Notes on MATLAB

Table of Contents

N.B. We use the symbol ■ in the text to denote the end of the discussion of an example.

1: Interactive Graphing

Matlab, which is a name contracted from MATrix LAB, is a widely used scientific software package. The goal of these notes is to teach you how to use it to

- construct graphs of curves and surfaces
- perform symbolic calculations, such as differentiation and integration
- write programs to facilitate carrying out the first two tasks
- present your results as a finished report suitable for printing or presentation.

In this section we will focus on interactive graphing. Later, we will consider aspects of Matlab that deal with numerical calculations, done both interactively and via programming.

We will make use of the Symbolic Toolbox, which is installed on the system in the Math department lab and comes bundled as part of the Student Edition of Matlab. Using the Symbolic Toolbox allows us to employ a language and style that are closer to everyday mathematics. This often makes it easier to figure out how to do things.

As we said, this chapter will involve a lot of interactive work with the program. This means "show and tell." We will demonstrate what to do in class, but not write much about how to do it. So coming to the lab will be very useful in figuring out how to complete the assignment described later.

First we will create plots of expressions in one variable.

1.1 Simple Plots

First declare a symbolic variable:

```
syms x
```

Matlab gives no response, but having declared the variable symbolic, we can enter expressions in this variable.

```
y=sin(x)
```

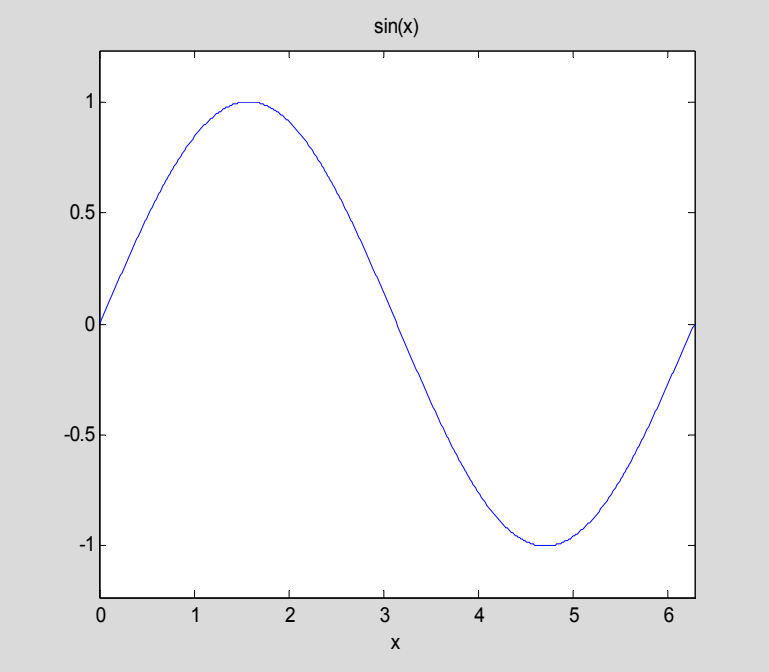
```
y =
sin(x)
```

After a brief delay while Matlab loads the package of symbolic commands, the system returns the definition you proposed.

Example 1.1**:** Plot $y = \sin(x)$ on the interval $[0, 2\pi]$. Enhance the graph with additional features.

Solution: We enter the following command at the prompt.

```
ezplot(y,[0,2*pi])
```



This figure appears in a new window, called a Figure window. We can customize many features of the graph interactively using the window. Among the features we will learn to change are

a) adding grid lines

b) adding or changing labels and tick marks on the axes

c) changing the color of the graph

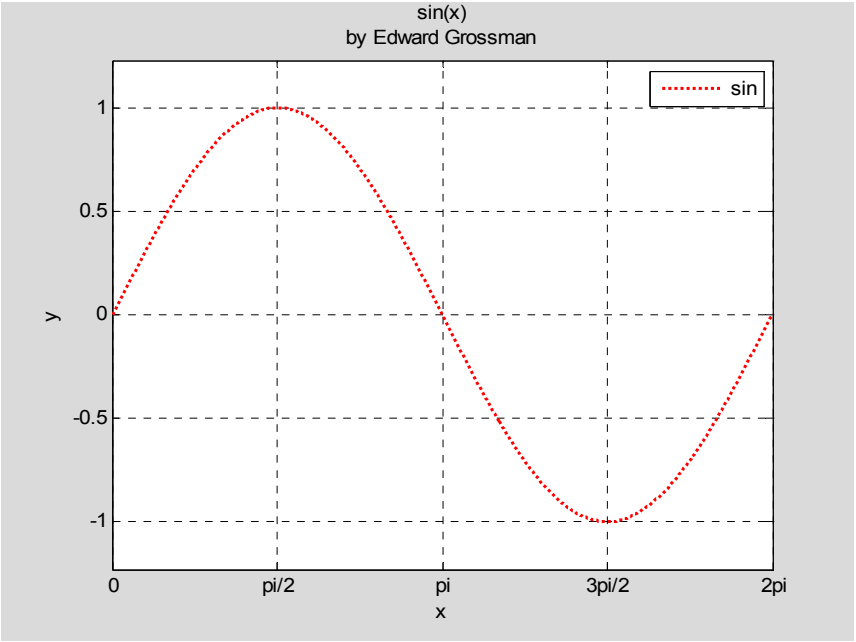
d) changing the plotting style of the graph, for example, dotted, or a heavier line

e) adding a legend (useful when we have more than one graph displayed on the same axes)

f) adding or changing the graph title

g) changing the tick marks on the axes

All of these can be done using the Plot Tools icon at the top of the Figure window. After some experimentation, you should be able to use this tool to produce the following picture:



■

Now suppose we want to add another graph to this one.

Example 1.2: Plot the *sine* function and its derivative on the same plot

Solution: In this case we know the derivative and we can simply define it directly. For more complicated functions, we can have Matlab compute it for us.

```
y2=cos(x)
```

```
y2 =
cos(x)
```

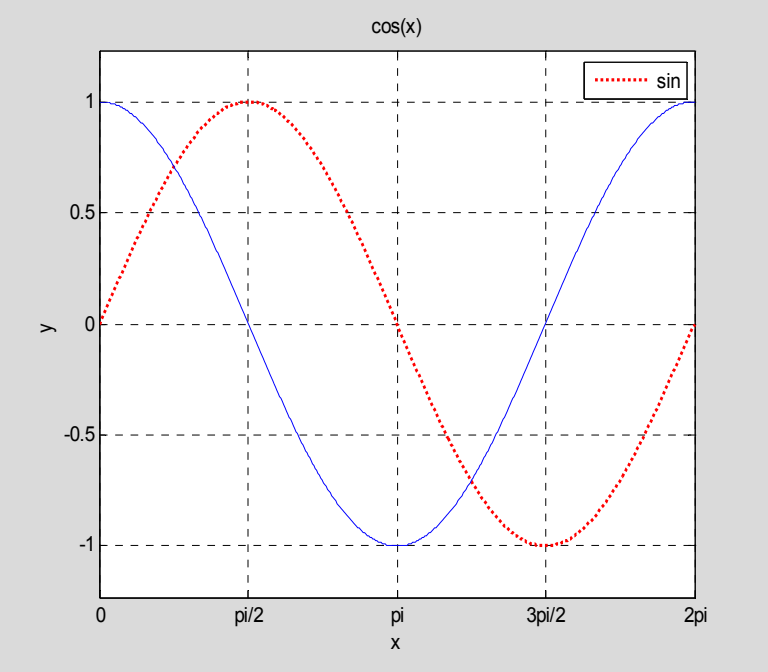
In order to add the graph of $\cos(x)$ to the previously drawn graph, we must tell Matlab not to erase the first graph when it draws the second one. This is done by typing the command

```
hold on
```

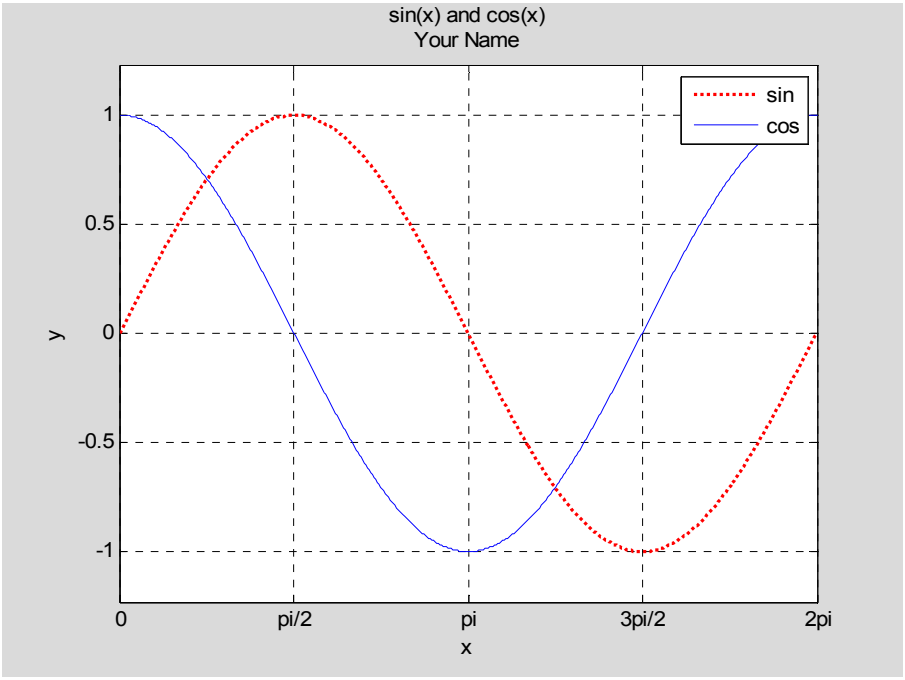
.

Then we simply repeat the `ezplot` command with y2 instead of y.

```
ezplot(y2,[0, 2*pi])
```



Besides adding the new graph to the figure window, Matlab changed the title and did not update the legend. We need to manipulate these features manually, using the Plot Tools windows. See if you can produce the following picture:



■

That was pretty easy. But there is one more challenge for this lesson. In Example 1.2 both graphs produced the same range. When the ranges are different, which is usually the case, the `ezplot` command uses the range of the last graph plotted. This may not produce the most useful looking graph.

Example 1.3: Plot the expression $x\sin(4x)$ and its derivative on the interval $[0, 2\pi]$.

Solution: This time we will use the Matlab command `diff` to compute the derivative. We can proceed as follows

```
y=x*sin(4*x); dy=diff(y,x)    % the diff command computes the derivative
dy =
sin(4*x)+4*x*cos(4*x)
```

(The semi-colon used in the input line suppresses the display of the output of the first command).

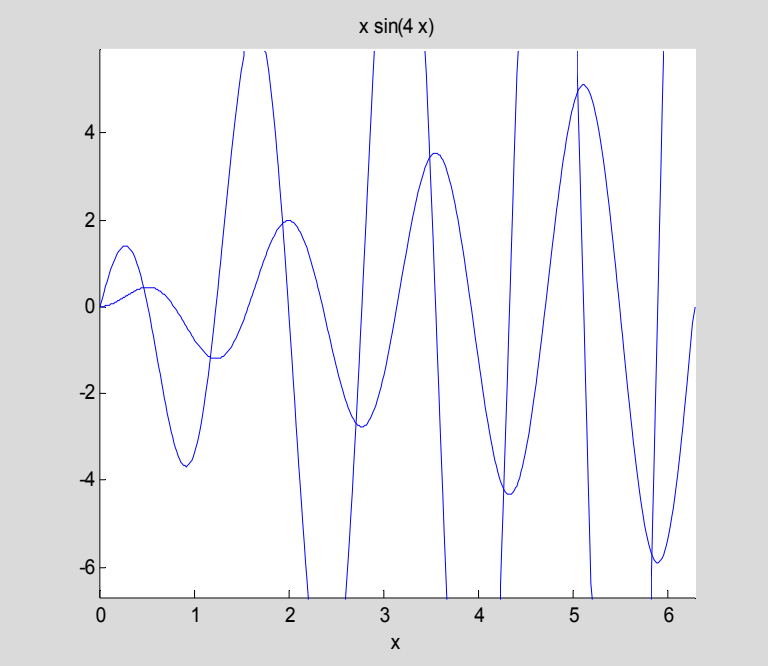
Before we create the plots, we must clear the previous graphs. We do this via

```
clf
```

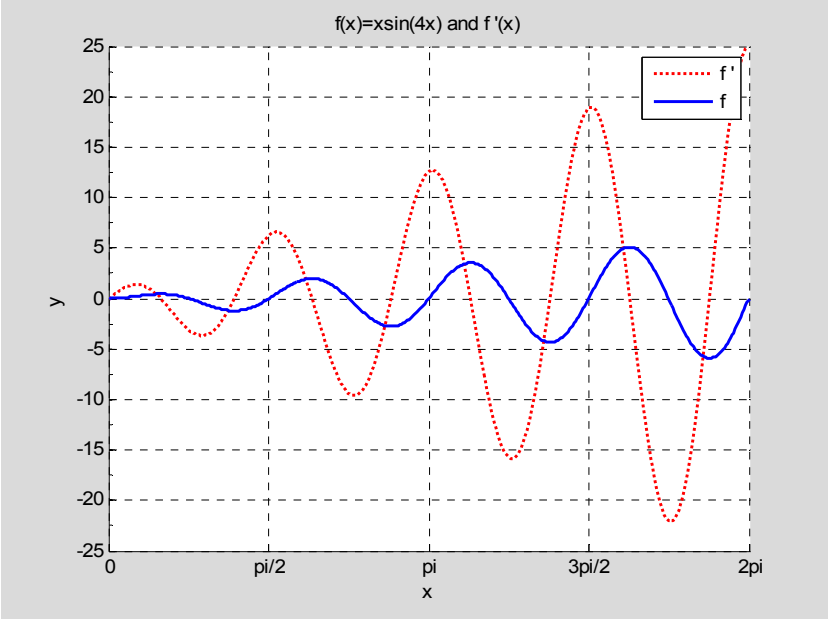
which stands for "clear figure." We must again ensure that "hold on" is in effect.

Now we use our `ezplot` command:

```
hold on;ezplot(dy,[0,2*pi]);ezplot(y,[0,2*pi])
```



Because we plotted the derivative first and the function second, part of the first graph was chopped off. Interactively, we can recover a more useful picture by adjusting the range of the axes. (This can also be done by simply reversing the plotting order, but discovering the best order would be time consuming when we had to deal with more than two expressions.) Use Plot Tools to obtain the picture below by changing the range of the *y*-axis and modifying the labels and tickmarks on the *x*-axis.



■

1.2 Summary

OK, that's it for the first lesson. Here's what we discussed in terms of commands:

`clf` – clear graphics figure

`diff` – compute a symbolic derivative with respect to a variable

`ezplot` – name says it all

`hold on` - draw new graph on top of old one (there is also `hold off` if you want to go back to the overwrite mode)

`syms` – declaring symbolic variables

Your exploration with Plot Tools should have also taught you a lot about the structure of Matlab's graphs. You also became familiar with manipulating commands in the Command Window. In addition to items a) to g) listed earlier, you should also know how to change the range of the axes. Not bad for the first lesson. Later we will see how to control these features in a program, so they do not have to be repeated when you need to change the input to a problem.

1.3 Exercises

In each of the following exercises you should produce the specified plot on the stated interval. Your plot should contain the following features:

a) a legend identifying the graph(s)

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 and appropriate tick marks on both axes

e) graph should display grid lines

f) A title similar to the one produced in the last graph above. If you will be handing in the output, the title should also contain YOUR NAME.

Exercise 1.1 Let $y = \sin(x^2)\cos(x)$. Produce a plot of y and y' over the interval $[-\pi, \pi]$. The x-axis tick marks should be at the points with coordinates $-\pi, -\pi/2, 0, \pi/2, \pi$.

Exercise 1.2 Let $y = \dfrac{2x}{\sqrt{x^2+1}}$. Produce a plot of y and y' over the interval $[-2, 2]$.
(Note: You can enter the square root either using exponents or the Matlab function `sqrt`.)

Exercise 1.3 Plot the expressions $y = \sqrt{1-x^2}$ and $y = -\sqrt{1-x^2}$ on the interval $[-1, 1]$. The two graphs should form the circle $x^2 + y^2 = 1$. Look up the `axis` command in Help to see how to get the picture to actually look like a circle, rather than an ellipse.

Exercise 1.4 Let $y = \dfrac{1}{5}\left(\dfrac{\sin 5x}{\sin x}\right)^2$. Produce a plot of y and y' over the interval $[-\pi, \pi]$.

From the graph, what is the value of $\lim\limits_{x \to 0} \dfrac{1}{5}\left(\dfrac{\sin 5x}{\sin x}\right)^2$? Confirm the result separately using what you know about limits.

Exercise 1.5 What is the smallest positive value of x (in radians) such that $\tan x = x$? Estimate the answer by plotting the graphs of $y = \tan x$ and $y = x$ on a suitable interval and examining the points of intersection. You can use the Data Cursor icon in the figure window to estimate coordinates of points as well as to leave a data marker that can be printed to show your estimate.

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.

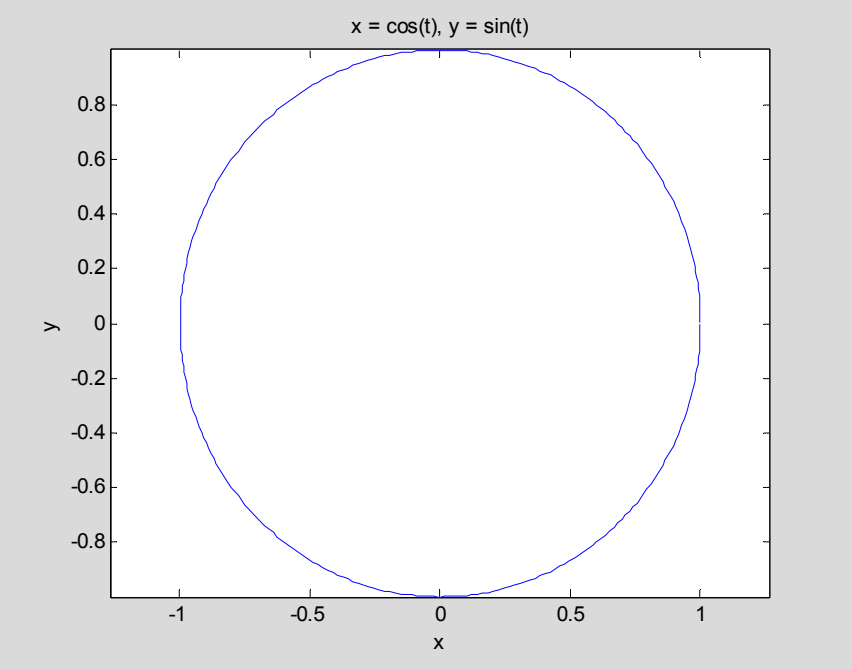
2.1 A Simple M-File

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.

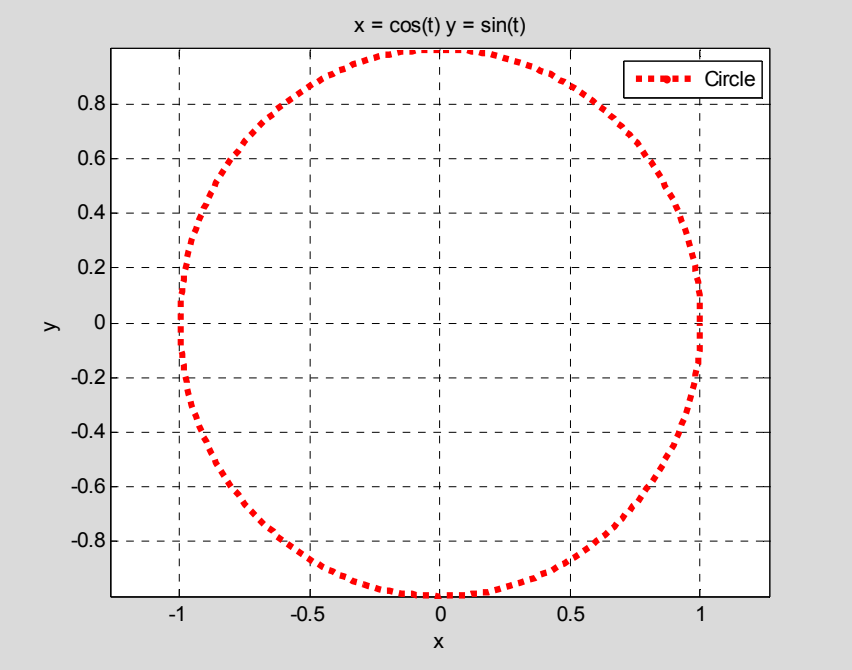
Here is the modified file which we save as **param2.m**:

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

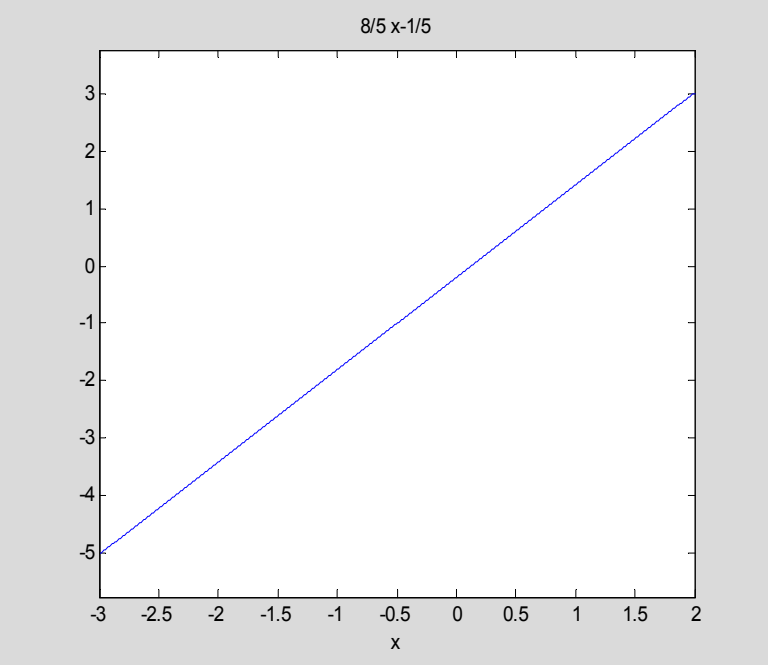
In this chapter we will learn an alternate approach to plotting in Matlab. This approach is more general than using the ez-commands. It will enable us to plot figures that can't be created using the ez-commands. It is particularly valuable for plotting in three dimensions.

