Maple for Math Majors

Roger Kraft
Department of Mathematics, Computer Science, and Statistics
Purdue University Calumet
roger@calumet.purdue.edu

7. Graphs of functions and equations

7.1. Introduction

Mathematics uses a lot of different kinds of graphs and Maple has a lot of commands for drawing graphs. This worksheet helps you to understand these different kinds of graphs. This understanding will then help you to keep straight and to use all the Maple graphing commands and it will also help you to better understand many ideas that are taught in calculus.

In the next section we describe nine kinds of graphs commonly used in a calculus course and seven Maple commands that are used to draw these graphs. Each of the rest of the sections in this worksheet gives examples and exercises that provide you with the details of working with a particular kind of graph and its associated Maple command.

[>

7.2. A review of graphs

Maple has a lot of graphing commands. This is because Maple can draw a lot of different kinds of graphs. To understand Maple's graphing abilities it helps to have a way of classifying graphs so that we can organize Maple's graphing commands according to what kind of graph they draw.

There are several ways of classifying graphs. For example, there are two dimensional graphs vs. three dimensional graphs, or graphs of equations vs. graphs of functions. In addition, graphs of functions can also be classified by how we draw the graph. For example we can graph inputs vs. outputs, we can graph just the outputs, or we can graph a vector field. We will use all of these classifications to help us understand graphs in general. From Maple's point of view though, the main way of classifying graphs is as either two dimensional or three dimensional. Here is a list of the principle kinds of two and three dimensional graphs that Maple can draw.

[>

In two dimensions, Maple can draw the following kinds of graphs:

1) The graph of a real valued function of one real variable.
2) The graph of a parametric curve in the plane
   (that is, the graph of a 2-dimensional vector valued function of one real variable).
3) The graph of a vector field in the plane
   (that is, the graph of a 2-dimensional vector valued function of two real variables).

4) The graph of an equation in two real variables.

In three dimensions, Maple can draw the following kinds of graphs:

5) The graph of a real valued function of two real variables.
6) The graph of a parametric curve in 3-dimensional space
(that is, the graph of a 3-dimensional vector valued function of one real variable).
7) The graph of a parametric surface in 3-dimensional space
(that is, the graph of a 3-dimensional vector valued function of two real variables).
8) The graph of a vector field in 3-dimensional space
(that is, the graph of a 3-dimensional vector valued function of three real variables).
9) The graph of an equation in three real variables.

Notice that the list includes two kinds of equations and seven kinds of functions. Before going into the details of drawing these kinds of graphs with Maple, we should look at an example of each one of them. But before doing even that, let us go in to more detail about how we are going to classify different kinds of graphs

[**>**

We will begin our classification of graphs by reviewing exactly what we mean by the "graph of an equation" and the "graph of a function". Then, by looking carefully at the definition of a graph of a function, we will see that functions can be classified in a certain way. Maple uses this classification of functions to organize its graphing commands, so understanding Maple's graphing commands boils down to understanding how we can classify functions. This classification of functions can also help you to understand other ideas in mathematics, for example, all the different kinds of derivatives that you learned about in calculus (e.g., ordinary, partial, tangent vector, gradient vector, Jacobian).

Recall that an equation is made up of an equal sign with an expression on either side of it. Equations ask a question; does the left hand side have the same value as the right hand side? If an equation has unassigned variables in it, then we can ask which values of the unassigned variables make the equation true (that is, what values of the variables solve the equation). The **graph of an equation** is a plot of those values for the unassigned variables that make the equation true. If the equation is an equation in one variable, then its graph is made up of points in the real line (which is not visually interesting, and Maple does not even have a command to "draw" such a graph). If the equation is an equation in two variables, then its graph is made up of points in the plane and the graph is usually, but not always, a curve in the plane. If the equation is an equation in three variables, then its graph is made up of points in 3-dimensional space and the graph is usually, but not always, a two dimensional surface. If the equation has four (or more) variables, then its graph is made up of points in four (or more) dimensional "space"; this is something we cannot easily visualize and Maple cannot draw (at least not directly), but it is important to realize that the graph does in fact exist.

[**>**

Recall that a function is made of three things, a set of inputs (domain), a set of outputs (codomain), and a rule for associating one output to each of the inputs. The **graph of a function** is a visualization of the relationship between the inputs and outputs of the function. Since a function is

made of three things, so is its graph. The graph has to have a visualization of the input set, a visualization of the output set, and a visualization of the rule. If a function has an m-dimensional input and an n-dimensional output, then the graph of the function is a plot in $(m + n)$-dimensional space with the m-dimensional input axes perpendicular to the n-dimensional output axes and a point plotted for every combination of an input with its output. This is a bit cumbersome to describe in words. The most important point to remember is that the graph of a function is drawn in a space whose dimension is the sum of the dimensions of the inputs and the outputs. We sometimes refer to this as an **input vs. output** kind of graph.

For the purpose of drawing graphs with Maple, we will consider three kinds of sets of inputs and outputs. An input or output set will be either the 1-dimensional real line, the 2-dimensional plane, or 3-dimensional space. That gives us nine kinds of functions to worry about, i.e., all possible combinations of the three kinds of input sets with the three kinds of output sets. But for some of these kinds of functions, trying to draw its graph creates a problem. If m and n are the dimensions of the inputs and the outputs and $m + n$ is more than three, then how do we draw the graph? This problem comes up for six of the nine possible kinds of functions we have to worry about (which ones are they?). We solve this problem by considering two other kinds of graphs for a function besides the graph which plots inputs vs. outputs.

One of these other kinds of graphs, a **parametric graph**, visualizes a function by plotting only the outputs of the function. This type of graph will be used for three kinds of functions, functions from the line to the plane, functions from the line to space, and functions from the plane to space. Notice that for two of these kinds of functions, an input vs. output kind of graph is not possible (why?). But for functions from the line to the plane, while an input vs. output graph is possible, it turns out that a parametric graph has traditionally been more useful.

The other kind of graph that we will consider is a vector field. A **vector field** is kind of an input vs. output graph, but the input axes are not shown perpendicular to the output axes. Instead the inputs and the outputs are draw on the same set of axes. This only works when the function has the same number of inputs as outputs. So we draw vector fields for functions from the plane to the plane (2-dimensional vector fields) and for functions from space to space (3-dimensional vector fields). (It is possible to draw a 1-dimensional vector field, but Maple does not have a command for this.)

If you have been following along very carefully, you know that there are still two kinds of functions that we have not considered. Functions from space to the line (real valued functions of three real variables) and from space to the plane (2-dimensional vector valued functions of three real variables) do not have any practical way of being visualized and Maple does not have any commands for visualizing them, so we will not consider these two cases any further.

So we end up with seven kinds of functions to graph. Two of these kinds of functions will have input vs. output graphs and will require two different Maple commands, **`plot`** and **`plot3d`**. Three kinds of functions will have parametric graphs and will require three different Maple commands,

plot, **spacecurve**, and **plot3d**. And two kinds of functions will be graphed as vector fields which will require two Maple commands, **fieldplot** and **fieldplot3d**. If we add in the two kinds of graphs of equations, equations in two variables and equations in three variables, which require two more Maple commands, **implicitplot** and **implicitplot3d**, then we have a total of nine kinds of graphs and Maple has seven different commands for drawing these nine kinds of graphs!

Now that we have briefly reviewed the idea of graphing functions and equations, let us look at an example of each of the nine kinds of graphs that we have mentioned. Each kind of graph will turn out to have its own Maple syntax. Without the classification we made above, all these different commands and syntaxes can become pretty confusing. Hopefully, understanding the kind of graph that we want to draw will help us to remember the kind of command to use.

First the four kinds of two dimensional graphs. For each of these graphs, be sure to click on the graph and then play with some of the buttons from the graphics context bar.

1) The graph of a real valued function of one real variable. These are the functions that you are most familiar with from the first two semesters of calculus. These are also the most common example of an input vs output kind of graph.

```
> plot( x+2*sin(x), x=-10..10 );
>
```

2) The graph of a parametric curve in the plane (that is, the graph of a 2-dimensional vector valued function of one real variable). These come up in second semester calculus and also in physics. They represent the path of motion of something moving in the plane. This is an example of where the parametric graph (output only graph) is more useful than the input-output graph of the function (which could be drawn; why?).

```
> plot( [t*sin(t), t*cos(t), t=0..3*Pi] );
```

Notice that the two formulas (why are there two?) are put in a list along with their range. It is the fact that the range is inside the list that makes this a parametric graph and not a graph of two functions. Try moving the range outside of the list and see what happens (do not forget to put a comma between the list and the range).

```
>
```

3) The graph of a vector field in the plane (that is, the graph of a 2-dimensional vector valued function of two real variables). These functions come up in a differential equations course and also in a vector calculus course. In those settings a 2-dimensional vector valued function of two variables is visualized as a vector field. Maple has a special command for graphing a vector field. Notice that a vector field is neither an input-output nor a parametric type of graph.

```
> plots[fieldplot]( [y/sqrt(x^2+y^2), x/sqrt(x^2+y^2)], x=-5..5,
  y=-5..5 );
>
```

4) The graph of an equation in two real variables. The simplest examples of these are the equations for the conic sections (i.e., hyperbola, parabola, ellipse). Here is a more complicated example of an equation and it graph.

```
> plots[implicitplot]( x^3+y^3-5*x*y=1/8, x=-3..3, y=-3..3 );
>
```

In three dimensions, Maple can draw the following five kinds of graphs: For each of these graphs, be sure to click on the graph and try rotating it. Also, play with all the buttons on the graphics context bar.

5) The graph of a real valued function of two real variables. These are the functions that are first studied in third semester calculus. These are also input vs. output graphs.

```
> plot3d( x^2+y^2, x=-2..2, y=-2..2 );
>
```

6) The graph of a parametric curve in 3-dimensional space (that is, the graph of a 3-dimensional vector valued function of one real variable). These come up in third semester calculus and also in physics. They represent the path of motion of something moving in space. This is another example of a function where the parametric graph is more useful than the graph of the function (the graph of the function would need four dimensions anyway).

```
> plots[spacecurve]( [t*sin(t), t*cos(t), t], t=0..9*Pi );
>
```

7) The graph of a parametric surface in 3-dimensional space (that is, the graph of a 3-dimensional vector valued function of two real variables). These come up at the end of third semester calculus and in vector calculus courses. (How many dimensions would the input-output graph of one of these functions need?)

```
> plot3d( [(2+sin(v))*cos(u), (2+sin(v))*sin(u), u+cos(v)],
>          u=-2*Pi..2*Pi, v=-2*Pi..2*Pi );
```

Try changing the brackets (**[** and **]**) to braces (**{** and **}**). Can you explain the resulting graph?

```
>
```

8) The graph of a vector field in 3-dimensional space (that is, the graph of a 3-dimensional vector valued function of three real variables). These functions come up in differential equations and vector calculus courses. In those settings a 3-dimensional vector valued function of three variables is visualized as a vector field. Maple has a special command for graphing a vector field in 3-dimensional space. Notice again that a vector field is neither an input-output nor a parametric type of graph.

```
> plots[fieldplot3d]( [2*x,2*y,1], x=-1..1, y=-1..1, z=-1..1,
>                      grid=[5,5,5] );
>
```

9) The graph of an equation in three real variables. The simplest examples of these are the quadric

surfaces that are studied in third semester calculus. But here is a far more complicated example.

```
> plots[implicitplot3d]( x^3+y^3+z^3+1 = (x+y+z+1)^3,
>                        x=-2..2, y=-2..2, z=-2..2 );
>
```

Finally, notice two things. First, the **plot** and **plot3d** command each handle two kinds of graphs (which are they?). That is why there are seven Maple commands for drawing nine kinds of graphs. Second, the other five of these seven Maple commands are contained in the **plots** package. We could load the **plots** package at this time and save ourselves some typing later in this worksheet, but we will not do that. Instead, we will always use the long name for these commands. This has two advantages. First, it helps to remind us that these commands are from the **plots** package and second, all the commands in this worksheet will always work, regardless of where in the worksheet you might jump in and start working from.

```
>
```

Exercise: Part (a) For each of the seven kinds of functions described above, describe what kind of derivative the type of function has. Give your answers in terms of ideas from calculus, not in terms of Maple.

Part (b) For each of the seven kinds of functions described above, explain how you might integrate a function of each type. Again, give your answers in terms of ideas from calculus, not in terms of Maple.

Part (c) What about the two kinds of equations? Can you differentiate an equation? Can you integrate an equation?

```
>
```

```
>
```

7.3. Graphs of functions of one variable

Maple's most basic graphing command is **plot**. This command draws graphs of real valued functions of one real variable. We graph such functions by giving **plot** the function and a range for the independent variable. Here is an example.

```
> plot(sin(x)/x, x=-5*Pi..5*Pi);
```

When **plot** draws the graph, it gives the horizontal axis the range we specified in the range for the independent variable. For the vertical axis, **plot** decides for itself what is an appropriate range. Usually the vertical range is from the minimum to the maximum of the function (as in the above graph). This can sometimes cause a graph to be a bit strange. For example, the next graph seems to be showing a function that has the horizontal axis as a horizontal asymptote. But look very carefully at the vertical axis. The vertical range is from 2 to 3. The piece of the vertical axis from 0 to 2 is missing!

```
> plot(2+exp(-x^2), x=-10..10);
```

If we want to, we can give **plot** a range for the dependent variable, instead of letting **plot** determine the vertical range itself. The next command fixes the ambiguity of the last graph by specifying a vertical range. Notice that we now see very clearly that the horizontal axis is not the horizontal asymptote for this function.

```
> plot(2+exp(-x^2), x=-10..10, 0..3);
>
```

Using a range for the dependent variable is especially useful when graphing a function that has a vertical asymptote. In the next graph, **plot** tries by default to accommodate in the graph the full range of the dependent variable. But this range is infinite, because of the vertical asymptote at $x = 1$. So the graph is not very useful.

```
> plot(1/(x-1), x=-1..3);
```

We can fix the graph and get a useful representation of the function by providing a range for the dependent variable.

```
> plot(1/(x-1), x=-1..3, -20..20);
```

Notice the vertical red line at $x = 1$. This is *not* Maple's way of drawing the vertical asymptote. This vertical line is an artifact of the way the **plot** command works. We will explain in the next worksheet why this line is there. We can make it go away by using the **plot** option **discont=true**.

```
> plot(1/(x-1), x=-1..3, -20..20, discont=true, color=red);
>
```

When using the **plot** command, it is important to make a distinction between the *range* and the *scale* of an axis. The range of an axis is what we have been discussing in the last few examples. It is the part of the number line displayed on the axis. The scale of an axis is how much actual length is assigned to one unit of the axis. Here is an example.

```
> plot(sin(x)/x, x=-3*Pi..3*Pi);
```

In this last graph, the range of the horizontal axis is from -3π to 3π (about –9.425 to 9.425). The range of the vertical axis is from –0.2 to 1. So the range of the horizontal axis is much more than the range of the vertical axis. On the other hand, the scale of the horizontal axis is much less than the scale of the vertical axis. Each unit on the horizontal axis (say from 0 to 1 on the horizontal axis) is much shorter in length than a unit on the vertical axis. The **plot** command did not use the same scales on the two axes. In general, once the ranges of the two axes have been determined, the **plot** command will choose scales for each axis so that the final graph is "well proportioned". The scales are usually chosen so that the graph is roughly a square. It is possible to force the **plot** command to use the same scale on both of the axes by using the **scaling=constrained** option to **plot**. Here is the last graph redrawn with the same scales on both axes.

```
> plot(sin(x)/x, x=-3*Pi..3*Pi, scaling=constrained);
```

If you compare the last two graphs, the horizontal scale is about the same in both graphs. It was the vertical scale that was modified in the last graph. Sometimes the **scaling=constrained** option helps the appearance of a graph and often it does not. A quick way to see the difference between constrained and unconstrained scaling is to click on a graph and look at the context bar at the top of

the Maple window. You will see a button on the context bar labelled **1:1**. This button switches the constrained scaling on and off. Try it on the last graph.

```
> 
```

When graphing real valued functions of one variable, it is important to realize that for many functions, no one graph can tell us everything we may need to know about the function. So the most important skill to develop when using the **plot** command is modifying the ranges on the horizontal and vertical axes to show different pieces of information about a function. Let us look at some examples and exercises about manipulating the ranges of a graph.

Consider the following rational function and a few of its graphs. Each graph will tell us something different about this function.

```
> f := (1-x^2)/(x-2);
```

The following graph from minus infinity to infinity tells us that the function has a vertical asymptote, two x-intercepts, and that the graph goes to minus infinity as x goes to plus infinity and it goes to plus infinity as x goes to minus infinity.

```
> plot(f, x=-infinity..infinity);
```

Now notice that the next graph tells us a bit more and a bit less. This graph is over a very large domain. It tells us exactly how the graph goes to plus and minus infinity. The graph is skew asymptotic to the line $y = -x$. But this graph obscures the fact that the function has two zeros, and even the vertical asymptote is a bit vague.

```
> plot((1-x^2)/(x-2), x=-100..100, -100..100);
```

The next graph allows us to see where the x-intercepts are and to approximate the local minimum between these intercepts.

```
> plot((1-x^2)/(x-2), x=-1.1..1.1);
```

And the next graph gives us a good approximation of the local maximum to the right of the vertical asymptote.

```
> plot((1-x^2)/(x-2), x=2..8, -10..-7);
```

It would be difficult for any one graph to convey all of the information that the above graphs show. After some trial and error, the following graph shows quite a bit about the function, except for providing good approximations of the x-intercepts and the local maximum and minimum.

```
> plot((1-x^2)/(x-2), x=-10..10, -15..15, discont=true,
  color=red);
```

```
> 
```

Exercise: Consider the following function.

$$\frac{x}{x+1} - \cos\left(\frac{40}{x}\right)$$

Part (a) Try to get a sense of what the graph of this function looks like.

```
> 
```

Part (b) Use a graph to find the most negative and the most positive x-intercepts for this function. (Hint: The positive answer is quite large.)

