3.2 Parametric and Polar Equations

In this section we will continue to pursue or exploration of curves drawn in the plane. We will look at three distinct categories of plane curves.

1. Parametric equations.
2. Polar equations.
3. Algebraic curves.

We begin with the first category.

Parametric Equations

As a first example, let's examine the curve defined by the following set of *parametric equations*.

$$\begin{aligned} x &= |1-t| \\ y &= |t|+2 \end{aligned} \tag{3.1}$$

Note that x and y are defined in terms of an independent parameter t.

The approach here is similar to the approach used to draw the graph of a function of one variable. First, create a table of points that satisfy the parametric equations. The usual technique is to select arbitrary values of the independent parameter t, then use the parametric **equations 3.1** to calculate x and y at each value of t, as shown in **Table 3.1**(a). The computations in **Table 3.1**(a) are simplified and summarized in **Table 3.1**(b).

t	$x = \lvert 1-t \rvert$	$y = \lvert t \rvert + 2$
-3	$x = \lvert 1-(-3) \rvert$	$y = \lvert (-3) \rvert + 2$
-2	$x = \lvert 1-(-2) \rvert$	$y = \lvert (-2) \rvert + 2$
-1	$x = \lvert 1-(-1) \rvert$	$y = \lvert (-1) \rvert + 2$
0	$x = \lvert 1-0 \rvert$	$y = \lvert 0 \rvert + 2$
1	$x = \lvert 1-1 \rvert$	$y = \lvert 1 \rvert + 2$
2	$x = \lvert 1-2 \rvert$	$y = \lvert 2 \rvert + 2$
3	$x = \lvert 1-3 \rvert$	$y = \lvert 3 \rvert + 2$

(a)

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

(b)

Table 3.1. Creating tables of points that satisfy the parametric **equations 3.1**.

[1] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

Matlab greatly eases the calculations shown in **Tables 3.1**(a) and (b). Create a column vector t ranging from -3 to 3 to match the values of t in **Table 3.1**(a). Compute vectors **x** and **y**.

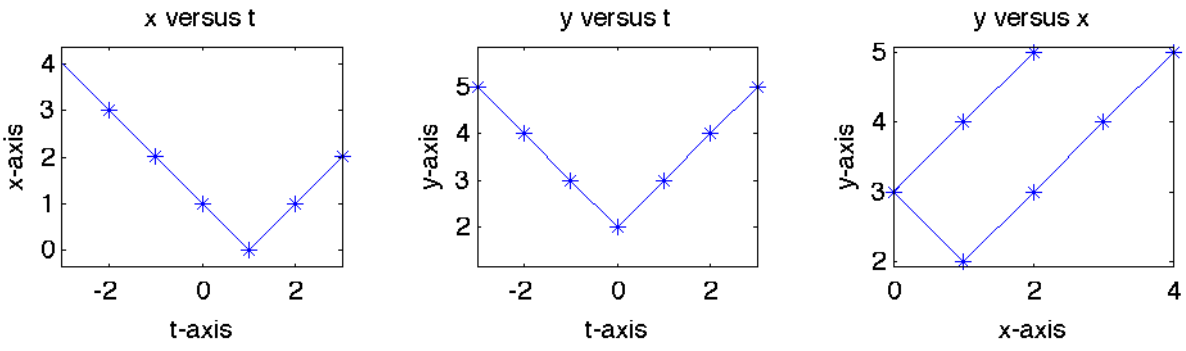
```
>> t=(-3:3).';
>> x=abs(1-t);
>> y=abs(t)+2;
```

We can build a matrix with the column vectors **t**, **x**, and **y** and display the results to the screen.

```
>> [t,x,y]
ans =
    -3     4     5
    -2     3     4
    -1     2     3
     0     1     2
     1     0     3
     2     1     4
     3     2     5
```

Note that these results match those in **Table 3.1**(b).

At this point, we have several choices: (1) we can plot **x** versus **t** (**Figure 3.1**(a)), or (2) we ccould plot **y** versus **t** (**Figure 3.1**(b)), or (3) we could plot **y** versus **x** (**Figure 3.1**(c)). Each figure has its advantages and disadvantages.



(a) **plot(t,x,'b*-')** (b) **plot(t,y,'b*-')** (c) **plot(x,y,'b*-')**

Figure 3.1. Plotting the data from **Table 3.1**(b).

The path in **Figure 3.1**(c) holds the most interest. If we think of (x, y) as the position of a particle in the plane at time t, then the path in **Figure 3.1**(c) describes the path taken by the particle in the plane over time.

Animating the Path of the Particle. Unfortunately, the path shown in **Figure 3.1**(c) is static, but we can rectify this with Matlab's **comet** command, which is used to animate the motion of the particle in the plane. Enter the following code into the Matlab editor. Execute the code, either with cell mode enabled in the editor, or as a script file.

```
close all
t=linspace(-3,3,1000);
x=abs(1-t);
y=abs(t);
comet(x,y)
```

If the animation takes inordinately long, you can stop the animation with `Ctrl+C` in the command window. Then try deleting the number of points in the **linspace** command (e.g., **x=linspace(-3,3,500)**). On the other hand, if the animation is too quick, increase the number of points in the **linspace** command (e.g., **linspace(-3,3,1500)**), then run the script again. The animation provided by Matlab's plot command is extremely valuable as it provides the user with a sense of the particle's motion in the plane over time.

Although we cannot display the animation provided by the **comet** in this document, we can indicate that the direction of motion is that shown in **Figure 3.2**.