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 $x = 2$.

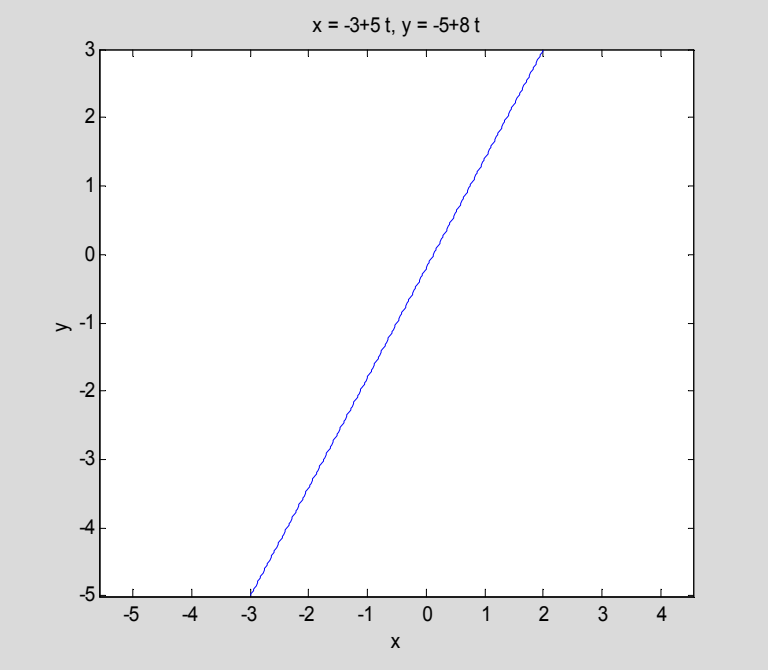
```
syms x
ezplot(8/5*x-1/5,[-3 2])
```



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.

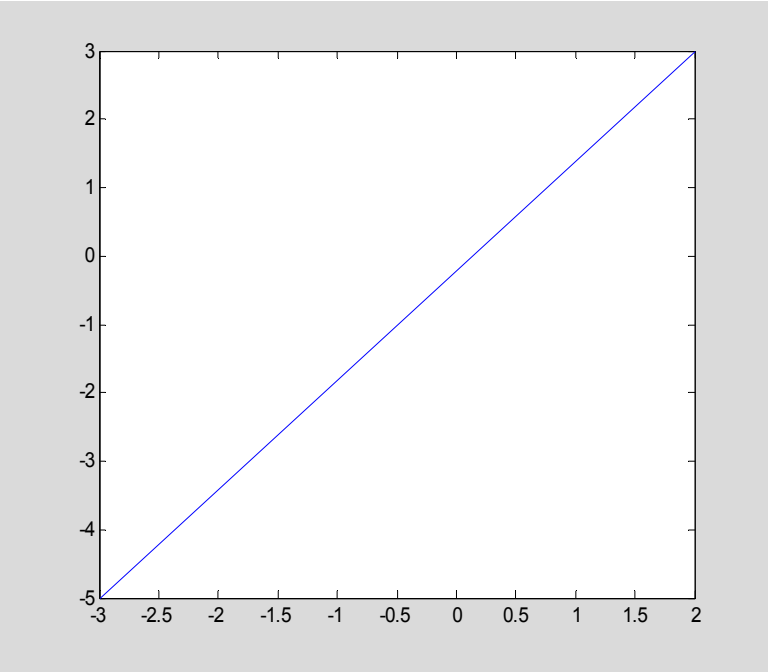
Solution:

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`:

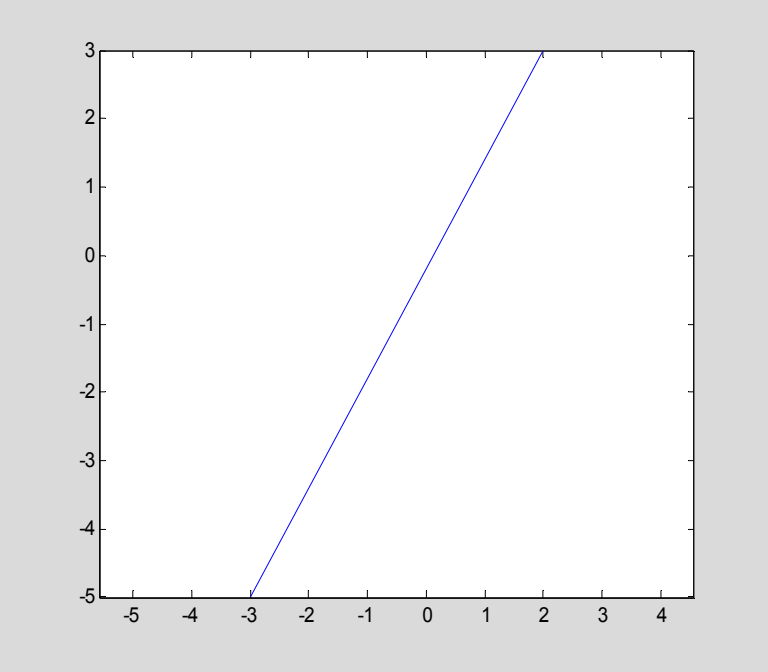
```
plot(x,y)
```



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.

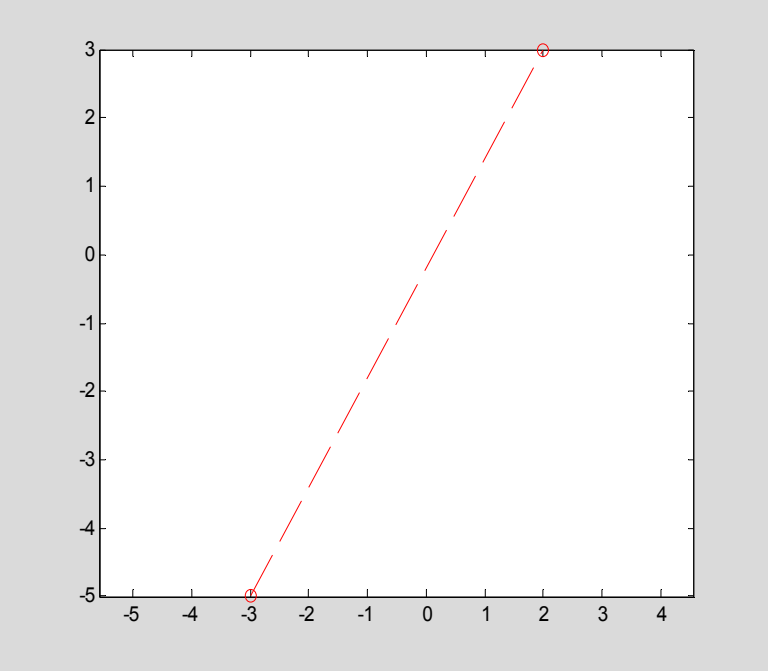
You may not like that the graph makes the line appear to have slope 1. We can get Matlab to display a visually correct representation by modifying the aspect ratio, which relates the unit of length displayed in the *x* direction to that displayed in the *y* direction. We can make these distances equal,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

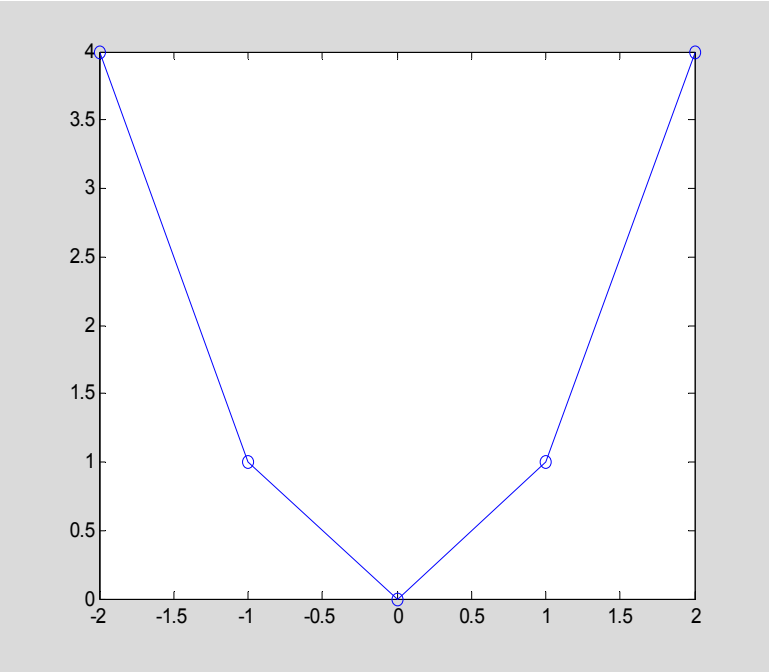
(Note: If you omitted the dot Matlab will complain.

```
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.)

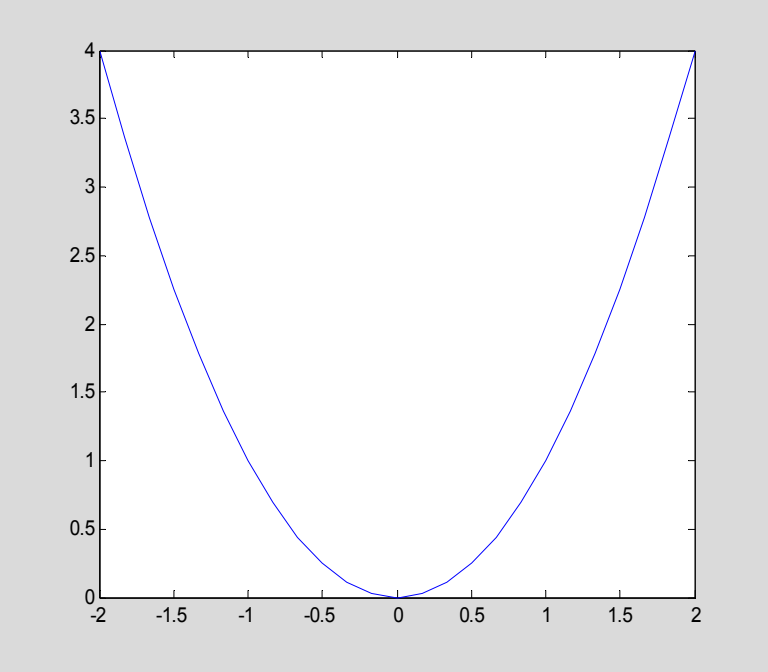
c) Now we can let Matlab plot the points whose *x* coordinates are taken from the *x* list and whose *y* coordinates are taken from the *y* list.

```
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)
```

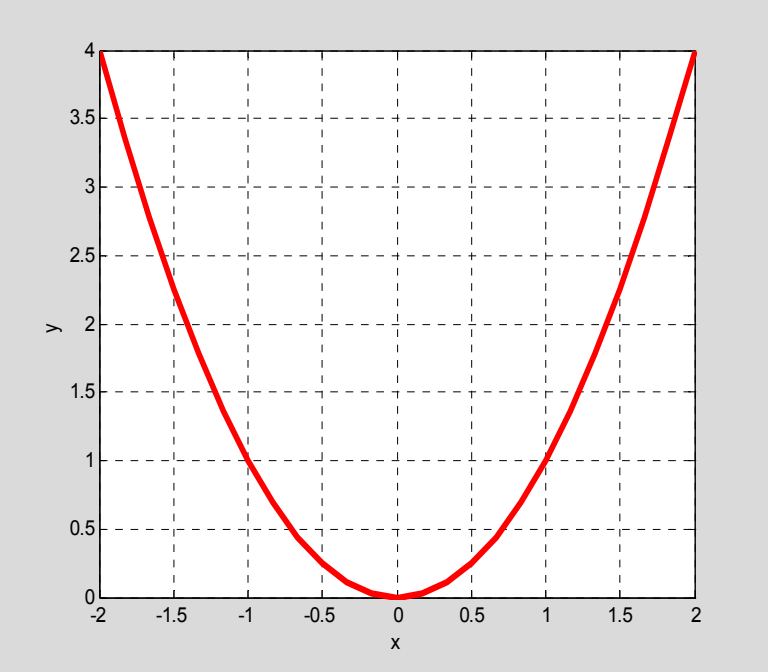


Much better. ■

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 1/2 1/4 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

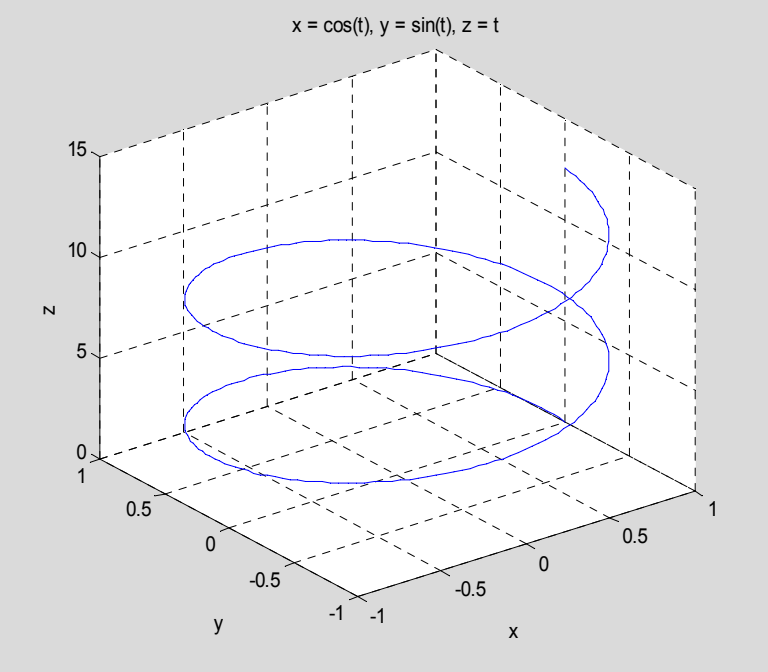
4.1.1 Curves

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:

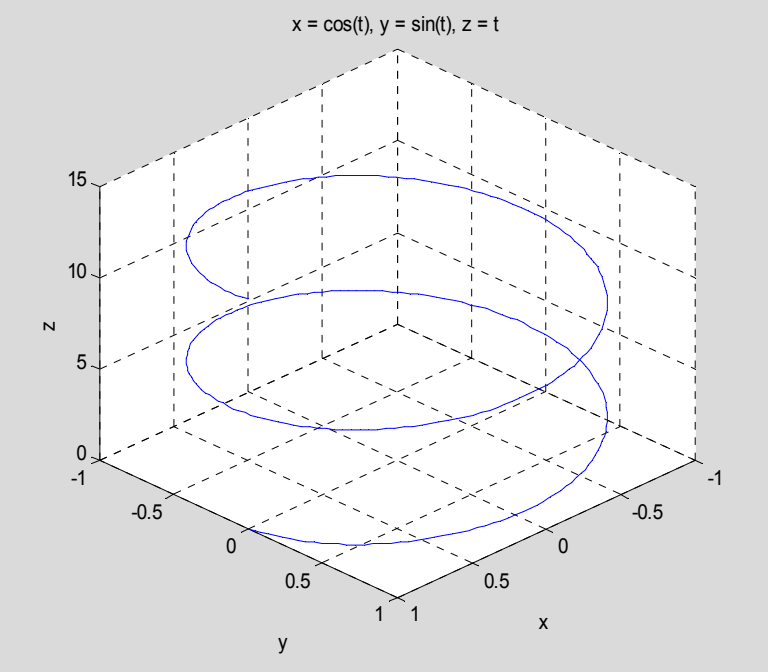
```
syms t;
x=cos(t); y=sin(t); z=t;
ezplot3(x,y,z, [0, 4*pi])    % The list [0, 4*pi] gives the range of the parameter t.
```



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. ■

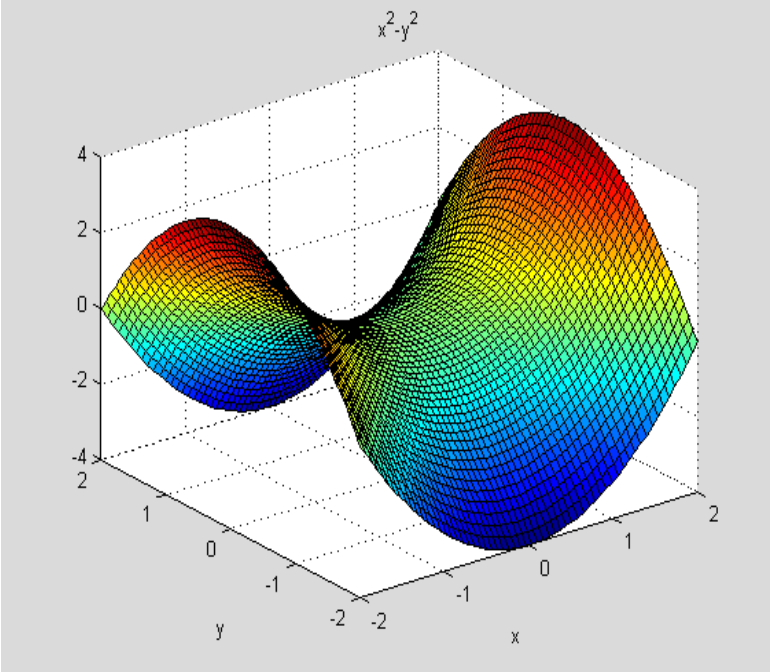
4.1.2 Surfaces

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.
```



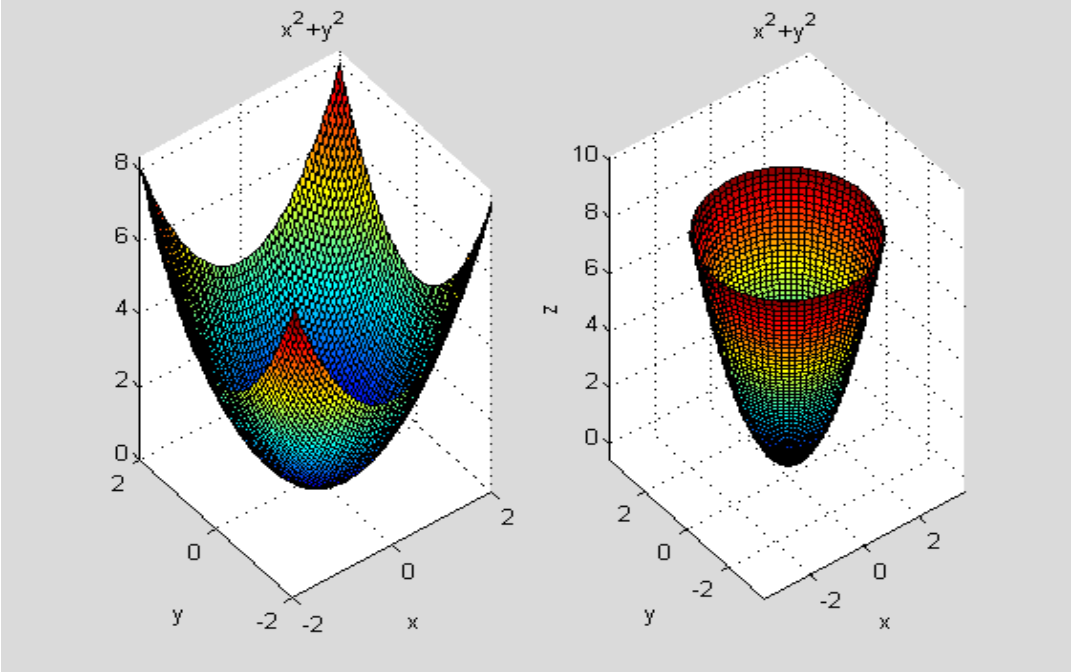
Again, the viewpoint is not quite standard, but the basic shape of the graph is immediately apparent. If it isn't, you can select the Rotate3D button and give yourself a tour of the surface until you throroughly understand it. Notice that the graph appears to be covered with a fine grid, as if it were 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

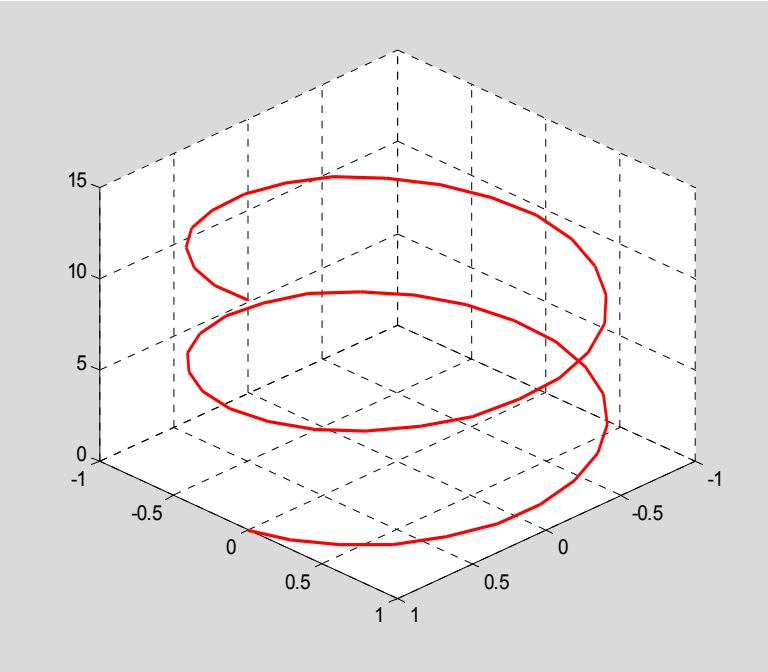
4.2.1 Curves

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.