[>
Part (c) How many x-intercepts does the graph of this function have between 1/10 and 10?
[>
Part (d) Use a graph to find the most negative local maximum and minimums for this function.
[>
Note: The function in this problem comes up in an interesting "real world" calculus problem. See *Calculus with Maple*, by Frank Hagan and Jack Cohen, and the chapter called "The Asteroid Problem".
[>

Exercise: Find ranges for the **plot** command so that the graph of $\cos(x)$ looks like
a) a horizontal line,
[>
b) a vertical line,
[>
c) a 45 degree line.
[>
Then do the same with the graph of $\tan(x)$.
[>

All in all, the **plot** command is not all that hard to use. For the most part, one learns to use it by looking at examples. We gave a lot of examples of its use in the first worksheet. In the rest of this section we look at examples of a few more ways of using **plot** that were not shown in the first worksheet.

When we use the **plot** command to graph several functions at the same time, we cannot give each function its own domain. For example, in the following graph, suppose we only want to graph the part of the parabola that is contained inside the semicircle.

```
[ > plot([x^2, sqrt(1-x^2)], x=-1..1, scaling=constrained);
```

To do this, we need to draw graphs for the parabola and semicircle separately and then combine the two graphs together using the **display** command from the **plots** package. First let us draw the semicircle.

```
[ > plot(sqrt(1-x^2), x=-1..1, color=green);
```

Now let us give this graph a name for later reference.

```
[ > plot1 := %;
```

Notice how, when we gave the graph a name, Maple returned a large amount of data. This data is Maple's internal description of the graph (it's a PLOT data structure). In another worksheet we will say quite a bit about these PLOT data structures but right now we are not very interested in seeing all of this data. So for now, whenever we want to name a graph we will use a colon at the end of the assignment command (instead of a semicolon) to suppress the printing of this data. (A quick way to make all of that output go away is to immediately type Ctrl-Z and then skip down to the next command.)

Now let us solve for the two intersection points of the parabola with the semicircle. First the negative intersection point.

```
> a := fsolve(x^2=sqrt(1-x^2), x, -1..0);
```

And now solve for the positive intersection point.

```
> b := fsolve(x^2=sqrt(1-x^2), x, 0..1);
```

Now graph the parabola between these two intersection points.

```
> plot(x^2, x=a..b, color=blue);
```

Now give this last graph a name (notice the colon at the end of the command).

```
> plot2 := %:
```

Now we can combine our two graphs using the **display** command.

```
> plots[display]([plot1, plot2], scaling=constrained);
>
```

Here is an interesting way to use **plot** together with the **display** command. First, we need to create a special kind of variable, an array.

```
> graphs := array(1..2,1..2);
```

Now let us draw four graphs and use the array variable to name them (recall that we discussed indexed names like these in the worksheet about variables and names).

```
> graphs[1,1] := plot(exp(-x^2)*sin(Pi*x^3), x=-2..2,
  color=blue):
> graphs[1,2] := plot(exp(-x^2), x=-2..2, color=red):
> graphs[2,1] := plot(-exp(-x^2), x=-2..2, color=green):
> graphs[2,2] := plot([exp(-x^2)*sin(Pi*x^3), exp(-x^2),
  -exp(-x^2)],
>                     x=-2..2, color=[blue,red,green]):
```

Now combine all four graphs into a two by two array of pictures.

```
> plots[display](graphs);
```

It is the fact that our four graphs are named by a single array variable that tells **display** to draw them in an array kind of format instead of superimposing all four of them on one set of axes.

```
>
```

So far we have drawn all of our graphs of real valued functions of one variable with the independent variable running along the horizontal axis and the dependent variable along the vertical axis. Of course, this is a pretty common way to draw graphs but it is not the only way. There are times when it is more convenient to draw a graph the other way, with the independent variable along the vertical axis. However, the **plot** command does not make it easy to graph a real valued function of one variable this way. Later in this worksheet, and also in the next worksheet, we will see several ways to graph a function with its independent variable along the vertical axis. The main point that we want to make here though is that the **plot** command gives one of the directions in the cartesian coordinate system a preferred status. The preferred direction is the horizontal direction and its preferred status is that this is the direction of the axis for the independent variable when graphing a

function. We commonly label the horizontal axis using x and the vertical axis using y. Using these labels, the default way for **plot** to graph a function f using cartesian coordinates is as $y = f(x)$ (and **plot** will not graph $x = f(y)$). Now let us consider some other coordinate systems on the plane.

The **plot** command can use several other coordinates systems on the plane when it graphs a real valued function of one variable. For every one of these non-cartesian coordinates systems, there is a preferred "direction" that is (somewhat arbitrarily) used for the independent variable of the function. The most common of these non-cartesian coordinate systems is polar coordinates. Here is how we graph the function $f(x) = \sin(x)$ using polar coordinates in the plane.

```
> plot(sin(x), x=0..Pi, coords=polar, scaling=constrained);
```

Here is a command that will draw a picture of "graph paper" in polar coordinates.

```
> plots[coordplot](polar, scaling=constrained);
```

In this coordinate system, the **plot** command uses the circular "direction" as the preferred direction for the independent variable and the radial "direction" is used for the dependent variable. We commonly label the circular direction with θ (the angle) and we label the radial direction with r (the radius). Using these labels, the default way for **plot** to graph a function using polar coordinates is as $r = f(\theta)$. The labels θ and r may be common for polar coordinates but there is nothing that says that we must use them and in fact the **plot** command in polar coordinates does not in any way prefer these labels. Here is a graph of the function $\cos(2u)$ in polar coordinates. Notice that here the **plot** command is using u as the label for the circular direction.

```
> plot(cos(2*u), u=0..Pi, coords=polar, scaling=constrained);
```

When graphing real valued functions of one variable using polar coordinates, the **plot** command will not graph a function with the radial direction as the independent variable. That is, if we use the common labels for the polar coordinates, **plot** will not graph $\theta = f(r)$, even though there is nothing to prevent us from defining such a graph. (In the next section, and also in the next worksheet, we will see a trick that does draw such a graph.)

```
>
```

Exercise: One of the non-cartesian coordinates systems that **plot** can use on the plane is called hyperbolic coordinates. Here is a picture of some "graph paper" in this coordinate system.

```
> plots[coordplot](hyperbolic, scaling=constrained);
```

Figure out which of the coordinate directions (the blue or the red one) is the preferred direction that is used as the independent variable when **plot** graphs a real valued function of one variable using the hyperbolic coordinate system.

```
>
```

Exercise: Here are examples of "graph paper" for two more coordinate systems.

```
> plots[coordplot](bipolar, scaling=constrained);
> plots[coordplot](logcosh, scaling=constrained);
```

The following help page lists 14 non-cartesian coordinate systems for the plane.

```
> ?plot,coords
```

Draw graph paper for several more of these coordinates systems. Use the online help to read about

the **coordplot** command and try to modify some of these pieces of graph paper. Pick a coordinate systems and try to figure out which of its two coordinates is used as the independent variable by the **plot** command.

```
> 
```

Exercise: The following command uses polar coordinates to graph a circle.

```
> plot(sin(t), t=0..Pi, coords=polar, scaling=constrained);
```

If we change the sin to a cos in the last command, we get the circle rotated 90 degrees clockwise.

```
> plot(cos(t), t=0..Pi, coords=polar, scaling=constrained);
```

The following command uses polar coordinates to graph a curve called a cochleoid.

```
> plot(sin(t)/t, t=-6*Pi..6*Pi, coords=polar,
  scaling=constrained);
```

If we change the sin to a cos in the last command, the graph is not rotated 90 degrees.

```
> plot(cos(t)/t, t=-6*Pi..16*Pi, coords=polar,
  scaling=constrained);
```

Find a way to rotate the cochleoid 90 degrees clockwise.

```
> 
```

```
> 
```

7.4. Graphs of parametric curves

We have two kinds of parametric curves, curves in two dimensional space and curves in three dimensional space. Curves in two dimensional space are drawn using a special case of the **plot** command. Curves in three dimensional space have the special command, **spacecurve**, from the **plots** package.

A curve in two dimensional space is described by two real valued functions, the component functions. Here is a simple example.

```
> plot([cos(t), sin(t), t=0..2*Pi]);
```

This should have given us a circle. The extra parameter in the next example makes the graph into a circle.

```
> plot([cos(t), sin(t), t=0..2*Pi], scaling=constrained);
```

Notice that it is the presence inside the brackets of the range for the independent variable that distinguishes this command from the command to draw the graph of two real valued functions. In the next example, we move the range outside the brackets.

```
> plot([cos(t), sin(t)], t=0..2*Pi, scaling=constrained);
```

Here is how we graph two parametric curves (so we have four real valued functions in two pairs). Notice that each parametric curve has its own range.

```
> plot([ [3*cos(t), 1/2*sin(t), t=0..2*Pi],
>        [cos(t)*sin(3*t), abs(t), t=-Pi..Pi] ],
>       axes=none);
```

```
> 
```

In the first section of this worksheet, we said that a curve in the plane is defined by a single function, a 2-dimensional vector valued function of a single variable. Here is a way to use a Maple function to emphasize that a curve is really defined by a single (vector valued) function. The function $f(t) = (t\cos(t), t\sin(t))$ defines a spiral curve. Here is the Maple definition of this function.

```
> f := t -> (t*cos(t), t*sin(t));
```

Here is how we can use the function **f** with the **plot** command to graph a spiral curve.

```
> plot( [f(t), t=0..4*Pi] );
```

This example shows that a curve is really defined by a single function. The two expressions that we used in each of the previous curves are really the component functions of the single (vector valued) function that defines each curve. It is usually more convenient to work with two expressions than with a single vector valued function, so for the rest of this section we will use expressions to describe parametric curves.

```
>
```

A curve in three dimensional space is described by three real valued (component) functions. Here is an example of a curve in the shape of a trefoil knot. Be sure to click on the graph and use the mouse to rotate it.

```
> plots[spacecurve]([(2+cos(3*t/2))*cos(t),
>                    (2+cos(3*t/2))*sin(t),
>                    sin(3*t/2)], t=0..4*Pi
>                  );
```

With the **spacecurve** command, the range of the independent variable can be either inside or outside of the brackets that enclose the three component functions. In the next example, the range is inside the brackets.

```
> plots[spacecurve]([(2+cos(3*t/2))*cos(t),
>                    (2+cos(3*t/2))*sin(t),
>                    sin(3*t/2), t=0..4*Pi]
>                  );
```

The range for a curve must be inside the brackets if we want to draw more than one curve at a time. (Notice how each curve has its own range which can be different from the other curves.)

```
> plots[spacecurve]({[(2+cos(3*t/2))*cos(t),
>                     (2+cos(3*t/2))*sin(t),
>                     sin(3*t/2), t=0..4*Pi],
>                     [5*cos(t), 5*sin(t), 2*cos(6*t), t=0..2*Pi]}
>                   );
```

Notice how, with the **spacecurve** command, multiple space curves are placed inside of a pair of braces, not brackets like in the **plot** command.

```
>
```

Exercise: Redraw one of the last space curves using a single (vector valued) Maple function to define the curve.

```
>
```

Exercise: Suppose we have a 2-dimensional vector valued function of one real variable and suppose that instead of its parametric graph we want its input-output graph. There is an easy way to draw the input-output graph. What is it?
(Hint: You will need to use the **spacecurve** command.)

```
>
```

Exercise: Try to explain why the following graph has the shape that it does. (Hint: Look at the graphs of the component functions.)

```
> plot([sin(t+sin(t)), cos(t+cos(t)), t=0..2*Pi],
  scaling=constrained);
>
```

Recall that in the previous section we mentioned that the **plot** command, when graphing a function, gives the horizontal axis in cartesian coordinates the preferred status of being the axis for the independent variable. When graphing parametric equations, the **plot** command does not give either direction a preferred status. It does however always treat the first expression after the opening bracket as the horizontal component and the second expression as the vertical component. We can use this to get a graph of a real valued function of one variable with the vertical axis as the axis for the independent variable. That is, using the common labels x and y for the horizontal and vertical axes, we can use a parametric curve to draw a graph of $x = f(y)$. For example, here is how we can graph $x = \sin(y)$ (as a parametric curve).

```
> plot([sin(y), y, y=0..2*Pi], scaling=constrained);
```

Here is the analogous way to graph $y = \sin(x)$ as a parametric curve.

```
> plot([x, sin(x), x=0..2*Pi], scaling=constrained);
```

Here is a graph of a quadratic function with the independent variable along the vertical axis (but notice that we are using the label x for the vertical axis here).

```
> plot([3*x^2+5*x-4, x, x=-5..3]);
>
```

Exercise: Use parametric curves to graph the function $x = y^3 - 5y - 1$ and its tangent line at the point $y = 1$.

```
>
```

After you have a good graph of the function and its tangent line, try zooming in on the point of tangency.

```
>
```

The **plot** command can also draw parametric curves in non-cartesian coordinate systems. For each coordinate system, the **plot** command has to make an arbitrary choice of which "direction" of the coordinate system has to come first after the opening bracket. In the case of the polar coordinate system, the first expression after the opening bracket is the radial coordinate and the second

expression is the angle. Here is an example showing this. We graph $r = \cos(2\,\theta)$ using parametric equations (compare this with the graph from the last section of the same function).

```
> plot([cos(2*t), t, t=0..Pi], coords=polar,
  scaling=constrained);
```

Notice a subtle difference in how **plot** handles the cartesian and polar coordinate systems. In the cartesian coordinate system the order of the parametric expressions inside the brackets is independent variable then dependent variable. For the polar coordinate system it is dependent variable then independent variable (where, by independent and dependent variable, we mean with respect to how **plot** treats them when graphing a function instead of parametric equations).

```
>
```

Exercise: In the last **plot** command, reproduced at the next prompt, what graph would you expect to get if you removed the option **coords=polar** from the command? Does the graph change from $r = \cos(2\,\theta)$ to $y = \cos(2\,x)$?

```
> plot([cos(2*t), t, t=0..Pi], coords=polar,
  scaling=constrained);
```

```
>
```

Exercise: The **spacecurve** command can graph parametric curves using cylindrical and spherical coordinates systems (among many others). For each of these two coordinate systems, figure out what the order of the parametric expressions is inside the brackets.

```
>
```

Knowing how **plot** handles parametric equations in polar coordinates, we can now draw a graph in polar coordinates of a function of the form $\theta = f(r)$. Here is a graph of $\theta = \sin(r)$.

```
> plot([r, sin(r), r=0..2*Pi], coords=polar);
```

```
>
```

Exercise: Study the last graph carefully. Explain why it has the shape that it does. Explain what the following graph is demonstrating.

```
> plot([ [r,sin(r),r=0..2*Pi], [t,1,t=0..3], [t,-1,t=0..6] ],
>        coords=polar);
```

```
>
```

Exercise: Use polar coordinates to draw a graph of a wedge from a circle.

```
>
```

At this point it is worth emphasizing how versatile parametric curves are. Notice in the last several examples how many kinds of graphs we have been able to draw using parametric equations. Besides drawing curves that are not the graph of any function, we have also used parametric curves to draw all four of the kinds of graphs that can be made from a function using cartesian and polar coordinates. For example, the following four commands use parametric equations to draw graphs of

$y = f(x)$, $x = f(y)$, $r = f(\theta)$, and $\theta = f(r)$ respectively, where f is the squaring function.

```
> plot( [x, x^2, x=-2..2] );
> plot( [y^2, y, y=-2..2] );
> plot( [theta^2, theta, theta=-2..2], coords=polar );
> plot( [r, r^2, r=-2..2], coords=polar );
```

Using parametric equations to graph functions is an important and useful technique. Make sure you understand exactly how the last four examples work. We will return to this idea of using parametric equations to graph functions in the section about parametric surfaces. In that section we will see that this technique lets us work around a couple of bugs in Maple.

```
>
```

Exercise: Try extending the range of the parameter for each of the last two polar graphs. Study these two graphs until they make sense to you.

```
>
```

We end this section with several subsections. Each subsection explores some interesting use of parametric equations.

7.4.1. Animating curves

A fun way to practice working with parametric curves is to work with animations. It is fairly easy to create interesting animations out of parametric curves. The next several exercises and examples use animations to both give you an idea of what can be done and to also get you to think some more about parametric curves.

