2.2 An Optimization Problem

Determine the rectangle with the biggest area that can be drawn into the $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ *ellipse.*

Enter the equation of the ellipse and draw it with the help of the **implicitplot** procedure from the **plots** package. In order to understand the task more easily, draw a rectangle into the ellipse as well. Naturally Maple is unable to visualize the ellipse with the *a* and *b* indefinite parameters that's why we have to substitute values in the places of the *a* and *b* parameters.

```
> x^2/a^2+y^2/b^2=1
```

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1 \qquad (1)$$

```
> ellipszis := subs(a=3, b=2, (1))
```

$$ellipszis := \frac{1}{9}x^2+\frac{1}{4}y^2=1 \qquad (2)$$

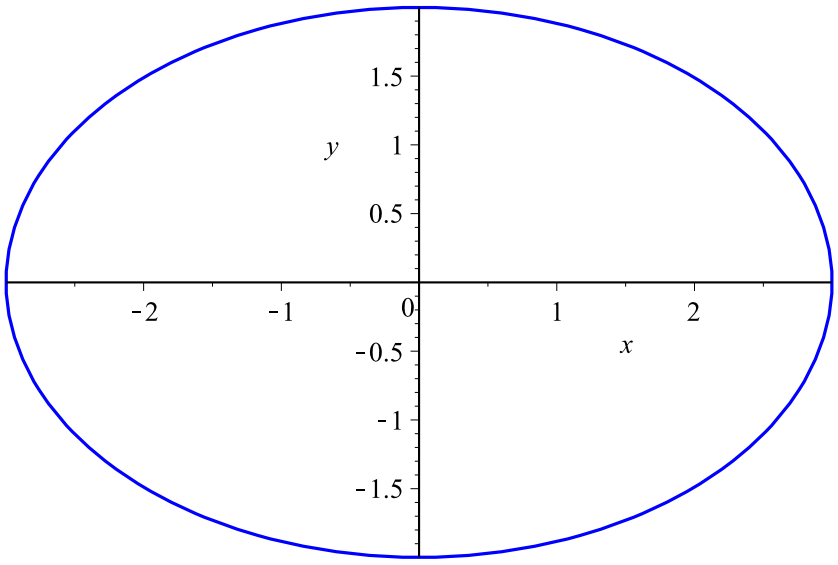
The name *ellipszis* variable was created from the equation of the ellipse containing indefinite parameters in such a way that we substituted 3 into the place of *a* and 2 into the place of *b*. Notice that the **subs** procedure allows us to give a sequence of equalities with which we describe the sequence of the **sequential** substitutions. As a first step, the system substitutes three to all the locations of the variable *a* then it substitutes two to all the locations of the variable *b* in the expression created in this way. The system repeats it as many times as many parameters are given to the **subs** procedure. In the worksheet which can be found at the end of this chapter we explain the differences between sequential and **simultaneous** substitutions. In our case the two kinds of substitutions, concerning their results, do not differ from each other.

So the name *ellipszis* already contains the equation of one concrete ellipse which is an implicit function thus it has to be drawn with the **implicitplot** procedure. We are going to use two procedures of the **plots** package in the case of the worksheet thus we may apply the **with(plots)** command so that we could refer to the procedures in the package with their short names.

The call of the **implicitplot** resembles the call of the **plot** procedure. Its first parameter is the equation of the implicit function to be drawn. Its second and third parameters are the domains on the *x* and *y* axes and finally these are followed by the usual options. It is important to note here that contrary to the **plot** procedure, it is mandatory to give the domains concerning the two axes in the case of the **implicitplot**.

```
> with(plots):
> implicitplot(ellipszis, x=-3..3, y=-2..2, color=blue, scaling=constrained, title='Ellipszis');
```

Ellipszis



Preparations are needed before the drawing of the rectangle. First, we need an arbitrary $[x_1, y_1]$ point of the ellipse. Let us observe that due to the typical symmetries of the ellipses the point $[x_1, y_1]$ obviously determines uniquely the rectangle that can be drawn into the ellipse. The vertices of the rectangle are $[x_1, y_1], [-x_1, y_1], [-x_1, -y_1]$ and $[x_1, -y_1]$ coordinated points.