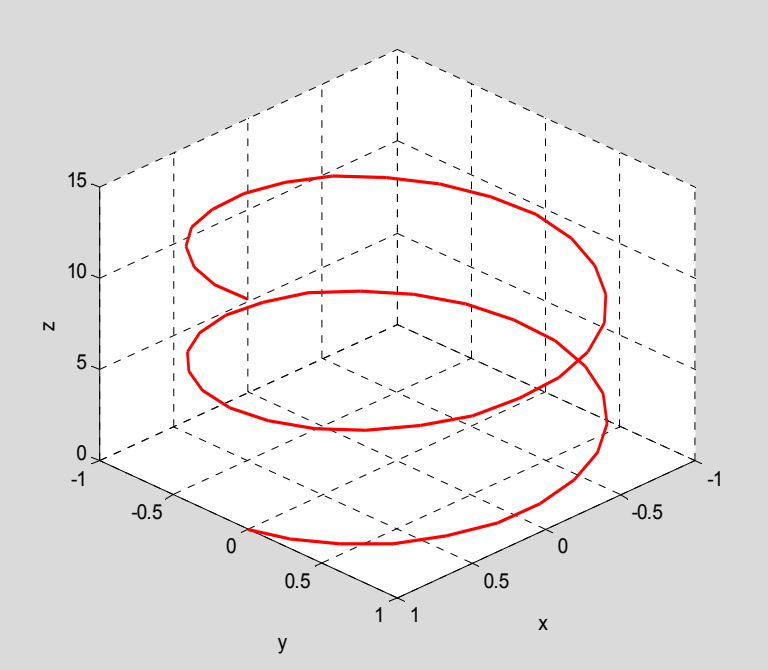
Solution: The appropriate command is now `plot3`.

```
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

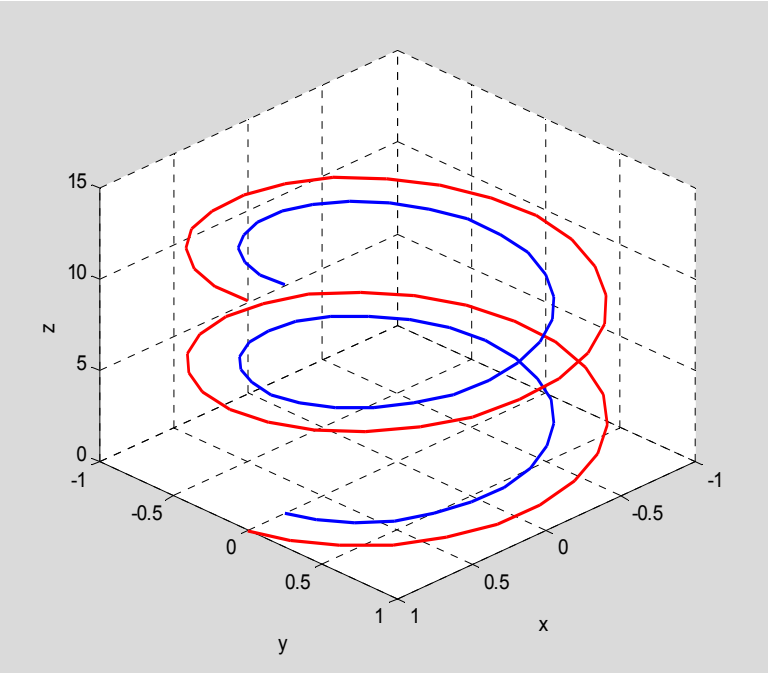
```
xlabel('x');ylabel('y');zlabel('z');
```



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,

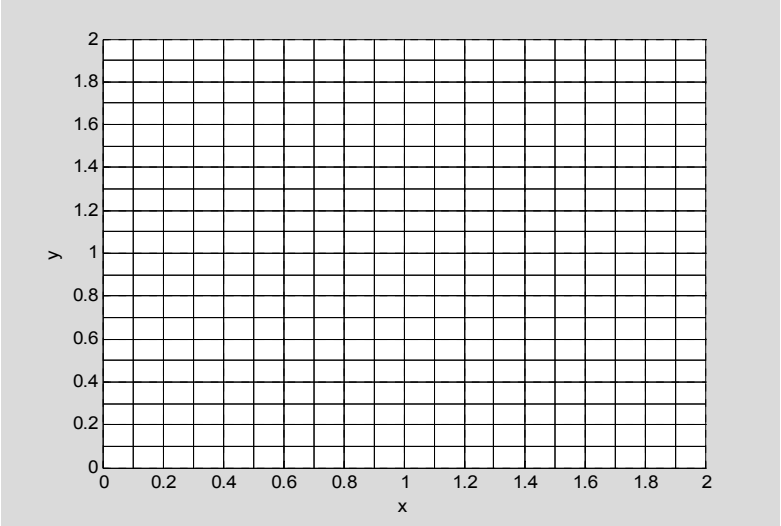
```
hold on;
x=.75*cos(t);y=.75*sin(t);z=t;
plot3(x,y,z, 'b', 'LineWidth', 2)
```



■

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.

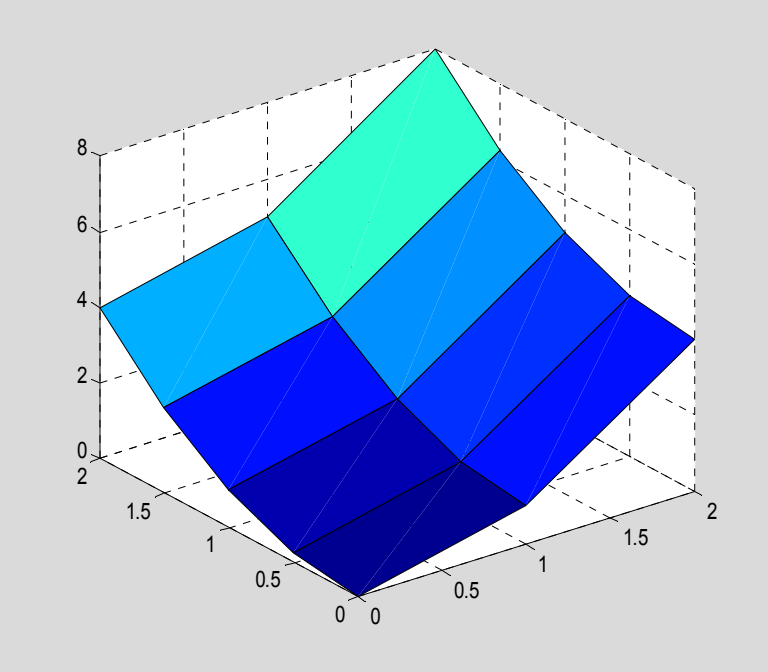
```
z=x.^2+y.^2
```

```
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.

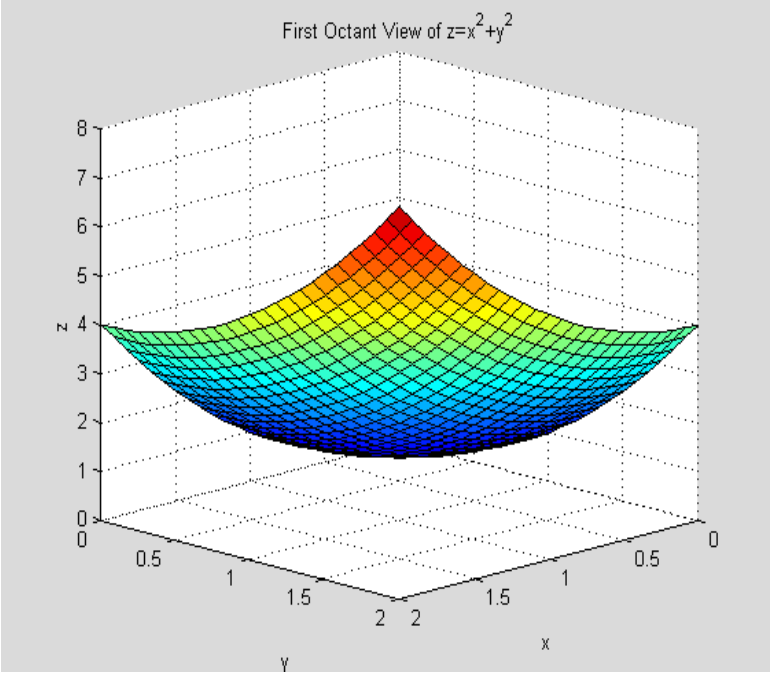
These points are connected to each other by line segments according to the layout in the grid. Matlab then creates colored "patches" from this collection of four-sided polygons. The patches typically look like parallelograms (in this example they actually are), though in fact they may not actually be 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).

5: Contour Plots

In this section we will study contour plots. These are related to curves defined implicitly as the solutions of an equation $f(x,y)=0$, so we will take a look at that topic as well.

5.1 Basic Contour Plots

Matlab has several commands for generating contour plots. We will omit the ez-commands because one can't easily customize them in ways that are particularly valuable for this type of plot.

The standard contour plot commands use the same general framework as `surf`. Namely, we generate a pair of matrices holding the x and y coordinates of the grid points, then generate the z values from these using a vectorized expression. In these notes we will analyze the contour plot of the surface $z=x^3+y^3-3xy$. We describe the process step-by-step, but the commands should be written to an M-file for convenient modification.

Example 5.1: Generate a contour plot of the function $z=x^3+y^3-3xy$ over the square $-2\le x\le 2$ and $-2\le y\le 2$.

Solution:

a) Select a plotting region:

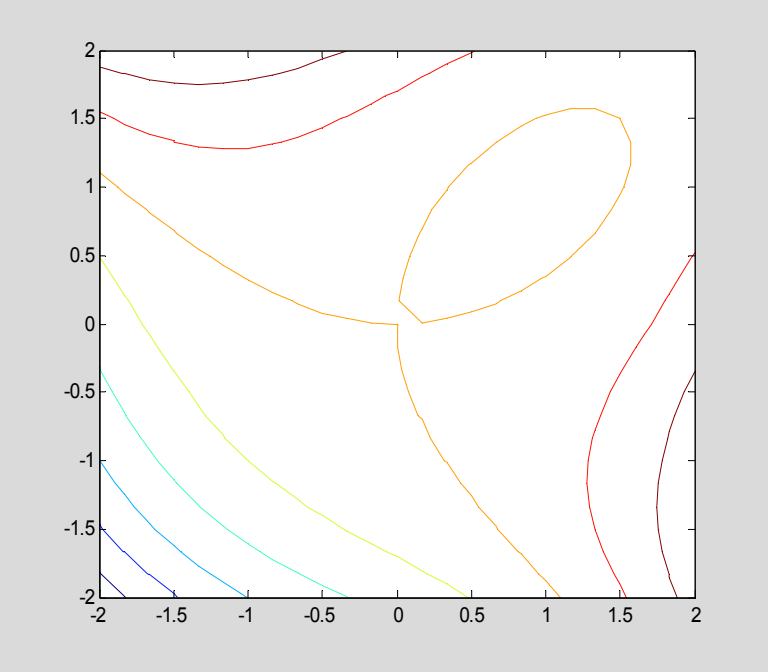
```
x=linspace(-2,2,25); y=linspace(-2,2,25);[x y]=meshgrid(x,y);
```

Now generate the z values using a vectorized expression for z.

```
z=x.^3+y.^3-3*x.*y;
```

We generate a 2D contour plot using the command `contour`.

```
contour(x,y,z)
```



■

Matlab generates a plot using a default number of level curves (contours with constant z value.) They are not labeled with the z value associated with each curve. The colors enable one to distinguish the relative heights on each curve. Reds and oranges correspond to larger z values than blues and purples. (To remember, think of the hot, red sun (high) and the cool, blue oceans (low) – OK, it helps me.)

5.2 Labeled Contour Plots

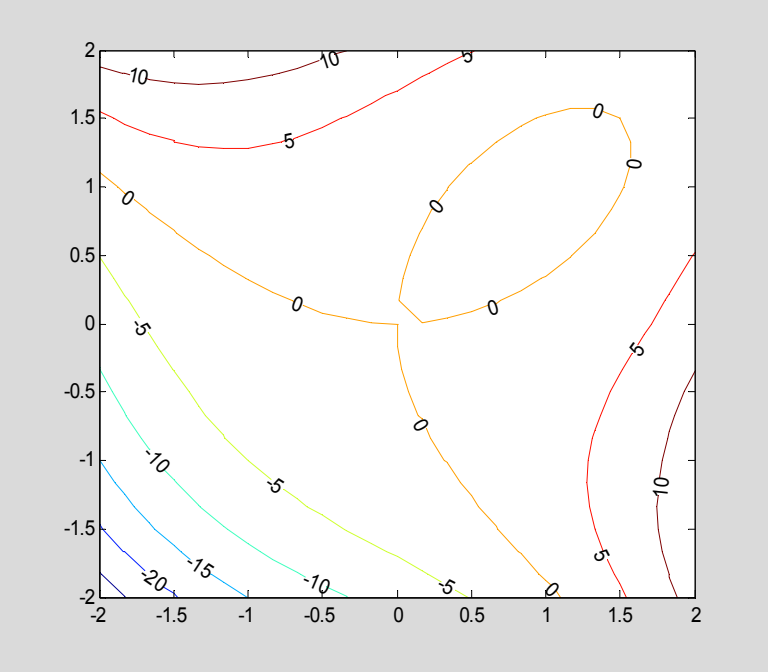
The first improvement over the basic contour plot would be to get Matlab to label the curves with the associated z value. We can do this by passing to the function `clabel` the so-called contour matrix (a matrix that holds all the data for drawing the curves) and the "handle" to the contour plot. You will recall from Chapter 2 that the handle is a name that we can ask Matlab to assign to identify a particular graphic object. We can access these bits of important information about the plot by assigning variables to the output of the `contour` command. Without going into excruciating details on all of this, let's see how it is done.

Example 5.2: Generate a labeled contour plot of the function $z = x^3 + y^3 - 3xy$ over the square $-2 \le x \le 2$ and $-2 \le y \le 2$.

Solution:

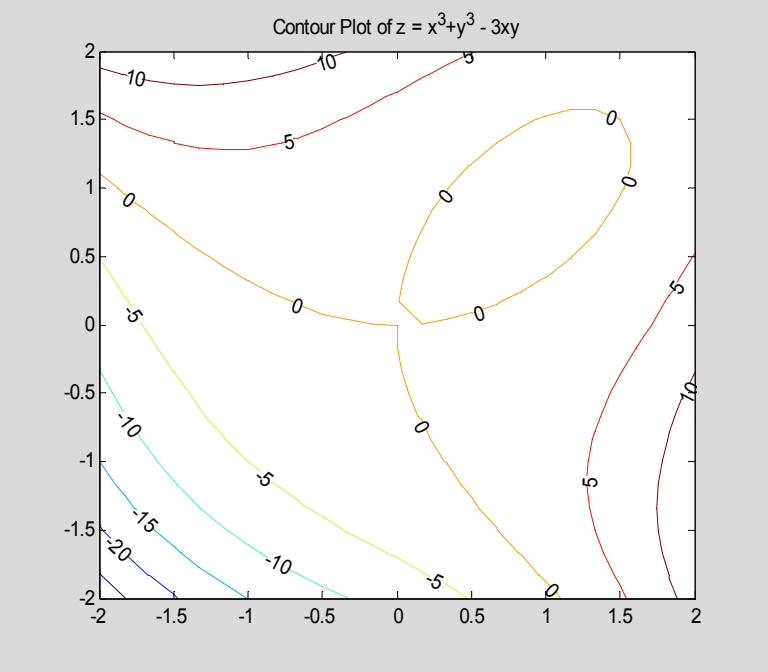
```
x=linspace(-2,2,25); y=linspace(-2,2,25);[x y]=meshgrid(x,y);
z=x.^3+y.^3-3*x.*y;
[C h]=contour(x,y,z);  %generates graph and assigns appropriate
variables to output for labeling
```

```
clabel(C,h);
```



This is much better. We can add a grid, if we wish, as well as axis labels and title in the usual way. Just to remind you, here is a nice title:

```
title('Contour Plot of z = x^3+y^3 - 3xy')
```



■

It is very instructive to explore this plot with the Data Cursor, which can be activated using an icon at the top of the figure window. (This feature may not work on some systems because of issues related to graphics card drivers.) As you move the cursor through the figure, the program displays both the coordinates of the point under the cursor and the *z* value (level value) of the point. Wouldn't it be great if you could click the mouse button and have the program generate the specific level curve that passes through a point, (in other words, connect all the *x*,*y* values at that same level)? Well, sorry, but we can't do this. However, as a consolation prize we can write our M-file to produce specific level curves.

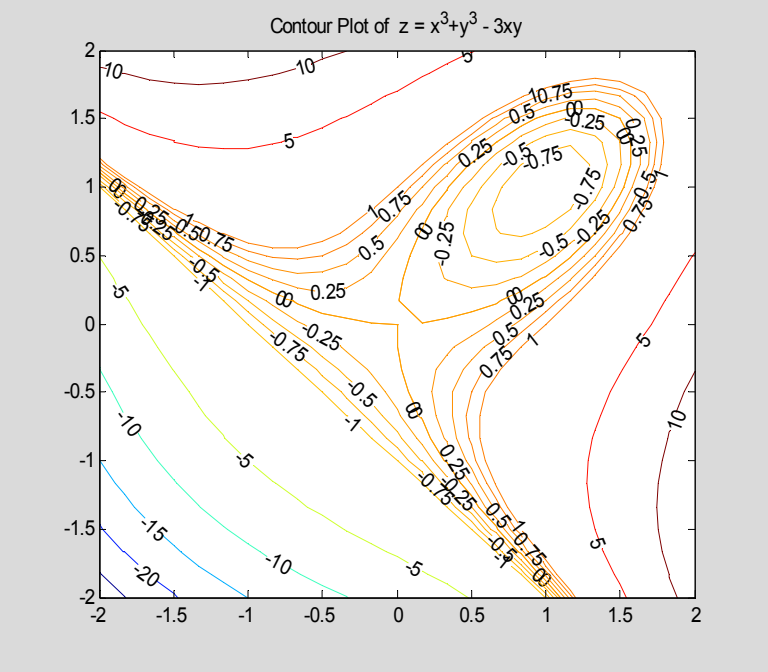
Programatically, we can get Matlab to add specific contour lines to the picture. We do this by creating a list of *z* values for which we want to see the level curves and pass this list to the `contour` command. We also pass the list to `clabel` so that the curves are labeled as well. In this case we would like to see level curves corresponding to values of *z* between -1 and +1.

Example 5.3: Add specific level curves to the contour plot generated in Example 5.2.

Solution:

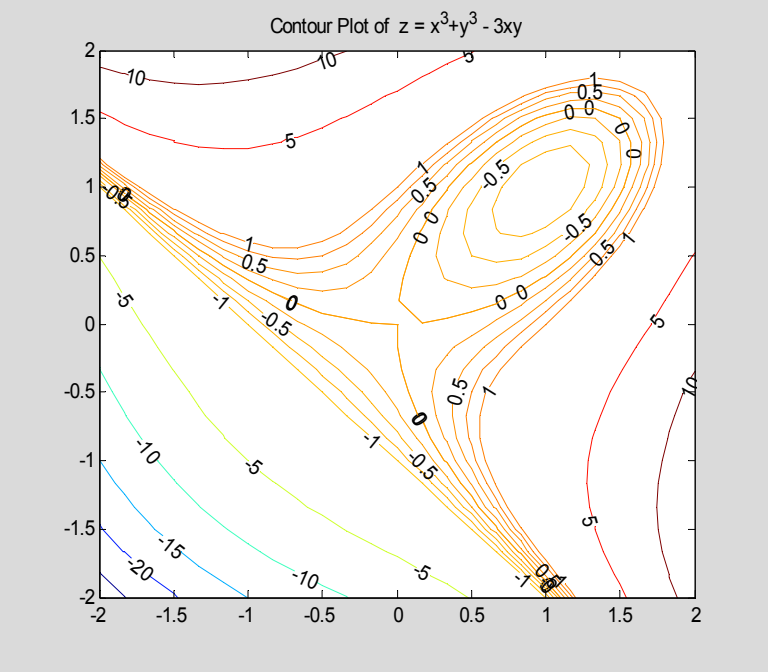
We assume the previous plot has been generated and is currently displayed. We then execute the following additional code:

```
vals=-1:.25:1; % level values to plot.  Recall the : operator described
% in the exercises of Chapter 3.
hold on
[C h]=contour(x,y,z,vals);
clabel(C,h,vals);  % add labels to the new curves
hold off
```



We seem to have generated too many labels. The picture appears cluttered. In this case we can either pass a shorter list to the `clabel` command, or we can use a neat interactive option to select the labels we care to see. The latter is done via the command `clabel(C,h,'manual')`. The help page for `clabel` contains a pretty good description of how this option works. As with all things interactive, it is most easily discussed by actually using it, which you will do in class. (N.B. This feature also has graphics driver issues, so may not work on all systems.) Here, we'll redo the plot with a truncated list of levels to be labeled.

```
vals=-1:.25:1;
hold on
[C h]=contour(x,y,z,vals);
clabel(C,h,-1:.5:1);  % fewer labels in this list
hold off
```



This is a bit less muddy. ■

Look at the picture. Do you see a point where the surface $z = x^3 + y^3 - 3xy$ might have a a maximum or minimum? What about a saddle point, where the function increases in some directions and decreases in others? By examining z values with the Data Cursor see how well you can approximate the location of these points. Later in the course, you will use methods of calculus to determine these points analytically. As we will see, such methods are often difficult to carry out, (though not in this case) so the simple technique used here in visually identifying the possible location of such points is very helpful.

There is one other strange phenomenon exhibited above. One of the level curves appears to be a straight line (which one?) You can find the equation of this line by examining the graph. Can you show algebraically that all points on this line produce the same value of z and therefore define a level curve? Try it.

5.3 Contour and Surface Plots

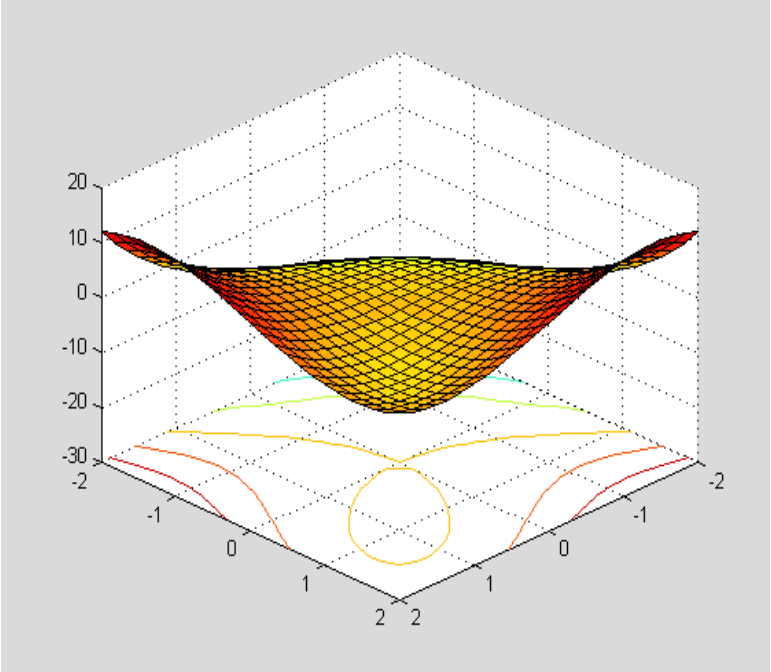
The graph of the surface can be visualized along with the contour curves using the command `surfc`. Here is the original contour set shown lying beneath the graph of the surface.

Example 5.4: Draw a combined contour and surface plot for the function $z = x^3 + y^3 - 3xy$.