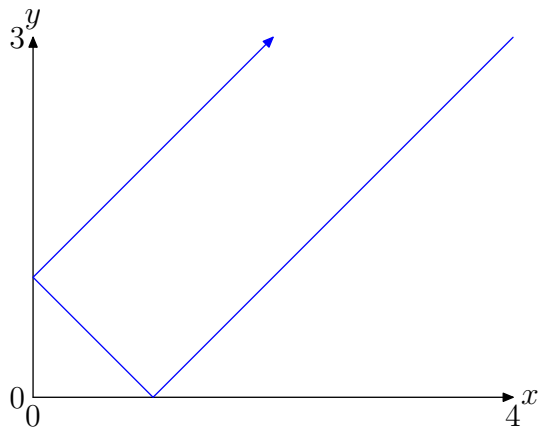


Figure 3.2. The **comet** command provides a sense of the direction of motion.

Let's look at another example.

▶ **Example 1.** *Consider the following set of parametric equations.*

$$\begin{aligned} x &= 3\cos 2t \\ y &= 5\sin 3t \end{aligned} \tag{3.2}$$

Plot x versus t, y versus t, then y versus x.

First, use the **linspace** command to create a vector of t-values. Then compute the vectors **x** and **y**. *Note: We assume that you are working either in cell-enabled mode in your editor, or you are placing the following commands in a script file.*

```
t=linspace(0,2*pi,500);
x=3*cos(2*t);
y=5*sin(3*t);
```

The following commands will produce the plot of x versus t in **Figure 3.3**(a).

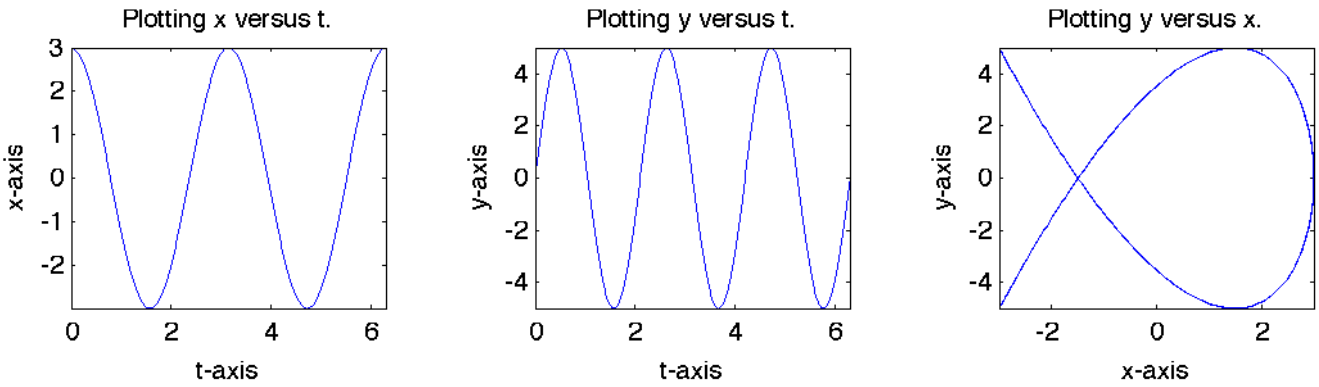
```
plot(t,x)
axis tight
xlabel('t-axis')
ylabel('x-axis')
title('Plotting x versus t.')
```

The next set of command will produce the plot of y versus t in **Figure 3.3**(b).

```
plot(t,y)
axis tight
xlabel('t-axis')
ylabel('y-axis')
title('Plotting y versus t.')
```

Finally, the following commands will produce a plot of y versus x in **Figure 3.3**(c).

```
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('Plotting y versus x.')
```



(a) (b) (c)

Figure 3.3. Plotting x and y versus t.

However, the static nature of each of the graphs in **Figure 3.3** don't reveal the character of the motion of the particle in the plane, as dictated by the parametric equations **3.2**. Only the **comet** command can reveal the motion. Although we cannot portray the dynamic motion of the particle in this printed document, you should try the following command to get a sense of the motion of the particle in the plane.

```
comet(x,y)
```

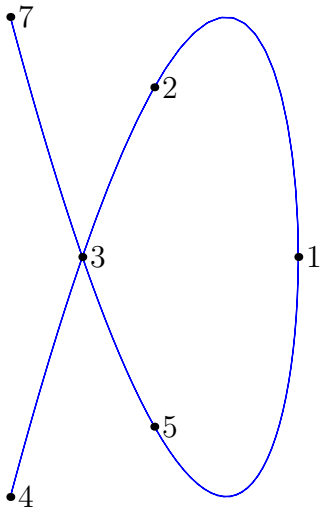


Figure 3.4. Indicating the motion of the particle using the **comet** command.

The **comet** command will show that the particle starts at postion 1 in **Figure 3.4**. It then begins its motion counterclockwise, traveling through the points at 2, 3

and stopping momentarily at the point at 4. It then retraces its route, returning through 3 and 2 to its original point at 1. It continues its clockwise journey through 5, passes through the point 3 again and stops momentarily at the point at 7. It then retraces its route back through 3 and 5, returning to its starting point at 1.

Polar Equations

The Cartesian coordinate system is not the only way to determine the position of a point in the plane. Rather than giving its x- and y-coordinates, we can determine the location of a point in the plane by indicating an angle and radial length, called *polar coordinates*, as shown in **Figure 3.5**.

We must make two important points regarding polar coordinates.

1. Angles are measured from the positive x-axis, often called the *polar axis*. The counterclockwise direction marks positive angles, the clockwise direction marks negative angles.
2. The value of r can be positive or negative. A negative r-value indicates a reversal of direction when measuring the radial distance. You "go backwards," so to speak.

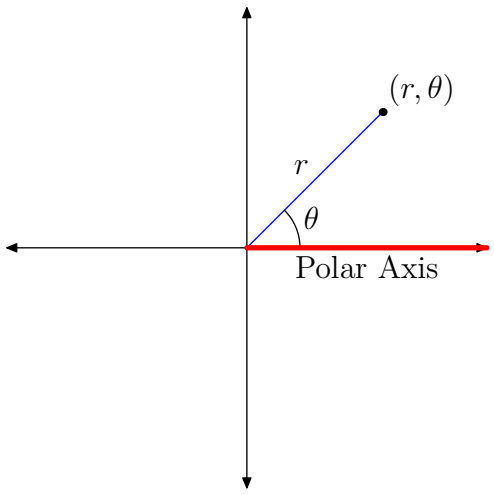


Figure 3.5. Defining polar coordinates.