Exercise: The following animation (from the first worksheet) uses parametric equations in cartesian coordinates. Convert the animation to use the graph in polar coordinates of a real valued function of a single variable.

```
> plots[animate]([(1+sin(t)*.5*cos(5*s))*cos(s),
>                 (1+sin(t)*.5*cos(5*s))*sin(s), s=0..2*Pi],
>                 t=0..2*Pi, scaling=constrained,
  numpoints=100,
>                 color=blue, axes=none, frames=50);
```

(Recall that this parametric curve defines a "circle" whose radius (given by the term `1+sin(t)*.5*cos(5*s)`) changes both with angle (the `s` variable) and with time (the `t` variable)).

```
>
```

Exercise: The next animation (also from the first worksheet) is based on the previous one. Some of the parameters have been changed and two circles are being morphed simultaneously. Can you convert this animation into one that uses graphs in polar coordinates of two real valued functions of a single variable?

```
> plots[animate]({[(1+2*sin(t)*cos(6*s))*cos(s),
>                  (1+2*sin(t)*cos(6*s))*sin(s), s=0..2*Pi],
```

```
> [(1+2*sin(t)*cos(6*s))*cos(s),
>  (1-2*sin(t)*cos(6*s))*sin(s), s=0..2*Pi]},
> t=0..2*Pi, scaling=constrained,
  numpoints=150,
> color=blue, axes=none, frames=100 );
>
```

Exercise: Create an animation of a line segment of length 2 π rolling itself up into a circle of radius one.

```
>
```

Maple has a special command in the **plots** package for animating curves in the plane, **animatecurve**. This command animates, in a sense, the drawing of a curve. Here is a simple example. (When you execute this command, it seems to produce an empty graph. Click on the graph and you will get an animation context bar at the top of the Maple window. Click on the play button to start the animation.)

```
> plots[animatecurve](sin(x), x=0..2*Pi);
```

Let us see how we can use this command to create an informative animation of a parametric curve.

```
> curves := array(1..3):
> curves[1] := plots[animatecurve](cos(x), x=0..2*Pi,
  frames=50):
> curves[2] := plots[animatecurve](sin(x), x=0..2*Pi,
  frames=50):
> curves[3] := plots[animatecurve]([cos(x), sin(x), x=0..2*Pi],
>                                  frames=50, color=blue):
> plots[display](curves);
```

The two graphs on the left are the component functions of the parametric curve on the right. It may be easier to follow the combined animations if you slow them down or even "single step" through the frames (using the buttons on the context bar). The animation looks better if you click on the graph and then use the mouse to enlarge it as much as possible (using the corners of the graph, much like you would enlarge any other window). Unfortunately, if you use either the context bar or the context menu to make the graphs have constrained scaling, then the graphs shrink in size by quite a bit. You can still stretch them out (a lot) to make their size reasonable again.

```
>
```

The **animatecurve** command does not work with curves in three dimensional space. And the **animate3d** command only works with surfaces in three dimensions so it also cannot animate a curve in space. So how can we create an animation, like that of **animatecurve**, for a parametric curve in space?

Let us create an animation of the parameterization of the trefoil knot

$$x = \left(2 + \cos\left(\frac{3t}{2}\right)\right)\cos(t),\ y = \left(2 + \cos\left(\frac{3t}{2}\right)\right)\sin(t),\quad z = \sin\left(\frac{3t}{2}\right).$$

We will do the animation two ways. The first way will have one end of the parameterization fixed and the other end of the parameterization sweeping over the curve until it gets back to the fixed starting point and closes the curve. The second animation will have the two endpoints of the parameterization moving away from the fixed starting point and sweeping out the curve until the two moving points meet and close the curve at a point "opposite" to the fixed starting point.

First, define a graph valued function that we will call **frames**. For any value of this function's input variable, the function returns a graph of a space curve.

```
> frames := s -> plots[spacecurve]([(2+cos(3*t/2))*cos(t),
>                                   (2+cos(3*t/2))*sin(t),
>                                   sin(3*t/2), t=0..s]);
```

Here is an example of evaluating the function **frames**. This evaluation draws half of the trefoil knot.

```
> frames(2*Pi);
```

Now use the **seq** command and the **frames** function to create a sequence of 50 graphs of pieces of the trefoil knot. The **seq** command will evaluate the **frames** function 50 times. Each call of the **frames** function will increment the input of **frames** a little bit and draw a slightly longer piece of the trefoil knot. (Notice the colon at the end of this command so that we do not see the huge amount of data that it creates.)

```
> seq( frames(4*Pi*i/50), i=1..50):
```

Now use the **display3d** command with the option **insequence=true** to create an animation out of the 50 frames that we just computed. (Try rotating the animation as it is running to see it from different angles. You can slow it down if you wish, or set it to loop continually.)

```
> plots[display3d]( [%], insequence=true);
>
```

Exercise: Copy the three commands that create the animation into a single execution group at the end of this exercise. Then modify the example so that the new animation will have the two endpoints of the parameterization moving away from a fixed point and sweeping out the curve until the two moving points meet and close the curve at a point "opposite" to the fixed starting point. (This animation will emphasize the symmetry of the curve.)

```
>
```

Exercise: There is a bug in the **animatecurve** command. Let us look at an example that brings out this bug. Here is a graph of a function.

```
> plot(sin(4*x), x=0..2*Pi);
```

Let us animate this last graph.

```
> plots[animatecurve](sin(4*x), x=0..2*Pi);
```

Now let us convert the graph of the function from cartesian to polar coordinates.

```
> plot(sin(4*x), x=0..2*Pi, coords=polar);
```

Now let us convert the animation from cartesian to polar coordinates (which should animate this last graph).

```
> plots[animatecurve](sin(4*x), x=0..2*Pi, coords=polar);
```

What went wrong? What did **animatecurve** do? Find a way to use **animatecurve** to animate the graph in polar coordinates.

```
>
```

```
>
```

7.4.2. Parametric squares

There are two common ways to draw a graph of the circle of radius one (and diameter two) centered at the origin. We can either graph the equation

$$x^2 + y^2 = 1$$

or we can graph the parametric curve

$$x = \cos(2\,\pi\,t),\ \ y = \sin(2\,\pi\,t),\ \ 0 \le t \text{ and } t \le 1.$$

Here are the Maple commands for doing both of these.

```
> plots[implicitplot](x^2+y^2=1, x=-1..1, y=-1..1,
  scaling=constrained);
```

```
> plot( [cos(2*Pi*t), sin(2*Pi*t), t=0..1],
  scaling=constrained);
```

```
>
```

There is an interesting equation whose graph is a square centered at the origin with sides of length two. The equation is

$$\max(|x|, |y|) = 1.$$

Here is its graph.

```
> plots[implicitplot](max(abs(x),abs(y))=1, x=-1..1, y=-1..1,
>                     scaling=constrained);
```

(Notice that the graph has two "clipped" corners. We will see in the next worksheet what causes this anomaly.)

```
>
```

Our goal in this section is to find a parameterization of this curve that is analogous to the parameterization of the circle, and then show an interesting connection between the equations $x^2 + y^2 = 1$ and $\max(|x|, |y|) = 1$.

The basic idea behind parameterizing the square is that we need two functions that "act like" $\cos(2\,\pi\,t)$ and $\sin(2\,\pi\,t)$ but parameterize the square instead of the circle. Recall that $\cos(2\,\pi\,t)$ provides the horizontal component of the motion in the parameterization of the circle, and $\sin(2\,\pi\,t)$ provides the vertical component of the motion around the circle. Now think about the kind of motion that we would want a parameterization of the square to describe. Start the

parameterization at the point (1,0) (just like for the circle) and assume that the parameterization will take one unit of time (just like for the circle). The horizontal component of the motion around the square should sit still at 1 for a while (how long?), then sweep linearly down from 1 to −1 (how long should this take?), then sit still at −1 for a while (again, how long?), then sweep linearly up from −1 to 1, and finally, sit still at 1 again for a while. Meanwhile, the vertical component of the motion around the square starts out sweeping linearly up from 0 to 1, then sitting still at 1 for a while, then sweeping linearly down from 1 to −1, then holding still at −1 for a while, and finally sweeping linearly up from −1 to 0.

Here is a piecewise defined function that implements the horizontal component that we need to parameterize the square.

```
> sq := t -> piecewise(frac(t) < 1/8, 1,
>                      frac(t) < 3/8, -8*frac(t)+2,
>                      frac(t) < 5/8, -1,
>                      frac(t) < 7/8, 8*frac(t)-6,
>                      frac(t) < 1,   1);
```

Here is a graph of **sq** compared to a graph of $\cos(2\pi t)$.

```
> plot([sq(t), cos(2*Pi*t)], t=0..1);
```

Now what about the vertical component? Notice that in the parameterization of the circle, the horizontal motion is really the same as the vertical motion, they are just 1/4 of a time unit "out of phase" with each other. To put it another way, $\sin(2\pi t)$ is just $\cos(2\pi t)$ shifted by 1/4, or $\sin(2\pi t) = \cos(2\pi (t - .25))$. So let us just do the same thing to define our vertical component. Here is a parametric curve defined by **sq(t)** and **sq(t-1/4)**.

```
> plot([sq(t), sq(t-1/4), t=0..1], scaling=constrained);
```

Almost, but not quite right. Let us look at the graphs of our component functions.

```
> plot([sq(t), sq(t-1/4)], t=0..1);
```

We see here that there is something wrong with the first part of **sq(t-1/4)**, the part between **0** and **1/8**. This part of **sq(t-1/4)** is defined by **sq(t)** with **t** between **-1/8** and **0**. Let us graph **sq** from **-1** to **1**.

```
> plot(sq, -1..1);
```

Now we can see what is wrong. We failed to make **sq** an even function and define it for negative numbers. Here is a revised definition of **sq** that fixes this problem.

```
> sq := t -> piecewise(frac(abs(t)) < 1/8, 1,
>                      frac(abs(t)) < 3/8, -8*frac(abs(t))+2,
>                      frac(abs(t)) < 5/8, -1,
>                      frac(abs(t)) < 7/8, 8*frac(abs(t))-6,
>                      frac(abs(t)) < 1,   1);
```

Now graph our component functions again.

```
> plot([sq(t), sq(t-1/4)], t=-1..1);
```

Here is what they look like compared to the components of the parameterization of the circle.

```
> plot([sq(t), cos(2*Pi*t)], t=-1..1);
> plot([sq(t-1/4), cos(2*Pi*(t-1/4))], t=-1..1);
```

```
>
```

Now use **sq** to parameterize the square in the same way that **cos** is used to parameterize the circle.

```
> plot([sq(t), sq(t-1/4), t=0..1], scaling=constrained);
```

Here is the square graphed with the circle.

```
> plot([ [sq(t), sq(t-1/4), t=0..1],
>        [cos(2*Pi*t), cos(2*Pi*(t-1/4)), t=0..1] ],
>        scaling=constrained);
```

Here is a graph of a number of "concentric" squares.

```
> squares := seq( [i*sq(t), i*sq(t-1/4), t=0..1], i=1..10):
> plot({squares}, scaling=constrained);
>
```

Exercise: Parameterize a regular octagon. (Try to parameterize a regular hexagon also.)

```
>
```

At the beginning of this section we mentioned that there is an interesting connection between the equations $x^2 + y^2 = 1$ and $\max(|x|, |y|) = 1$. The next three exercises help you to find this connection.

Exercise: Here is another equation whose graph is a square, but this time the square is rotated to have its corners on the axes. The equation is $|x| + |y| = 1$. Here is its graph. (In the next worksheet we will find out why this graph is not quite exactly right.)

```
> plots[implicitplot]( abs(x)+abs(y)=1, x=-1..1, y=-1..1,
>                      scaling=constrained);
```

Find a parametric representation of this square.

```
>
```

Exercise: The family of equations $|x|^\alpha + |y|^\alpha = 1$, for α between 1 and 2, have graphs that are somewhere between a square and a circle. That is, when $\alpha = 1$ the graph of the equation is the square from the last exercise, and when $\alpha = 2$ the graph is the circle of radius one centered at the origin. Here is an example with α between 1 and 2.

```
> plots[implicitplot]( abs(x)^1.5+abs(y)^1.5=1, x=-1..1,
  y=-1..1,
>                      scaling=constrained);
```

Use your parameterization from the previous exercise to create an animation of a square morphing into a circle.

```
>
```

Exercise: What happens if you let α be less than 1, or greater than 2, in the above family of equations? What happens if α becomes very large? Use your animation from the last exercise to

watch α grow large.

```
>
```

What is the connection between the equations $x^2 + y^2 = 1$ and $\max(|x|, |y|) = 1$?

```
>
```

In the rest of this section we try out some interesting modifications of our parameterization of the square using **sq**.

Here is an interesting modification to **sq**. We will replace the flat "tops" of **sq** with parabolas. This will make the new **sq** a bit more cosine like.

```
> new_sq := t -> piecewise(
>    frac(abs(t)) < 1/8, 64*(1-c)*frac(t)^2+c,
>    frac(abs(t)) < 3/8, -8*frac(abs(t))+2,
>    frac(abs(t)) < 5/8, -64*(1-c)*(frac(abs(t))-1/2)^2-c,
>    frac(abs(t)) < 7/8, 8*frac(abs(t))-6,
>    frac(abs(t)) < 1,   64*(1-c)*(frac(abs(t))-1)^2+c
> );
```

The following equation was used to help define **new_sq**. This equation solves for the coefficients a, b, and c in the general quadratic polynomial $a\,x^2 + b\,x + c$ so that the parabola goes through the points (-1, 1/8) and (1, 1/8).

```
> x:=-1/8:
> y:=1/8:
> solve( {a*x^2+b*x+c=1, a*y^2+b*y+c=1}, {a,b,c});
> x, y := 'x', 'y':
```

The parameter **c** determines the maximum height of the parabolic segments in the graph of **new_sq**. Here is a graph of **new_sq** with a specific value for **c**.

```
> c := 5/4;
> plot(new_sq, -1..1);
```

Here is how this function compares with $\cos(2\,\pi\,t)$.

```
> plot([new_sq(t), cos(2*Pi*t)], t=-1..1);
```

Now use **new_sq** to graph a parametric curve.

```
> plot([new_sq(t), new_sq(t-1/4), t=0..1],
  scaling=constrained);
```

Try a few different values for the parameter **c**.

```
> c := 3/2;
> plot([new_sq(t), new_sq(t-1/4), t=0..1],
  scaling=constrained);
> c := 1/2;
> plot(new_sq(t), t=-2..2);
> plot([new_sq(t), new_sq(t-1/4), t=0..1],
  scaling=constrained);
```

You should try other values of **c** also.

```
>
```

Let us see how we can draw families of these "squares". First, redefine **new_sq** so that it is explicitly a function of **c**.

```
> new_sq := (t,c) -> piecewise(
>    frac(abs(t)) < 1/8, 64*(1-c)*frac(t)^2+c,
>    frac(abs(t)) < 3/8, -8*frac(abs(t))+2,
>    frac(abs(t)) < 5/8, -64*(1-c)*(frac(abs(t))-1/2)^2-c,
>    frac(abs(t)) < 7/8, 8*frac(abs(t))-6,
>    frac(abs(t)) < 1,   64*(1-c)*(frac(abs(t))-1)^2+c
> );
```

Now create a sequence of parameters for the **plot** command, each parameter with a different values of **c**, and use them in a **plot** command.

```
> shapes:=seq(
>           [new_sq(t,1/2+i/6), new_sq(t-1/4,1/2+i/6), t=0..1],
>         i=0..6):
> plot([shapes], scaling=constrained);
```

When **c** is 1, **new_sq** parameterizes a square. When **c** is less than 1, the "squares" bulge inwards toward the center, and when **c** is greater than1 the "squares" bulge outwards.

```
>
```

Here is another family of these shapes. This family of graphs is created in a slightly different way. Here we use the **display** command, plus we parameterize the color of the curves.

```
> shapes:=seq(
>           plot( [new_sq(t,1/4+i/10), new_sq(t-1/4,1/4+i/10),
  t=0..1],
>                 color=COLOR(RGB,0,0,.5+i/40) ),
>         i=1..20):
> plots[display]([shapes], scaling=constrained);
```

Make this into a movie.

```
> plots[display]([shapes], scaling=constrained,
  insequence=true);
```

Make the movie periodic.

```
> shapes := shapes, seq( shapes[-i], i=1..nops([shapes])):
> plots[display]([shapes], scaling=constrained,
  insequence=true);
>
```

We can draw a very elegant spiral using **new_sq**.

```
> plot([t*new_sq(t, 1/2), t*new_sq(t-1/4, 1/2), t=0..4],
>       scaling=constrained);
>
```

Exercise: Replace the flat tops of `sq` (or, to put it another way, the parabolic tops of `new_sq`) with piecewise linear tops. So instead of being flat topped like `sq`, or with a curved top like `new_sq`, the new function will have a "slanted roof" top. Use the new function to draw closed parametric curves and spiral parametric curves.

```
>
```

Exercise: Change the definition of `new_sq` so that it has two parameters, one for setting the peak of each of the parabolic "tops". (Then the new version of `new_sq` need no longer be symmetric about 1/2.) Draw some parametric curves using this new version of `new_sq`.

```
>
```