Solution:

Assuming the data values x, y, and z have been obtained as above we use

```
surfc(x,y,z); view([1 1 1])
```



As usual, rotating the figure gives a clearer sense of the relationship between the level (contour) curves and the surface. In the exercises we will explore how the contour curves can be drawn on the surface. ■

5.4 Implicit Plots

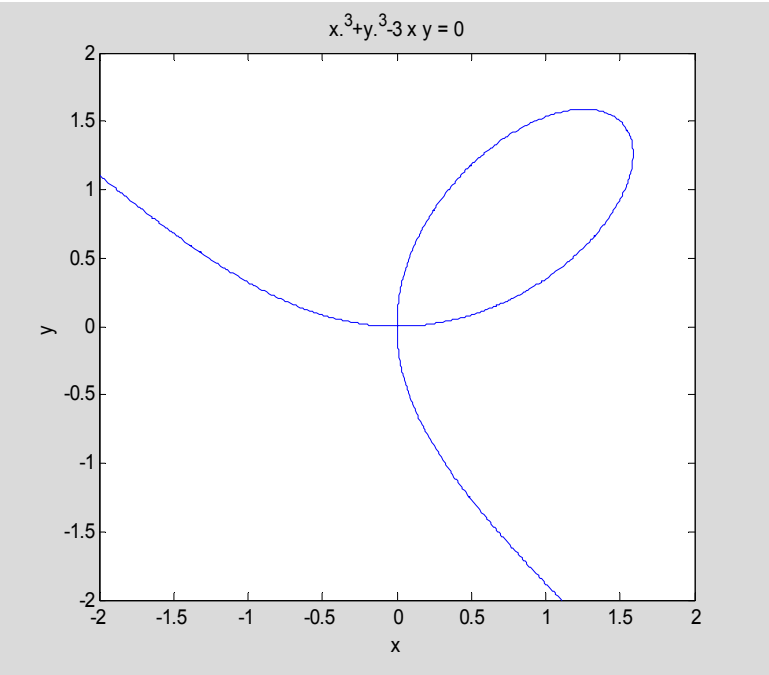
The level curves shown in a countour plot of $z = f(x, y)$ are plane curves given by setting z equal to a constant c. In other words, they describe the location of points where $f(x, y) = c$. In the example considered above these are the curves $x^3 + y^3 - 3xy = c$. For most such curves, we cannot solve for x or y as functions of the other variable and we say the curves are defined by implicit equations (remember implicit differentiation). The contour plot can be viewed as a collection of such implicit curves for different values of c. Sometimes, however, we would like a picture of simply one or two such curves and the mechanism of generating a contour plot is not the right way to achieve this. In fact, sometimes Matlab errs in the shape of curves it forms in the contour plots because of the limited information with which it is working.

Suppose for example that we want to plot the single curve $x^3 + y^3 - 3xy = 0$, which is one of the curves in the contour plot generated earlier. We can actually get Matlab to plot this using our old friend `ezplot`. When we used `ezplot` before, we passed symbolic expressions to it. However, since we are working here with numeric variables, we use an alternate method – passing the expression as a string.

Example 5.5: Draw a plot of the implicitly defined curve $x^3 + y^3 - 3xy = 0$.

Solution:

```
ezplot('x.^3+y.^3-3*x.*y', [-2 2])
```



The string should be the vectorized form of the expression $f(x, y)$. Matlab assumes you want to plot $f(x, y) = 0$. If you want $f(x, y) = c$, just move the constant to the left side and include it in the string. The domain is indicated by the entry [-2 2]. Matlab assumes that both x and y will vary over the same bounds. In other words, you are limited to sketching over a square region. You should compare the picture above with the zero level curve in the contour plot. You will see that Matlab had trouble in the latter plot in analyzing the proper shape of the curve near the origin. ■

5.5 Summary

We introduced three new commands, but each has a number of options with which you need to be acquainted to make effective use of them. Of course, everything we've said, and more, can be found in Help.

`contour`: - draw a 2D contour plot of a function of two variables

`clabel`: - generate labels for a 2D contour plot – various options for specifying the labeled curves.

`surfc`: - generate a surface plot and a contour plot lying below it.

`ezplot`: - generate a plot of an implicit curve $f(x, y) = c$

5.6 Exercises

Exercise 5.1. Let $f(x,y) = 2x^2 + xy + y^4$. Write an M-file that uses `subplot` to generate a 1 x 2 array of plots with the following specifications.

a) The left plot should use `surfc` to generate a plot of the graph $z = f(x,y)$ and contours over the region $-1 \le x \le 1$ and $-1 \le y \le 1$. Use a 25 x 25 grid. Display **grid** lines, **labels** on all axes, and a **title** that includes the formula for the function. (Use a multi-line title if necessary.)

b) The right plot should

i) display the default contour plot of $f(x,y)$ over the square [-1,1]x[-1,1] using a 25 x 25 grid, with labels on all contour lines.

ii) add five additional contours at levels between 0 and 0.5.

iii) label the additional contours using `clabel`.

iv) include **labels** on the axes and a **title**

c) Does the function have any maximum or minimum values in the domain $[-1,1]\times[-1,1]$? If so use either the Data Cursor or the coordinate grid to estimate the x and y values at the maximum/minimum points. Specify these estimates in a comment line in your M-file.

Publish your M-file (include your name in a comment). The total should be at most two pages.

Exercise 5.2 Suppose $f(x,y) = \sin(3y - x^2 + 1) + \cos(2y^2 - 2x)$.

a) Write an M-file that will produce a labeled contour plot for $f(x,y)$ over the region $S = \{(x,y) \mid -2 \le x \le 2, -1 \le y \le 1\}$.

b) Based on the contour plot you found in a) estimate the coordinates of two saddle points of the function in the region S defined in a). Mark the points using the Data Cursor.

Exercise 5.3 Suppose $f(x,y) = \sin(3x + y) - 2\cos(x - y)$.

a) Write an M-file that will produce 15 **labeled** contour curves for $f(x,y)$ over the square $S = \{(x,y) \mid 0 \le x \le 2, 0 \le y \le 2\}$.

b) Based on the contour plot you found in a) determine whether the function has any critical points in the square S defined in a). If there are any such points, provide **estimates** from the graph for their x and y coordinates and provide a **justification** from the graph as to whether these are **relative maxima**, **minima** or **saddle points**. Indicate your reasons as comments on the M-file and publish the contour plot and the M-file.

Exercise 5.4 Write an M-file that uses `ezplot` to display the graph of $2x^2 + xy + y^4 = 1$ and the circle $x^2 + y^2 = 1$, on the same figure. Your graph should show

a) A graphing window where x and y vary from -1.2 to $+1.2$.

b) Display of grid lines

c) The circle displayed in a dotted line style.

d) A legend displaying the equation of each curve and a title with your name

e) The curves meet at the points $(0, \pm 1)$ and at two other points. Using the Data Cursor, (if available) display labels at the approximate locations of these two other points. (To display multiple data points, right click the data marker box and choose the "Create New Datatip" option.) Otherwise, estimate the coordinates of these additional intersection points using the grid lines. Write your estimate as a comment in your M-file.

Publish your M-file. Be sure to include your name.

Exercise 5.5 Run the M-file below (you can download the file as `contourplot.m` from the course site). What does the file do? Explain what is done by each of the shaded lines.

```
% 3D Contour plotting
a=-2;b=2;
c=-2;d=2;
x=linspace(a,b,25);
y=linspace(c,d,25);
[x y] =meshgrid(x,y);
z=-x.*y.*exp(-x.^2-y.^2);
clf;
hold on
[C h]=contour3(x,y,z,'k');
set(h,'LineWidth',1.5);
surf(x,y,z);
view([1 1 1]);
grid on
shading interp
xlabel('x');ylabel('y');zlabel('z');
hold off
```

6: Symbolic and Numerical Calculations

We have touched on symbolic objects earlier in our discussion of the ez-graphing techniques, which serve as a parallel plotting world for symbolic users.

Now we want to consider using Matlab to do symbolic calculations. We will also consider how we do numerical calculations with symbolic expressions. For example, how do we compute a derivative symbolically and then find numerical values of the derivative? We begin first with a discussion of Matlab functions.

6.1 Matlab Functions

From the point of view of computing, a function is a structure *f* that accepts one or more inputs and produces one or more outputs. Matlab has two mechanisms for defining functions – the M-file function and the anonymous function. An M-file function, as the name suggests, is a function that is created as a special type of M-file. The built-in Matlab functions are M-functions. Such functions are saved by the system and can be accessed at a later date. Typically, one uses an M-function when constructing a function requires multiple lines of code. For example, in addition to carrying out its principal calculation, one might want the function to check that the user has entered a legal input, and if not, issue an error message and not proceed. In this course, we will not be interested in creating such carefully crafted functions so we will have little need to write M-file functions. Instead, we will focus on the more useful **anonymous function** construct.

6.1.1 Anonymous Functions

Although, as the name suggests, an anonymous function may indeed be used without giving it a name, more commonly we will attach names to these functions. Here is a very simple illustration of the construction.

Example 6.1: Write a Matlab anonymous function to define $f(x) = x^2$. The function should be vectorized so it accepts vector inputs and computes the result for each component of the vector.

Solution:

```
f=@(x)x.^2
f =
    @(x)x.^2
```

The symbol @ initiates the definition. This is followed by a list of the input variables, in parentheses, separated by commas if there is more than one variable. Finally, we state the formula defining the function. If we wish to have the function act component-wise on arrays, the dot operator should be included in this formula. The function has been named

f. Note that it is not necessary to declare *x* as symbolic in order to use it as a placeholder for the variable within an anonymous function ■

Such a function can be used in much the same way you use functions in ordinary mathematics, with the added feature of being vectorized. Thus, if

```
x=[1 2 3]
x =
     1     2     3
```

then entering

```
f(x)
```

produces the square of each input value.

```
ans =
     1     4     9
```

However, if *x* has not been declared to be symbolic then a request such as

```
f(x)
??? Undefined function or variable 'x'.
```

produces an error message.

We can define functions of more than one variable in a similar fashion.

Example 6.2: Define an anonymous function $g(x,y)=x^2+y^2$. Vectorize the function to accept input arrays.

Solution:

```
g=@(x,y)x.^2+y.^2
g =
    @(x,y)x.^2+y.^2
```

Then `g(1,2)` is

```
ans =
     5
```

and if `x=[1 2 -1]` and `y=[3 2 1]` then

```
x =
     1     2    -1
y =
     3     2     1
```

and `g(x,y)`

```
ans =
    10     8     2
```

consists of the values $g(1,3)$, $g(2,2)$ and $g(-1,1)$. ■

6.2 Symbolic Functions

So far, when using anonmymous functions we did not require that the function variables represent symbolic quantities. When we add that option we obtain symbolic functions, which closely resemble ordinary mathematical functions. Let us first declare a pair of symbolic variables.

```
syms x y  .
```

We can now create symbolic functions of these variables.

Example 6.3: Define the function $h(x,y) = x^2 + y^2$ as a symbolic, vectorized function of x and y.

Solution:

```
h=@(x,y)x.^2+y.^2
h =
    @(x,y)x.^2+y.^2
```

Then not only can we compute `h(2,3)`

```
ans =
    13
```

but we can also ask for symbolic quantities, such as `h(x,x^2)`

```
ans =
x^2+x^4
```

An important point to observe, however, is that we must never assign any numeric value to the symbolic variables themselves. Doing so will destroy their symbolic character. Thus, if we want to compute the value of h at each pair (x, y) from the arrays [1, 2, -1] and [3, 2, 1] we can do so by assigning suitable upper-case letters as names for the input arrays. Remember, Matlab is case-sensitive, so that x and X represent different quantities.

`X=[1,2,-1]` and `Y=[3,2,1]`

```
X =
     1     2    -1
Y =
     3     2     1
```

Then `h(X,Y)`

yields the desired result.

```
ans =
    10     8     2
```

■

6.2.1 Calculus: - Differentiation

One purpose in using the Symbolic Toolbox is to have the system perform complicated manipulations using its built-in symbolic processing. A particularly useful feature is symbolic differentiation. Later we will also take up integration. The `diff` command, which we have briefly encountered in Chapter 1, computes derivatives of a symbolic expression.

Example 6.4: Find the derivative of $y = x\sin(2x)$.

Solution: Assuming that x has been declared symbolic we proceed as follows:

```
y=x.*sin(2*x); diff(y)

ans =

sin(2*x)+2*x*cos(2*x)
```

Notice that although we started with a vectorized expression, the formula that is returned for the derivative is not vectorized. We can get Matlab to supply the appropriate dot operators using the `vectorize` command.

```
vectorize(ans)

ans =

sin(2.*x)+2.*x.*cos(2.*x)
```

As in this case, the `vectorize` command usually supplies more dot operators than are actually needed. ■

We can compute higher order derivatives by including an extra parameter in the `diff` function.

Example 6.5: Compute the 3rd derivative of $y = x\sin(2x)$.

Solution: Add the order of differentiation as a second argument in the `diff` function.

```
diff(y,3)
```

gives the 3rd derivative.

```
ans =

-12*sin(2*x)-8*x*cos(2*x)
```

■

Notice that the input to `diff` is an expression, and the command returns an expression as output. Sometimes it is useful to have a derivative function. This is particulary so when we want to make numerical evaluations of the derivative. We can get Matlab to convert the expression for the first derivative generated above, $\sin(2x) + 2x\cos(2x)$, into a function, as follows:

Example 6.6: Construct a derivative function from the symbolic derivative formula.

Solution:

Suppose we start with the symbolic function $f(x) = x\sin(2x)$. We want to construct the function $f'(x)$. Moreover, both functions should be vectorized. Here's how to do it.

First let's define $f(x)$.

```
f=@(x)x.*sin(2*x)
```

```
f =
    @(x)x.*sin(2*x)x
```

Then we construct the derivative expression via

```
dy=diff(f(x))
dy =
sin(2*x)+2*x*cos(2*x)
```

Finally, we can create the appropriate derivative function using the `subs` command.

```
df=@(x)subs(dy)
df =
    @(x)subs(dy)
```

This is a perfectly general method for converting an expression defined through some other procedure, in this case as a result of applying the `diff` function, into an anonymous function. [1]

When we evaluate the function *df*, the `subs` command passes any input values to the expression *dy*. If we tried to do this construction without the `subs` command, it would not produce the desired result. (Try it!) In our case, everything works fine. For example,

```
df(x)
ans =
sin(2*x)+2*x*cos(2*x)
```

returns the expression for the derivative, *dy*. Although it does not appear from the latter formula that the derivative function *df* is vectorized, in fact it is.

```
df([pi,pi/2])
ans =
    6.2832   -3.1416
```

returns the numerical value of the derivative at π and $\pi/2$. ■

Everything we have done for functions of a single variable works equally well for functions of several variables. The only extra bookkeeping is that the `diff` command requires that we specify the variable of differentiation.

Example 6.7: Find $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, and $\frac{\partial^4 f}{\partial y^2 \partial x^2}$ for the function $f(x,y) = \sin(x^2 + y^2)$.

Solution: Define a symbolic function *f* of two variables.

```
f=@(x,y)sin(x.^2+y.^2);
```
Then we have

```
zx=diff(f(x,y),x)
```

[1] I am grateful to Prof. Peter Brinkmann for this construction.

```
zx =
2*cos(x^2+y^2)*x
```

for z_x and

```
zy=diff(f(x,y),y)
zy =
2*cos(x^2+y^2)*y
```

for z_y.

If we wish to generate (vectorized) partial derivative functions we can do so through

```
fx=@(x,y)subs(zx);
```

and `fy=@(x,y)subs(zy);` .

Then, for example

using the arrays `X=[1, 2 ,-1]` and `Y=[3, 2, 1]` , we get

```
X =
     1     2    -1
Y =
     3     2     1
```

and we have `fx(X,Y)`

```
ans =
   -1.6781   -0.5820    0.8323
```

The required mixed partial cannot be computed in a single application of the `diff` function. The latter can only compute derivatives with respect to one variable at a time. To compute mixed partials, we use the theorem that says the order of differentiation is not important. Thus, to find $\frac{\partial^4 f}{\partial y^2 \partial x^2}$, we first find $z_{xx} = \frac{\partial^2 f}{\partial x^2}$ and then take the second partial of this result with respect to *y*.

```
zx2=diff(f(x,y),x,2);
```

This is the partial z_{xx}.

Then take the second partial of this with respect to *y*.

```
zx2y2=diff(zx2,y,2)
zx2y2 =
16*sin(x^2+y^2)*y^2*x^2-8*cos(x^2+y^2)*x^2-8*cos(x^2+y^2)*y^2-
4*sin(x^2+y^2)
```

■

The `pretty` command can be used to write this last expression in a somewhat more legible form, though if mathematically typeset output is desired one must use the `latex` command to convert the expression to the mathematical markup language latex. The latter result can then be printed through a program that generates mathematical expressions from such input. Unfortunately, it is not (yet) possible to include computer generated latex output in a published version of an M-file, (you can add explicit latex code), but undoubtedly this will be an added feature in a future release.

```
pretty(zx2y2)
```

```
          2    2   2  2          2    2   2          2    2   2
  16 sin(x  + y ) y  x  - 8 cos(x  + y ) x  - 8 cos(x  + y ) y

                  2    2
        - 4 sin(x  + y )
```

6.3 Tangent Planes

The following M-file uses some of the ideas discussed above to draw the graph of a function and its tangent plane at a specified point. We highlight the lines that use material discussed in this section. The M-file is available as *tanplane.m* on the course website. One may have to rotate the figure to get a view that shows the tangency to the best effect.

```
% Surface and tangent plane plot
% User Input
a=0; b=1.5; % x interval for plotting
c=0; d=1.5;  % y interval for plotting
% grid size parameters
m=15;
n=15;
% x, y coordinates of point of tangency
X0=1;
Y0=1;
%function to plot
syms x y
f=@(x,y)sin(x.^2+y.^2);  % anonymous function
% End user input
% Begin Matlab computations
% Construct 1st derivative functions
zx=diff(f(x,y),x);
zy=diff(f(x,y),y);
fx=@(x,y)subs(zx);
fy=@(x,y)subs(zy);
% Plot Surface
clf
hold on
X=linspace(a,b,m); % use X, Y, Z for numeric values
Y=linspace(c,d,n);
[X Y] = meshgrid(X,Y);
Z=f(X,Y);
surf(X,Y,Z); view([1 1 1]);
% Construct symbolic function for z value on tangent plane
Z0=f(X0,Y0);
A=fx(X0,Y0);
B=fy(X0,Y0);
z=@(x,y)Z0+A*(x-X0)+B*(y-Y0);
% Now we plot the tangent plane, but use only the four
% boundary points of the plotting region to get an
%uncluttered picture
X=linspace(a,b,2);
Y=linspace(c,d,2);
```

```
[X Y] = meshgrid(X,Y);
Z=z(X,Y);
surf(X,Y,Z,'FaceColor','cyan') % adjust the color of the
%patch
% The next line marks the point of tangency
plot3(X0,Y0,Z0, 'or','MarkerSize',2,'MarkerFaceColor','r')
grid on
xlabel('x');ylabel('y');zlabel('z');hold off
```

6.4 Summary

We summarize the commands used in this section:

@: - symbol to initiate an **anonymous** function

diff: - compute a symbolic derivative or partial derivative (note that `diff` can also apply to arrays and produces the so-called first difference array. This is used in computing numerical approximations for derivatives.)

pretty: - display a symbolic expression in a style that is closer to standard math output.

subs: - substitutes input values for variables in symbolic expressions. Useful for converting symbolic expressions to (vectorized) anonymous functions.

vectorize: - inserts dot operands into an expression

6.5 Exercises

Exercise 6.1 Write an M-file in which you define each of the following functions and have Matlab compute the indicated derivative, and the value of **the derivative** for the specified inputs. Publish the M-file, making sure that you divide the published document into cells, each of which produces the answer for a single part. Do not forget to include your name at the top of the file.

a) $f(x)=x^7-3x^5+5x^4+3$: $f'''(x)$: evaluate at $x=1:5$.

b) $f(x)=e^{x\sin x}+2\ln(x^2+1)$: $f''(x)$: evaluate at $x=-pi:pi/4:pi$ (Note: use exp to enter the exponential function and log for the natural logarithm.)

c) $f(x,y)=\dfrac{x^2+y^2}{(x+y)^2}$: $f_{xy}(x,y)$: evaluate at all pairs $(1,1),(2,2),(3,3),(4,4),(5,5)$

d) $g(x,y,z)=x^3y^2z+\sqrt{x^2+y^2+z^2}$: $g_{xyz}(x,y,z)$: evaluate at the eight vertices of the cube with coordinates at $x=\pm1$, $y=\pm1$, $z=\pm1$

Exercise 6.2 Write an M-file in which you define the function $f(x)=\sin(x^2)\cos x$ and have Matlab compute the first two derivatives as functions *df* and *df2*. Using `plot`,

create a graph of the function *f* and the first two derivatives over the interval $[-\pi, \pi]$. Use different line styles for each plot and a label to distinguish which is which. You may want to truncate the y-axis display to the interval $[-10,10]$. This can be done using the command `ylim`. Publish your result, after dividing the M-file into appropriate cells so that the output is closely related to the input statements.

Exercise 6.3 Using the tangent plane M-file above as a model, write an M-file that plots the graph of the function $f(x) = e^{-x}\cos(2x)$ over the interval $[0, \pi]$ and draws the tangent line to the graph at the point with *x* coordinate $\pi/4$. Matlab should be used to compute the necessary formulas and values for derivatives.

7: Parametric Surfaces

So far we have examined surfaces defined as the graph of an equation $z = f(x, y)$. However, there are many interesting surfaces that are not graphs of functions. For example, the sphere given as the set of points satisfying $x^2 + y^2 + z^2 = 1$. This surface can be pieced together from its upper and lower hemispheres defined by $z = \sqrt{1 - x^2 - y^2}$ and $z = -\sqrt{1 - x^2 - y^2}$, respectively. For the purposes of graphing, however, this produces a poor picture because the domain of these functions (the inside of the unit circle) is not so easily approximated by a rectangular grid.

We can generalize the method of representing a surface by introducing the idea of a parametic representation. This is analogous to the parametric description of curves. Namely, we say that a surface is given parametrically by the parameters *s* and *t*, if there are functions *f*, *g*, and *h* of the variables *s* and *t* such that any point (x, y, z) on the surface satisfies

$$x = f(s,t) \quad y = g(s,t) \quad z = h(s,t).$$

Here *s* and *t* vary over some region in the *st*-plane, usually a rectangle. We can use this representation to generate points (x, y, z) on the surface, from which Matlab can construct a 3D plot. Let's consider some examples.