Plotting points in polar coordinates is best accomplished on polar coordinate graph paper, an example of which is shown in **Figure 3.6**. To plot the point $Q(1, 60°)$, you rotate 60° in the counterclockwise direction from the polar axis, then move 1 unit outward in the radial direction to obtain the point $Q(1, 60°)$

shown in **Figure 3.6**. In similar fashion, rotate $-60°$ in a clockwise direction from the polar axis, then move 1 unit outward in the radial direction to obtain the point $R(1, -60°)$ shown in **Figure 3.6**. Finally, to understand how to proceed when r is negative, consider the point $S(-1, 30°)$. To plot this point, first rotate $30°$ in a counterclockwise direction from the polar axis, but then reverse the radial direction, moving "backward 1 unit" to obtain the point $S(-1, 30°)$ shown in **Figure 3.6**.

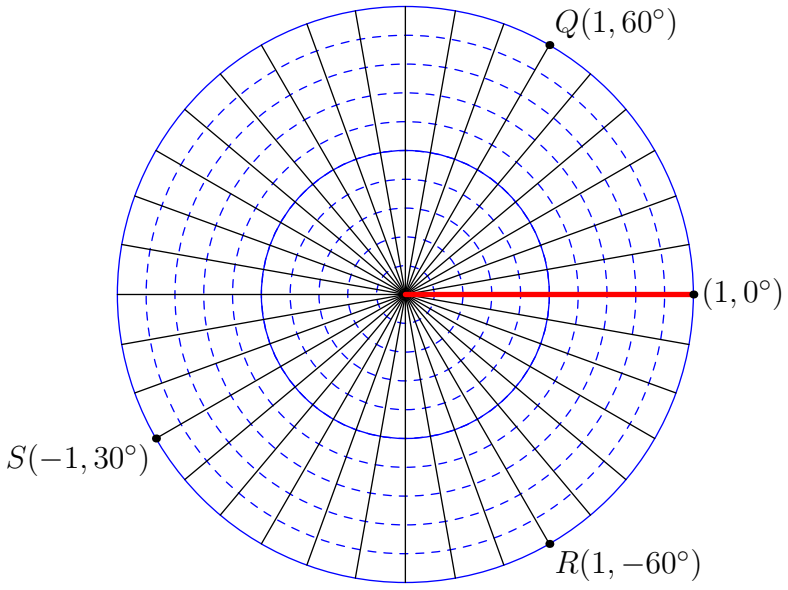


Figure 3.6. Polar coordinate graph paper.

The difficulty with polar coordinates is that unlike Cartesian coordinates, points can have multiple representations. For example, the point Q could also be the point $Q(1, 420°)$, the point $Q(-1, 240°)$, or even the point $Q(1, -300°)$. Multiple representations can cause difficulty in trigonometry and calculus, but we won't worry much about it in this course.

Matlab fully understands polar coordinates. It has commands **cart2pol** and **pol2cart** to transform Cartesian coordinates to polar coordinates, and vice-versa. As we are most intersted in the plotting of polar equations, we begin by first investigating Matlab's **polar** command.

▶ **Example 2.** *Sketch the graph of the polar equation* $r = 2\sin 2\theta$.

Note that the radial length r is given as a function of the angle θ. This is typical of the form of most polar equations. To use Matlab's **polar** command to plot the graph of $r = 2\sin 2\theta$, use the following commands, either in a script or in the cell-enabled editor. Executing these commands produces the *four-leave rose* shown in **Figure 3.7**.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
polar(theta,r)
```

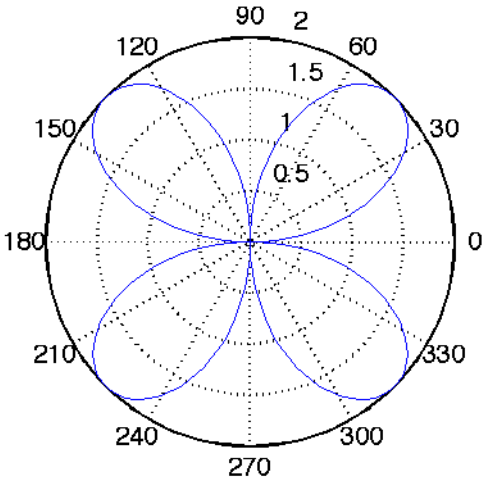


Figure 3.7. A four-leaf rose.

Note that Matlab's **polar** command plots the rose on a form of polar coordinate graph paper.

Polar to Cartesian Coordinates. In order to obtain more control over plots of polar equations, we will need to know how to transform polar coordinates into Cartesian coordinates. To do so, we need to understand a basic trigonometric definition. Consider the image in **Figure 3.8**, where the polar angle and radius are θ and r, respectively. In addition, we've labeled the x- and y-axes and the point $P(x, y)$.

Here are the standard trigonometric definitions of sine and cosine.

Definition 3. *Given the radius r and polar angle θ, we determine the Cartesian coordinates of the point P as follows.*

$$x = r\cos\theta \qquad \text{and} \qquad y = r\sin\theta \tag{3.3}$$

We can use **Definition 3** to change the polar form of the rose in **Example 2** to Cartesian form.

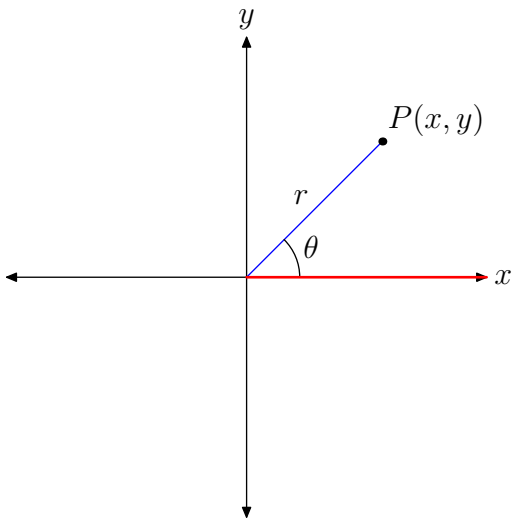


Figure 3.8. Transforming polar to Cartesian coordinates.

▶ **Example 4.** *Use Matlab to sketch the graph of the polar equation* $r = 2\sin 2\theta$ *by first transforming polar coordinates to Cartesian form.*

We can compute vectors **r** and θ as before.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
```