Exercise: Replace the linear "sides" of `new_sq` with parabolic segments. Then the graph of this new function will be piecewise parabolic and there will be two parameters in the definition of the function (or as many as 5 parameters if you do not want to make the parabolic "sides" symmetric). Draw some parametric families with this new function.

```
>
```

```
>
```

7.4.3. The `tubeplot` command

Finally, let us consider one last topic about parametric curves. A very nice command in the `plots` package is `tubeplot`, which takes a parametric curve in three dimensions and graphs a tube around the curve. So, in a sense, this command allows us to convert a one dimensional curve in space into a two dimensional surface. Here is an example of a tube formed around the trefoil knot.

```
> plots[tubeplot]( [(2+cos(3*t/2))*cos(t),
>                   (2+cos(3*t/2))*sin(t),
>                   sin(3*t/2), t=0..4*Pi] );
```

We can change the radius of the tube by using the `radius` option to `tubeplot`.

```
> plots[tubeplot]( [(2+cos(3*t/2))*cos(t),
>                   (2+cos(3*t/2))*sin(t),
>                   sin(3*t/2), t=0..4*Pi], radius=0.4);
```

In the next sequence of commands, we "cut away" part of the tube and combine it with a graph of the trefoil knot to show how the knot runs through the center of the tube.

```
> g1:=plots[tubeplot]([(2+cos(3*t/2))*cos(t),(2+cos(3*t/2))*sin
  (t),
>                      sin(3*t/2), t=Pi/2..Pi], radius=0.5):
> g2:=plots[tubeplot]([(2+cos(3*t/2))*cos(t),(2+cos(3*t/2))*sin
  (t),
>                      sin(3*t/2), t=2*Pi..5*Pi/2],
  radius=0.5):
```

```
> g3:=plots[tubeplot]([(2+cos(3*t/2))*cos(t),(2+cos(3*t/2))*sin
  (t),
>                      sin(3*t/2), t=3*Pi..7*Pi/2],
  radius=0.5):
> g4:=plots[spacecurve]([(2+cos(3*t/2))*cos(t),(2+cos(3*t/2))*s
  in(t),
>                      sin(3*t/2), t=0..4*Pi]):
> plots[display](g1,g2,g3,g4);
```

Try changing the curve in the last graph into a (very narrow) tube, so that the shape of the curve is more visible.

```
>
```

The radius of the tube can be give by a function, so the radius need not be constant along the whole tube.

```
> plots[tubeplot]([(2+cos(3*t/2))*cos(t),
>                  (2+cos(3*t/2))*sin(t),
>                  sin(3*t/2), t=0..4*Pi],
  radius=.4+.3*cos(2*t));
>
```

```
>
```

7.5. Graphs of functions of two variables

Maple's other basic graphing command is **plot3d**. This command draws graphs of real valued functions of two real variables. We graph such functions by giving **plot3d** a function and two ranges, one for each of the two independent variables. Here is an example. Be sure to click on this graph with the mouse and try rotating it.

```
> plot3d(cos(2*x)+y^2, x=-3..3, y=-3..3);
```

When you rotate the graph with the mouse, look at the context bar at the top of the Maple window. On the left edge of the context bar there are two boxes with numbers in them that change as you rotate the graph. These numbers describe the **orientation** of the graph. If you rotate a graph into a position that you think is especially nice, you can tell the **plot3d** command to draw the graph with that position by specifying the orientation numbers to the **plot3d** command using the **orientation** option. Here is the function from the last graph but with an orientation specified in the **plot3d** command.

```
> plot3d(cos(2*x)+y^2, x=-3..3, y=-3..3, orientation=[135,-90]);
```

There are many other buttons on the context bar for three dimensional graphs. Try playing with these buttons to see what they do.

```
>
```

A sometimes useful feature of the **plot3d** command is the ability to graph in "black and white" by using the **shading=zgreyscale** option. Graphs that are drawn this way will sometimes print

better on a black and white laser printer than graphs that are drawn in full color.

```
> plot3d(sin(x)+sin(y/2), x=-6..6, y=-6..6, shading=zgreyscale);
>
```

Recall that the **plot** command allowed the specification of two ranges, one range for the independent variable and one range for the dependent variable. The range for the dependent variable was especially useful for graphing functions that had vertical asymptotes. We might therefore expect the **plot3d** command to allow a third range for its dependent variable. But it does not. The **plot3d** command always chooses the range for the dependent variable and it automatically takes care of the fact that a function might "blow up" somewhere. As an example we will graph the function $1/(x\,y)$. Compare graphing this function with graphing $1/x$ using the **plot** command.

```
> plot3d(1/(x*y), x=-3..3, y=-3..3);
>
```

The **plot3d** command can draw graphs of several functions at the same time. Here is a graph of two functions. Notice that the two functions must be put inside of a pair of braces (brackets would mean something else here).

```
> plot3d({x*y-1, x^2+y^2}, x=-10..10, y=-10..10);
>
```

Exercise: Do the two surfaces in the last graph touch each other?

```
>
```

Here is an example of graphing a function and one of its tangent planes. We do this example using Maple functions instead of expressions. First define the function.

```
> f := (x,y) -> -x^2-y^2;
```

Now define the point where we want to compute the tangent plane.

```
> x0,y0 := 2,2;
```

Now define the function that defines the tangent plane (we use the name **tpf** for "**t**angent **p**lane of **f**").

```
> tpf := (x,y) -> f(x0,y0) + D[1](f)(x0,y0)*(x-x0) +
  D[2](f)(x0,y0)*(y-y0);
```

The next command returns the expression for the tangent plane function, just so that we can see what it looks like.

```
> tpf(x,y);
```

Now graph the original function and its tangent plane function.

```
> plot3d({f, tpf}, -5..5, -5..5);
```

The last graph used a single **plot3d** command to draw two surfaces. Using a single **plot3d** command to draw multiple surfaces can be convenient, but often we can improve a graph by using separate **plot3d** commands for each surface and then combining the graphs using **display**. Using separate **plot3d** command lets us, for example, give each surface its own domain. The next execution group improves the last graph by giving the tangent plane a smaller domain.

```
> graph1 := plot3d(f, -5..5, -5..5):
> graph2 := plot3d(tpf, -2..5, -2..5):
> plots[display](graph1,graph2);
>
```

Exercise: Here is the graph of the function and its tangent plane as expressions. For one thing, notice how much more quickly this graph is drawn.

```
> plot3d({-x^2-y^2, -8-4*(x-2)-4*(y-2)}, x=-5..5, y=-5..5);
```

The tangency in this graph is at the point (2,2,-8). Modify the **plot3d** command to "zoom in" on the point of tangency until the function and its tangent plane are just barely distinguishable from each other.

```
>
```

Exercise: Choose some other function and some other point and draw a graph of the function and its tangent plane at the point. Try graphing a function and several tangent planes at once (using Maple functions).

```
>
```

The **plot3d** command can draw graphs over regions that are not rectangular. This can have quite an effect on the appearance of the graph of a function. For example, the following graph of $f(x, y) = x^2 + y^2$ has a rectangular domain.

```
> plot3d(x^2+y^2, x=-4..4, y=-4..4, axes=boxed);
```

Notice how the graph has a very scalloped top edge. If you look at the graph of this function in most calculus books, the graph will look much more bowl like than the above graph. The following command redraws the graph with a circular, instead of rectangular, domain.

```
> plot3d(x^2+y^2, x=-4..4, y=-sqrt(16-x^2)..sqrt(16-x^2),
  axes=boxed);
```

To see the shape of the region that the function is being graphed over, use the mouse to rotate the graph so that you are looking straight down the *z*-axis onto the *xy*-plane. Here is the same function graphed over a region that is bounded by a piece of a parabola on one edge and a straight line on another edge.

```
> plot3d(x^2+y^2, x=-4..4, y=-1/2*(x+4)..-(x/2)^2+4, axes=boxed);
```

Here is a two dimensional graph of the region that the above graph is drawn over. Try lining up the above graph so that it looks similar to this next graph.

```
> plot([-1/2*(x+4), -(x/2)^2+4], x=-4..4,
>       scaling=constrained, color=black);
```

Notice that the region for the graph of the surface was defined by two functions of *x*. One function defines the "top edge" of the region and the other function defines the "bottom edge". It is also possible to use two functions of *y* to define a region over which a surface is graphed. In this case, one function will define the "left hand edge" and the other function will define the "right hand edge". Here is an example using the same surface as above. The "right hand edge" of the graphing region is a sine curve.

```
> plot3d(x^2+y^2, x=-4..sin(Pi*y)+3, y=-4..4, axes=boxed);
```

Here is the region drawn by itself in two dimensions (notice that this time we need to use parametric curves to outline the region). Again, line up the above graph so that it looks similar to the next graph.

```
> plot([[-4,y,y=-4..4],[sin(Pi*y)+3,y,y=-4..4],
>       [x,4,x=-4..3],[x,-4,x=-4..3] ],
>      scaling=constrained, color=black);
>
```

Exercise: The following graph of the function $f(x, y) = x^2 - y^2$ has a very curved bottom.

```
> plot3d(x^2-y^2, x=-4..4, y=-4..4, axes=framed);
```

Find a shape for the domain of the graph so that the graph of this function has a flat bottom.

```
>
```

In a previous exercise, we drew a graph of a function and one of its tangent planes. Here is the graph once again.

```
> plot3d({-x^2-y^2, -8-4*(x-2)-4*(y-2)}, x=-5..5, y=-5..5);
```

Let us modify the region that this graph is drawn over so that the graph of the function has a flat bottom. We will draw the graph over a circular domain.

```
> plot3d({-x^2-y^2, -8-4*(x-2)-4*(y-2)},
>         x=-5..5, y=-sqrt(25-x^2)..sqrt(25-x^2));
```

This graph is a bit strange looking. The tangent plane is now a very large disk, and it does not look very good. We really should graph the function over a circular domain and graph the tangent plane over a rectangular domain. We can do this if we use two separate **plot3d** commands and then combine their graphs using the **display** command.

```
> graph1 := plot3d(-x^2-y^2, x=-5..5,
  y=-sqrt(25-x^2)..sqrt(25-x^2)):
> graph2 := plot3d(-8-4*(x-2)-4*(y-2), x=0..4, y=0..4):
> plots[display](graph1,graph2);
>
```

Exercise: We used expressions throughout this last example. Modify the example to use Maple functions throughout, including using functions to define the shape of the region to draw **graph1** over.

```
>
```

An important way to study a function of two variables is to look at its level sets. A level set for a surface is all the points on the surface that have the same elevation. The **plot3d** command has the **style=contour** option for drawing a function's level sets. Here is a graph of the level sets for a bowl shaped surface.

```
> plot3d(3*x^2+5*y^2, x=-5..5, y=-5..5, style=contour,
  axes=framed);
```

If you rotate this last graph so that the vertical axis is straight up and so that you are viewing the edge of the *xy*-plane, then you will see that the level sets are all parallel to the *xy*-plane, that is, each level set is all the points on the surface that are the same height above (or below) the *xy*-plane.

```
> 
```

From just the level sets it can be difficult to visualize the shape of a surface. There is another option to **plot3d** that graphs the surface with the level sets drawn on the surface.

```
> plot3d(3*x^2+5*y^2, x=-5..5, y=-5..5, style=patchcontour);
> 
```

The next command uses a non rectangular domain so that the level sets do not break up into pieces. Notice that the idea of drawing a graph so that its top (or bottom) edge is "flat" is the same as drawing the graph so that its top (or bottom) edge is a level set.

```
> plot3d(3*x^2+5*y^2, x=-5..5,
  y=-sqrt(15-3/5*x^2)..sqrt(15-3/5*x^2),
>        style=contour);
> 
```

The **plot3d** command also has a **contours** option for specifying the number of level sets to draw or for specifying the specific elevations to use for the level sets. Here is an example that request three level sets.

```
> plot3d(3*x^2+5*y^2, x=-5..5,
  y=-sqrt(15-3/5*x^2)..sqrt(15-3/5*x^2),
>        contours=3, style=patchcontour);
```

Here is an example that specifies exactly which level sets to draw.

```
> plot3d(3*x^2+5*y^2, x=-5..5,
  y=-sqrt(15-3/5*x^2)..sqrt(15-3/5*x^2),
>        contours=[10,20,65], style=patchcontour);
> 
```

Here is an example of a more interesting surface and some of its level sets. Click on this graph with the mouse and look at the context bar. There is a group of seven buttons near the middle of the context bar. These buttons let you switch between seven different **style** options. The third and fifth buttons (from the left) are for the styles **patchcontour** and **contour**. Try them.

```
> plot3d(x*y*exp(-(x^2+y^2)), x=-2..2, y=-2..2, style=contour);
> 
```

Closely related to the idea of a function's level sets is the notion of a contour diagram for a function. A contour diagram for a function is made by pushing level sets for the function down into the *xy* -plane. So a contour diagram is a two dimensional representation of a surface in three dimensions. Maple has the **contourplot** command from the **plots** package for drawing two dimensional contour diagrams for functions of two variables.

```
> plots[contourplot](x*y*exp(-(x^2+y^2)), x=-2..2, y=-2..2);
```
(Notice that the last graph is strictly two dimensional. You cannot rotate it.) The **contourplot** command has several options. For example, the following command specifies more contour lines, it specifies that the area between the contours should be shaded in, and the shading should shift from green to blue. (The default shading is from red, for low contour values, to yellow, for high contour values.) The shading helps to distinguish between peaks and valleys in the graph of the surface.
```
> plots[contourplot](x*y*exp(-(x^2+y^2)), x=-2..2, y=-2..2,
>                    contours=15, filled=true,
  coloring=[green,blue]);
> 
```

We just mentioned that a contour diagram is made by pushing the level sets for a surface down into the *xy*-plane. Here is a nice way to see how this happens. The next command draws a (two dimensional) contour diagram.
```
> plots[contourplot]((-3*y)/(x^2+y^2+1), x=-10..10, y=-10..10,
>                     contours=17, axes=boxed);
```
The next command draws a three dimensional graph of the level sets for the same function as in the last command. But the next command sets the orientation of the graph so that we are looking straight down the *z*-axis onto the *xy*-plane. So the next graph appears at first to be a two dimensional contour diagram (i.e., the same one as the last graph). But you can rotate it to see that the level curves in this diagram are really floating in space. Looking straight down the *z*-axis has the visual affect of pushing the level sets down into the *xy*-plane.
```
> plot3d((-3*y)/(x^2+y^2+1), x=-10..10, y=-10..10, style=contour,
>         contours=17, axes=boxed, orientation=[-90,0]);
> 
```

Just as the **plot** command can use noncartesian coordinates on the plane when it graphs a function of one variable, the **plot3d** command can use noncartesian coordinates on three dimensional space when it graphs a function of two variables. For example, here is the same function graphed using three different coordinate systems in space, cartesian, cylindrical, and spherical.
```
> plot3d(x^2+y^2, x=-2..2, y=-2..2);
> plot3d(x^2+y^2, x=-2..2, y=-2..2, coords=cylindrical);
> plot3d(x^2+y^2, x=-2..2, y=-2..2, coords=spherical);
```
And just as the **plot** command must, for each coordinate system on the plane, (arbitrarily) choose one coordinate direction for the independent variable of the function, the **plot3d** command must, for each coordinate system on three dimensional space, (arbitrarily) choose two coordinate directions for the independent variables of the function. In addition, the **plot3d** command must choose an ordering for the two independent variables, that is, a way to match up each of the two ranges in the **plot3d** command with one of the two preferred coordinate directions.
```
> 
```

For example, consider the cartesian coordinate system in space. Let us use the common labels *x*, *y*,

and z for the coordinates. The **plot3d** command of course chooses x and y as the independent variables. The first range in the **plot3d** command is associated to x and the second range to y. In addition, the x and y coordinates are drawn on a graph so as to form a right hand coordinate system. So given a function f of two variables, the **plot3d** command will by default draw the graph of $z = \mathrm{f}(x, y)$. There are times when some other graph may be desirable, for example $y = \mathrm{f}(x, z)$, and we will see later how this can be done.

Exercise: The following five commands graph two different functions. Which commands are graphing the same function? Explain why.

```
> plot3d(cos(2*x)+y^2, x=-Pi..Pi, y=-3..3);
> plot3d(cos(2*x)+y^2, y=-3..3, x=-Pi..Pi);
> plot3d(cos(2*y)+x^2, y=-3..3, x=-Pi..Pi);
> plot3d(cos(2*v)+u^2, u=-Pi..Pi, v=-3..3);
> plot3d(cos(2*u)+v^2, v=-3..3, u=-Pi..Pi, orientation=[135,45]);
>
```

Exercise: Two students are arguing over whether or not, given a function f of two variables, **plot3d** can always graph both $z = \mathrm{f}(x, y)$ and $z = \mathrm{f}(y, x)$. The first student claims it can and gives this example. Let $\mathrm{f}(u, v) = u \sin(v)$.

```
> f := (u,v) -> u*sin(v);
```

The next two commands seem to graph $z = \mathrm{f}(x, y)$ and $z = \mathrm{f}(y, x)$ respectively.

```
> plot3d(f(x,y), x=-2*Pi..2*Pi, y=-2*Pi..2*Pi, axes=framed);
> plot3d(f(y,x), x=-2*Pi..2*Pi, y=-2*Pi..2*Pi, axes=framed);
```

On the other hand, the second student points out that the next command graphs $z = \mathrm{f}(x, y)$, and there is no way, using the Maple function form of plotting, to graph $z = \mathrm{f}(y, x)$ without redefining f.

```
> plot3d(f, -2*Pi..2*Pi, -2*Pi..2*Pi, axes=framed);
```

Who do you think is more correct? Should we say that **plot3d** can always graph both $z = \mathrm{f}(x, y)$ and $z = \mathrm{f}(y, x)$, or should we say that **plot3d** graphs only $y = \mathrm{f}(x, y)$?

```
>
```

Next we want to look at how **plot3d** makes choices for the cylindrical and spherical coordinates systems. First the cylindrical coordinate system.