7.1 Basic Examples

Example 7.1: Plot the **circular cylinder** $x^2 + y^2 = 1$.

Solution: We use the θ and *z* variables of a cylindrical coordinate system for the parametric representation. Namely, on the surface of the cylinder we can take

$$x = \cos\theta \quad y = \sin\theta \quad z = z$$

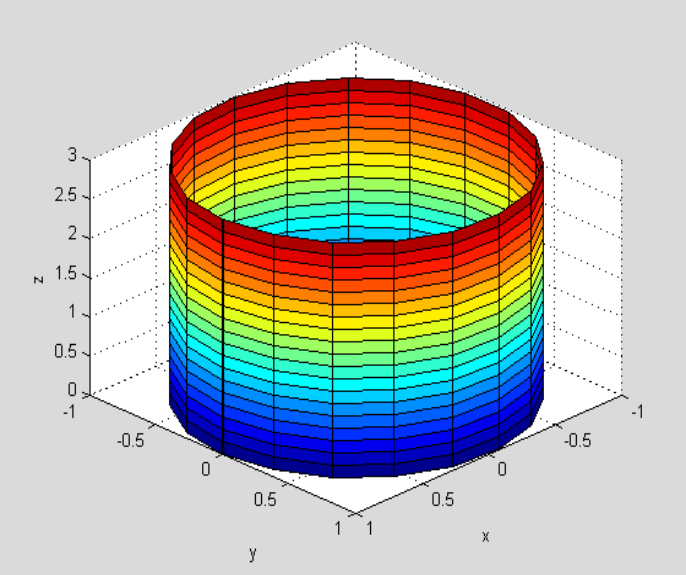
where $0 \le \theta \le 2\pi$ and $0 \le z \le 3$, (for example). To plot, we set up a meshgrid in the θ, z variables and determine the *x*, *y*, and *z* values from the above equations. Let's see how it is done. For simplicity, we will use *t* instead of θ,

```
t=linspace(0,2*pi,20);
z=linspace(0,3,20);
[t,z]=meshgrid(t,z);
```

Now we compute the values of *x*, *y*, and *z* corresponding to the values of *t* and *z* in the grid. Since $z = z$, we do not have to do any additional computation to compute these values. Once we have the data values for the points on the surface, we are in business and can use the `surf` command as usual.

```
x=cos(t); y=sin(t);
surf(x,y,z)
grid on; view([1 1 1]);
```

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



■

In generic terms the curves we see on a parametric surface plot are the images on the surface of the grid lines in the *s*, *t*-plane where *s* or *t* is constant. For example, in the plot above, the circles are curves on which *z* is constant, while the vertical lines are curves on which θ (i.e. *t* in our notation) is constant.

We can carry out similar constructions for cylinders parallel to the other axes. For example, to plot the cylinder $x^2 + z^2 = 1$, use the parametric equations $x = \cos t$, $z = \sin t$ and $y = y$, with $0 \le t \le 2\pi$ and $a \le y \le b$, for some interval $[a,b]$.

Plotting cylinders evidently depends on being able to parametrize the trace curve that defines the cylindrical cross-section. Here are some additional commonly occurring examples:

Parabolic cylinder: $y = x^2$: Use *x* and *z* as parameters. The equations are $x = x$, $y = x^2$, $z = z$, with $a \le x \le b$ and $c \le z \le d$. We have actually used this construction earlier in Exercise 5.5

Elliptic Cylinder: $\frac{x^2}{9} + \frac{y^2}{16} = 1$. Here we can let $x = 3\cos t$ and $y = 4\sin t$, while $z = z$ is the other parameter. The range of the variables is $0 \le t \le 2\pi$ and $c \le z \le d$.

Example 7.2: **Surface of Revolution**. Describe parametric equations for the surface of revolution obtained by revolving a curve in the *x*, *z* plane around the *z*-axis. Apply the method to plotting a cone.

Solution: The cylinder in Example 7.1 is obtained by rotating around the *z*-axis the straight line $x = 1$ in the *xz*-plane. More generally, we can rotate around the *z*-axis any curve with equation $x = f(z)$ in the *xz*-plane. The resulting surface can be described

using the angle of rotation t and the z coordinate as parameters. In fact, the general equations are

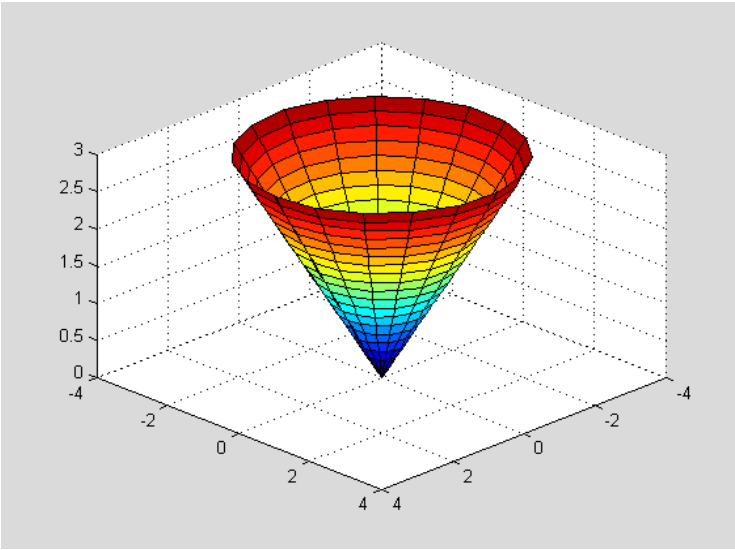
$$x = f(z)\cos t \quad y = f(z)\sin t \quad z = z\,.$$

For example, if we take the line $x = z$ and rotate it around the z-axis we obtain a cone. This can be described by the parametric equations

$$x = z\cos t \quad y = z\sin t \quad z = z\,.$$

This description is also clear from the rectangular equation of the cone, namely $z^2 = x^2 + y^2$. A Matlab plot using a meshgrid in the t, z-plane gives:

```
t=linspace(0,2*pi,20);z=linspace(0,3,20);
[t z]=meshgrid(t,z);
x=z.*cos(t);y=z.*sin(t);
surf(x,y,z); grid on; view([1 1 1]);
```



■

Example 7.3: Parametrize a **sphere** and use the parametrization to plot the sphere.

Solution: The parametrization can be obtained using spherical coordinates. For example, to plot $x^2 + y^2 + z^2 = 4$ use the equations

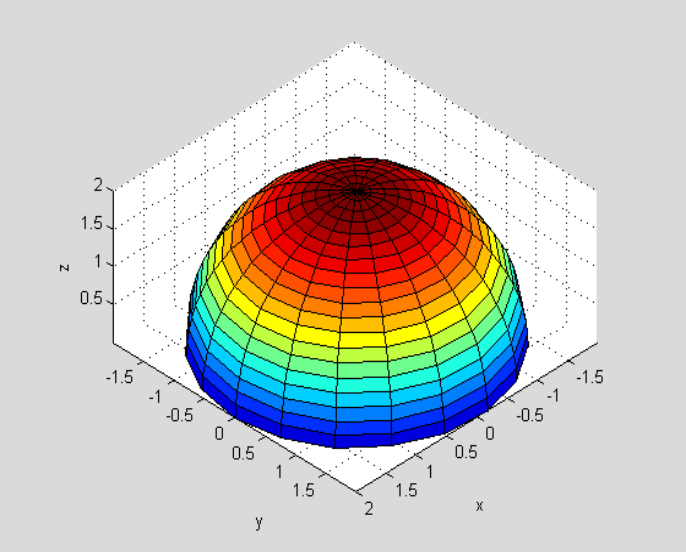
$$x = 2\cos t\sin s \quad y = 2\sin t\sin s \quad z = 2\cos s$$

For simplicity in typing, we have identified t with θ and s with ϕ in the usual notation. Thus, for the full sphere we would let $0 \le t \le 2\pi$ and $0 \le s \le \pi$. To plot the upper hemisphere we would restrict s to the interval $0 \le s \le \pi/2$. The Matlab plot is:

```
t=linspace(0,2*pi,20);s=linspace(0,pi/2,20);
[s t]=meshgrid(s,t);
```

```
x=2*cos(t).*sin(s);  y=2*sin(t).*sin(s);  z=2*cos(s);
surf(x,y,z); view([1 1 1]); grid on;
xlabel('x');ylabel('y');zlabel('z');
axis equal
```

The last command simply draws the sphere without distorting distances, so the appearance is actually spherical, instead of egg-shaped.



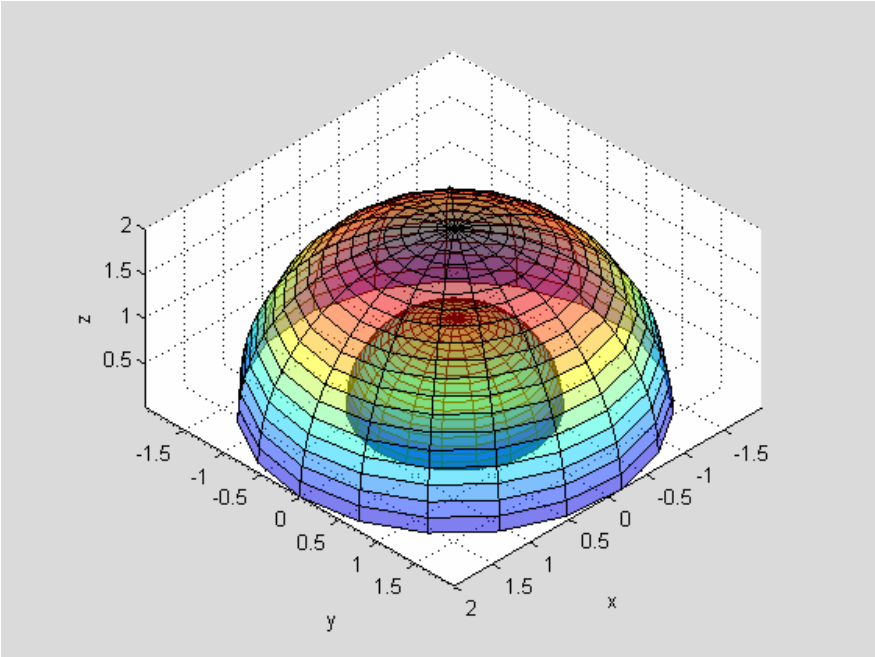
■

Example 7.4: Draw a plot of two concentric spheres with the outer sphere rendered as semi-transparent.

Solution: Using `hold on` we can easily create plots showing multiple surfaces. To assist our visualization it is sometimes useful to adjust the transparency of one of the surfaces. This is done with the property ***FaceAlpha***. This takes a numeric value between 0 and 1, with a value of 1 signifying an opaque surface (the default) and a value of 0 denoting a completely clear surface. Observe the effect of using this command in the following figure, which shows the hemisphere of radius one inside the hemisphere of radius two. The latter has been drawn as a semi-transparent surface.

```
% Hemisphere of radius two
clf; hold on;
t=linspace(0,2*pi,20);s=linspace(0,pi/2,20);
[s t]=meshgrid(s,t);
x=2*cos(t).*sin(s);  y=2*sin(t).*sin(s);  z=2*cos(s);
surf(x,y,z, 'FaceAlpha',.5);
view([1 1 1]); grid on;
xlabel('x');ylabel('y');zlabel('z');
% Hemisphere of radius one
t=linspace(0,2*pi,20);s=linspace(0,pi/2,20);
```

```
[s t]=meshgrid(s,t);
x=cos(t).*sin(s);  y=sin(t).*sin(s);  z=cos(s);
surf(x,y,z);
axis equal;
hold off
```



■

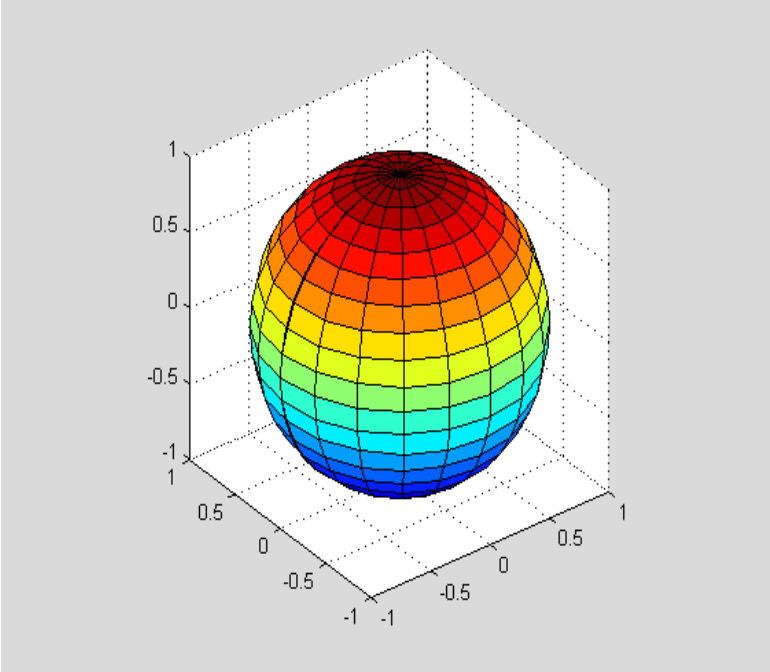
7.2 Built-in Parametric Plotting

By way of completeness, we mention that Matlab has commands for doing some of the parametric plots described above.

Example 7.5: Use the `sphere` command to draw spheres of radius one centered at the origin and at the point (1,0,0).

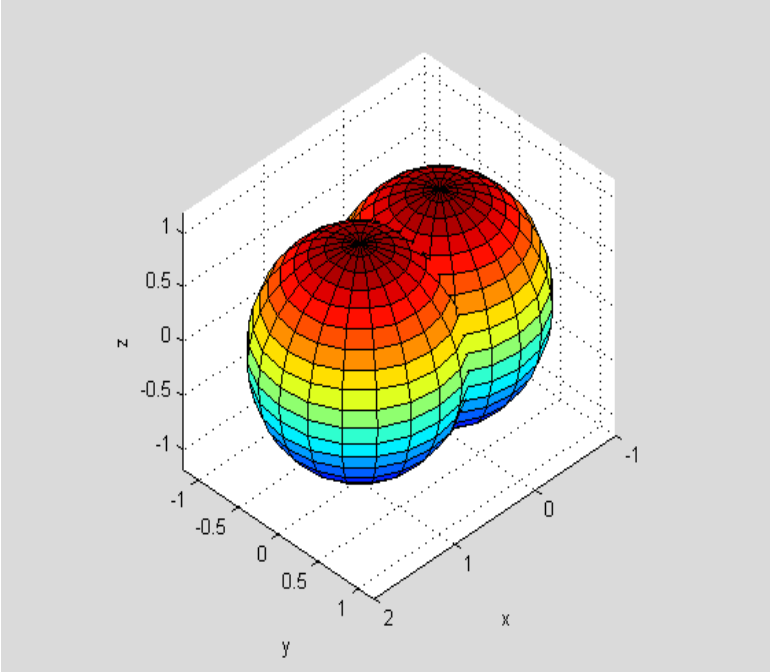
Solution: The command `sphere` generates a plot of a sphere of radius one centered at the origin. Thus

```
sphere; axis equal
```



The second sphere, with its center shifted along the x-axis, can be obtained from the first by translating all points on the latter sphere by the vector $\mathbf{v} = \langle 1 \quad 0 \quad 0 \rangle$. In order to do this, we first need to generate those data points. We can obtain the points by assigning variables $[x, y, z]$ as the output of the `sphere` command. Then we add 1 to the x coordinates of these points, leaving the others fixed. The following code will implement this and draw the new graph, together with the sphere generated above.

```
clf; hold on;
[x y z]=sphere;
surf(x,y,z)   % plots sphere centered at the origin
x=x+1;  % translates the x-coordinates
surf(x,y,z);  % plots translated sphere
axis equal; hold off;
grid on; view([1 1 1]);
xlabel('x');ylabel('y');zlabel('z');
```



■

7.3 Summary

We showed how to construct surface plots based on the parametric representation of a surface. Matlab contains some built-in parametric plotters, of which we discussed in detail the `sphere` command. Other parametric plotters are the `ellipsoid` command and the `cylinder` command. The latter can be used to construct surfaces of revolution whose rotation axis is the *z* axis. Geometric transformations are needed if you wish to use this command to plot cylindrical surfaces with other axes of rotation.

7.4 Exercises

All exercises should be done by writing appropriate M-files and publishing the results.

Exercise 7.1 Plot the portion of the parabolic cylinder $z = 4 - x^2$ that lies in the first octant with $0 \le y \le 4$. (Use parameters *x* and *y*. Determine the range of values for *x* to only plot points in the first octant.)

Exercise 7.2 Rotate the curve $x = 1 + \cos z$, for $0 \le z \le 2\pi$, around the *z*-axis. Use Matlab to draw the surface of revolution. Use `axis equal` if necessary for a non-distorted view.

Exercise 7.3 Draw the portion of the cone $y^2 = x^2 + z^2$ for $0 \le y \le 3$. (This is obtained by rotating the line $z = y$ around the *y*-axis. Use the angle of rotation *t* and the *y* variable as parameters.)

Exercise 7.4 Draw the ellipsoid whose equation is $\frac{x^2}{4} + \frac{y^2}{16} + \frac{z^2}{2} = 1$ in two ways.

a) Using explicit parametric equations base on spherical coordinates. (Observe: The quantities $x' = x/2$, $y' = y/4$, and $z' = z/\sqrt{2}$ satisfy the equation $(x')^2 + (y')^2 + (z')^2 = 1$. Use spherical coordinates for $x', y',$ and z' to parametrize the latter surface.) Use `axis equal` to obtain a non-distorted view.

b) Using the built-in function `ellipsoid`.

Exercise 7.5 The circle of radius one with center $(2, 0, 0)$ in the x, z-plane has parametric equations $x = 2 + \cos t$, $y = 0$, $z = \sin t$, where $0 \le t \le 2\pi$. If we revolve that circle around the z-axis we obtain a donut-shaped figure called a **torus**. Letting s denote the angle of rotation about the z-axis show that the parametric equations of this surface are

$$x = (2 + \cos t)\cos s \quad y = (2 + \cos t)\sin s \quad z = \sin t,$$

where $0 \le t \le 2\pi$ and $0 \le s \le 2\pi$. Using Matlab, draw the surface. (N.B. Use `axis equal` to obtain a non-distorted view.)

Exercise 7.6 Create a single plot showing the cylinders $x^2 + z^2 = 1$ and $y^2 + z^2 = 1$ within the box where $-2 \le x \le 2$, $-2 \le y \le 2$, and $-2 \le z \le 2$. Later we will compute the volume enclosed by the two cylinders. Make one cylinder semi-transparent using the `FaceAlpha` property.

Exercise 7.7 Create a single plot showing the portion of the parabolic cylinder $z = 4 - x^2$ that lies in the first octant with $0 \le y \le 3$, (see Exercise 7.1) and the first octant portion of the vertical plane $y = x$, with $0 \le z \le 4$. Draw the plane with a single patch using a 'FaceColor' of 'cyan'. Make the cylinder semi-transparent so you can see the vertical plane inside the cylinder.

Exercise 7.8 Create a single plot that shows the sphere $x^2 + y^2 + z^2 = 4$ and the cylinder $(x-1)^2 + y^2 = 1$. Use `axis equal` to obtain a non-distorted view. Make the sphere semi-transparent so you can see the cylinder inside of it.

8: Field Plots

8.1 Vector Field Plots

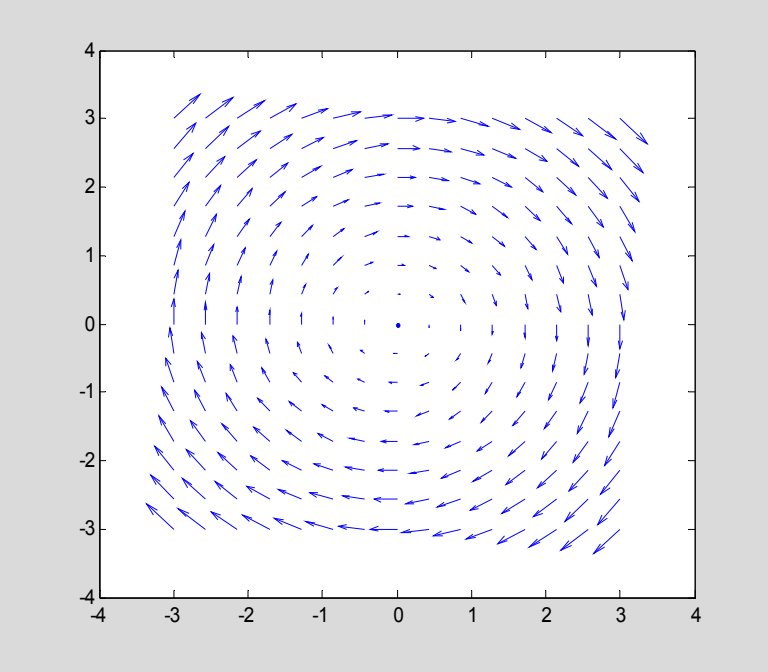
A function $f(x,y)$ of two variables assigns a scalar to each pair (x,y). Sometimes, we need to consider a function that assigns a vector to each point (x,y). For example, if we have a fluid flowing through some region (in the plane), at any time we can measure the velocity vector of the fluid at the point (x,y). This assigns a vector to the point. We call such an assignment a vector field. We can visualize a vector field by plotting a small arrow in the direction of the vector, but with its initial point based at the point (x,y). The matlab command `quiver` allows us to create such displays.

Example 8.1: Suppose a velocity field is given by $\mathbf{v}(x,y) = \langle y, \quad -x \rangle$. Use the `quiver` command to plot the field.

Solution: First, let's make sure we understand what we are trying to do. For example, the field associates with the point (1,1) the vector $\mathbf{v}(1,1) = \langle 1, \quad -1 \rangle = \mathbf{i} - \mathbf{j}$. We want to plot this vector with initial point $(1,1)$. Of course, this vector has length $\sqrt{2}$ and plotting lots of such vectors at their actual scale would quickly produce an unintelligible mess. Instead, Matlab scales the vectors (without changing their directions) so that the vectors that are drawn do not overlap. That's what `quiver` does for us.