Next, note that **Definition 3** provides

$$x = r\cos\theta \qquad \text{and} \qquad y = r\sin\theta.$$

We use this definition to change to Cartesian coordinates.

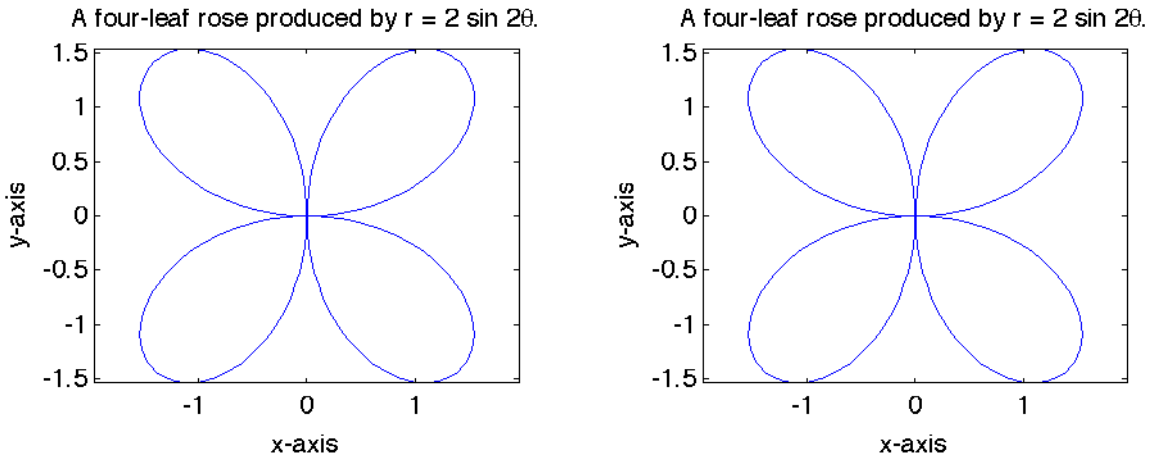
```
x=r.*cos(theta);
y=r.*sin(theta);
```

The **plot** command will now produce the image shown in **Figure 3.9**(a). We've added a command **axis equal** which sets the aspect ratio so that equal tick mark increments on the x- and $y-$ axes are equal in size.

```
plot(x,y)
axis equal
xlabel('x-axis')
ylabel('y-axis')
title('A four-leaf rose produced by r = 2 sin 2\theta.')
```

Matlab's **pol2cart** will handle the change to Cartesian coordinates if you wish to use that instead. The following commands produce an identical rose in **Figure 3.9**(b).

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
xlabel('x-axis')
ylabel('y-axis')
title('A four-leaf rose produced by r = 2 sin 2\theta.')
```



(a) $x = r\cos\theta$ and $y = r\sin\theta$. (b) **pol2cart**

Figure 3.9. The four-leaf rose in Cartesian coordinates.

The Dynamics are Missing. The static result in **Figures 3.9**(a) and (b) hide the motion of the particle in the plane. We strongly encourage our readers

to try the following code to get a feel for the motion of the particle as it traces the rose.

```
theta=linspace(0,2*pi,200);
r=2*sin(2*theta);
[x,y]=pol2cart(theta,r);
comet(x,y)
```

Algebraic Curves

Often, we are presented with an equation in x and y where it would be difficult or impossible to solve for one variable in terms of the other. In this case, the techniques for plotting that we've provided thus far will not work. However, the class of *algebraic curves* is a fascinating subject, so we would like to develop technique for plotting members of this class.

Matlab has a suite of functions called the *EZ* functions (short for "Easy") that will provide a quick sketch with a minimum of preparation. They don't offer the fine-grained control of the **plot** command, but if your goal is a simple quick sketch, they will suffice for that purpose. Here are the members of the "ez-suite."

ezcontour	**ezmeshc**	**ezpolar**
ezcontourf	**ezplot**	**ezsurf**
ezmesh	**ezplot3**	**ezsurfc**

We will concentrate on using the **ezplot** command to sketch the graphs of algebraic curves, but we encourage our readers to try some of the other commands. Help is available for each command in the form of **help ezplot**. For example, if you want help for drawing contours or level curves, you will find the information you need with the command **help ezcontour**.

Let's look at an example of an algebraic curve.

▶ **Example 5.** *Sketch the graph of the* lemniscate *given by the equation*

$$(x^2 + y^2)^2 = 2(x^2 - y^2). \tag{3.4}$$

First, note that solving for one variable in terms of the other is a seemingly impossible task (or at least extremely difficult). We will use the implicit nature of the **ezplot** command to sketch a graph of **equation (3.4)**. As the **ezplot** command is meant as a quick hack, we'll return to entering commands at the prompt in the command window.

First, set one side of the equation equal to zero, as **ezplot**'s implicit plotting routine expects a function of the form $f(x, y) = 0$.

$$(x^2 + y^2)^2 - 2(x^2 - y^2) = 0 \tag{3.5}$$

The **ezplot** routine allows us to enter the left-hand side of **equation (3.5)** as a string (remember that strings are delimited by single apostrophes). The following command will produce the image in **Figure 3.10**(a).