Let us use the common labels θ, r, and z for the coordinates in the cylindrical coordinate system. When graphing a function in cylindrical coordinates, the **plot3d** command chooses the angular and vertical directions (i.e., θ and z) as the independent variables and the radial direction (i.e., r) as the dependent variable. In addition, the first range in the **plot3d** command will be associated to the radial direction and the second range will be associated to the vertical direction. In other words, given a function f of two variables, the **plot3d** command with cylindrical coordinates will draw the graph of $r = \mathrm{f}(\theta, z)$. So for example, we graph a cylinder by graphing a constant function.

```
> plot3d(3, theta=0..2*Pi, z=-6..6, coords=cylindrical,
```

```
   axes=boxed);
```
The equation $r = \sin(3\theta)$ in polar coordinates graphs a three petal rose in the plane. Here is a "cylinder" over this three petal rose.
```
> plot3d(sin(3*t), t=0..2*Pi, z=0..1/2, coords=cylindrical,
  axes=boxed);
```
The above graph does not really look very good. We can improve it by using another option. (We will see in the next worksheet what this option does and why it is needed.)
```
> r := sin(3*theta);
> plot3d(r, theta=0..2*Pi, z=0..1/2, coords=cylindrical,
  axes=boxed,
>        grid=[40,40]);
```
Recall that the equation $r = a\,(1 + 2\cos(\theta))$ in polar coordinates defines a limacon in the plane with a "diameter" determined by a. Here is an example with $a = 5$.
```
> plot(5*(1+2*cos(t)), t=0..2*Pi, coords=polar,
  scaling=constrained);
```
If we graph the function $f(\theta, z) = 5\,(1 + 2\cos(\theta))$ as a function of two variables with cylindrical coordinates, the graph will be a "cylinder" over the above limacon.
```
> r := 5*(1+2*cos(theta));
> plot3d(r, theta=0..2*Pi, z=-1..1, coords=cylindrical,
>        axes=boxed, grid=[40,40]);
```
Now let the "diameter" of the limacon vary with z. In the next graph, every horizontal cross section is a limacon, but the "diameters" depends on z.
```
> r := (1+z^2)*(1+2*cos(theta));
> plot3d(r, theta=0..2*Pi, z=-1..1, coords=cylindrical,
>         axes=boxed, grid=[40,40]);
>
```

Given a function of one variable, it is easy to use cylindrical coordinates to graph its surface of revolution around the z-axis. Here is a simple example.
```
> sqrt(z);
> plot3d(%, theta=0..2*Pi, z=0..4, coords=cylindrical);
```
Here is an animation of the one dimensional graph being revolved around the z-axis to create the surface of revolution.
```
> p := t -> plot3d(sqrt(z), theta=0..t, z=0..4,
  coords=cylindrical):
> plots[display]( [ seq(p(2*Pi*i/50), i=1..50) ],
  insequence=true);
```
Here are a few other surfaces of revolution.
```
> exp(-z^2);
> plot3d(%, theta=0..15*Pi/8, z=-2..2, coords=cylindrical);
> z+sin(z);
> plot3d(%, theta=0..2*Pi, z=0..8*Pi, coords=cylindrical);
```

```
> 1+sin(z)/z;
> plot3d(%, theta=0..2*Pi, z=-3*Pi..3*Pi, coords=cylindrical);
```
Here is the previous function again, but with a different range. This example shows that occasionally the **plot3d** command can draw misleading graphs. The following surface should not be broken into two pieces.
```
> 1+sin(z)/z;
> plot3d(%, theta=0..2*Pi, z=-4*Pi..4*Pi, coords=cylindrical);
```
Here is a torus (a donut shaped surface) drawn as a surface of revolution for two functions. (Try graphing each of the two surfaces by themselves.)
```
> 2+sqrt(1-z^2), 2-sqrt(1-z^2);
> plot3d( {%}, theta=0..2*Pi, z=-1..1, coords=cylindrical);
```
In the next section we will see how to draw surfaces of revolutions around the other two axes.
```
>
```

Exercise: Part (a) Use **plot3d** with cylindrical coordinates to graph a sphere of radius 4 centered at the origin.
```
>
```
Part (b) Now drill a cylindrical hole of radius three through the sphere along the z-axis. Draw the remaining part of the sphere along with the wall of the cylindrical hole.
```
>
```
Part (c) Cut away part of the graph from part (b) so that you can see the space between the wall of the sphere and the wall of the hole. Make both a horizontal and a vertical cutaway.
```
>
```

Given a function f of two variables, there are a total of six ways that we could graph f in cylindrical coordinates. We could choose to graph $r = f(\theta, z)$, $r = f(z, \theta)$, $z = f(\theta, r)$, $z = f(r, \theta)$, $\theta = f(r, z)$, or $\theta = f(z, r)$. By default, the **plot3d** command will only draw the first of these six possible graphs. In the next section of this worksheet we will see how we can draw all six of these graphs by using parametric equations. And in the next worksheet we will see how we can draw all of these graphs by defining new versions of cylindrical coordinates.

Why is it that **plot3d** defaults to $r = f(\theta, z)$? Recall that in polar coordinates we usually graph functions of the form $r = f(\theta)$. And since cylindrical coordinates is just polar coordinates with the variable z added, it seems reasonable to just consider the z coordinate as another independent variable and, by analogy to polar coordinates, graph functions of the form $r = f(\theta, z)$.

Of the six possible graphs that we could make in cylindrical coordinates, there is one other graph that would seem to be a very reasonable choice as the default graph for **plot3d**. Since the default graph in rectangular coordinates is of the form $z = f(x, y)$, and since (x, y) and (r, θ) both coordinatize the plane, then by analogy to rectangular coordinate it would seem reasonable for **plot3d** to graph $z = f(r, \theta)$. Such graphs can in fact be very useful. For example, suppose that in a calculus class we want to find the volume under the graph of a function $f(x, y)$ and over the cardioid

defined by $r = 1 + \cos(\theta)$, and we want to visualize this volume before computing it. So we need to draw a graph of $f(x, y)$ over the cardioid $r = 1 + \cos(\theta)$. We might try to do this using **plot3d**'s ability to graph over non rectangular domains in rectangular coordinates, but that would be difficult. What we would like to do is convert the function to cylindrical coordinates using $g(r, \theta) = f(r\cos(\theta), r\sin(\theta))$ and then draw a graph of $z = g(r, \theta)$ using cylindrical coordinates with the variable θ ranging between 0 and 2π and the variable r ranging between 0 and $1 + \cos(\theta)$. But **plot3d** cannot graph a function of the form $z = g(r, \theta)$, so we will come back to this example in the next section.

```
>
```

Now let us turn to spherical coordinates. Let us use the common labels ρ, θ, and ϕ to represent the coordinates. When graphing a function of two variables in spherical coordinates, the **plot3d** command uses θ and ϕ as the independent variables. The first range in the **plot3d** command will be associated to θ and the second range will be associated to ϕ. So given a function f of two variables, the **plot3d** command with spherical coordinates will draw the graph of $\rho = f(\theta, \phi)$. For example, we can graph a sphere using spherical coordinates by graphing a constant function. The following command graphs a sphere but with a bit of its top removed and with a vertical slice taken out. Try closing each of these holes (one at a time).

```
> plot3d(1, t=0..7*Pi/4, p=Pi/8..Pi, coords=spherical);
>
```

Exercise: Here is a graph of a function of two variables in spherical coordinates. This is a "bumpy sphere" with a hole in the bottom.

```
> r := 1+.2*cos(5*theta)*cos(5*phi);
> plot3d(r, theta=0..2*Pi, phi=0..3*Pi/4, coords=spherical);
```

How would you change the size of the hole? How would you cut away the front half of the graph? How would you make the bumps bigger or smaller? More or less numerous? Make a 3D animation of the hole growing and shrinking. Make another animation of a "pulsating sphere" with the bumps growing and shrinking.

```
>
```

In the last section we worked with parametric curves. Let us see how we can combine parametric curves with graphs of functions in two variables. First, let us draw some curves that lie on surfaces. The easiest way to draw a graph of a curve on a surface is to use the surface's function to "lift" a curve off of the plane and into the surface. Here is an example. We will lift a spiral up to the graph of a paraboloid.

```
> g1:=plot3d(x^2+y^2, x=-2..2, y=-sqrt(4-x^2)..sqrt(4-x^2),
>            style=hidden, shading=xy ):
> g2:=plots[spacecurve](
>          [t*cos(20*t), t*sin(20*t),
  (t*cos(20*t))^2+(t*sin(20*t))^2],
>            t=0..2, color=black, numpoints=200 ):
```

```
> plots[display](g1,g2);
> r := 'r':
```

Try removing **g1** from the **display** command so that you see only the spiral curve.

```
>
```

Exercise: Redraw the last graph but with the spiral curve in the *xy*-plane added as a kind of "shadow" of its lifted version on the paraboloid. Draw the graph with both the surface in the graph and with the surface removed.

```
>
```

Here is an example of a curve drawn in cylindrical coordinates on a surface drawn in rectangular coordinates.

```
> g1:=plot3d(x^2-y^2, x=-1..1, y=-sqrt(x^2+1)..sqrt(x^2+1),
>            style=patchnogrid):
> r := sin(4*t):
> g2:=plots[spacecurve]( [r, t, (r*cos(t))^2-(r*sin(t))^2],
>          t=0..2*Pi, coords=cylindrical, color=black,
  numpoints=100):
> plots[display](g1,g2);
> r := 'r':
```

```
>
```

Exercise: Change the curve drawn on the surface from a rose to the cardioid $r = 1 + 2\cos(\theta)$.

```
>
```

Here is an example of a curve drawn in rectangular coordinates on a surface drawn with cylindrical coordinates (a sine curve running up and down the side of a cylinder).

```
> g1:=plot3d(1, theta=0..2*Pi, z=0..3*Pi, coords=cylindrical,
>            style=hidden, shading=xy ):
> g2:=plots[spacecurve]( [2/3*sin(2*t), sqrt(1-(2/3*sin(2*t))^2),
  t],
>            t=0..3*Pi, color=black, numpoints=100 ):
> plots[display](g1,g2);
```

```
>
```

Exercise: Modify the last example so that the sine curve lies in the *yz*-plane inside the cylinder and then make the cylinder "see through" so that you can see the sine curve inside the cylinder.

```
>
```

Exercise: Part (a) Modify the sine curve on a cylinder example so that the cylinder has a diameter that depends on *z* (try $2 + z$ as the expression for the diameter). Make sure that the graph of the sine curve stays on the graph of the new surface.

[>

Part (b) Now modify the graph from part (a) so that the amplitude of the sine curve also increases as it moves up the graph in the *z* direction (and again, keep the graph of the new sine curve on the graph of the surface).

[>

Exercise: Draw a curve on the surface of a sphere.

[>

Exercise: Take any one of the above examples of a curve drawn on a surface and convert the curve into a (pretty narrow) tube plot. Try this both with and without the surface itself in the graph.

[>

Now let us look at an example from calculus of a curve on a surface. Recall that the partial derivative with respect to x of a function $f(x, y)$ at a point (x_0, y_0) is computed by holding y fixed at y_0 and then computing the ordinary derivative of $f(x, y_0)$ (which is a function of only one variable) at x_0. Geometrically, this means that we slice the graph of $f(x, y)$ through the point $(x_0, y_0, f(x_0, y_0))$ with a vertical plane parallel to the *yz*-plane, which gives us a curve in the slicing plane, and then compute the slope of the line tangent to this curve. Here is a picture of this for the function $f(x, y) = -x^2 - y^2$ and the point $(x_0, y_0) = (2, 2)$. Make sure you understand each step of this execution group and how it relates to the definition of the partial derivative.

```
> f := (x,y) -> -x^2-y^2;
> x0,y0 := 2,2;
> tl := x -> f(x0,y0)+D[1](f)(x0,y0)*(x-x0);
> graph1:=plot3d(f, -5..5, -5..5):
> graph2:=plots[implicitplot3d](y=y0, x=-5..5, y=-5..5,
  z=-50..0):
> graph3:=plots[spacecurve]( [t, y0, f(t,y0)], t=-5..5,
  color=black ):
> graph4:=plots[spacecurve]( [t, y0, tl(t)], t=-2..5,
  color=black):
> plots[display](graph.(1..4), style=patchnogrid, axes=framed);
```

[>

Exercise: Part (a) Modify the last example so that it demonstrates the partial derivative with respect to y at the point (x_0, y_0).

[>

Part (b) Now combine the last two graphs and demonstrate both partial derivatives at the point (x_0, y_0).

[>

Part (c) Add the tangent plane at the point (x_0, y_0) to your graph from part (b).

[>

[>

7.6. Graphs of parametric surfaces

Just as the **plot** command can graph both functions of one variable and also parametric curves in the plane, the **plot3d** command can graph both functions of two variables and parametric surfaces in three dimensional space. Before going over the details of parametric surfaces, let us quickly review the case of the **plot** command.

Here is a graph of one function of a single variable.

```
[ > plot( cos(t), t=0..2*Pi );
```

Here is a graph of two functions, each of a single variable.

```
[ > plot( [cos(t), sin(t)], t=0..2*Pi );
```

Now if we move the range inside the brackets, the graph becomes a parametric curve in the plane, and the two functions are the horizontal and vertical component functions of the curve.

```
[ > plot( [cos(t), sin(t), t=0..2*Pi] );
[ >
```

Exercise: Here is a graph of three functions, each of a single variable.

```
[ > plot( [cos(t), sin(t), 2*t], t=0..2*Pi );
```

What if we now move the range inside the brackets?

```
[ > plot( [cos(t), sin(t), 2*t, t=0..2*Pi] );
```

How do we fix the last command so that it draws an appropriate curve?

```
[ >
```

Now let us look at the analogous uses of the **plot3d** command. Here is a graph of a single function of two variables.

```
[ > plot3d( u*cos(v), u=-1..1, v=-Pi..Pi );
```

Here is a graph of two functions of two variables.

```
[ > plot3d( {u*cos(v), u*sin(v)}, u=-1..1, v=-Pi..Pi );
```

And here is a graph of three functions of two variables.

```
[ > plot3d( {u*cos(v), u*sin(v), v}, u=-1..1, v=-Pi..Pi );
```

Now replace the braces with brackets, and the graph becomes a parametric surface, and the three functions become the *x*, *y*, and *z* components of the parameterization.

```
[ > plot3d( [u*cos(v), u*sin(v), v], u=-1..1, v=-Pi..Pi );
```

Notice a key difference in the syntax of the **plot** and **plot3d** commands. If we want to graph several functions of a single variable, the **plot** command allows us to place the list of functions inside either a pair of braces or a pair of brackets, but if we want to graph a parametric curve, we need the (two) functions along with their range inside of a pair of braces. On the other hand, if we want to graph several functions of two variables, the **plot3d** command requires that we put the functions inside of a pair of braces, and if we want to graph a parametric surface we need the (three)

functions inside of a pair of brackets and the (two) ranges outside of the brackets.

```
> 
```

In the first section of this worksheet we said that a parametric surface is defined by a single function, a 3-dimensional vector valued function of two real variables. Here is a way to use a Maple function to emphasize that a parametric surface is really defined by a single (vector valued) function. The function $f(u, v) = (u\cos(v), u\sin(v), v)$ defines the surface in the previous example. Here is the Maple definition of this function.

```
> f := (u,v) -> [u*cos(v), u*sin(v), v];
```

Here is how we use this function to graph the parametric surface.

```
> plot3d( f(u,v), u=-1..1, v=-Pi..Pi );
```

This example emphasizes that a parametric surface is defined by a single function. But it is usually more convenient to work with three expressions than with a single Maple function, so for the rest of this section we will express parametric surfaces by using three expressions for the three component functions of the parameterization.

```
> 
```

In general, finding equations for parametric surfaces is tricky business. Most third semester calculus books will have several nice examples of interesting parametric surfaces. Here are a few of examples and a couple of exercises that are typical of what you will find in a calculus book.

```
> plot3d( [sin(u), u*sin(v), u*cos(v)], u=-Pi..Pi, v=0..3*Pi/2 );
```

```
> plot3d( [(u-sin(u))*cos(v), (1-cos(u))*sin(v), u],
>           u=0..2*Pi, v=0..3*Pi/2 );
```

```
> [ (2+cos(theta))*cos(phi),
>   (2+cos(theta))*sin(phi),
>   sin(theta) ];
> plot3d( %, theta=0..15*Pi/8, phi=0..15*Pi/8, title="Torus" );
```

```
> [ 2+cos(theta)+r*cos(theta/2),
>   2+sin(theta)+r*cos(theta/2),
>   r*sin(theta/2) ];
> plot3d( %, theta=0..2*Pi, r=-1/2..1/2, title="Mobius strip" );
> 
```

Exercise: Redraw one of the last parametric surfaces using a single (vector valued) Maple function to define the parameterization.

```
> 
```

In the section on parametric curves, we saw that animations of the parameter sweeping out a curve were helpful in visualizing how the parameterization works. We can do something similar in the case of parametric surfaces. An animation of a parametric surface can "unfold" one parameters at a

time. Such an animation can help us to understand the role played in the parameterization by that parameter. Here is an example. Here is a standard parameterization of the sphere.

```
> [sin(phi)*cos(theta), sin(phi)*sin(theta), cos(phi)]:
> plot3d( %, theta=0..2*Pi, phi=0..Pi, title="Sphere" );
```

$(\theta, \phi) \rightarrow [\sin(\phi)\cos(\theta), \sin(\phi)\sin(\theta), \cos(\phi)]$ with θ between 0 and 2π and ϕ between 0 and π.