To use the command, we define a grid of points (x,y) in some rectangle on which we want to plot the vector field. This grid is constructed as we have done for plotting surfaces. Then we compute the components of the vector field at the grid points and pass this to the `quiver` command. We illustrate with the vector field **v**.

```
x=linspace(-3,3,15);y=x;
[x y]=meshgrid(x,y);
vx=y; vy=-x;  % compute components of the vector field
quiver(x,y,vx,vy)
```



In this case it would seem that the fluid is flowing clockwise around the origin, which may not have been so obvious from just looking at the formula for **v**. ■

8.2 Gradient Plots

8.2.1 2D Gradient Fields

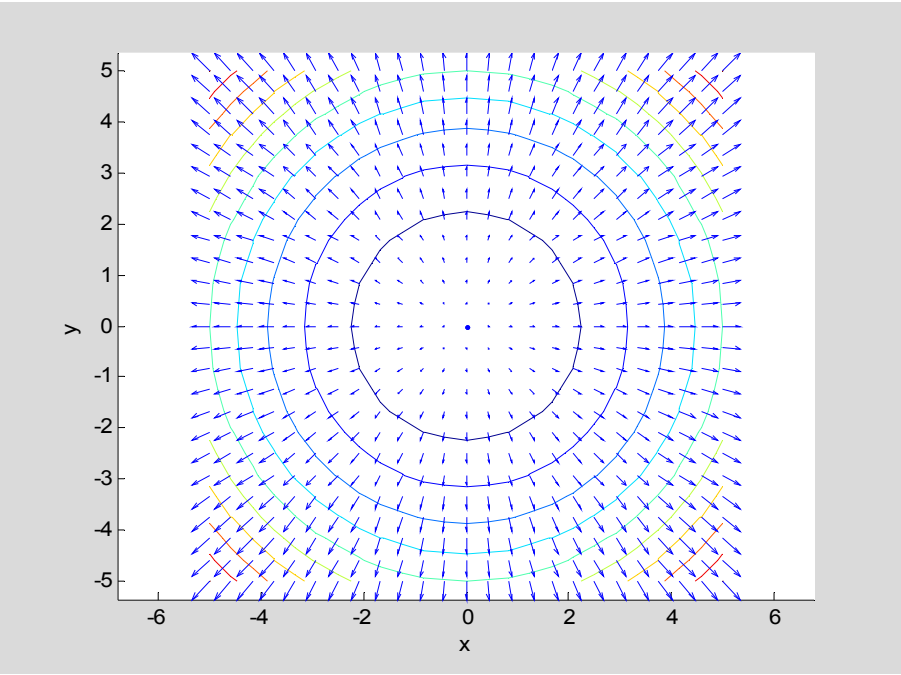
The gradient of a function $f(x,y)$ provides us with an important example of a vector field. This field is given by $\nabla f = \langle f_x \quad f_y \rangle$. We can illustrate some of the principle geometric properties of the gradient using the `quiver` command. For one, the gradient is orthogonal to the level curves of *f*, i.e. the curves defined by $f(x,y) = c$. We can illustrate this by plotting a contour plot and then superimposing the quiver plot on this. Let's do this for the simple case of $f(x,y) = x^2 + y^2$, whose contour lines (level curves) are concentric circles around the origin.

Example 8.2: Draw a gradient field plot of the function $f(x,y) = x^2 + y^2$ in the square defined by $-5 \le x \le 5$ and $-5 \le y \le 5$.

Solution:

```
x=linspace(-5,5,25);y=x;
[x y]=meshgrid(x,y);
z=x.^2+y.^2;
clf; hold on;
contour(x,y,z);
vx=2*x; vy=2*y;  % define components of gradient
quiver(x,y,vx,vy);
```

```
axis equal;  % useful for depicting orthog relationship
xlabel('x'); ylabel('y');
```



The gradient field is clearly perpendicular to the level curves. Moreover, as we approach the origin, the gradients become progressively shorter, until the gradient vanishes at the $(0,0)$. This simply says that $f_x = 0$ and $f_y = 0$ so the point $(x, y) = (0,0)$ is a critical point of f. ■

8.2.2 3D Gradient Fields

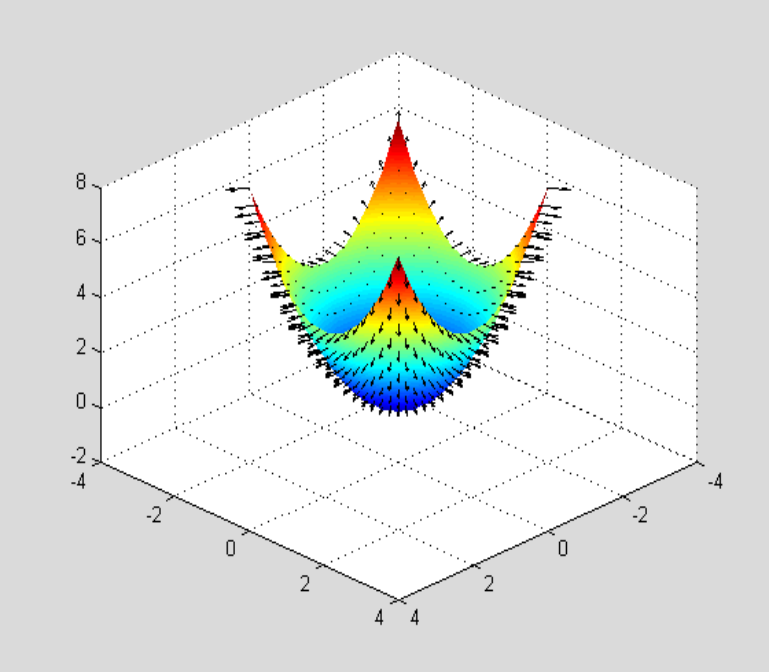
For a function $F(x, y, z)$ of three variables, the gradient ∇F is perpendicular to the level surface $F(x, y, z) = c$. We can plot such a level surface (often using parametric plotting) and generate a vector field of normals from the gradient. The command `quiver3` allows us to visualize the normal field. We illustrate with two examples:

Example 8.3: Draw the surface $z = x^2 + y^2$ over the rectangle $-2 \le x \le 2$, $-2 \le y \le 2$ and display the surface normals.

Solution: We can write the equation of the surface as $F(x, y, z) = x^2 + y^2 - z = 0$, so that the normal has components $\langle 2x \quad 2y \quad -1 \rangle$ (This also follows from our earlier work on the tangent plane to surfaces given by $z = f(x, y)$.) We compute the x, y, and z coordinates of points on the surface and then compute the components of the vector field from this formula.

```
x=linspace(-2,2,20); y=x;
[ x y] =meshgrid(x,y);
```

```
z=x.^2+y.^2;
vx=2*x; vy=2*y; vz=-1;  % the z-component is constant
clf; hold on;
surf(x,y,z);
shading interp  % removes grid lines from the surface
quiver3(x,y,z, vx,vy,vz, 'k');  % black arrows
view([1 1 1])
grid on;
```



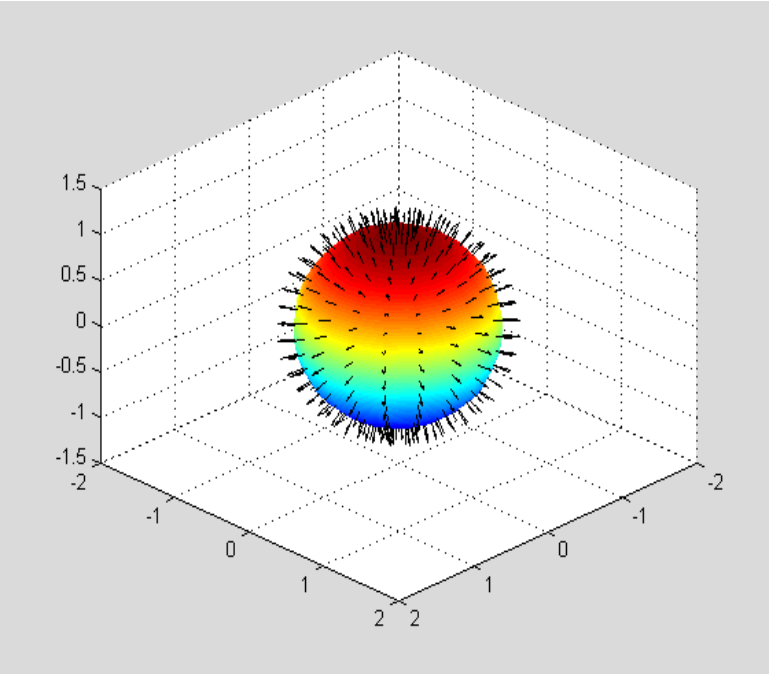
■

Example 8.4: Draw the sphere $x^2 + y^2 + z^2 = 1$ with the normal vectors (a "hairy" sphere).

Solution: This time we use spherical coordinates (See Chapter 7) to give a parametric representation of the sphere.

```
u=linspace(0,2*pi,20); v=linspace(0,pi,20);
[u v]=meshgrid(u,v);
x=cos(u).*sin(v);
y=sin(u).*sin(v);
z=cos(v);
vx=2*x; vy=2*y;vz=2*z; % gradient components
clf; hold on;
surf(x,y,z);
shading interp;
```

```
quiver3(x,y,z,vx,vy,vz,'k');
view([1 1 1])
grid on
```



■

8.3 Summary

quiver: `quiver(x, y, vx, vy)` draws vectors given by $\langle vx, vy \rangle$ with initial point (x, y). Lengths of vectors are scaled to avoid overlapping.

quiver3: `quiver3(x,y,z,vx,vy,vz)` draws vectors given by $\langle vx, vy, vz \rangle$ at the point (x, y, z). Vector lengths are scaled to avoid overlapping.

shading `interp`: removes gird lines from surface plot, leaving color scheme intact.

8.4 Exercises

Exercise 8.1 Using `contour`, plot at least 10 level curves of the function $f(x, y) = x^2 - y^2$ on the rectangle $-3 \le x \le 3$ and $-3 \le y \le 3$. Add a plot of the gradient field to the plot of the level curves. Make sure you adjust the scales using `axis equal` to preserve orthogonality. Publish your M-file with the graphic output.

Exercise 8.2 Using the parametric representation $x = \cosh(t)$, $y = \sinh(t)$, plot the curve $x^2 - y^2 = 1$. Include portions of both branches of this hyperbola. Then add the normal vectors to the plot of the curve. (Your plot should only show arrows attached to the hyperbola, not covering a 2D region.) Make sure you adjust the scales using `axis equal` to preserve orthogonality. Publish your M-file, including the graphic output.

Exercise 8.3 Plot the surface $x^2 + 3y^2 + 4z^2 = 9$ by parametrizing it (see Chapter 7 on parametric plotting for how to do this using spherical coordinates). On the same graph plot the normal vector field to this surface. Make sure you adjust the scales using `axis equal` to preserve orthogonality. Publish your M-file, including graphic output.

Exercise 8.4 Recreate the picture in Example 8.4 but using the `sphere` command to generate the points on the sphere.

Exercise 8.5 Matlab has a `gradient` command that approximates a gradient numerically. Look up how to use the command and then use it to generate the output in Example 8.3

9: Integration

9.1 Single Variable Integration

9.1.1 Symbolic Integration

If the Symbolic Toolbox is available, Matlab can calculate exact values of definite or indefinite integrals, as you learned to do in calculus II. For example, suppose we want to find $\int x\sin x\,dx$ (i.e. the antiderivative) and $\int_0^{\pi/4} x\sin x\,dx$. We first declare x a symbolic variable.

```
syms x
```

Example 9.1: (Indefinite Integral) Find $\int x\sin x\,dx$

Solution:

```
int(x*sin(x))
ans =
sin(x)-x*cos(x)
```

Matlab returns a specific antiderivative, without a constant of integration. ■

Example 9.2: (Definite Integral) Find $\int_0^{\pi/4} x\sin x\,dx$**.**

Solution:

```
int(x*sin(x),0,pi/4)
ans =
1/2*2^(1/2)-1/8*pi*2^(1/2)
```

In more traditional form we would write this as $\frac{1}{2}\sqrt{2}-\frac{1}{8}\pi\sqrt{2}$. If we wish a **numerical value** we can use the `double` command.

```
double(ans)
ans =
    0.1517
```

■

If you defined a symbolic function f (either anonymously or in an M-function) and you wish to find $\int f(x)dx$ you do not have to retype the expression $f(x)$ inside the `int` command. Just enter `int(f(x))`. For example, we have

```
f=@(x)x*sin(x);
int(f(x))
```

yielding

```
ans =
sin(x)-x*cos(x)
```

9.1.2 Numerical Integration

Matlab's ability to perform single variable symbolic integration is impressive. It can obtain answers to integration problems that are beyond the scope of elementary calculus.

For example, the result of

```
int(1/sqrt(1+x^2))
ans =
asinh(x)
```

may at least be familiar to you (it is the inverse hyperbolic sine function). But the answer to

```
int(1/sqrt(1+x^4))
ans =
1/(1/2*2^(1/2)+1/2*i*2^(1/2))*(1-
i*x^2)^(1/2)*(1+i*x^2)^(1/2)/(1+x^4)^(1/2)*EllipticF(x*(1/2*2^(1/2)+1/2*
i*2^(1/2)),i)
```

is certainly baffling, as it involves complex numbers (the *i* terms), as well as a new function `EllipticF`. There are a host of such special functions which arise in many advanced analytical problems. They are built into Matlab, much as sine and cosine, and Matlab uses them when theory indicates they are relevant.

Of course, even these special functions may fail to provide an answer to a problem. For example,

Example 9.3: Use Matlab to evaluate $\int \sqrt{1+\log x}\, dx$.

Solution:

```
int(sqrt(1+log(x)))
Warning: Explicit integral could not be found.
> In sym.int at 58
ans =
int((1+log(x))^(1/2),x)
```

■

In this case, Matlab gives up and issues a warning that it could not solve your problem. What do you do if you want to compute $\int_1^2 \sqrt{1+\log x}\, dx$? (Recall that in Matlab the natural logarithm is written as `log`, not ln.)

Fortunately, Matlab has a very precise numerical integration routine based on Simpson's Rule, which you learned about in calculus II. We won't go into the details of the method. The command is called `quad`, which is short for quadrature, a somewhat obsolete term

used for integration. The `quad` command works slightly differently than `int`. Instead of entering the expression that you want to integrate, it expects that you enter a handle to a function defining the expression. Moreover, your function needs to be vectorized, i.e. it must use "dot" operations or built-in functions such as `sin`, `sqrt`, etc, that are automatically vectorized.

Example 9.4: Use the `quad` command to evaluate $\int_0^{\pi/4} x \sin x \, dx$.

Solution:

```
quad(@(x)x.*sin(x),0,pi/4)


ans =

    0.1517
```

Of course, this agrees with the earlier result found using `int`. ■

In this example, we entered the function anonymously into the command. We can also first define the function using the @ construction. In that case, we simply pass the name of the function to the `quad` command (not the value $f(x)$ as we did for `int`). We can now numerically integrate the example that had stymied the symbolic processor.

Example 9.5: Use the `quad` command to evaluate $\int_1^2 \sqrt{1+\log x} \, dx$.

Solution:

```
g=@(x)sqrt(1+log(x));
quad(g,1,2)
ans =

    1.1743
```

■

In general, we would expect Matlab's numerical answer to be reliable. Nonetheless, it is a good idea to check things when you can do so. For example, we can try to perform a symbolic integration and, assuming Matlab can carry it out, compute a numerical approximation from the latter. If there are discrepancies, we will have to investigate more deeply to figure out which answer, if any, is correct. This may not be easy, but one way to do so is to write your own simple routine that gives an approximate answer. In the case of integration, we can use the simple scheme of mid-point Riemann sums. This works as follows

Mid-point Riemann Sums to approximate $\int_a^b f(x)\,dx$,

a) Select a whole number n.
b) Divide the interval of integration $[a,b]$ into n equal subintervals each of length $dx = \dfrac{b-a}{n}$.

c) Compute the mid-point of each subinterval. These are the n points $x_1 = a + \frac{dx}{2}$, $x_2 = x_1 + dx$, …, $x_n = x_{n-1} + dx$. The last point will be $b - \frac{dx}{2}$.

d) Compute the Riemann Sum
$$f(x_1)dx + f(x_2)dx + \cdots + f(x_n)dx = (f(x_1) + f(x_2) + \cdots + f(x_n))dx$$

e) The latter sum will approach $\int_a^b f(x)\,dx$ as n increases.

It is very easy to implement these steps as an M-file. We do this for the function $f(x) = \sqrt{1 + \log x}$. The following code has been saved as the M-file `midpoint.m` and the reader can modify it for use with a different example.

Example 9.6: Compute the Mid-Point Riemann Sum Approximation to $\int_1^2 \sqrt{1 + \log x}dx$

Solution:

```
f=@(x)sqrt(1+log(x));  % function should be vectorized
a=1; b=2;  % interval of integration
n=20;  % number of subintervals.
% The rest of the code just implements the method.
dx=(b-a)/n;
x=a+dx/2:dx:b-dx/2;  % generate the midpoints
y=f(x);  % compute the function values at the midpoints
RS=sum(y)*dx  % sum command adds the entries in the vector y
```

```
RS =

    1.1744
```

■

This is in sufficient agreement with the earlier result for us to accept the latter. If we had selected a larger number of subintervals we would have, in theory, seen closer agreement. In practice, using an extremely large a number of subintervals may introduce rounding errors that will degrade the theoretical accuracy of the method. This is more of a concern in multivariate integration.

9.2 Multiple Integration

9.2.1 Symbolic Double Integration

The `int` command can be nested to carry out iterated symbolic double integration. Matlab's graphing facilities may be able to assist you in visualizing the region of integration in order to set up the double integral. However, we assume at this point that you have formulated an iterated double integral to solve your problem. Let's do some examples illustrating how to do the integration. First we declare both x and y symbolic variables.

```
syms x y
```

Example 9.7: Compute $\int_0^1\int_0^2 x^2+y^2 dxdy$ **(Integration over a rectangle)**

Solution: To avoid excessive parentheses, we prefer to do the inner integral separately and then pass the result to the outer integral.

```
inner=int(x^2+y^2,x,0,2) % we must specify the variable of integration x
int(inner,0,1) % the variable of integration now defaults to the
remaining variable.
inner =
8/3+2*y^2
ans =
10/3
```

We could have suppressed the display of the inner integral. For those who prefer, the entire calculation can be entered in one line as

```
int(int(x^2+y^2,x,0,2),0,1)
ans =
10/3
```

■

If the region of integration is not a rectangle, we may enter appropriate formulas for the boundaries.

Example 9.8: Evaluate $\int_0^1\int_0^x x^2+y^2 dydx$

Solution:

```
inner=int(x^2+y^2,y,0,x); int(inner,0,1)
ans =
1/3
```

■

9.2.2 Numeric Double Integration

It is much more common that a symbolic integration package will fail to evaluate a double integral because of the increased complexity arising after the initial integration. It is therefore desirable to have a method for approximating such integrals numerically. Matlab has a procedure `dblquad` for doing this. This procedure works well for rectangular regions of integration, provided the integrand is continuous on the domain, including along the boundary. With some tweaking it is possible to use it on non-rectangular domains, but the results are often unreliable. We therefore do not recommend its use. In the next section we describe an alternative procedure called `quadvec`, which is not part of the Matlab system but has been developed by the Mathworks, works quite well, and can easily be extended to do numeric triple and higher order integrations, at least in principle.

As with `quad`, the `dblquad` command expects its function input to be the handle of a Matlab function. Moreover, this function must accept vectorized inputs.

Example 9.9: Evaluate $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ using `dblquad`.

Solution:

```
f=@(x,y)x.^2+y.^2;  %define anonymous function.  Note the . operations.
dblquad(f,0,2,0,1)  % first give x-limits, then y-limits


ans =
    3.3333
```

To the displayed precision, this agrees with the exact answer 10/3 computed above. ■

We can also do a quick check on the accuracy of this result by creating our own Riemann Sum approximation. This is similar to what we did above in the single variable case. We generate a 2D grid of midpoints and evaluate the function at each grid point. These values are summed and the result multiplied by $dx\,dy$ to produce a Riemann Sum approximation to the double integral. The only unexpected revision in the code is that to sum all the entries in a 2D array (matrix) we have to apply the `sum` command twice. On a 2D array, the `sum` command simply adds all entries in each column. We then apply the command again to add those entries. For example,

```
A=[1 2; 3 4]  % enters a 2 x 2 matrix
A =
     1     2
     3     4
sum(A)  % sums entries in each column
ans =
     4     6
sum(ans)  % sums the entries in the last vector, thus giving total for A
ans =
    10
sum(sum(A))  % this sums all entries in A.  Notice the similarity to
using a double integral.
ans =
    10
```

Example 9.10: Construct an M-file to find a mid-point Riemann sum approximation to a double integral (over a rectangle). (This file is available as `dblmidpoint.m`.)

Solution:

```
f=@(x,y)x.^2+y.^2;  % function should be vectorized
```

```
a=0; b=2;  % x axis limits of integration
c=0; d=1;  % y axis limits of integration
m=20;  % number of x-subintervals.
n=20; % number of y-subintervals
% The rest of the code implements the method.
dx=(b-a)/m; dy=(d-c)/n;
x=a+dx/2:dx:b-dx/2;  % generate the x-midpoints
y=c+dy/2:dy:d-dy/2;  % generate the y-midpoints
[x y]=meshgrid(x,y); % 2D array of grid coordinates of midpoints
z=f(x,y);  % compute the function values at the midpoints
RS = sum(sum(z))*dx*dy
```

```
RS =
    3.3312
```

The answer only agrees to the second decimal place with the more precise value using `dblquad`. Using a larger number of subintervals will produce closer agreement. ■

9.2.3 Alternative Numeric Double Integration

The `dblquad` command is awkward to use with non-rectangular domains and often suffers from a lack of accuracy on such domains. One might think that nested `quad` commands could be used to numerically evaluate a double integral, much as nested `int` commands can be used to symbolically evaluate double integrals. Let's see what happens when we try to carry out such a plan. We consider $\int_0^1\int_0^2 x^2+y^2 dxdy$. First we define a numeric inner integral by entering

```
inner=@(y)quad(@(x)x.^2+y.^2,0,2)  % computes a function that gives the
numeric value of the inner integral for any value y of outer integration
```

```
inner =
    @(y)quad(@(x)x.^2+y.^2,0,2)
```

We can evaluate such a function in the usual way. For example,

```
inner(1.5)
```

```
ans =
    7.1667
```

This is just $\int_0^2 (x^2 + (1.5)^2)dx$. The `inner` function passes the value $y = 1.5$ to the integrand in `quad` and then `quad` performs a numerical integration. Note, however, that the function `inner` is not vectorized. If we ask

```
inner([1, 1.5])
```

```
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> @(x)x.^2+y.^2
Error in ==> quad at 63
y = f(x, varargin{:});
Error in ==> @(y)quad(@(x)x.^2+y.^2,0,2)
```

we get a series of error messages. Why is this important? Because, if we want to integrate the `inner` function using `quad` (to complete the double integral) we must supply the function handle of a vectorized function and `inner` is not vectorized. Thus, writing

```
quad(inner, 0,1)
```

```
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> @(x)x.^2+y.^2
Error in ==> quad>quadstep at 118
y = f(x, varargin{:});
Error in ==> quad at 78
...
```

produces a similar collection of error messages.