```
>> ezplot('(x^2+y^2)^2-2*(x^2-y^2)')
```

Three facts warrant our attention.

1. When the implicit function is entered as a string, as it is above, Matlab vectorizes the string (makes it "array smart") before passing it along to whatever routine is used to produce the curve. That is, Matlab sends along the string **'(x.^2+y.^2).^2-2*(x.^2-y.^2)'**
2. With the usage above, the **ezplot** routine uses a default domain $[-2\pi, 2\pi]$ and range $[-2\pi, 2\pi]$.
3. The **ezplot** routine labels axes and provides a title automatically. It determines the variables and title by parsing the string passed to the routine.

We can change the domain and range if we wish. The following command will produce the image in **Figure 3.10**(b).

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

In this form, $[-2, 2]$ is used for both the domain and range. The effect is seen by examining the scale on the axes in **Figure 3.10**(a). Note that both domain and range are set to $[-2, 2]$.

The domain and range can be set independently. The following command will produce the image in **Figure 3.10**(c). Note that the domain is $[-2, 2]$, but the range is now set to $[-1, 1]$.

```
>> ezplot('(x^2+y^2)^2-2*(x^2-y^2)',[-2,2,-1,1])
```

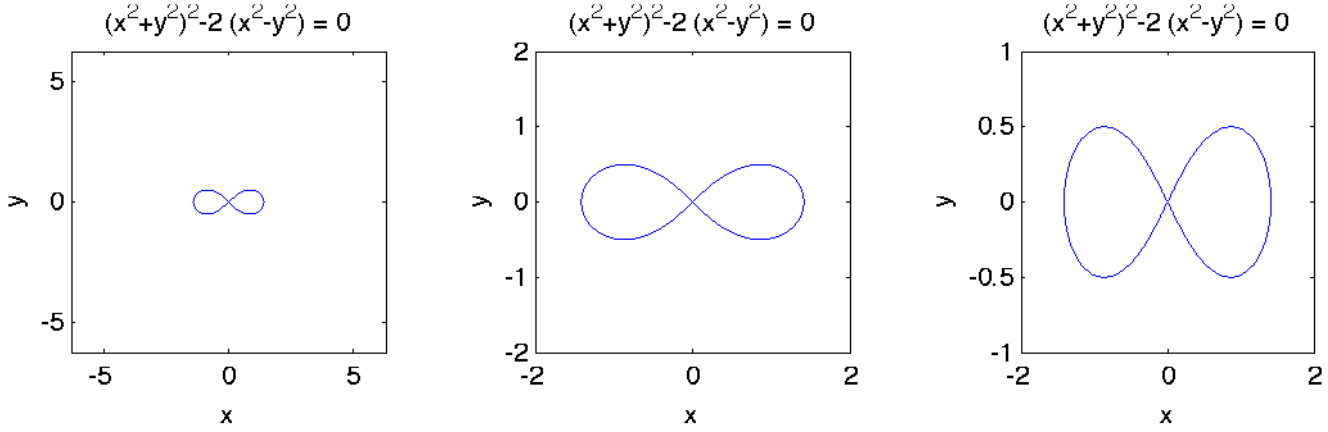


Figure 3.10. Changing domain and range.

Anonymous Functions. Matlab's **ezplot** routine will accept an anonymous function in lieu of a string. You can create an anonymous function for the lemniscate with the following command.

```
>> f = @(x,y) (x.^2+y.^2).^2-2*(x.^2-y.^2);
```

Anonymous functions are powerful constructs that were first introduced in Matlab 7. We'll have much more to say about them as the course progresses, but let's first comment on the construction of this particular anonymous function.

1. The variable **f** is assigned a *function handle*, which is essentially a number that identifies where to look in the memory of your machine for the function definition.
2. Next comes the function handle symbol **@**.
3. The **@** symbol is followed (in parentheses) by the input variables for the anonymous function. The order is important here, at least as far as the **ezplot** command is concerned. The first input variable is placed on the horizontal axis, the second on the vertical axis.
4. The input variables are followed by the function definition; i.e., the output value of the function.
5. As we saw above, **ezplot** will vectorize strings and make them "array smart." This is not the case with anonymous functions used with **ezplot**, so one must use array operators to make the anonymous function definition "array smart."

Whenever you create an anonymous function at the command prompt, it is a good idea to test the function. Note that if we substitute $x = 1$ and $y = 2$, the result should be

$$(x^2+y^2)^2-2(x^2-y^2)=(1^2+2^2)^2-2(1^2-2^2)=25+6=31.$$

If we evaluate the anonymous function **f** at the point $(1,2)$, we see that it is working properly.

```
>> f(1,2)
ans =
    31
```

Note that Matlab uses the function notation of mathematics to evaluate the anonymous function at the given values of x and y.

All looks well, so let's create the graph in **Figure 3.11** with the following command.