One way to understand this parameterization is to see the $\cos(\theta)$ and $\sin(\theta)$ terms as doing a horizontal rotation of a radius, which is the $\sin(\phi)$ term. The radius term starts out at 0 when ϕ is 0, and the radius then grows to 1 and then shrinks back to 0 as ϕ goes from 0 to π. The height above the *xy*-plane of the radius that is being rotated is given by the $\cos(\phi)$ term, which starts at 1 and decreases to -1.

```
>
```

Let us animate this parameterization by unfolding the parameterization in each of the θ and ϕ "directions". We cannot use the **animate3d** command for these animations since it has a restriction that the ranges in the command must be given by constants and it is precisely the ranges that we want to animate. So we need to use the **display** command with the **insequence=true** option. The following "graph valued function" makes the animation easier to describe. Notice that the input to the function is the range of one of the parameters of the parameterization. This first animation shows the rotation caused by the θ parameter.

```
> p := t -> plot3d(
>                  [sin(phi)*cos(theta), sin(phi)*sin(theta),
  cos(phi)],
>                  theta=0..t, phi=0..Pi
>                 );
> plots[display]([seq( p(2*Pi*i/100), i=1..100)],
>                title="Animated Sphere",
>                orientation=[-60,60],
>                insequence=true );
```

The next animation shows how the ϕ parameter determines the radius swept out by the θ parameter.

```
> p := s -> plot3d(
>                  [sin(phi)*cos(theta), sin(phi)*sin(theta),
  cos(phi)],
>                  theta=0..2*Pi, phi=0..s
>                 );
> plots[display]([seq( p(Pi*i/100), i=1..100)],
>                title="Animated Sphere",
>                orientation=[30,100],
>                insequence=true );
```

The next animation shows how we can unfold both parameters at once. The resulting animation is interesting to watch, but it probably does not help one to understand the parameterization as much as the previous two animations.

```
> p := (t,s)->plot3d(
>                  [sin(phi)*cos(theta), sin(phi)*sin(theta),
  cos(phi)],
>                  theta=0..t, phi=0..s
>                 );
> plots[display]([seq( p(2*Pi*i/100, Pi*i/100), i=1..100)],
>                 title="Animated Sphere",
>                 orientation=[-60,90],
>                 insequence=true, scaling=constrained );
>
```

Here are three similar animations for the torus parameterization.

```
> p := t -> plot3d([(2+cos(theta))*cos(phi),
>                   (2+cos(theta))*sin(phi),
>                   sin(theta)],
>                  theta=0..t, phi=0..2*Pi
>                 );
> plots[display]([seq( p(2*Pi*i/100), i=1..100)],
>                 title="Animated Torus",
>                 orientation=[30,160],
>                 insequence=true );
```

```
> p := s -> plot3d([(2+cos(theta))*cos(phi),
>                   (2+cos(theta))*sin(phi),
>                   sin(theta)],
>                  theta=0..2*Pi, phi=0..s
>                 );
> plots[display]([seq( p(2*Pi*i/100), i=1..100)],
>                 title="Animated Torus",
>                 orientation=[-90,15],
>                 insequence=true );
```

```
> p := (t,s)->plot3d([(2+cos(theta))*cos(phi),
>                     (2+cos(theta))*sin(phi),
>                      sin(theta)],
>                     theta=0..t, phi=0..s
>                    );
> plots[display]([seq( p(2*Pi*i/100, 2*Pi*i/100), i=1..100)],
>                 title="Animated Torus",
>                 orientation=[-110,-160],
>                 insequence=true );
>
```

Exercise: Study the following parameterization. How would you get more spirals to show in the graph? What if you wanted this to be a spiral water trough with no top?

```
> plot3d( [(2+sin(v))*cos(u),
>          (2+sin(v))*sin(u),
>          u+cos(v)], u=0..4*Pi, v=0..2*Pi );
>
```

Exercise: Can you make this one into an open topped water trough? How can you get more spirals? (Hint: Experiment with the various constants in the formulas and try to get a sense of what they determine.)

```
> [ (1-u)*(3+cos(v))*cos(4*Pi*u),
>   (1-u)*(3+cos(v))*sin(4*Pi*u),
>   3*u+(1-u)*sin(v) ];
> plot3d( %, u=0..1, v=0..2*Pi, orientation=[-14,76] );
>
```

Exercise: Let f(x) be a real valued function of one variable. Find a way to parameterize the surface of revolution generated by revolving the graph of f(x) around the x-axis. Graph a few surfaces of revolution for different functions f.

```
>
```

Exercise: Part (a) Here are four different parameterizations of the sphere of radius one centered at the origin. For each parameterization try to change just one range so that the parameterization draws only the upper hemisphere. If you cannot do this by changing just one range, explain why.

```
> [ sin(phi)*cos(theta), sin(phi)*sin(theta), cos(phi) ];
> plot3d( %, theta=0..2*Pi, phi=0..Pi, title="Sphere" );
>
> [ cos(phi)*cos(theta), cos(phi)*sin(theta), sin(phi) ];
> plot3d( %, theta=0..Pi, phi=0..2*Pi, title="Sphere" );
>
> [ sin(phi)*cos(theta), sin(phi)*sin(theta), cos(phi) ];
> plot3d( %, theta=0..Pi, phi=0..2*Pi, title="Sphere" );
>
> [ sqrt(1-z^2)*cos(theta), sqrt(1-z^2)*sin(theta), z ];
> plot3d( %, theta=0..2*Pi, z=-1..1, title="Sphere" );
```

Part (b) Repeat part (a) but this time try to draw only the right hemisphere.

```
>
```

Part (c) Repeat part (a) but this time try to draw 3/4 of the sphere (say, the upper hemisphere and the left or right half of the lower hemisphere).

```
>
```

Exercise: This exercise compares the second of the four sphere parameterizations above with a parameterization of a torus. Explain how the following minor change in the sphere parameterization

manages to become half of a torus. First, the sphere parameterization again.

```
> [ cos(phi)*cos(theta), cos(phi)*sin(theta), sin(phi) ];
> plot3d( %, theta=0..Pi, phi=0..2*Pi, title="Sphere" );
```

Now a minor change that makes it into a parameterization of a torus.

```
> [ (2+cos(phi))*cos(theta), (2+cos(phi))*sin(theta), sin(phi) ];
> plot3d( %, theta=0..Pi, phi=0..2*Pi, title="1/2 Torus" );
>
```

Exercise: The following parameterization draws a "can". Explain how this parameterization works by comparing it to the standard parameterization of a sphere. Also, how would you make this can's height twice what its diameter is? How would you turn this into a parameterization of a can with no lid?

```
> f := x -> piecewise(x<1/4, 4*x, x<3/4, 1, x<=1, -4*x+4);
> g := x -> piecewise(x<1/4, 1, x<3/4, 2-4*x, x<=1, -1);
> [ f(phi)*cos(theta), f(phi)*sin(theta), g(phi) ]:
> plot3d( %, theta=0..2*Pi, phi=0..1, title="Can" );
>
```

Exercise: How would you graph two (or more) parametric surfaces at the same time, but with different parameter ranges for each surface? (Compare this with using **plot** or **spacecurve** to draw several parametric curves.)

```
>
```

Recall that in the previous section we mentioned that the **plot3d** command, when graphing a function of two variables, gives the x and y coordinates the preferred status of being the independent variables and **plot3d** always draws a graph of $z = f(x, y)$. When graphing a parametric surface, **plot3d** does not give any coordinate direction a preferred status. It does however always treat the first expression after the opening bracket as the x-component, the second expression as the y-component, and the third expression as the z-component. We can use this to get other kinds of graphs from a function of two variables. That is, we can use parametric equations to draw any of the six graphs $z = f(x, y)$, $z = f(y, x)$, $y = f(x, z)$, $y = f(z, x)$, $x = f(y, z)$, or $x = f(z, y)$. For example, here is how we can graph $x = y^2 + z^2$ (as a parametric surface).

```
> plot3d( [y^2+z^2, y, z], y=-2..2, z=-2..2);
```

Here is the analogous way to graph $y = x^2 + y^2$ as a parametric surface.

```
> plot3d([x, y, x^2+y^2], x=-2..2, y=-2..2);
>
```

Exercise: Use the scroll bar on the Maple window so that you can see both of the last two graphs at the same time. Use the mouse to rotate the two graphs into the same position. Then notice how similar mouse motions on each graph can cause different rotations of the graphs. Also, notice that each graph can be rotated around an axis that the other graph cannot be rotated about.

```
>
```

The **plot3d** command can draw parametric surfaces using other spatial coordinates systems besides rectangular coordinates. Here is an example that uses cylindrical coordinates. This is a ribbon winding its way up the side of a paraboloid.

```
> plot3d([sqrt(theta+t), theta, theta+t], theta=0..7*Pi, t=0..3,
>         coords=cylindrical, style=patchnogrid, grid=[50,50]);
```

Here is a similar example on the surface of a sphere and done with spherical coordinates. It looks like an apple being peeled.

```
> plot3d( [1, 8*t+s, t], t=0..Pi, s=-2..2,
>          coords=spherical, style=patchnogrid, grid=[50,50]);
```

Here is a sphere with its surface being peeled off of it.

```
> g1:=plot3d( [1+.2*(Pi-t), 8*t+s, t], t=0..Pi, s=-2.4..2.4,
>          coords=spherical, style=patchnogrid, grid=[50,50]):
> g2:=plot3d(1, theta=0..2*Pi, phi=0..Pi,
>           coords=spherical, style=hidden, grid=[50,50] ):
> plots[display](g1,g2, scaling=constrained);
>
```

In the previous section we mentioned that the **plot3d** command, when using cylindrical coordinates to graph a function of two variables, gives the θ and z coordinates the preferred status of being the independent variables and **plot3d** always draws a graph of $r = f(\theta, z)$. When graphing a parametric surface in cylindrical coordinates, **plot3d** does not give any coordinate direction a preferred status. It does however always treat the first expression after the opening bracket as the r -component, the second expression as the θ-component, and the third expression as the z -component. As you might expect by now, given any function of two variables, we can use parametric equations to draw any of the six graphs $r = f(\theta, z)$, $r = f(z, \theta)$, $\theta = f(r, z)$, $\theta = f(z, r)$, $z = f(\theta, r)$, or $z = f(r, \theta)$. For example, here are the six different graphs of a constant function (over a non rectangular domain) in cylindrical coordinates.

```
> plot3d([1, theta, z], theta=0..2*Pi, z=0..theta,
  coords=cylindrical);
> plot3d([1, theta, z], theta=0..z,    z=0..2*Pi,
  coords=cylindrical);
> plot3d([r, 1, z],     r=0..2*Pi,     z=0..r,
  coords=cylindrical);
> plot3d([r, 1, z],     r=0..z,        z=0..2*Pi,
  coords=cylindrical);
> plot3d([r, theta, 1], theta=0..2*Pi, r=0..theta,
  coords=cylindrical);
> plot3d([r, theta, 1], theta=0..r,    r=0..2*Pi,
  coords=cylindrical);
>
```

Exercise: Draw the six different graphs in cylindrical coordinates of the function $f(u, v) = u^2 - v^2$, with the domain u between -1 and 1 and v between 2 and 3. For each graph, make sure that the graph makes sense for you.

```
> 
```

In the previous section we mentioned the problem of visualizing the graph of a function $z = f(x, y)$ over the cardioid $r = 1 + \cos(\theta)$. One way to do this is to use $g(r, \theta) = f(r\cos(\theta), r\sin(\theta))$ to convert the function to cylindrical coordinates and then use cylindrical coordinates to graph $z = g(r, \theta)$ with the variable θ ranging between 0 and 2π and the variable r ranging between 0 and $1 + \cos(\theta)$. The problem with this idea is that the **plot3d** command with polar coordinates will only graph functions of the form $r = g(\theta, z)$. We get around this limitation of **plot3d**, and get the graph that we want, by using parametric equations in polar coordinates to graph $z = g(r, \theta)$. Here is an example with $f(x, y) = x^2 + y^2$, so $g(r, \theta) = r^2$.

```
> plot3d([r,t,r^2], r=0..1+cos(t), t=0..2*Pi,
  coords=cylindrical);
```

The next graph shows that the previous graph is correct. The next graph redraws the previous graph and combines it with a graph of the paraboloid. From the next graph we see that the previous graph really is part of the paraboloid.

```
> g1:=plot3d([r,t,r^2], r=0..1+cos(t), t=0..2*Pi,
  coords=cylindrical):
> g2:=plot3d(x^2+y^2, x=-2..2, y=-sqrt(4-x^2)..sqrt(4-x^2),
>            style=wireframe):
> plots[display](g1,g2);
```

Here is a way to add the "walls" to the volume under the graph of $f(x, y) = x^2 + y^2$ and over the cardioid $r = 1 + \cos(t)$, so that we can see the exact shape of this volume.

```
> g1:=plot3d([r,t,r^2], r=0..1+cos(t), t=0..2*Pi,
  coords=cylindrical):
> g2:=plot3d([1+cos(t), t, (1+cos(t))^2*z], t=0..2*Pi, z=0..1,
>            coords=cylindrical):
> plots[display](g1,g2);
```

```
> 
```

Exercise: The graph named **g2** in the last example draws the walls of the volume. The walls are drawn as a parametric surface in cylindrical coordinates. Redraw the walls of the volume as the graph of a function $r = h(\theta, z)$ in cylindrical coordinates.

```
> 
```

Exercise: Draw a graph of the volume under the function $f(x, y) = x + y$ and over petal in the first quadrant of the three leaf rose $r = \sin(3\theta)$. Be sure to draw both the top of the volume and its side walls.

```
> 
```

In the previous section we mentioned that the **plot3d** command, when using spherical coordinates to graph a function of two variables, gives the θ and ϕ coordinates the preferred status of being the independent variables and **plot3d** always draws a graph of $\rho = f(\theta, \phi)$. When graphing a parametric surface in spherical coordinates, **plot3d** does not give any coordinate direction a preferred status. It does however always treat the first expression after the opening bracket as the ρ -component, the second expression as the θ-component, and the third expression as the ϕ -component. And of course, given any function of two variables, we can use parametric equations to draw any of the six graphs $\rho = f(\theta, \phi)$, $\rho = f(\phi, \theta)$, $\theta = f(\rho, \phi)$, $\theta = f(\phi, \rho)$, $\phi = f(\theta, \rho)$, or $\phi = f(\rho, \theta)$. For example, here are the six different graphs of a constant function (over a non rectangular domain) in spherical coordinates.

```
> plot3d([1, theta, phi], theta=0..2*Pi, phi=0..theta,
  coords=spherical);
> plot3d([1, theta, phi], theta=0..phi,  phi=0..2*Pi,
  coords=spherical);
> plot3d([rho, 1, phi],   rho=0..2*Pi,   phi=0..rho,
  coords=spherical);
> plot3d([rho, 1, phi],   rho=0..phi,    phi=0..2*Pi,
  coords=spherical);
> plot3d([rho, theta, 1], theta=0..2*Pi, rho=0..theta,
  coords=spherical);
> plot3d([rho, theta, 1], theta=0..rho,  rho=0..2*Pi,
  coords=spherical);
>
```

Exercise: Draw the six different graphs in spherical coordinates of the function $f(u, v) = u^2 - v^2$, with the domain u between 0 and $\pi/2$ and v between $\pi/2$ and π.

```
>
```

Here are some general exercises using parametric surfaces.