The M-function `quadvec` uses the `quad` function (actually a more precise version known as `quadl`), but automatically vectorizes the input function you give it. This allows us to plug in function handles of non-vectorized functions and `quadvec` will vectorize them before passing them to the `quad` function. Assuming `quadvec` is in the path accessed by Matlab we can then compute $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ as follows.

Example 9.11: Compute the value of $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ using the routine `quadvec`.

Solution:

```
inner=@(y)quadvec(@(x)x^2+y^2,0,2);
quadvec(inner,0,1)
```

```
ans =
    3.3333
```

Notice that we did not have to vectorize even the initial function expressions, because `quadvec` automatically takes care of that. Naturally, we can write the entire expression as a nested function, though the parentheses get rather thick. The @ designations in effect serve to indicate the variable of integration in each part.

```
quadvec(@(y)quadvec(@(x)x^2+y^2,0,2),0,1)
```

```
ans =

    3.3333
```

■

One additional advantage of this command is that it allows us to easily carry out numeric integrations over non-rectangular regions.

Example 9.12: Evaluate $\int_0^1 \int_0^{\sqrt{1-x^2}} x^2 + y^2 dydx$ numerically using `quadvec`.

Solution:

```
inner = @(x)quadvec(@(y)x^2+y^2,0,sqrt(1-x^2));
quadvec(inner,0,1)
```

```
Warning: Minimum step size reached; singularity possible.
> In quadl at 95
  In quadvec at 22
  In @(x)quadvec(@(y)x^2+y^2,0,sqrt(1-x^2))
  In quadvec>g at 26
  In quadl at 64
  In quadvec at 22
ans =
    0.3927
```

Although the command issued a number of warnings, it did produce the correct numerical value, (to the default tolerance of 10^{-6}), as we can verify in this case using symbolic integration:

```
syms x y;
int(int(x^2+y^2,y,0,sqrt(1-x^2)),0,1);
double(ans)
```

```
ans =

    0.3927
```

■

9.2.4 Triple Integrals

Much of what we have said extends to triple integrals. We can nest the `int` function to duplicate the computation of iterated triple integrals. Similarly we can iterate `quadvec` to handle examples that can only be done numerically. We demonstrate both methods by

computing a common integral. We do not recommend the built-in routine `triplequad`, unless you need to compute a triple integral over a rectangular box with sides parallel to the coordinate planes.

Example 9.13: Find the exact value of $\int_0^1 \int_0^z \int_0^y ze^{-y^2}\,dxdydz$ using the `int` command.

Solution: We need to create three symbolic variables. Then we use `int` to carry out the successive integrations. We show the intermediate steps so the reader can compare the computation with the standard evaluation done by hand.

```
syms x y z
intx=int(z*exp(-y^2),x,0,y)
inty=int(intx, y, 0, z)
int(inty, 0, 1)
double(ans)  % convert to numeric answer
```

```
intx =
z*exp(-y^2)*y
inty =
-1/2*z*exp(-z^2)+1/2*z
ans =
1/4*exp(-1)
ans =
    0.0920
```

■

Now we repeat the calculation using `quadvec`. The steps are very similar to those displayed above, except we pass function handles at each call of the `quadvec` function.

Example 9.14: Approximate the value of $\int_0^1 \int_0^z \int_0^y ze^{-y^2}\,dxdydz$ using the `quadvec` command.

Solution:

The computation generates many warnings, similar to those displayed above in Example 9.12. In order not to fill the page with these warnings, we suppress that part of the output.

```
warning off all;  %turn off warnings
f=@(x,y,z)z*exp(-y^2);
intyz=@(y,z)quadvec(@(x)f(x,y,z),0,y);
intz=@(z)quadvec(@(y)intyz(y,z), 0, z);
quadvec(intz,0,1)
warning on  % turn on warnings at exit
```

```
ans =
    0.0920
```

This agrees with the result obtained earlier using symbolic iterated triple integrals.

Notice that whenever the integrand involves more than one variable, we have to create a function handle from this integrand that only involves the single variable that we wish to integrate. Thus, the *@variable* symbol at each stage serves as a reminder of the corresponding d*variable* in the traditional integral. We could have done this at the end as well, writing `quadvec(@(z)intz(z),0,1)`, though it is superfluous at that stage.■

9.3 Summary

We discussed both numerical and symbolic single and multiple integrals. We used the following built-in Matlab functions:

`int`: computes symbolic integrals; can be nested to do multiple integrals. Input must be a symbolic expression.

`double`: converts an exact (symbolic) value to double precision floating point (decimal).

`quad`: single variable numeric integration. Input must be the handle to a vectorized function of a single variable. Cannot be nested to do multiple numeric integration.

`dblquad` : double integral numeric integration. Input must be the handle to a vectorized function of two variables. Cannot be applied directly to non-rectangular domains. See also **`triplequad`** for triple integrals over 3D boxes.

In addition, we discussed several M-files as enhancements or checks on the system files.

`midpoint`: implements midpoint Riemann sum approximation to a single integral

`dblmidpoint`: implements midpoint Riemann sum approximation to a double integral over a rectangle.

`quadvec`: vectorizing numerical integrator. Can be used for multivariate numeric integration.

9.4 Exercises

Exercise 9.1: Let *R* be the region in the plane bounded by $y = x^2$, $y = 0$, and $x = 1$. Set up an iterated double integral to evaluate $\iint_R e^{y/x}\,dA$.

Your work should produce a single M-file that gives the following output:

a) Uses Matlab's symbolics to find the exact value of the double integral.

b) Uses `quadvec` to obtain a numerical estimate for the double integral.

Publish your work to html.

Note: In publishing your work, insert cell dividers after each output that you want displayed. This way the output will appear directly after the input that generates it, rather than having all the output at the end, where one can't tell what it goes with. See the Cell menu for the command to enter a cell divider.

Exercise 9.2: Using Matlab write an M-file that does the following:

a) Draws the region *R* between the curves $y = (1 - x^{2/3})^{3/2}$ and $y = 1 - x$ for $0 \le x \le 1$.

b) Assuming the region *R* (in 9.2a) is a plane lamina of density 1, finds its exact mass using symbolic double integrals. Convert to a numerical value using `double`.

c) Use the `quadvec` command to confirm the answer in b) by numerical calculation.

d) Uses Matlab to find the exact expressions for the moments of the region *R* about the *x* and *y* axes.

e) Using d) and either the answer to b) or c) find a numerical approximation to the coordinates of the center of mass of *R*. (N.B. The symmetry of the region suggests that you should find $\bar{x} = \bar{y}$.)

Publish your M-file in html. Make sure you indicate by suitable comments what you are calculating in each part. Also, see the note in Exercise 9.1 concerning separation of the output when you publish the M-file.

Exercise 9.3 A solid hemisphere defined by $0 \le x^2 + y^2 + z^2 \le 1$ and $z \ge 0$ has a density given by $\delta = \sqrt{x^2 + y^2 + z^2}$.

a) Set up a triple integral in spherical coordinates to find the mass of the hemisphere. Use Matlab to evaluate the iterated triple integral. (You should verify the result by hand.)

b) Determine the height $z = z_0$ such that half of the mass of the hemisphere lies between $z = 0$ and $z = z_0$. Use Matlab to carry out any necessary computations. Check your work by hand!

Exercise 9.4 Evaluate the triple integral $\int_0^3 \int_0^2 \int_0^1 x^2 + y^2 + z^2 dxdydz$ using **four** different methods in Matlab:

a) Exactly, using symbolic integration with the `int` command.

b) numerically, using the command `triplequad`.

c) numerically, using the command `quadvec`.

d) writing your own routine, `trpmidpoint` to estimate the mid-point Riemann sum approximation to the triple integral. Model your code after the code for `dblmidpoint`.

Include the results of using the first three methods in one M-file, clearly separating the parts and their output using cell dividers. (See note in Exercise 9.1) Submit the results using method d) in a separate M-file that contains the code for implementing the computation.

Exercise 9.5 Referring to the plot described in Exercise 7.6, using double integrals find the volume of the region inside both cylinders and lying above the *xy*-plane. Your M-file should generate

a) The picture of the 3D region whose volume is requested (see Exercise 7.6)

b) the picture of the 2D region of integration

c) the volume computed using iterated integration and the `int` function.

Your M-file should be divided into cells with the output requested above apppearing after the appropriate cell.

Exercise 9.6 Referring to the plot described in Exercise 7.8, find the volume of the region inside the sphere and above the *xy*-plane that lies within the cylinder. Your M-file should generate

a) The picture of the 3D region whose volume is requested (see Exercise 7.8)

b) the picture of the 2D region of integration

c) the volume computed using iterated integration and the `int` function.

Your M-file should be divided into cells with the output requested above apppearing after the appropriate cell.

10: Sequences and Series

Infinite series play an important role in many areas of applied mathematics, such as probability theory, differential equations, and signal processing, to name a few. In addition, the theoretical questions related to the notion of convergence provide the basis for the underlying theory of calculus, on which much of modern mathematics rests.

Matlab cannot help you determine what convergence test to use to justify the convergence or divergence of a particular sequence or series. It can however do two things that are both useful. First, it can compute accurate numerical approximations that can help you assess the existence of a limit and its numerical value. Second, using the Symbolic Toolbox, it can in many cases provide you with a symbolic expression for the value of a limit. In this chapter, we will consider how to use Matlab in each of these ways.

10.1 Numerical Estimation

10.1.1 Sequences

A sequence is an ordered list of (real) numbers, which we usually refer to in a subscript notation as $a_1, a_2, \ldots, a_n, \ldots$. From a more mathematical perspective, a sequence defines a function, *a*, whose domain is the set of natural numbers. The value of this function at $n = 1$ is written as a_1, instead of the more traditional $a(1)$, similarly a_2 instead of $a(2)$, etc. From the point of view of Matlab, we can therefore define a sequence as we do any other function, except we will use letters such as *k*, and *n*, for the independent variables, rather than *x* and *y*, and we will also often use letters such as *a* and *b* for the function names, rather than *f* and *g*.

Example 10.1: Define the sequence $a_k = \dfrac{(-1)^k k}{k^2+1}$ as an anonymous function of *k*. Generate the first 10 values in the sequence.

Solution: Since we want to generate a list of values, we should define the function as a vectorized expression.

```
a=@(k)(-1).^k.*k./(k.^2+1)
```

```
a =
    @(k)(-1).^k.*k./(k.^2+1)
```

Now we evaluate this function on the sequence 1:10 to find its first ten values.

```
a(1:10)
```

```
ans =
  Columns 1 through 7
   -0.5000    0.4000   -0.3000    0.2353   -0.1923    0.1622   -0.1400
```

```
  Columns 8 through 10
    0.1231   -0.1098    0.0990
```

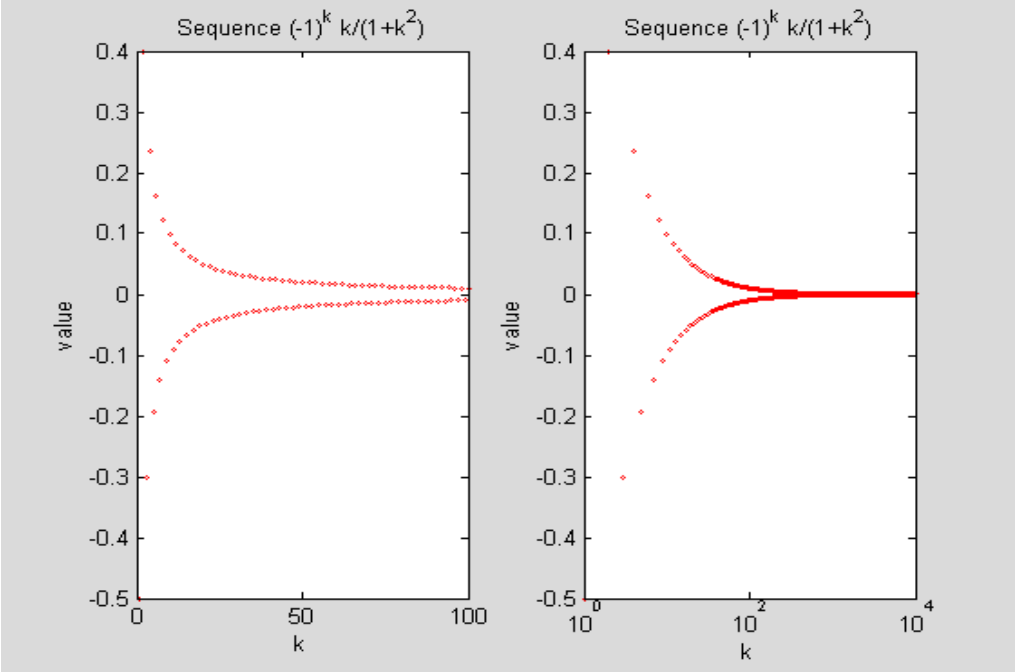
As expected, the terms alternate in sign and are getting smaller in magnitude as k increases. ■

In this example, it is mathematically pretty obvious that $\lim_{k\to\infty}\frac{(-1)^k k}{k^2+1}=0$, since the quotient $k/(1+k^2)$ tends to zero. We can get a visual sense of that behavior by plotting the values of the sequence a_k for a large run of values of k. We do this is in the following example, using the first 100 values in the sequence (not actually a very long run) and then using 10,000 values.

Example 10.2: Plot the first 100 and then the first 10,000 values of the sequence $\frac{(-1)^k k}{1+k^2}$.

Solution: We generate two plots, placing them side by side using the `subplot` command.

```
clf;
n=100;
a=@(k)(-1).^k.*k./(k.^2+1);
vals=a(1:n);
subplot(1,2,1);
plot(vals,'or','MarkerSize',2); %plot sequence values using red 'o'
marker, with no connecting line
title('Sequence (-1)^k k/(1+k^2)');
xlabel('k');ylabel('value');
n=10000;
vals=a(1:n);
subplot(1,2,2);
semilogx(vals, 'or', 'MarkerSize',2); %Log scale on k axis
title('Sequence (-1)^k k/(1+k^2)');
xlabel('k');ylabel('value');
```



Note that the plot commands did not specify the values of the independent variable. If these are not given, Matlab plots the sequence values against 1,2,…*n*, where *n* is the number of entries, which is what we wanted anyway. We did not connect the points, because technically the sequence only consists of the values generated. There is nothing wrong with connecting the points as a visual guide, as long as you realize that the segments joining successive points have no other meaning.

In the second plot we have used a semi-log plot in which the *k* axis is plotted on a logarithmic scale. In this type of plot, successive powers of 10 are equally spaced. Such plots (and the related `loglog` plot) are useful when the domain and/or range encompass values spread over several orders of magnitude (powers of 10). The behavior within each decile (power of ten) is then more clearly displayed. Note that between 1 and 10 we only have 10 values to plot, which show up as discrete points. In the interval from 11 to 100 there are 90 values, and many more in the larger deciles. Squeezing these into the allotted space produces the appearance of a smooth graph.

Both plots clearly show that the numbers in the sequence are approaching zero or at least some numerically very small value. ■

We have examined a sequence given by an explicit formula. An alternate way to generate a sequence is through recursion. In this process, we state values for one or more initial terms of the sequence. The recursive definition shows how to compute additional terms. For example, the famous Fibonacci sequence is determined by the recursion:

$$f_1 = 1, \quad f_2 = 1, \quad f_k = f_{k-1} + f_{k-2} \quad \text{if } k \geq 3 .$$

From the last formula or recursion, we have $f_3 = f_2 + f_1 = 2$, $f_4 = f_3 + f_2 = 2+1 = 3$, $f_5 = f_4 + f_3 = 3+2 = 5$, and so on. If we wished to find f_{100}, we would need to find all the Fibonacci numbers for $k < 100$, which makes for considerably more work than when we examined the explict sequence in Example 10.1. Nonetheless, here are the commands of a short M-file to compute that result. The reader ought to enter this as an M-file and observe how the Matlab editor handles the looping "for…end" command when it is typed.

Example 10**.3:** Compute the 100th Fibonacci using the Symbolic Toolbox.

Solution: We use the symbolic toolbox so as to compute the exact integer value. The program computes a vector *f* whose components are the first 100 Fibonacci numbers:

```
clear f ;

% Clears f of any existing values. Useful when dealing with vector
quantities so that previously computed components are not accidentally
retained.

n=100; % the number of Fibonnaci numbers you want to compute.

f(1)=sym(1);f(2)=sym(1);

% Initialize the list to symbolic integer values using sym.  Further
calculations will automatically be done in exact arithmetic.

for k=3:n
   f(k)=f(k-1)+f(k-2);
          % recursive step; Note semicolon to suppress output.
end;       % for loop to repeatedly execute recursive statement
f(n)    % display final value


ans =

354224848179261915075
```

As you can see the number is quite large. If we had run the program by setting `f(1)=1` and `f(2)=1` the computation would have run in normal machine arithmetic and we would have obtained an approximation to the above value, namely

```
double(ans)


ans =

  3.5422e+020
```

.

■

Recursively defined sequences play an important role in many areas of mathematics, but we only mention them here for the sake of completeness. From a computational point of view they present more of a challenge than explicitly defined sequences, because at each stage the next computed value depends on previously computed values. Using the usual floating point arithmetic on the computer this may lead to magnification of small round-off errors at each stage into significant errors in the long term computation.

10.1.2 Series

We can associate with a numerical sequence a_k a new sequence, called the sequence of partial sums, defined by

$$\begin{aligned} s_1 &= a_1 \\ s_2 &= a_1 + a_2 \\ s_3 &= a_1 + a_2 + a_3 \\ &\cdots \\ s_n &= a_1 + a_2 + \cdots + a_n \\ &\cdots \end{aligned}$$

If the sequence of these partial sums has a limit L we say the the infinite series $\sum_{k=1}^{\infty} a_k$ converges to L. We write this as $\sum_{k=1}^{\infty} a_k = L$.

The partial sum sequence can also be defined recursively by $s_1 = a_1$ and for $n \geq 2$, by $s_n = s_{n-1} + a_n$. This definition assumes that the values of a_n have been computed elsewhere (or are given by some formula that can be evaluated for each n).

Concretely, you can think of the relationship between the sequences a_k and s_k as follows. The numbers a_k represent weekly deposits (if positive) or withdrawals (if negative) from a bank account. The partial sums give the total accumulation in the account each week. How does your pattern of weekly deposits affect the pattern of your total accumulations? Well, the theory of infinite series you are studying in class tries to provide some answers to that question. Here we will use Matlab to investigate the sequence of partial sums in a manner similar to our earlier investigation of sequences.

The principal tool in doing this is the command `cumsum`, short for cumlative sum. Following our analogy with a bank account, this simply computes the accumulated sum in your account at the end of each time period. For example, with

```
x=[1 2 3 4];
```

we have

```
cumsum(x)
```

```
ans =
     1     3     6    10
```

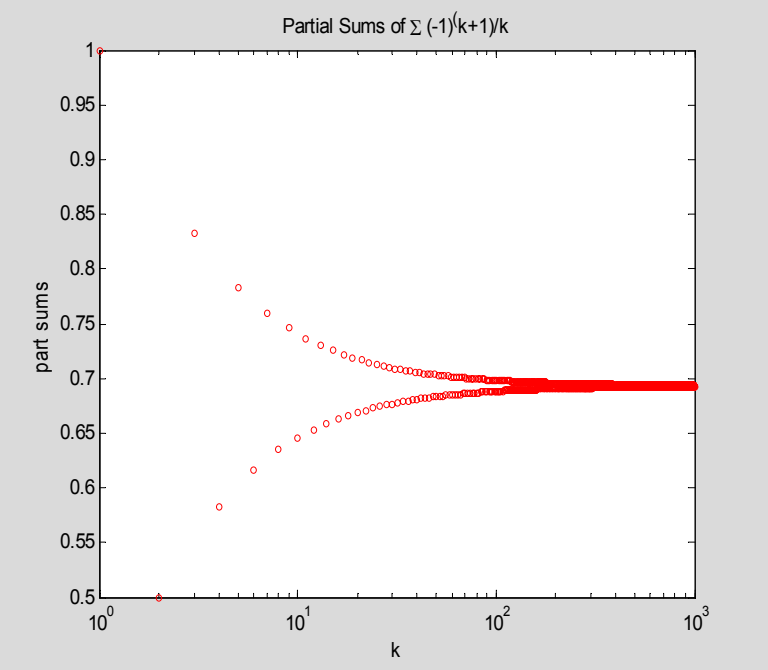
.

In short, `cumsum` generates the sequence of partial sums associated with the infinite series $\sum_{k=1}^{\infty} a_k$. We can plot this sequence to generate a visual picture of the behavior of the partial sum sequence.

Example 10**.4:** Make a semilog plot of the partial sums of the first 1000 terms of the series $\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}=1-\frac{1}{2}+\frac{1}{3}+\cdots$. Does the plot give evidence of the convergence of the series?