```
>> ezplot(f,[-2,2,-1,1])
```

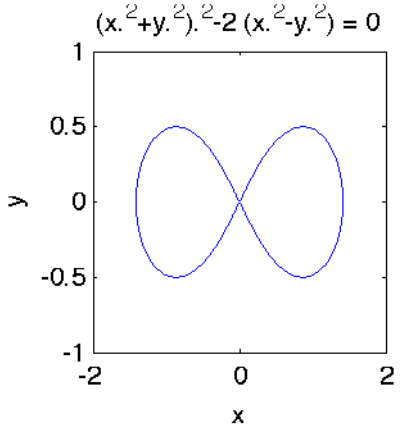


Figure 3.11. Constructing the lemniscate with an anonymous function.

Note that the result in **Figure 3.11** is identical to that found with string input in **Figure 3.10**(c).

3.2 Exercises

The general parametric form of *Lissajous curves* is given by the set of parametric equations

$$\begin{aligned} x &= a\cos(\omega_x t - \delta_x) \\ y &= b\sin(\omega_y t - \delta_y). \end{aligned} \tag{3.6}$$

There is a theorem that says that the curve will eventually repeat (or close) if the quotient ω_x/ω_y is rational. In **Exercises 1-4**, use the set of Lissajous **equations (3.6)** to sketch y versus x for the given constants. Investigate until you have a minimal domain for t that allows the curve to close.

1. $a = 2$, $b = 3$, $w_x = 2$, $w_y = 3$, $\delta_x = 1$, $\delta_y = 0$

2. $a = 5$, $b = 3$, $w_x = 4$, $w_y = 5$, $\delta_x = 0$, $\delta_y = 0$

3. $a = 2$, $b = 3$, $w_x = 3$, $w_y = 5$, $\delta_x = 0$, $\delta_y = 0$

4. $a = 5$, $b = 4$, $w_x = 4$, $w_y = 7$, $\delta_x = 1$, $\delta_y = 2$

Imagine a circle of radius b rolling about the circumference of a circle of radius a without any slippage occurring. A point P on the circumference of the outer circle of radius b traces out a curve called an *epicycloid*. The parametric equations of the epicylcoid are

$$\begin{aligned} x &= (a+b)\cos t - b\cos((1+a/b)t) \\ y &= (a+b)\sin t - b\sin((1+a/b)t). \end{aligned}$$

In **Exercises 5-8**, use Matlab to sketch the graph of the epicycloid for the given values of a and b. In each case, determine a minimal domain on t so that the curve closes.

5. $a = 8$ and $b = 5$.

6. $a = 5$ and $b = 3$.

7. In the case where $a = 2b$, a *nephroid* is produced. Set $a = 4$ and $b = 2$.

8. In the case where $a = b$, a *cardioid* is produced. Set $a = b = 2$.

---

9. The *Lemniscate of Bernoulli* has parametric equatons

$$\begin{aligned} x &= \frac{\cos t}{1+\sin^2 t} \\ y &= \frac{\cos t \sin t}{1+\sin^2 t}. \end{aligned}$$

Sketch the graph of the lemniscate for $0 \le t \le 2\pi$.

In **Exercises 10-13**, use Matlab's **polar** command to produce a plot of the given polar equation.

10. $r = 1 + \cos\theta$

11. $r = 1 - 2\cos\theta$

12. $r = 3 - 2\sin\theta$

13. $r = 3\sin 2\theta$

In **Exercises 14-17**, use the transformations

$$x = r\cos\theta \quad \text{and} \quad y = r\sin\theta$$

to plot the given polar equations in Cartesian coordinates.

14. $r = \cos 2\theta$

15. $r = -3\cos\theta$

16. $r = 1 - 3\cos\theta$

17. $r = 1 + 2\cos\frac{\theta}{2}$

In **Exercises 18-21**, use Matlab's **pol2cart** command to tranform the polar coordinates of the given polar equation into Cartesian coordinates. Use the **plot** command to plot the result.

18. $r = -1 - \sin\theta$

19. $r = 2 - 4\sin\theta$

20. You will need to carefully select the domain to obtain a good sketch of $r = 4/(1 + \sin\theta)$.

21. $r = e^{\theta/6}$

---

22. The curve defined by the polar equation

$$r = e^{\sin\theta} - 2\cos 4\theta + \sin\left(\frac{2\theta - \pi}{24}\right)$$

is called the **butterfly curve**. Sketch the polar graph on the interval $0 \le \theta \le 2\pi$ and use the result to explain the name of this curve. What happens when you extend the interval on θ, say to $[0, 12\pi]$?

23. Sketch the curve defined by the polar equation

$$r = (\sin 4\theta)^4 + \cos 3\theta.$$

This result is called the **Henri's Butterfly**, discovered by Henri Berger, a student in David Cohen's prcalculus class at UCLA in the spring of 1988.

24. Sketch the curve defined by the polar equation

$$r = \cos^2 5\theta + \sin 3\theta + 0.3.$$

This result is called the **Oscar's Butterfly**, discovered by Oscar Ramirez, a student in David Cohen's prcalculus class at UCLA in the fall of 1991.

In **Exercises 25-28**, use Matlab's **ezplot** command to plot the given algebraic curve. Use the form

ezplot(fun2,[xmin,xmax,ymin,ymax])

to provide a good viewing window for the curve. In each case, **fun2** should be entered as a string.

25. $x^4 + x^2y^2 + y^4 = x(x^2 + y^2)$, known as the "Bean Curve."

26. $(x^2 - 1)(x - 1)^2 + (y^2 - 1)^2 = 0$, known as the "Bicuspid."

27. $y^2(16 - x^2) = (x^2 + 8y - 16)^2$, known as the "Cocked Hat."

28. $y^4 - 16y^2 = x^4 - 9x^2$, known as the "Devil's Curve."

In **Exercises 29-32**, use Matlab's **ezplot** command to plot the given algebraic curve. Use the form

ezplot(fun2,[xmin,xmax,ymin,ymax])

to provide a good viewing window for the curve. In each case, **fun2** should an anonymous function.

29. $x^4 = 4(x^2 - y^2)$, known as the "Eight Curve."

30. $(x^2+y^2)(x(x+5)+y^2) = 12xy^2$, known as "Kepler's Folium."

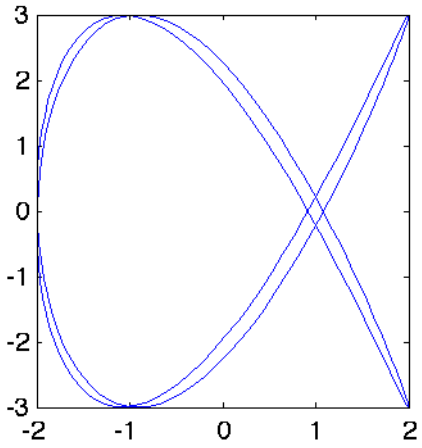
31. $16y^2 = x^3(8 - x)$, called a "Piriform."

32. $(x^2-1)^2 = y^2(3+2y)$, called the "Knot Curve."

3.2 Answers

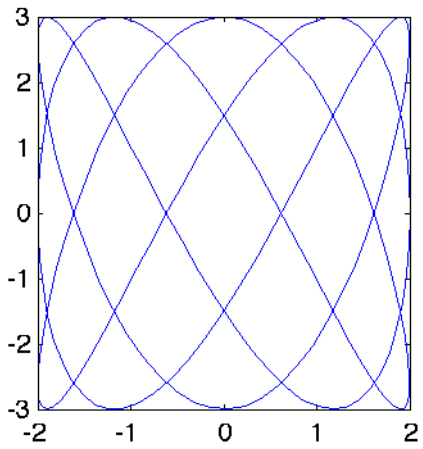
1.