Exercise: Given a non vertical plane in three dimensional space and a point on the plane, draw a square centered at the point and lying in the plane. Also draw the projection of the square onto the *xy* -plane.

```
>
```

Exercise: Part (a) Given a plane through the origin, draw a disk centered at the origin and lying on the plane.

```
>
```

Part (b) Given a plane and a point on the plane, draw a disk centered at the point and lying on the plane and draw the projection of the disk onto the *xy*-plane.

```
>
```

Exercise: In Section 7.4.2, we defined a function **sq** that we used to parameterize a square. Use **sq** to modify the standard parameterization of the sphere into a parameterization of a cube.

```
>
```

Exercise: Parameterize various kinds of "tori" using combinations of sine, cosine and **sq** functions.

```
>
```

Exercise: Experiment with various parameterizations from this section by replacing trig functions in the parameterizations with the **sq** function. Also, try using some of the variations on the **sq** function that were define in Section 7.4.2.

```
>
```

We end this section with three animations of parameterizations that we used earlier. The following two animations provide clues for doing the exercise about parameterizing just parts of a sphere.

```
> p := s -> plot3d( [ cos(phi)*cos(theta),
>                     cos(phi)*sin(theta),
>                     sin(phi) ],
>                   theta=0..Pi, phi=0..s,
>                   style=patchcontour ):
> plots[display]([seq( p(2*Pi*s/100), s=1..100)],
>                title="Animated Sphere",
>                orientation=[30,100],
>                insequence=true );
```

```
> p := s -> plot3d( [ sin(phi)*cos(theta),
>                     sin(phi)*sin(theta),
>                     cos(phi) ],
>                   theta=0..Pi, phi=0..s,
>                   style=patchcontour ):
> plots[display]([seq( p(2*Pi*s/100), s=1..100)],
>                title="Animated Sphere",
>                orientation=[-45,60],
>                insequence=true );
```

The next animation shows one of the sphere parameterizations becoming a torus parameterization.

```
> p := s -> plot3d( [ (s+cos(phi))*cos(theta),
>                     (s+cos(phi))*sin(theta),
>                     sin(phi) ],
>                     theta=0..7*Pi/4, phi=0..7*Pi/4 ):
> plots[display]([seq( p(2*s/100), s=0..100)],
>                title="Sphere to 1/2 Torus",
>                orientation=[-60,35],
>                insequence=true );
```

```
>
```

For Maple V Releases 4 and 5

Here is an example of how we can make use of parametric graphs of functions. In Maple 5 Releases 4 and 5 there is a bug (at least on Windows 98 there is) in the way that the **plot3d** command handles "non rectangular domains" in either cylindrical or spherical coordinates and we can get around this bug by graphing functions in their parametric form. (I do not know if this bug is also in Maple V Release 5.1.)

First an example of a non rectangular domain in cylindrical coordinates. Notice that the range of the second variable (z) depends on first variable (θ). The following graph is not correct.

```
> plot3d(1, theta=0..2*Pi, z=0..theta, coords=cylindrical);
```

Here is what the graph was supposed to have looked like. Notice that all we are doing here is converting the above graph of a function into its equivalent graph as a parametric surface.

```
> plot3d([1, theta, z], theta=0..2*Pi, z=0..theta,
  coords=cylindrical);
```

Here is how we can reproduce the buggy graph above as a parametric surface. The next command is drawing a graph of the form $z = f(r, \theta)$ but the radial variable is called **theta** in the next command, the angular variable is called **z**, and the function f is the constant function **1**.

```
> plot3d([theta, z, 1], theta=0..2*Pi, z=0..theta,
  coords=cylindrical);
```

So now we see that the **plot3d** command somehow gets coordinate directions mixed up when we try to use non rectangular domains with the graph of a function in cylindrical coordinates.

```
>
```

Now here is an example of using a "non rectangular" domain in spherical coordinates. This graph is not correct.

```
> plot3d(1, t=0..2*Pi, p=Pi/4+.2*sin(5*t)..Pi,
  coords=spherical);
```

Here is what the graph was supposed to have looked like. Notice that all we are doing here is converting the above graph of a function into its equivalent graph as a parametric surface.

```
> plot3d([1, t, p], t=0..2*Pi, p=Pi/4+.2*sin(5*t)..Pi,
  coords=spherical);
```

Here is how we can reproduce the buggy graph above as a parametric surface. The next command is drawing a graph of the form $\phi = f(\rho, \theta)$ but the radial variable is called **t** and the angular variable is called **p** and the function is the constant function **1** (so ϕ is constantly equal to 1 in the graph, which explains the cone like slope of the surface).

```
> plot3d([t, p, 1], t=0..2*Pi, p=Pi/4+.2*sin(5*t)..Pi,
  coords=spherical);
```

As in the case of cylindrical coordinates, **plot3d** somehow gets coordinate directions mixed up when we try to use non rectangular domains with the graph of a function in spherical coordinates.

```
> 

> 
```

7.7. Graphs of equations

Maple can draw graphs of equations in two and three variables, it can graph equations in several coordinate systems, and it can graph more than one equation at a time. For example, here is how we can draw "graph paper" for cartesian coordinates.

```
> plots[implicitplot]({x=-2, x=-1, x=0, x=1, x=2,
>                      y=-2, y=-1, y=0, y=1, y=2},
>                      x=-3..3, y=-3..3, axes=framed);
```

And here is some "graph paper" for polar coordinates.

```
> plots[implicitplot]({r=0, r=1, r=2, r=3, r=4,
>                     theta=0, theta=Pi/4, theta=Pi/2,
  theta=3*Pi/4,
>                     theta=Pi, theta=5*Pi/4, theta=3*Pi/2,
  theta=7*Pi/4},
>                     r=0..4, theta=0..2*Pi, axes=framed,
  coords=polar);
```

It does not do much good to draw "graph paper" in three dimensions (why?). Instead, here are graphs of three "coordinate planes" for each of the rectangular, cylindrical, and spherical coordinate systems.

```
> plots[implicitplot3d]({x=0, y=0, z=0}, x=-1..1, y=-1..1,
  z=-1..1);
> plots[implicitplot3d]({r=1, theta=0, z=0},
>                       r=0..2, theta=0..2*Pi, z=-2..2,
>                       coords=cylindrical);
> plots[implicitplot3d]({rho=1, theta=Pi, phi=Pi/4},
>                       rho=0..1.5, theta=Pi/8..15*Pi/8,
  phi=0..Pi,
>                       coords=spherical);
> 
```

Exercise: Explain why the following graph is not a circle

```
> plots[implicitplot](r=1, theta=0..2*Pi, r=0..2,
>                     coords=polar, scaling=constrained);
```

Explain why the following graph is not a sphere and explain in detail exactly why it has the shape that it does.

```
> plots[implicitplot3d](rho=1, theta=0..2*Pi, phi=0..Pi,
  rho=0..1,
>                       coords=spherical, scaling=constrained);
> 
```

Exercise: Explain in detail why the following graph has the shape that it does.

```
> plots[implicitplot3d](1-u=sin(w)/2, u=0..2, v=0..7, w=0..2*Pi,
>                       coords=cylindrical);
>
```

Exercise: In third semester calculus you learn that the element of volume in cylindrical coordinates is given by $dV = r\,dr\,d\theta\,dz$. Draw a picture of an element of volume in cylindrical coordinates. (Hint: The element of volume has six faces. Draw three graphs, each of a pair of opposite faces, and then combine the three graphs together.)

```
>
```

Exercise: In third semester calculus you learn that the element of volume in spherical coordinates is given by $dV = \rho^2 \sin(\phi)\,d\rho\,d\theta\,d\phi$. Draw a picture of an element of volume in spherical coordinates.

```
>
```

Exercise: Here are eight different ways to graph a sphere. Explain how each one works.

```
> plots[implicitplot3d](x^2+y^2+z^2=16, x=-4..4, y=-4..4,
  z=-4..4);
> plots[implicitplot3d](rho=4, rho=0..4, theta=0..2*Pi,
  phi=0..Pi,
>                       coords=spherical);
> plots[implicitplot3d](r^2+z^2=16, r=0..4, theta=0..2*Pi,
  z=-4..4,
>                       coords=cylindrical);
> plot3d([4*sin(v)*cos(u),4*sin(v)*sin(u),4*cos(v)], u=0..2*Pi,
  v=0..Pi);
> plot3d([x, sqrt(16-x^2)*cos(theta), sqrt(16-x^2)*sin(theta)],
>        x=-4..4, theta=0..2*Pi);
> plot3d(4, theta=0..2*Pi, phi=0..Pi, coords=spherical);
> plot3d(sqrt(16-z^2), theta=0..2*Pi, z=-4..4,
  coords=cylindrical);
> plot3d({sqrt(16-x^2-y^2), -sqrt(16-x^2-y^2)}, x=-4..4,
>        y=-sqrt(16-x^2)..sqrt(16-x^2));
>
```

When using **implicitplot** or **implicitplot3d**, the order of the ranges is very important. As some of the above exercises demonstrate, these commands use the order of the ranges to determine exactly which coordinate a variable represents. A variable named **r** need not represent radius in polar coordinates. A variable named **r** in an **implicitplot** equation will represent the radial coordinate in polar coordinates only if the first range given is for **r** (and of course the

coords=polar option is used). On the other hand, there is nothing wrong with using the variable **x** to represent the radial coordinate in polar coordinates, we just have to list the range for **x** first (and use the **coords=polar** option).

```
> 
```

Exercise: Explain why the following two graphs look the way they do.

```
> plots[implicitplot](y=x^2-1, y=-1..3, x=-2..2);
> plots[implicitplot](x^2-1=y, x=-2..2, y=-1..3);
```

Predict what the following graph will look like before drawing it.

```
> plots[implicitplot](y^2-1=x, y=-1..3, x=-2..2):
> 
```

We can use the ordering of ranges in **implicitplot** commands to find another way to draw nonstandard graphs of functions. Recall that if f(s) is a real valued function of a single variable, then we can draw graphs of either $y = f(x)$ or $x = f(y)$ in cartesian coordinates or we could draw graphs of either $r = f(\theta)$ or $\theta = f(r)$ in polar coordinates. By default, the **plot** command will draw only $y = f(x)$ in cartesian coordinates and $r = f(\theta)$ in polar coordinates. Earlier we saw how to use the **plot** command with parametric equations to draw the graphs of $x = f(y)$ and $\theta = f(r)$. We can also draw these graphs using **implicitplot**. Here are the graphs of $x = \sin(y)$ and $\theta = \sin(r)$ drawn using **implicitplot**.

```
> plots[implicitplot](x=sin(y), x=-1..1, y=0..2*Pi);
> plots[implicitplot](theta=sin(r), r=0..4*Pi, theta=0..2*Pi,
>                     coords=polar);
> 
```

Exercise: Use **implicitplot** to draw the graph of $y = \sin(x)$ in cartesian coordinates and the graph of $r = \sin(\theta)$ in polar coordinates.

```
> 
```

Given a real valued function f(u, v) of two real variables we can use the **implicitplot3d** command to draw any one of the six possible graphs of f in each of rectangular, cylindrical, and spherical coordinates.

```
> 
```

Exercise: Use **implicitplot3d** to graph the function $z = \sin(x^2 + y^2)$ using cylindrical coordinates. Also graph this function using **plot3d** and both rectangular and cylindrical coordinates. Try to make the graphs as nearly equivalent as you can. Which graph turns out the "best", which the "worst"?

```
> 
```

Exercise: For the function f(u, v) = v sin(u) use **implicitplot3d** with spherical coordinates to

draw graphs of $\rho = f(\phi, \theta)$, $\theta = f(\phi, \rho)$, and $\phi = f(\theta, \rho)$. For each graph, find ranges for the variables that produce an interesting graph. Also, redraw each of these graphs using **plot3d** and parametric equations.

```
>
```

Earlier in this worksheet we drew contour diagrams for functions of two variables. These contour diagrams are closely related to graphs of equations. Suppose we have a function $f(x, y)$ of two variables. If we let c be any number, then we can make an equation of the form $f(x, y) = c$. The graph of an equation of this form is a level curve for the function f. A level curve for the function f is a curve in the plane such that the graph of f has constant elevation over this curve. If we use **implicitplot** to graph several level curves, then we get a contour diagram for f. Here is an example with the function $f(x, y) = x^2 - y^2$.

```
> f := (x,y) -> x^2-y^2;
> plots[implicitplot]({f(x,y)=0, f(x,y)=2, f(x,y)=4,
>                                  f(x,y)=-2,f(x,y)=-4}, x=-5..5,
  y=-5..5);
```

Here is the equivalent **contourplot** command.

```
> plots[contourplot](f(x,y), x=-5..5, y=-5..5,
  contours=[-4,-2,0,2,4]);
```

Notice that the equation $f(x, y) = c$ does not define a level set for the graph of f. We defined level sets as curves of constant elevation in three dimensional space and the equation $f(x, y) = c$ has its graph in two dimensional space. Here is a **plot3d** command that draws the level sets that are equivalent to the above level curves. Notice how you can rotate the next graph and see that these curves are really curves in three dimensional space and that they lie on the graph of f.

```
> plot3d(f(x,y), x=-5..5, y=-5..5,
>        style=contour, contours=[-4,-2,0,2,4],
>        orientation=[-90,0], axes=normal, view=[-5..5, -5..5,
  -5..5]);
>
```

```
>
```

7.8. Graphs of vector fields

Maple has two commands for drawing vector fields. The **fieldplot** command draws vector fields in the plane and **fieldplot3d** draws vector fields in space. Recall that a vector field in the plane is defined by a 2-dimensional vector valued function of two real variables and a vector field in space is defined by a 3-dimensional vector valued function of three real variables. Here is a simple example of a function that defines a vector field in the plane, $f(x, y) = (2x, 2y)$. If we let p_0 represent a point in the plane and we let (x_0, y_0) be the coordinates of p_0, then the value of $f(x_0, y_0)$ gives us the horizontal and vertical coordinates of a vector to be drawn at the point p_0. So at p_0 we would draw a vector with horizontal component $2x_0$ and vertical component $2y_0$. If we do this at

every point *p* in the plane, then we will draw a vector field that always points away from the origin, and the length of every vector in the field is twice the distance of the base of the vector from the origin. Here is a graph of this vector field.

```
> plots[fieldplot]( [2*x, 2*y], x=-3..3, y=-3..3);
>
```

Notice one thing right away about this graph. While the directions of the vectors in the graph are accurate, the lengths of the vectors are *not* accurate. For example, at the point (1,1) there is supposed to be a 45 degree vector with length $2\sqrt{2}$, which would put the tip of the vector past the point (3,3). But at (1,1) in the above graph there is a very short 45 degree vector. The vectors in the graph do get longer as they get further from the origin, but the lengths are nowhere near what they should be (why do you think that is?). A vector field drawn by **fieldplot** is not meant as a literal representation of the true vector field. The graphs drawn by **fieldplot** are meant to give an impression of how a vector field looks. These graphs usually give us a good qualitative (instead of quantitative) information about a vector field. In the above example, the graph shows us that every vector in the field points away from the origin and that the length of each vector is proportional to its distance from the origin, and this gives us a good idea of what the true vector field looks like.

```
>
```

Exercise: How would you expect the graphs of the vector fields $f(x, y) = (2x, 2y)$ and $g(x, y) = (3x, 3y)$ to differ? How would the graph of $h(x, y) = (-x, -y)$ compare with the graphs of f and g?

```
>
```

A vector field in the plane is defined by a single function but that function has two components, each of which is a real valued function of two variables. In the above **fieldplot** command we used two expressions in a list to describe the vector field, one expression for each component function. By making use of Maple functions, we can emphasize that a vector field is really defined by a single function. Here is a Maple function that defines the vector field used above

```
> f := (x,y) -> [2*x, 2*y];
```

Here is a **fieldplot** command that uses the Maple function **f**.

```
> plots[fieldplot]( f(x,y), x=-3..3, y=-3..3);
```

Recall from the section on vector valued functions in the last worksheet that there is unfortunately no standard way to work with vector valued functions. So for example, the following obvious version of the last command does not work.

```
> plots[fieldplot]( f, -3..3, -3..3);
```

And if we define **f** in the following common way (with parentheses instead of brackets)

```
> f := (x,y) -> (2*x, 2*y);
```

then we have to modify slightly the way we use **f** in **fieldplot**.

```
> plots[fieldplot]( [f(x,y)], x=-3..3, y=-3..3);
```

The most common way to use **fieldplot** is to use expressions for each of the component functions of the vector field and for the most part that is the way we will work with the command (just as we did with parametric curves and parametric surfaces, both of which also use vector valued

functions).

```
> 
```

Here are some examples of the kinds of vector fields that come up in a course on vector calculus or differential equations.

```
> plots[fieldplot]( [-y, x], x=-5..5, y=-5..5);
> plots[fieldplot]( [ln(1+y^2), ln(1+x^2)], x=-5..5, y=-5..5);
> plots[fieldplot]( [x/2, -y/3], x=-2..2, y=-2..2);
```

Here are a few 3-dimensional vector fields. It is not easy to get much information out of these kinds of graphs.

```
> plots[fieldplot3d]( [y, z, x], x=-2..2, y=-2..2, z=-2..2);
> plots[fieldplot3d]( [y/z, -x/z, z/4], x=-2..2, y=-2..2,
  z=1..3);
> plots[fieldplot3d]( [-x, -y, -z], x=-2..2, y=-2..2, z=-2..2);
> 
```

There is an important special class of vector fields that comes up very often in mathematics, gradient vector fields. This kind of vector field is used often enough that Maple has two special commands for drawing them, **gradplot** for 2-dimensional gradient fields and **gradplot3d** for 3-dimensional gradient fields.