Solution:

```
clf
a=@(k)(-1).^(k+1)./k; % define the sequence
n=1000;
vals=a(1:n);
s=cumsum(vals);
semilogx(s, 'or', 'MarkerSize',2)
xlabel('k');ylabel('part sums');
title('Partial Sums of \Sigma (-1)^(k+1)/k')
% \Sigma codes for Greek letter upper case sigma
```



The series seems to be approaching a limit. The term

```
s(1000)
```

is

```
ans =
    0.6926
```

and based on the graph, this should be a good approximation to the limit of the series, though this cannot be ascertained from the numerical computation alone. Comparing this partial sum to the previous one

```
s(999)
```

```
ans =

    0.6936
```

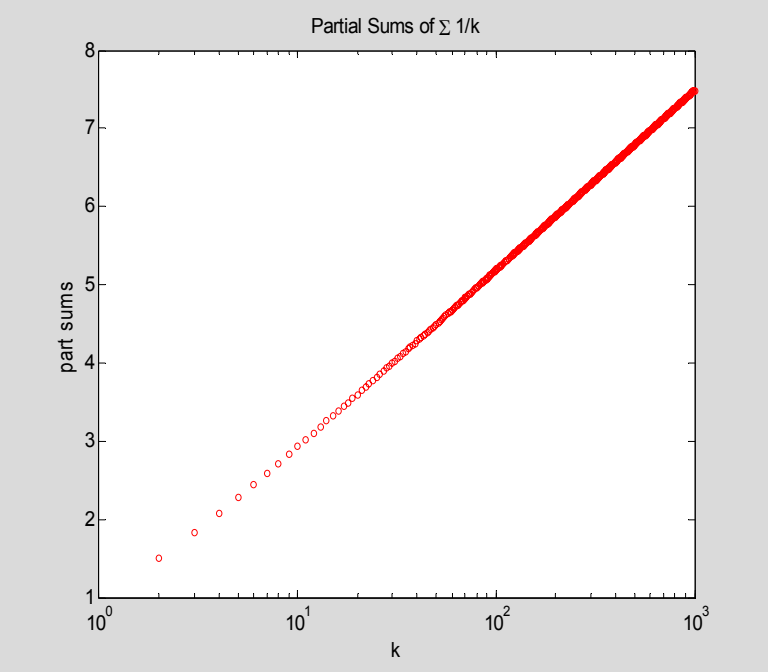
we have some confidence in asserting that the limit value is between .692 and .694, or in less precise terms $.693 \pm .001$. In this case, the error estimates associated with alternating series provide a rigorous justification for this conclusion. ■

The harmonic series $\sum_{k=1}^{\infty} 1/k$ is an important example of a divergent series whose terms go to zero. Is such behavior detectable using Matlab?

Example 10**.5:** Make a semilog plot of the partial sums of the first 1000 terms of the harmonic series. Discuss the convergence of the series as it relates to the plot.

Solution:

```
clf
a=@(k)1./k; % define the sequence
n=1000;
vals=a(1:n);
s=cumsum(vals);
semilogx(s, 'or', 'MarkerSize',2)
xlabel('k');ylabel('part sums');
title('Partial Sums of \Sigma 1/k')
```



The graph is strikingly different from the graphs for the partial sums of a convergent series. While the picture does not preclude that the partial sums converge, it certainly provides no compelling reason to think so, and a strong suggestion that they do not. Nonetheless, only a mathematical argument based on examining properties of the sequence can actually settle that question. ■

10.2 Symbolic Limits

The Symbolic Toolbox in Matlab is capable of finding limits of many sequences and series. These may be expressed in terms of familiar functions and mathematical constants, or may involve values of special functions. Many of these results rest on mathematical analysis that is quite advanced and well beyond the level of this course. However, these results are of interest because they often point to mysterious connections between seemingly unconnected parts of mathematics. The exercises will present more examples for the reader to consider.

Example 10**.6:** Find the value of $\lim_{k\to\infty}\left(1+\frac{1}{k}\right)^k$.

Solution:

First declare *k* symbolic. Then use the `limit` function.

```
syms k
limit((1+1/k)^k,inf) % inf is the Matlab abbreviation for "infinity"


ans =
exp(1)
```

The answer, which is *e*, is expressed as a value of the exponential function. ■

More impressive are the results obtained for infinite series. The command `symsum` computes the symbolic limit.

Example 10.**7:** Find the value of $\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}$.

Solution:

```
syms k
symsum((-1)^(k+1)/k,1,inf) % last two arguments give the summation range
```

```
ans =
log(2)
```

The answer is expressed in term of the natural log function. Note the value is

```
double(ans)
```

```
ans =
    0.6931
```

The reader should compare this to the numerical approximation computed in Example 10.4. ■

10.3 Summary

A sequence is a function whose domain is the set of natural numbers. Such functions can be defined using the standard method of anonymous functions or can be constructed through recursive definitions.

We can analyze the limit behavior of sequences and series through numerical computation. The results can be usefully presented in graphical format. The Symbolic Toolbox provides commands that can be used in many cases to find exact formulas for limits.

The following commands were introduced in this chapter:

`semilogx`: - plot with independent variable displayed in a logarithmic scale

`cumsum`: - computes sequence of partial sums of a numerical sequence

`limit`:- compute a symbolic limit of an explicitly given sequence

`symsum`: - compute a symbolic expression for a sum of terms of an explicitly given sequence

10.4 Exercises

Exercise 10.1 Let $a_k = k^{1/k}$. Write an M-file that

a) plots (you can use a regular plot, instead of a semilog plot) the first 100 values of the sequence as individual points

b) based on the numeric values generated makes an estimate of the limit of the sequence

c) uses the Symbolic Toolbox to find the exact limit.

Publish your M-file. The results of a), b), and c) should appear as separate output from different cells in the file. Don't forget to include your name at the top.

Exercise 10.2 Repeat the instructions in Exercise 10.1 for the sequence $a_k = \left(1 - \frac{2}{k}\right)^k$.

Exercise 10.3 Investigate which if any of the following sequences have limits. Provide all possible justifications for your answer using a variety of Matlab computations and graphs. Present your work in the form of a published M-file with appropriate comments:

a) $a_k = \sin(k)$

b) $b_k = (\sin(k))^k$

c) $c_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} - \ln k$

Exercise 10.4 Every year your bank account earns 5% interest on the principal on deposit for that entire year. In addition, you make an additional deposit of \$1000 into the account at the end of each year and no withdrawals. Let P_n denote the principal in the account at the beginning of year *n*.

a) For $n \geq 2$, express the principal P_n in terms of the principal P_{n-1}.

b) Assuming your initial principal is $P_1 = \$1000$, find the principal at the beginning of year 21. How much of that principal is interest? Make a graph showing the growth of the principal from year 1 to year 21.

c) Assuming the same initial deposit of \$1000 and a 5% interest rate, what is the minimum you would have to deposit each year (assuming equal deposits) in order for your accumulation to reach \$50,000 by the beginning of year 21? Display your result graphically.

Exercise 10.5 Write an M-file to generate a `semilogy` plot of the first 100 Fibonacci numbers. What can you deduce from the pattern of the plot for large *k*? Extend the plot for a larger range of the index. Use your plot to develop an approximate explicit formula for the *k*th Fibonacci number.

Exercise 10.6 Modify the Fibonacci sequence so that for $n \geq 3$ the recursion is $f_k = f_{k-1} + f_{k-2} + (-1)^k k$. How large is the 100th value of this sequence? Produce an appropriate plot showing the values of the numbers in this sequence.

Exercise 10.7 Investigate the convergence or divergence of each of the following series. You should use numerical and/or symbolic computations to arrive at your conclusions. To the extent possible you should confirm your answers using the theory covered in the course. Present the response to each part as a separate published M-file.

a) $\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2^k}$

b) $\sum_{k=1}^{\infty} \frac{1}{k!}$ (The quantity $k!$ can be entered in Matlab as `factorial(k)`.)

c) $\sum_{k=1}^{\infty} \frac{\sin(k)}{k}$

d) $\sum_{k=1}^{\infty} \frac{\sin(k)}{k^2}$

e) $\sum_{k=1}^{\infty} \frac{1}{k(k+1)(k+2)}$

f) $\sum_{k=1}^{\infty} \frac{1}{k^2}$

g) $\sum_{k=1}^{\infty} \frac{(-1)^k}{k^2+1}$

11: Power Series

One of the main applications of the theory of numerical series is to the study of series of functions. In particular, when the functions take the simple form of powers of the variable multiplied by a constant, we speak of a power series. Such series are a workhorse of many numerical methods in mathematics because they allow one to approximate complicated functions by much simpler ones. Infinite series comprised of trigonometric functions, so-called Fourier series, play a similar role when studying periodic phenomena.

Matlab is quite useful in helping us visualize the convergence process and approximating a limit when one exists. In many cases, it can also provide closed formulas for the limit function of an infinite series. We will use the Symbolic Toolbox in our analysis, since restricting the treatment to just numerical manipulations in Matlab presents some technical difficulties that would obscure the mathematical ideas.

11.1 Plotting Power Series

A power series is an infinite series of the form

$$\sum_{k=0}^{\infty} a_k (x-c)^k$$

where the a_k and c are real numbers. The principal question concerning such an object is determining the set of values of the variable x for which the series converges. The theory of series informs us that convergence occurs in some interval of length $2R$ centered around the point c, where R may be zero or infinity. This is called the *interval of convergence* and the quantity R is called *the radius of convergence*. If $0 < R < \infty$, further analysis is required to determine whether the series converges at the endpoints $c-R$ and $c+R$.

Assuming you have determined the interval of convergence for a particular power series, we will write an M-file that plots the series and a partial sum $\sum_{k=0}^{n} a_k (x-c)^k$, where n is chosen by the user. We will consider the example

$$\sum_{k=1}^{\infty} \frac{2^k}{k} x^k ,$$

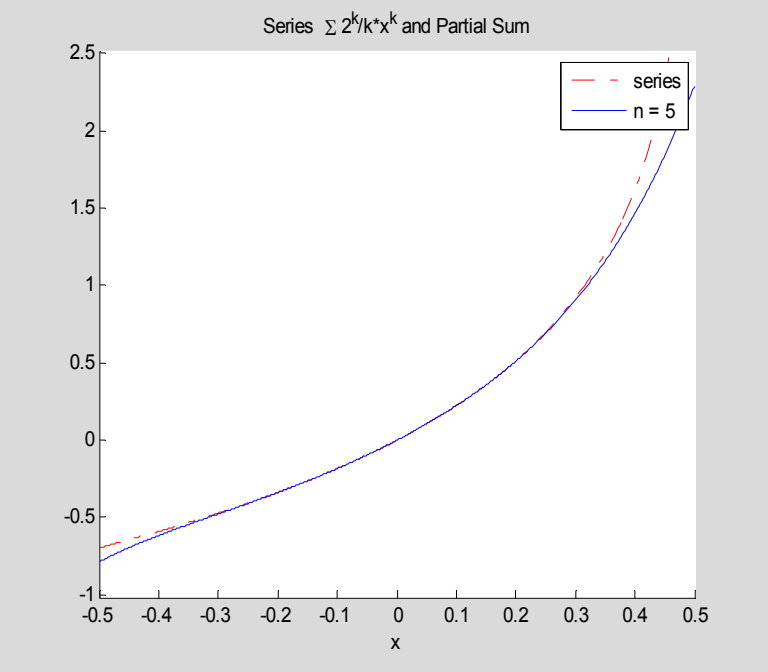
which converges in the interval $|x| < 1/2$.

Example 11.1: Plot the limit of the series $\sum_{k=1}^{\infty} \frac{2^k}{k} x^k$ and the partial sum $\sum_{k=1}^{5} \frac{2^k}{k} x^k$ on the interval $[-1/2, 1/2]$.

Solution:

Since we will be generating and plotting symbolic expressions we will be working with `ezplot`, which we studied at the beginning of the course.

```
syms k x
clf; hold on
%%%%% User Input %%%%%%%
a=@(k)2^k/k;  % define coefficient sequence
f=@(x)symsum(a(k)*x^k,k,1,inf); %define series
p=@(x,n)symsum(a(k)*x^k,k,1,n); % define partial sum
n=5;  % degree of partial sum
x1=-1/2;x2=1/2; % endpoints of plotting interval
%%%%% End User Input %%%%%%%
h=ezplot(f(x), [x1,x2]);  % plot limit function
set(h, 'Color','r', 'LineStyle','-.'); % use handle to set properties
ezplot(p(x,n), [x1,x2]); % plot partial sum; leave default style
legend('series',['n = ', num2str(n)]);
% num2str converts n into a string
title(['Series  \Sigma ',char(a(k)), '*x^k and Partial Sum' ]);
hold off
```



The two graphs are almost indistinguishable throughout the interval of convergence. The `symsum` command actually computes the limit of the power series. Namely,

```
symsum(2^k/k*x^k, k, 1, inf)
```

```
ans =
-log(1-2*x)
```

For convenience, this M-file is available in the course M-file folder as `powseries.m`. Note, this M-file will only work for power series for which Matlab is able to find a closed form expression. For example, even though the series $\sum_{k=1}^{\infty} \frac{x^k}{\sqrt{k}}$ converges on the interval $(-1,1)$, Matlab cannot determine an explicit formula for the series and therefore is unable to execute the `ezplot` command. ■

11.2 Taylor Series

A power series determines a function on its interval of convergence. We can recover the power series coefficients from this function by successive differentiation. Namely, if

$$f(x) = \sum_{k=0}^{\infty} a_k (x-c)^k$$

then $a_k = \frac{f^{(k)}(c)}{k!}$. If we only have the function $f(x)$ to begin with, we can use the latter formula for a_k (provided f is infinitely differentiable) to define a power series. This series is called the *Taylor series* of the function f around $x = c$, or in powers of $x - c$. For most functions this series converges to the original function, at least on the interior of the interval of convergence for the series. The partial sum,

$$p_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k$$

is called a *Taylor polynomial* (of degree *n*).

We illustrate computations with Taylor polynomials in Matlab and then leave it to the reader to exhibit graphically the convergence of these series through a simple modification of our previous program.

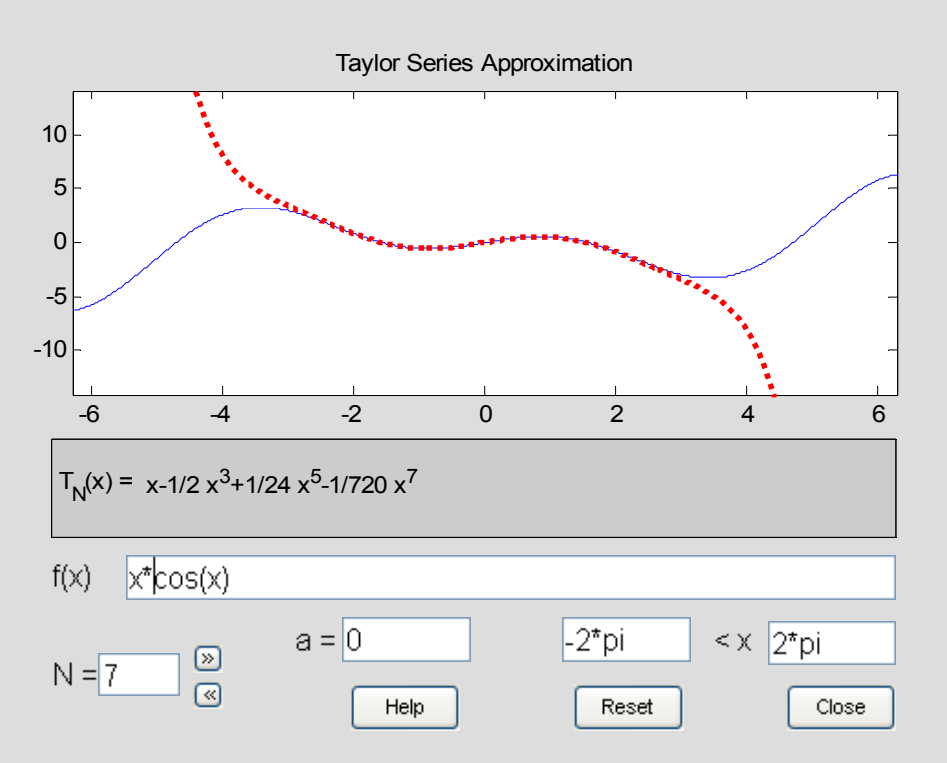
Example 11.2: Let $f(x) = \sin x$. Find the Taylor polynomial $p_9(x)$ with center $c = 0$ (i.e. Maclaurin series) and compare the values of $p_9(\pi/2)$ and $\sin(\pi/2) = 1$.

Solution: The `taylor` command computes Taylor polynomials for a given function. We convert the output to a polynomial function using the method described in Example 6.6.

```
syms x
tpoly=taylor(sin(x),10)  % defines taylor polynomial expression
p=@(x)subs(tpoly);  % taylor polynomial function
approx_error= abs(p(pi)-sin(pi)) % absolute difference in approximation

tpoly =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9
approx_error =
    0.0069
```

■

Although it is a straightforward matter to modify the M-file `powseries.m` to have it plot a function and its Taylor polynomial, we can save ourselves some work by using a built-in Matlab M-file for that purpose. This is the file `taylortool`. When entered from the command line, this brings up an interactive dialog box as shown below:



The user can change the function and various other settings and observe the effect on convergence.

11.3 Summary

Power series can be easily defined through the Symbolic Toolbox using the `symsum` command. The `taylor` command generates Taylor polynomials.

The following commands were introduced in this chapter:

`taylor`: - generate taylor polynomial for a symbolic expression. See help page for options to specify center and degree of expansion.

`num2str`: - convert a number to a string. This is useful in constructing informative titles and legends that contain a reference to a numerical parameter. Usually this is concatenated with other text using [text1, text2, …] to produce a meaningful annotation.

11.4 Exercises

Exercise 11.1 Find the interval of convergence for each of the following series. (Use the usual methods to do so.) Modify the file `powseries.m` so as to generate two plots: the first plot showing the series and an appropriate partial sum, and the second, in a separate window, showing the absolute difference (error) between the two functions.

Select your partial sum so it provides a reasonable approximation to the series throughout most of the interval of convergence. Publish your result. Make sure your graphs have clear titles that explain their content.

a) $\sum_{k=1}^{\infty}(-1)^k \frac{x^k}{k^2}$

b) $\sum_{k=0}^{\infty}\frac{3^k x^k}{2k+1}$

Exercise 11.2 The file `powseries.m` only works for power series for which Matlab can find an explicit sum. Modify the file so that instead of drawing the graph of the entire power series and the *n*th partial sum, it draws the graphs of the *n*th and (2*n*)th partial sums over the convergence interval. Don't forget to modify the legend and title appropriately.

In each of the following examples, determine the interval of convergence of the series in the usual way. Then use your modified M-file to plot a pair of partial sums that exhibit the convergence of the partial sums to a limit.

a) $\sum_{k=1}^{\infty}(-1)^k \frac{x^k}{\sqrt{k}}$

b) $\sum_{k=0}^{\infty}\frac{x^k}{4+k^2}$

Exercise 11.3 Modify the M-file `powseries.m` so that it displays the graph of the function $f(x) = x\sin x$ over the interval $[-2\pi, 2\pi]$ and the graph of the Taylor polynomial with center $c = 0$ and degree 8. Use an appropriate title and legend. Organize the file so that the data on the function, plotting interval, center of the

expansion, degree of the approximation appear in a clearly marked "User Entry Section" near the beginning. Publish your file and its output to html.

Exercise 11.4 The Lagrange error estimate says that

$$\left|f(x)-p_n(x)\right| \le \frac{\left|f^{(n+1)}(\theta)\right|}{(n+1)!}\left|x-c\right|^{n+1} \le \frac{M_{n+1}}{(n+1)!}\left|x-c\right|^{n+1}$$

where, $p_n(x)$ is the degree *n* Taylor polynomial at with center *c*, θ is some number between *x* and *c*, and M_{n+1} is the maximum absolute value of the $n+1$st derivative on the interval containing *x* and *c*. We can use Matlab to estimate this error estimate and compare it to the actual error.

In each of the following, make a plot of the appropriate derivative function to estimate its maximum absolute value on the stated interval. From this result determine an upper bound for the Lagrange error estimate and compare this upper bound with the absolute difference between $f(x)$ and $p_n(x)$ found by plotting $\left|f(x)-p_n(x)\right|$. Publish each example in a separate M-file.

a) $f(x)=\sin x$, $c=0$, $n=3$, interval: [−.5,.5]

b) $f(x)=\tan x$, $c=0$, $n=5$, interval: [−.5,.5]

c) $f(x)=\sin x$, $c=\pi/4$, $n=5$, interval: [40°, 50°] (N.B. You will need to convert to radians)

d) $f(x)=\ln(1+x)$, $c=0$, $n=7$, interval: [0,.5]

References

Electronic:

a) Matlab Help – Getting Started – a thorough introduction to all aspects of Matlab

b) www.mathworks.com/access/helpdesk/help/techdoc/matlab.html This is the online version of the above documentation, in case you do not have access to Matlab on your computer.

c) www.utexas.edu/its/rc/tutorials/matlab/ This is a comprehensive online tutorial from the University of Texas.

Books:

a) *A Guide to MATLAB, 2e: for Beginners and Experienced Users*, by Brian Hunt, et. al., Cambridge University Press, 2006

b) *Getting Started with Matlab 7: A Quick Introduction for Scientists and Engineers*, by Rudra Pratap, Oxford University Press, 2006

c) *MATLAB Guide,* 2e, by D. Higham & N. Higham, SIAM, 2006

d) *MATLAB Primer,* 7e, by T. Davis and K. Sigmon, CRC Press, 2005

Command Index for *Notes on Matlab*

e column labeled "Page" refers to the page on which the command is first introduced (occasionally a second page is listed w
nificant new information occurs on the second page). When two related commands are listed, the page references for each
arated by a /. We have left room in the comments section for the reader to write his or her own marginal notes on each comma
s makes for a useful review guide in preparing for the final. Entries in **boldface** mark essential commands whose use must be w
erstood.

egory	Page	Comments	Category	Page	Comments
nbolic Math			**Graphics**		
'	2.4		**axis**	2.4, 3.9	
	1.5		**clabel**	5.3	
ble	9.1		**clf**	1.5	
lot;ezplot3	1.1/4.1		contour; contour3	5.1/5.10	
urf	4.2		ellipsoid	7.8	
	9.1		FaceAlpha	7.4	
	10.8		FaceColor	6.8	
ty	6.6		gradient	8.6	
s	6.5		grid on/off	2.3	
ns; sym	1.1/11.4		**hold on/off**	1.3	
nsum	10.9		**legend**	2.4	
or	11.3		**Line specs**	2.3, 3.9	
ortool	11.4		LineStyle	2.4	
			LineWidth	2.4	
merical Math			MarkerSize	6.8	
anon. ction)	**6.1**		**meshgrid**	4.7	
on(:)	**3.10**		**plot; plot3**	3.2/4.4	
nsum	**10.5**		**quiver; quiver3**	8.1/8.3	
t" operator	**3.5**		semilogx; semilogy	10.2	
pace	**3.5**		**set**	2.4	

merical Math		**Comments**	**Graphics**		**Comments**
d; dblquad	**9.2/9.6**		shading	8.4	
dvec	**9.8**		sphere	7.5	
exp, log, log10, cos, tan, pi	**various**		subplot	4.3	
n	9.4, 9.6		**surf**; surfc	4.8/5.6	
torize	6.4		**title**	2.4	
			view	4.2	
c mmands			**xlabel; ylabel; zlabel**	3.7 (4.5)	
r	10.4		xlim; ylim	6.9	
.end	10.4		Xtick, Ytick,	3.11	
t	2.6		Xticklabel, Yticklabel	3.11	
2str	11.2				
se	4.11				
comment)	2.1				
icolon (;)	2.3				