```
a=2; b=3;
wx=2; wy=3;
dx=1; dy=0;
t=linspace(0,2*pi,200);
x=a*cos(wx*t+dx);
y=b*sin(wy*t+dy);
comet(x,y)
plot(x,y)
```



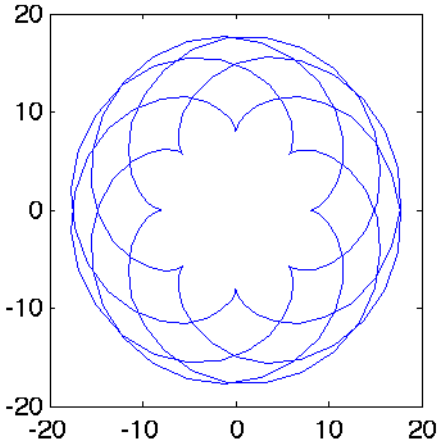
3.

```
a=2; b=3;
wx=3; wy=5;
dx=0; dy=0;
t=linspace(0,2*pi,200);
x=a*cos(wx*t+dx);
y=b*sin(wy*t+dy);
comet(x,y)
plot(x,y)
```



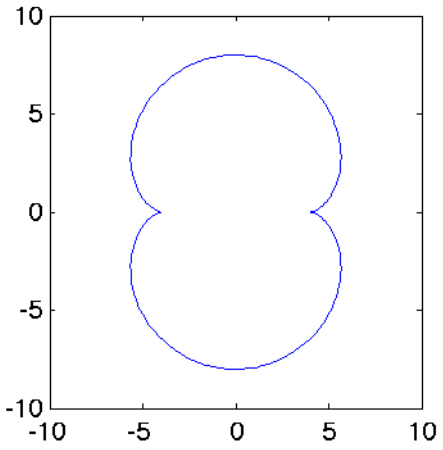
5.

```
a=8; b=5;
t=linspace(0,10*pi,200);
x=(a+b)*cos(t)-b*cos((1+a/b)*t);
y=(a+b)*sin(t)-b*sin((1+a/b)*t);
plot(x,y)
```



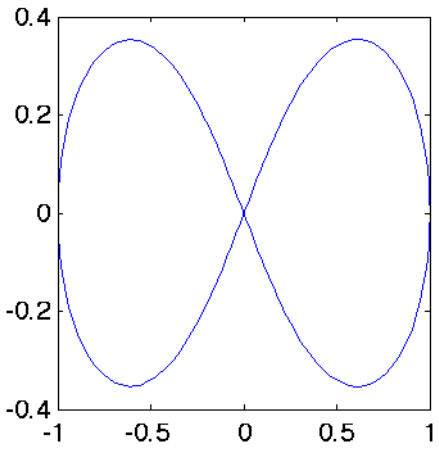
7.

```
a=4; b=2;
t=linspace(0,2*pi,200);
x=(a+b)*cos(t)-b*cos((1+a/b)*t);
y=(a+b)*sin(t)-b*sin((1+a/b)*t);
plot(x,y)
```



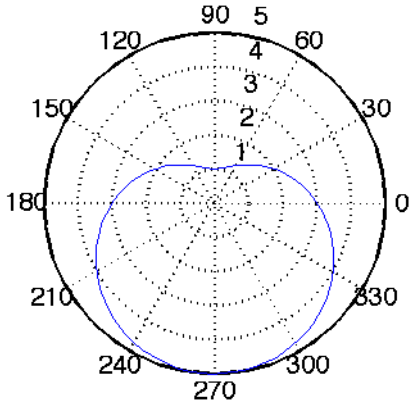
9.

```
t=linspace(0,2*pi,200);
x=cos(t)./(1+sin(t).^2);
y=cos(t).*sin(t)./(1+sin(t).^2);
plot(x,y)
```



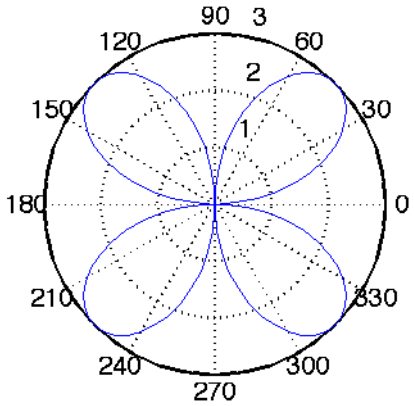
11.