Recall from third semester calculus that if we have a real valued function of two real variables, then its derivative at a point is a vector whose two components are the two partial derivatives of the function at the point. If we compute the derivative of the function at every point in its domain, then we get a vector field of gradient vectors. This vector field is called a gradient field and the original function is called gradient field's potential function. Let us look at an example. Consider the function $f(x, y) = \sin(x + y)$. Here are two ways to draw its gradient field, using **fieldplot** and using **gradplot**. First, let us define the function to Maple.

```
> f := (x,y) -> sin(x)+sin(y);
```

Here is how we can compute its two partial derivative functions.

```
> D[1](f); D[2](f);
```

Here is how we can use **fieldplot** to draw the gradient vector field for **f**. We give **fieldplot** the two partial derivatives of **f** as the components of the vector field.

```
> plots[fieldplot]( [ D[1](f), D[2](f) ], -6..6, -6..6 );
```

Here is how we use **gradplot** to draw the same gradient vector field for **f**. We only need to give **gradplot** the potential function.

```
> plots[gradplot]( f(x,y), x=-6..6, y=-6..6 );
> 
```

Recall from calculus that if p is any point in the domain of the potential function $f(x, y)$, then the gradient vector at p is perpendicular to the level curve passing through p. Let us demonstrate this by drawing both the gradient field and the contour diagram for **f** in the same graph. (The parameter **b** is

there to make it easier to change the domain of the graph.)

```
> b := 3:
> plots[contourplot]( f(x,y), x=-b..b, y=-b..b ):
> plots[gradplot](    f(x,y), x=-b..b, y=-b..b ):
> plots[display](%, %%);
```

If you look closely at the above graph, you can see that all of the gradient vectors are perpendicular to the level curves. (This is a bit easier to see if you zoom in on the graph a bit. Try letting **b** equal 3 or 2.) A bit more can be said about the directions of the gradient vectors. Recall that the color coding on level curves goes from red for low values to yellow for high values. The gradient vectors are pointing from red curves toward yellow curves. This shows, as you learned in calculus, that the gradient vectors point in the direction of steepest increase in the potential function. To help you see that the gradient vectors are pointing "uphill", compare the above graph with the following three dimensional graph of **f** and its level sets.

```
> b := 3:
> plot3d( f(x,y), x=-b..b, y=-b..b, style=contour,
  orientation=[-90,0] );
>
```

Exercise: Draw a combined graph of the level curves and gradient field for the potential function $f(x, y) = \frac{3y}{x^2 + y^2 + 1}$.

```
>
```

So far in this section, we have always considered a function of the form $(u, v) = f(x, y)$ as representing a vector field in the plane, that is u and v are the horizontal and vertical components of a vector that we draw at the point with coordinates (x, y). But there are other ways of interpreting this kind of function. Let us look briefly at one of these other interpretations. We can use a 2-dimensional vector valued function of two variables as a **change of variables**. Here is an example. Consider the function $(u, v) = f(r, \theta)$ defined by $f(r, \theta) = (r\cos(\theta), r\sin(\theta))$. If we let p represent a point in the plane, and suppose that p has polar coordinates (r, θ), then $(u, v) = (r\cos(\theta), r\sin(\theta))$ will be the cartesian coordinates for p. In other words, given a point in the plane, the function f takes as input the polar coordinates of the point and returns the cartesian coordinates for the same point. The function f changes the polar coordinates of a point into cartesian coordinates, and so we call it a change of variables. Of course, the function f can also be given a vector field interpretation. Let us demonstrate using both of these interpretations for f. Let us define f to Maple.

```
> f := (r,theta) -> [r*cos(theta), r*sin(theta)];
```

Here is a list of four points in the plane given in polar coordinates. These point are at the four corners of a square.

```
> points := [ [sqrt(2),   Pi/4], [sqrt(2), 3*Pi/4],
>             [sqrt(2), 5*Pi/4], [sqrt(2), 7*Pi/4] ];
```

Let us graph these points using polar coordinates.

```
> plot( points, style=point, coords=polar );
```

Now let us apply the change of variables function **f** to these points and get a list of the cartesian coordinates for the points. In the following command, **p** represents an ordered pair from the list **points**, **p[1]** is the first number in **p** (i.e., the radial coordinate) and **p[2]** is the second number in **p** (i.e., the angular coordinate).

```
> seq( f(p[1],p[2]), p=points );
```

Now graph the points using cartesian coordinates.

```
> plot( [%], style=point );
```

We just used **f** to change the coordinates of four points in the plane from polar to cartesian. Now let us interpret **f** as a vector field and graph the vector field.

```
> plots[fieldplot]( f(x,y), x=-6..6, y=-6..6 );
```

So we have given the same function **f** two very different interpretations, as a change of variables and as a vector field. Neither interpretation is more correct than the other. Some 2-dimensional vector valued functions of two variables are more useful as a vector field and some are more useful as a change of variables.

```
>
```

Exercise: The function $T(\rho, \theta, \phi) = (\rho \sin(\phi) \cos(\theta), \rho \sin(\phi) \sin(\theta), \rho \cos(\phi))$ can be interpreted as a change of variables or as a vector field. Create a short list of spherical coordinates for some points in space, plot the list using spherical coordinates, then use T to change the coordinates to rectangular coordinates and plot the points using rectangular coordinates, and then graph T as a vector field. (Note: To plot points in space, you need to use the **pointplot3d** command from the **plots** package. The **plot3d** command does not plot points.)

```
>
```

```
>
```

7.9. Online help for graphing and visualization

Maple has extensive graphing abilities. Besides the two main graphing commands, **plot** and **plot3d**, and the main package of graphing commands, **plots**, Maple has a lot of other graphing facilities scattered throughout a number of packages. There is a lot of online documentation and examples for these facilities. Below we try to outline this documentation. First we mention the documentation for the **plot** and **plot3d** commands and their most important options, and then we outline the documentation for the most important commands within the **plots** package. (Notice, as you go along, that a help page like **?plot,coords** is about an *option* to the **plot** command, and a help page like **?plots,coordplot** is about a *command* in the **plots** package.) Finally, at the end of this section we try to outline most of the documentation for Maple's other graphing commands and packages.

When you click on any two-dimensional or three-dimensional graph, the menus available at the top of the Maple window and the context bar just below the menus change appropriately. Here are two help pages that describe the items in the graphics menus.

```
> ?plot2dMenuItems
> ?plot3dMenuItems
```
And here are two help pages that briefly describe the items in the graphics context bars.
```
> ?style2
> ?style3
```

At this point we should also mention that on a few platforms, Maple has defined two special commands that try to make it easier to draw basic graphs. These are the **smartplot** and **smartplot3d** commands. The graphs drawn by these commands can be manipulated by right clicking on the graph to bring up a context menu. These context menus have more items in them than the context menus for regular graphs.
```
> ?smartplot
> ?smartplot3d
```
See also the following help page.
```
> ?contextmenu
```

Maple's most basic graphing command is of course **plot**.
```
> ?plot
```
The **plot** help page has almost no information about the options to **plot**. These are all described in the following important help page.
```
> ?plot,options
```
Many of the special features of the **plot** command, like parametric graphs, polar graphs, using infinity in a range, etc., have their own help pages.
```
> ?plot,multiple
> ?plot,color
> ?plot,style
> ?plot,ranges
> ?plot,infinity
> ?plot,function
> ?plot,parametric
> ?plot,polar
> ?plot,coords
```
Most of the options to the **plot** (and **plot3d**) command translate directly into pieces of a PLOT data structure. In a later worksheet we will say a lot more about PLOT data structures. The following command brings up a description of the PLOT data structure and all of its pieces. Sometimes, looking up the PLOT data structure analogue of a **plot** option will provide some clue about the option that is not in the option's documentation.
```
> ?plot,structure
```

We mentioned the **discont=true** option of the **plot** command. This option does not have its own help page. But the **discont=true** option to **plot** makes use of a Maple function called **discont**. The following help page is about the **discont** function and therefore it also sheds

some light on the **discont** option of **plot**.
[**> ?discont**
The **fdiscont** function, which is also used by **plot** with the **discont=true** option, does the same thing as **discont** but it works numerically instead of symbolically.
[**> ?fdiscont**

Maple's basic three dimensional graphing command is **plot3d**.
[**> ?plot3d**
The **plot3d** help page says very little about the options to the **plot3d** command. All of these options are explained in the next help page.
[**> ?plot3d,options**
The **plot3d** command has two other help pages about some of its special features.
[**> ?plot3d,coords**
[**> ?plot3d,colorfunc**

The **plot3d** command has the **style=contour** option for drawing the level curves of a surface. Two closely related commands from the **plots** package are **contourplot** and **contourplot3d**. The **contourplot** command draws all of the level curves of a surface as level sets in the plane. The **contourplot3d** command draws the level curves as curves in space. (Notice that the **contourplot3d** command seems to be equivalent to the **plot3d** command with the **style=contour** option.) The following help page describes both **contourplot** and **contourplot3d**.
[**> ?plots,contourplot**

If you are drawing a lot of graphs and they all are using the exact same options, it might be convenient to redefine the default **plot** and **plot3d** options using the **setoptions** and **setoption3d** commands from the **plots** package.
[**> ?setoptions**
[**> ?setoptions3d**

Maple can draw graphs in 15 two-dimensional coordinate systems and 31 three-dimensional coordinate systems. The next help page lists all of these coordinate systems, and, additionally, gives the formulas for the coordinate transformation of each coordinate system to cartesian coordinates.
[**> ?coords**
The next two help pages briefly summarize the two and three dimensional coordinate systems respectively.
[**> ?plot,coords**
[**> ?plot3d,coords**
The next two help pages describe the commands for drawing pictures of the two and three dimensional coordinate systems. For the two-dimensional coordinate systems, the **coordplot** command draws a picture of "graph paper" for each coordinate system. For the three-dimensional coordinate systems the **coordplot3d** command draws a surface of constant value for each of the

three coordinate variables.

```
> ?plots,coordplot
> ?plots,coordplot3d
```

The command for defining your own coordinate systems is **addcoords**. (You have to **readlib** this command in order to use it.)

```
> ?addcoords
```

It is worth mentioning that the **plots** package contains three functions for graphing in non cartesian coordinate systems that seem to be redundant with options for the **plot** and **plot3d** commands. The **polarplot** command seems to be equivalent to the **plot** command with the **coords=polar** option, the **cylinderplot** command seems to be equivalent to the **plot3d** command with the **coords=cylinder** option, and the **sphereplot** command seems to be equivalent to the **plot3d** command with the **coords=spherical** option.

```
> ?plots,polarplot
> ?plots,cylinderplot
> ?plots,sphereplot
```

Closely related to the idea of using different coordinate systems in the plane or in space is the idea of using a different coordinate system on the real line when graphing a real valued function of one variable. As the following three help pages describe, Maple can draw graphs of real valued functions of a single variable with a logarithmic scale on either or both of the axes.

```
> ?plots,logplot
> ?plots,semilogplot
> ?plots,loglogplot
```

The **plot3d** command only draws graphs of surfaces, that is, graphs of real valued functions of two variables and parametric surfaces. To draw curves in three dimensions Maple needs a special command, **spacecurve**, from the **plots** package

```
> ?plots,spacecurve
```

The **tubeplot** command lets us convert a one dimensional curve in three dimensional space into a two dimensional "tube".

```
> ?plots,tubeplot
```

There are two commands for graphing equations, one that draws two dimensional graphs of equations in two variables, and one that draws three dimensional graphs of equations in three variables.

```
> ?plots,implicitplot
> ?plots,implicitplot3d
```

There are two commands for graphing vector fields, one for two dimensional vector fields in the plane and one for three dimensional vector fields in space.

```
> ?plots,fieldplot
> ?plots,fieldplot3d
```

In addition, there are two commands for the special case of drawing gradient vector fields.

```
> ?plots,gradplot
> ?plots,gradplot3d
```

One very important command from the **plots** package is the **display** command, which can be used to combine several (usually simple) graphs into a more complicated graph.

```
> ?plots,display
```

One way that **display** can combine several graphs together is as the frames of an animation. This is done by using the **insequence=true** option to **display**. There is no separate help page for the **insequence** option. The previous help page includes a description of this option and several examples of its use. Another way to create animations is by using the **animate** and **animate3d** commands. These commands provide an easy way to make simple animations, but they are not as versatile as the **insequence=true** option to **display**.

```
> ?animate
> ?animate3d
```

There is a special animate command specifically for animating curves in the plane. In particular, this command makes nice animations of parametric curves.

```
> ?animatecurve
```

Maple animations can be used to create animated GIF files for use in web pages on the Internet. A brief explanation of this, along with an explanation of some other graphics formats that Maple can produce, is in the next help page.

```
> ?plot,device
```

The plot devices described in the last help page are used as options in either the **plotsetup** or **interface** commands.

```
> ?plotsetup
> ?interface
```

An interesting command from the **plots** package is **matrixplot**, which lets you draw a three dimensional visualization of the contents of a matrix.

```
> ?plots,matrixplot
```

In the New User's Tour there is a worksheet containing examples of using some basic graphics commands.

```
> ?newuser,topic05
```

Besides the **plot** and **plot3d** commands and the commands mentioned above from the **plots** package, Maple has many other commands for drawing graphs. In the rest of these paragraphs we show where to get additional information about most of these commands.

There are many graphing commands in the **plots** package that we have not yet mentioned. The next page is a summary of the entire **plots** package and it contains hyperlinks to the help pages of all the commands in the package.

```
> ?plots
```

The **plottools** package has a number of special functions for drawing "graphical objects", like an arrow. The next page is an overview of this package and it contains hyperlinks to the help pages of the all the commands in the package.

```
> ?plottools
```

Maple also has a **geometry** package that can be used to create illustrations of ideas and theorems from two dimensional Euclidean geometry. Here is a help page giving an overview of this package.

```
> ?geometry
```

Within this package you can work with the following kinds of geometric objects.

```
> ?geometry,objects
```

You can apply the following types of transformations to the geometric objects.

```
> ?geometry,transformation
```

The **draw** command in the **geometry** package is used to actually draw the geometric objects that you define using the package's commands.

```
> ?geometry,draw
```

Here is a help page that contains several examples of using the **geometry** package. You can cut each example out of the help page and paste it into a worksheet and then execute the example.

```
> ?geometry,examples
```

And here is a worksheet that has more examples of the **geometry** package. You can execute these examples directly in this worksheet.

```
> ?examples,geometry
```

For doing Euclidean geometry in three dimensions Maple has the **geom3d** package. Here is an overview of this package.

```
> ?geom3d
```

Within this package you can work with the following kinds of geometric objects.

```
> ?geom3d,objects
```

You can apply the following types of transformations to the geometric objects.

```
> ?geom3d,transformation
> ?geom3d,transform
```

The **draw** command in the **geom3d** package is used to actually draw the geometric objects that you define using the package's commands.

```
> ?geom3d,draw
```

Here is a worksheet that has some examples of the transformations available in the **geom3d** package. You can execute these examples directly in this worksheet.

```
> ?examples,transform
```

Maple has many commands for working with polyhedra. These commands are contained in three packages, **plots**, **plottools**, and **geom3d**. I do not really understand the division of labor involved here. I'll try to give some pointers to the documentation. In the **plots** package there is the

polyhedraplot command.

```
> ?plots,polyhedraplot
```

There is a command that will list the polyhedra supported by the **polyhedraplot** command.

```
> ?polyhedra_supported
```

Here is the list of the polyhedra supported by **polyhedraplot**. It is interesting to see the names of so many kinds of polyhedra. (Can you say parabidiminishedrhombicosidodecahedron?)

```
> polyhedra_supported();
```

In the **plottools** package there are five commands that act as an interface to the **polyhedraplot** command.

```
> ?plottools,dodecahedron
> ?plottools,hexahedron
> ?plottools,icosahedron
> ?plottools,octahedron
> ?plottools,tetrahedron
```

The **plottools** package has a few commands for modifying polyhedra, for example the **stellate** command.

```
> ?plottools,stellate
```

The **geom3d** package has a number of commands that can be used to define polyhedra. The polyhedra defined with these commands can be drawn using the **draw** command from the same package. The commands for creating polyhedra are organized into three groups.

```
> ?geom3d,RegularPolyhedron
> ?geom3d,QuasiRegularPolyhedron
> ?geom3d,Archimedean
```

The **geom3d** package has a command for creating the dual of a given polyhedron.

```
> ?geom3d,duality
```

Like the **plottools** package, the **geom3d** package has a command to stellate a polyhedron.

```
> ?geom3d,stellate
```

In addition, the **geom3d** package has a command for faceting a polyhedron.

```
> ?geom3d,facet
```

There are six worksheets that give examples of how to use the **geom3d** package to work with polyhedra. You can execute the examples directly in these worksheets. Some of the examples in these worksheets create very striking polyhedra.

```
> ?examples,regular
> ?examples,archi
> ?examples,dual
> ?examples,stellate
> ?examples,facet
> ?examples,transform
```

Here are several other packages with some specialized plotting and visualization commands in them.

```
> ?student
> ?DEtools
```

```
> ?PDEtools
> ?stats,statplots
```

Here are two worksheets that contain examples of using the **DEtools** package to graph solutions of differential equations. You can execute the examples directly in these worksheets.

```
> ?examples,deplot
> ?examples,deplot3d
```

Here is a worksheet with some examples from the **statplots** (sub) package.

```
> ?examples,statplots

>
```