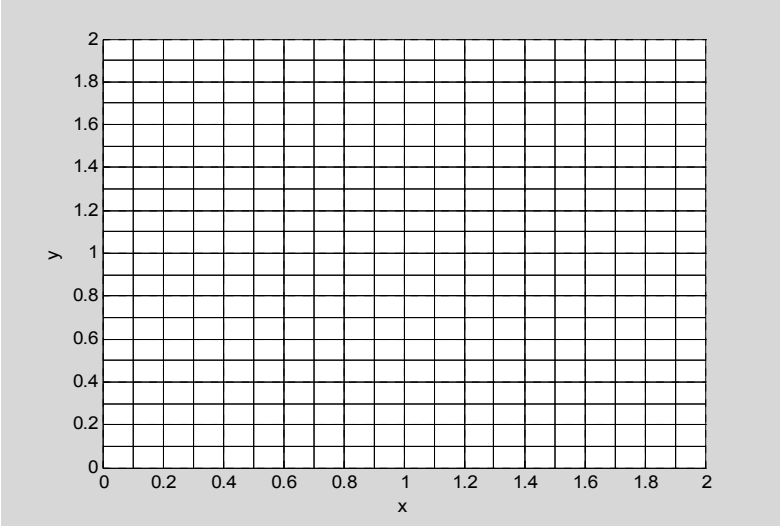
```
hold on;
x=.75*cos(t);y=.75*sin(t);z=t;
plot3(x,y,z, 'b', 'LineWidth', 2)
```



■

4.2.2 Surface Plots

Matlab plots surfaces in a way that is analogous to the way it plots curves. The latter are plotted as connected line segments. Surfaces are constructed as connected 4-sided patches. It is useful to understand Matlab's construction and not just mimic the steps. Suppose we want to plot the surface $z = x^2 + y^2$ over the square region $0 \le x \le 2$ and $0 \le y \le 2$. The idea is to divide this region into a rectangular grid. In the example shown below we use a 20 x 20 grid.



The vertices of the grid have coordinates $(0,0),(.1,0),(.2,0),\ldots,(2,0)$ in the bottom row, to $(0,2),(.1,2),(.2,2),\ldots,(2,2)$ in the top row. In this case we have 21 x 21 =441 grid vertices. Matlab stores the information about the grid vertices in a pair of matrices, i.e. rectangular arrays of numbers. In this case these would be 21 x 21 matrices. The *x* matrix contains the *x* coordinate of each grid vertex. All rows of the *x* matrix are identical. Each one just consists of the numbers 0, .1, .2, .3, …, 2, since these repeat in every row. The *y* matrix records the *y* values of the grid vertices. The top row of this matrix holds the *y* values for the points at the **bottom** of the grid. Thus, this row is 0, 0, 0, …, 0. The next row holds the *y* values from the second row from the bottom. Thus, this row is .1, .1, …, .1, etc. We will see below how Matlab constructs these matrices.

Once we have created these matrices, called the *meshgrid*, Matlab needs to calculate the *z* value on the surface $z = x^2 + y^2$ for each grid vertex. This is where we use array operations, as we did in plotting curves. When we write $z = x.\hat{}\,2 + y.\hat{}\,2$, Matlab will take every value in the *x* matrix described above and the value in the corresponding location in the *y* matrix and compute the value of *z* associated with that pair. Moreover, it stores these values in a matrix for *z* of exactly the same size (in this example 21 x 21), so that it knows what value of *z* goes with which pair of *x* and *y* values. This is crucial in enabling Matlab to create a coherent picture.

Let's illustrate these steps in a small problem. Instead of using 21 grid points in each direction, we will use three in the *x* direction and five in the *y* direction. Thus the *x* interval $[0,2]$ will be divided by the grid points 0, 1, and 2. The *y* interval will be divided by the points 0, .5, 1, 1.5, 2 . The `meshgrid` command is used to construct the matrices *x* and *y* described above. We do this as follows:

```
x=linspace(0,2,3) % list of three x values in interval [0,2]
y=linspace(0,2,5) % list of five y values in interval [0,2]
[ x y]=meshgrid(x,y)     % matrices describing 2D grid points
```

```
x =
```

```
     0     1     2

y =

          0    0.5000    1.0000    1.5000    2.0000

x =

     0     1     2
     0     1     2
     0     1     2
     0     1     2
     0     1     2

y =

          0          0          0
     0.5000     0.5000     0.5000
     1.0000     1.0000     1.0000
     1.5000     1.5000     1.5000
     2.0000     2.0000     2.0000
```

In the first two lines we create lists of the x and y points in each subinterval. Matlab will use these as inputs to the `meshgrid` command to construct the matrices that we described above. The `meshgrid` command takes the two input arrays x and y and produces two output matrices. It does this simply by repeating the entries of the array x a certain number of times and doing the same for y, after transposing the latter to a column array. Since we no longer have a need for the original list of x and y values, we assign these names as the output of `meshgrid`, as in the statement. This is a common practice in programming that helps one economize on names of variables. Normally, we would terminate each of the statements above with a semicolon, since the output is not at all of interest to us. Next we compute z.