```
theta=linspace(0,2*pi);
r=3-2*sin(theta);
polar(theta,r)
```



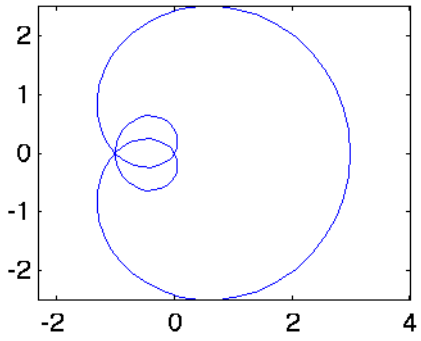
13.

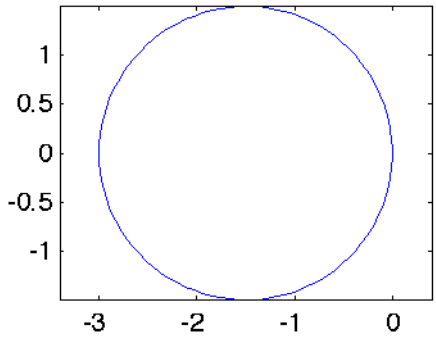
```
theta=linspace(0,2*pi);
r=3*sin(2*theta);
polar(theta,r)
```



15.

```
theta=linspace(0,pi);
r=-3*cos(theta);
x=r.*cos(theta);
y=r.*sin(theta);
plot(x,y)
axis equal
```



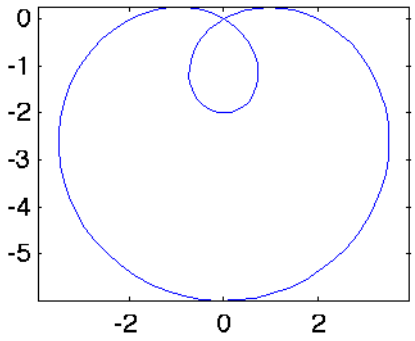


19.

```
theta=linspace(0,2*pi);
r=2-4*sin(theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```

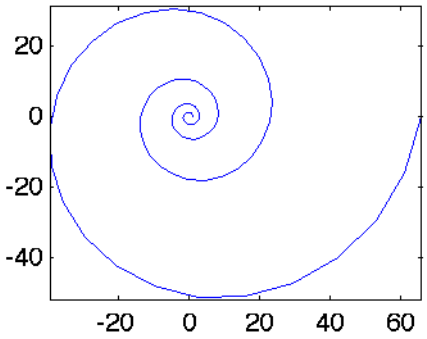
17.

```
theta=linspace(0,4*pi);
r=1+2*cos(theta/2);
x=r.*cos(theta);
y=r.*sin(theta);
plot(x,y)
axis equal
```



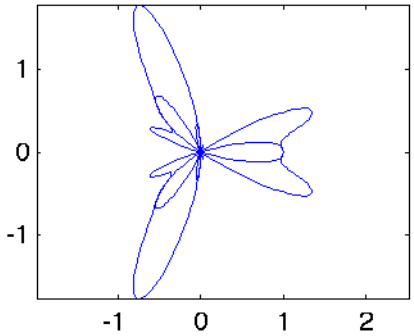
21.

```
theta=linspace(0,8*pi);
r=exp(theta/6);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```



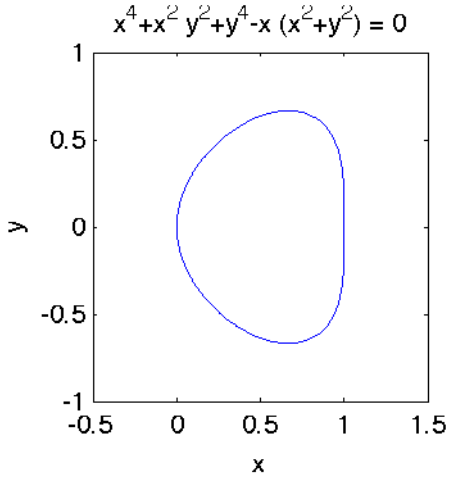
23.

```
theta=linspace(0,2*pi,500);
r=sin(4*theta).^4+cos(3*theta);
[x,y]=pol2cart(theta,r);
plot(x,y)
axis equal
```



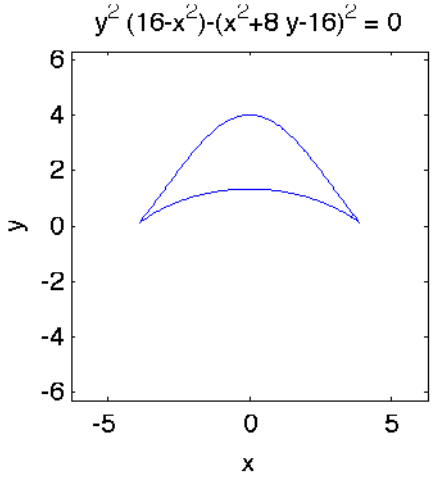
25.

```
fun2='x^4+x^2*y^2+y^4-x*(x^2+y^2)';
ezplot(fun2,[-0.5,1.5,-1,1])
```



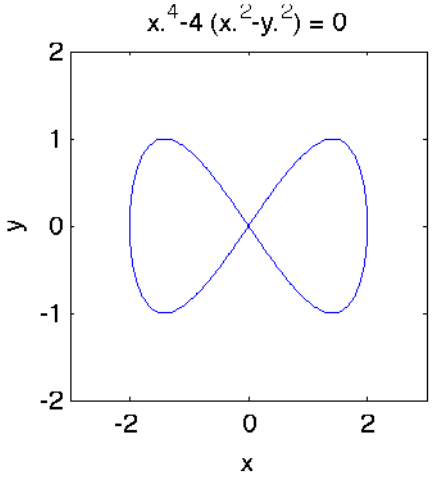
27.

```
fun2='y^2*(16-x^2)-(x^2+8*y-
16)^2';
ezplot(fun2)
```



29.

```
fun2=@(x,y) x.^4-4*(x.^2-y.^2);
ezplot(fun2,[-3,3,-2,2])
```



31.

```
fun2=@(x,y) 16*y.^2-x.^3.*(8-
x);
ezplot(fun2,[-1,10,-8,8])
```