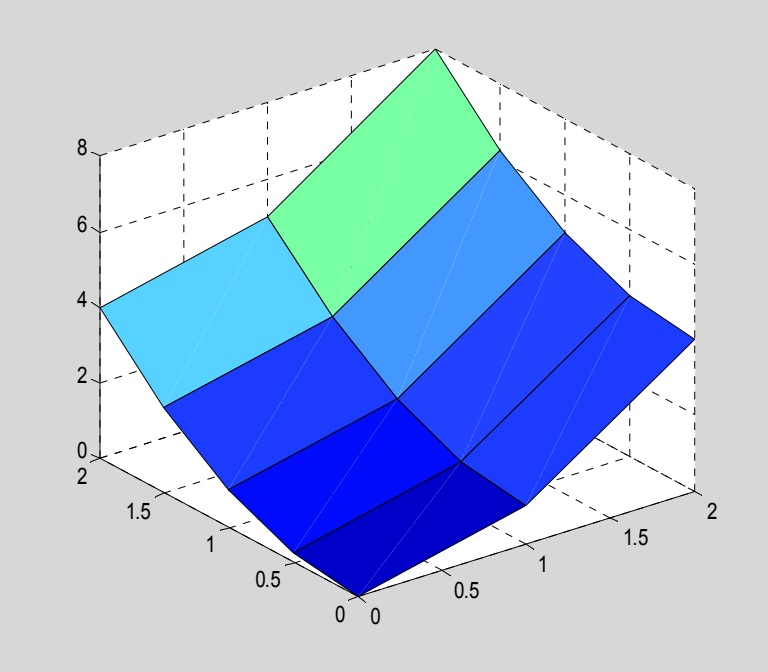
```
z=x.^2+y.^2
```

```
z =

          0    1.0000    4.0000
     0.2500    1.2500    4.2500
     1.0000    2.0000    5.0000
     2.2500    3.2500    6.2500
     4.0000    5.0000    8.0000
```

The reader ought to check that the values in the z matrix are obtained by applying the formula for z to the respective entries in the x and y matrices.

Now to complete the process, we use the `surf` command:

```
surf(x,y,z)
```



At this point the picture is not very informative, but it illustrates Matlab's methodology. Each location in the matrices x, y, and z determines a point (x_0, y_0, z_0) on the surface.

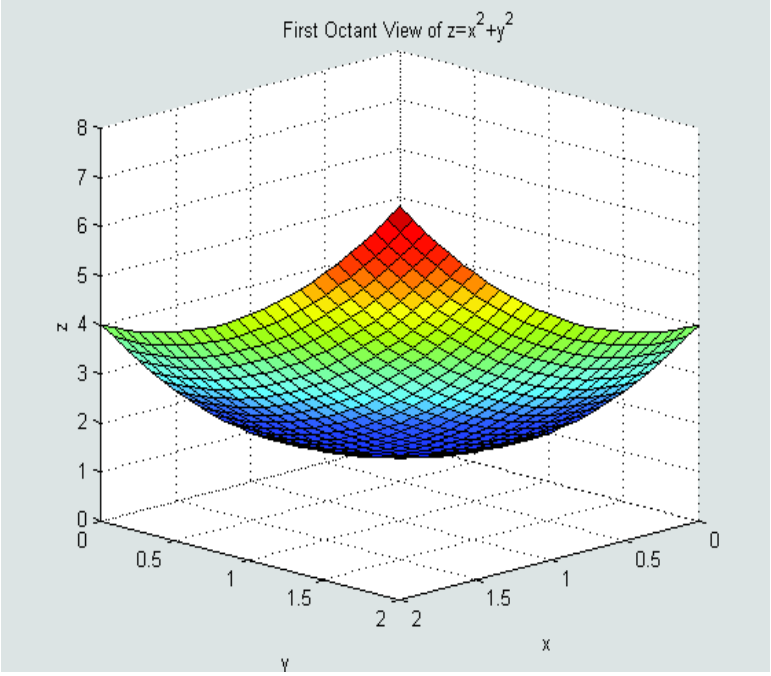
These points are connected to each other by line segments according to the layout in the grid. Matlab then creates colored "patches" from this collection of four-sided polygons. The patches typically look like parallelograms (in this example they actually are), though in fact they may not actually be planar (in general, each consists of two triangles with the same color, attached along a common diagonal edge). When the grid is fine enough we get the appearance of a smooth surface.

Although a finer grid gives a better approximation to the surface, it requires greater computational effort to render all the information on the screen. As a result, when we attempt to interact with the 3D model using some of the interactive features we may find the results slow and jerky. This is somewhat dependent on the processing speed of your individual computer. As a rule of thumb, you should probably limit exploratory work to grids of no more than 25 x 25 points.

Example 4.5: Use `surf` to plot $z = x^2 + y^2$ over the rectangle $[-2,2]\times[-2,2]$.

Solution: Normally, the following lines would be part of an M-file. They produce the figure shown below.

```
x=linspace(0,2,25); y=linspace(0,2,25);
[x y]=meshgrid(x,y);
z=x.^2+y.^2;
surf(x,y,z); view([1 1 .4]); xlabel('x');ylabel('y');zlabel('z');
title('First Octant View of z=x^2+y^2')
```



The `title` command shows how to place a nicely formatted mathematical expression in your title. ■

The curves on the surface created by the grid pattern are approximations to the sections of the surface by vertical planes $x = const$ and $y = const$. In effect, each patch in the interior is bounded by two pairs of such curves, since on the edge of any patch only one coordinate changes at a time. In the next lesson we will consider how to obtain sections of the surface where $z = const$, which are known as contour curves.

4.3 Summary

We discussed the following commands related to plotting:

`ezplot3`: - plotting 3D space curves. Has a nice animation option.

`ezsurf`: - ez plotter for 3D surfaces given by functions. Convenient, but not so easy to customize output. Particularly difficult to combine 3D plots.

`plot3`: - 3D curve plotter using data points

`meshgrid`:- creates matrices of 2D grid points from interval decomposition

`surf`: - 3D surface plotter using data points. Output needs to be customized, but provides greatest flexibility

Some extras we touched on:

`subplot`:- create an array of plots that are displayed and printed together

`view`:- set the viewpoint for drawing the 3D scene. The view can be specified in terms of a point in space or through a pair of angles. See the help page on `view` for details.

4.4 Exercises 

Exercise 4.1 Let $\mathbf{r}(t) = \langle \cos t, \quad \cos 2t, \quad \cos 3t \rangle$, over the interval $0 \le t \le 2\pi$.

a) Plot this parametric curve using `ezplot3`. Use the 'animate' option to view how the curve is traversed as the parameter varies. How many times is the curve traversed when *t* varies over the parameter interval? What is the smallest parameter interval for which the curve will be traversed exactly once?

b) The following M-file is supposed to display the parametric plot in a) viewed from (1,1,1) and then display three views of the same plot showing the projection into the coordinate planes. Complete the missing arguments in the code and place titles that state the projection that is being viewed on that screen. Note: The `pause` command halts execution of the M-file until the user presses a key. You may print a screen while the program is paused.

```
syms t
x=cos(t);y=cos(2*t); z=cos(3*t);
clf
ezplot3(x,y,z,[0, 2*pi]);
view([1 1 1]);
title(???)
pause
view(???);
title(???)
pause
view([1 0 0])
title(???)
pause
view([0 1 0])
title(???)
```

c) Print your M-file and the four screens that it generates. Make sure to include your name. You can also publish this to HTML. In doing so, you should first remove the `pause` commands (which cause the publishing program to "hang."). Then, insert cell dividers after each `view` command. Put appropriate titles in each cell divider line. The various graphs should now print following the commands that generated them.

d) BONUS!! This is not a Matlab problem. Find an equation of the form $y = f(x)$ for the projection of the curve into the *x*,*y* plane. Find an equation of the form $z = g(x)$ for the projection of the curve into the *x*,*z* plane. Can the projection into the *y*,*z* plane be described as the graph of a function $z = h(y)$? Justify your answer by referring to the figure you obtained in part b).

Exercise 4.2 Using `plot3`, write an M-file that draws

a) the helix with parametric equations $x = \cos t$, $y = \sin t$, $z = t$, for $0 \le t \le 2\pi$.

b) On the same graph, but in a different **color**, **line style**, and **thickness**, draws a portion of the tangent line to the helix at the point $(\sqrt{2}/2, \sqrt{2}/2, \pi/4)$. The portion of the tangent line should be chosen to extend as far as possible within the first octant. Include **labels** on the coordinate axes, and a **grid**. Publish to HTML. Make sure your name appears in the printout (you can include it in a comment line, for example), and the combined graphs use a view that clearly shows the tangent line.

Exercise 4.3 Using `ezsurf` make a plot of the graph of $z = 2x^3 + xy^2 + 5x^2 + y^2$ over the rectangle $-2.5 \le x \le .6$ and $-3 \le y \le 3$.

Exercise 4.4 Using `surf` make a plot of the graph of $f(x,y) = (x^2 - y^2)^2$ over the region $[-2,2]\times[-2,2]$. Use a 25x25 grid.

Exercise 4.5 Write an M-file to plot the parabolic cylinder $y = x^2$ using the `surf` command. You will need to treat *x* and *z* as independent variables, using a meshgrid in the *x*,*z* plane and generating the *y* values using the formula. Use a 15 x 15 grid of the rectangle $-2 \le x \le 2$ and $-2 \le z \le 2$. Locate the **viewpoint** at (1,1,1), add **labels** to each axis, and put a **title** on the graph that includes the **formula** for the surface. Publish to HTML. (Be sure to include **your name** in a comment).