> $x_1 := 1.5$

$$x_1 := 1.5 \quad \textbf{(3)}$$

> $subs(x = x_1, ellipszis)$

$$0.2500000000 + \frac{1}{4} y^2 = 1 \quad \textbf{(4)}$$

> $y_1 := fsolve(\%, y, 0..3)$

$$y_1 := 1.732050808 \quad \textbf{(5)}$$

Since we want to draw an arbitrary rectangle, we have chosen the value 1.5 for x_1 and substituted it into the equation of the ellipse. Attention! We are dealing with an implicit function thus in this case the result is an equality and not immediately the value of the function itself.

We solved this equality with the **fsolve** procedure for y and received the value of y_1 for which we had been looking for. Why did we use the **fsolve** procedure instead of the **solve**? The answer is that because the real approximation of the value of the function is adequate for our purposes. But what is more important is that the equation to be solved has two solutions which the **solve** procedure gives us but we only need one of them, e.g. the positive one. In the case of **solve** we could have selected that out of the sequence of the results only with further commands. On the contrary, **fsolve** accepts an interval as an option with which we can give the root clearly.

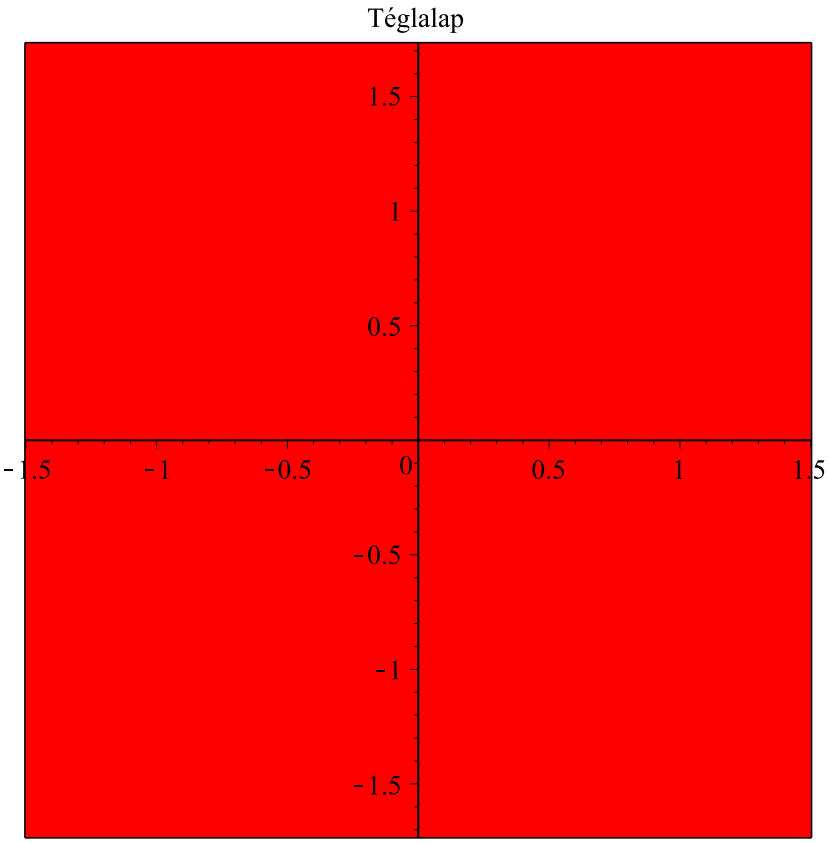
It is also recommended to take a closer look at the result of the **subs** instruction mentioned above. Isn't it surprising that we find a real approximation and an exact rational number within a Maple object? And isn't this in contrast with the strongly emphasised requirements of the exact arithmetic? Well, the solution is not so surprising but it is rather unusual. The answer is that we substituted 1.5 instead of $\frac{3}{2}$ into the equation. Number 1.5 is a floating-point number and in this case Maple gives the result as a floating point to the significant places the number of which was given in the **Digits** environment variable. Those readers who are keen on experimenting should try the $x_1 = \frac{3}{2}$ substitution.

After this we can create and draw the rectangle.

```
> T := [[x1, y1], [-x1, y1], [-x1, -y1], [x1, -y1]]
```

$$T := [[1.5, 1.732050808], [-1.5, 1.732050808], [-1.5, -1.732050808], [1.5, -1.732050808]] \quad \textbf{(6)}$$

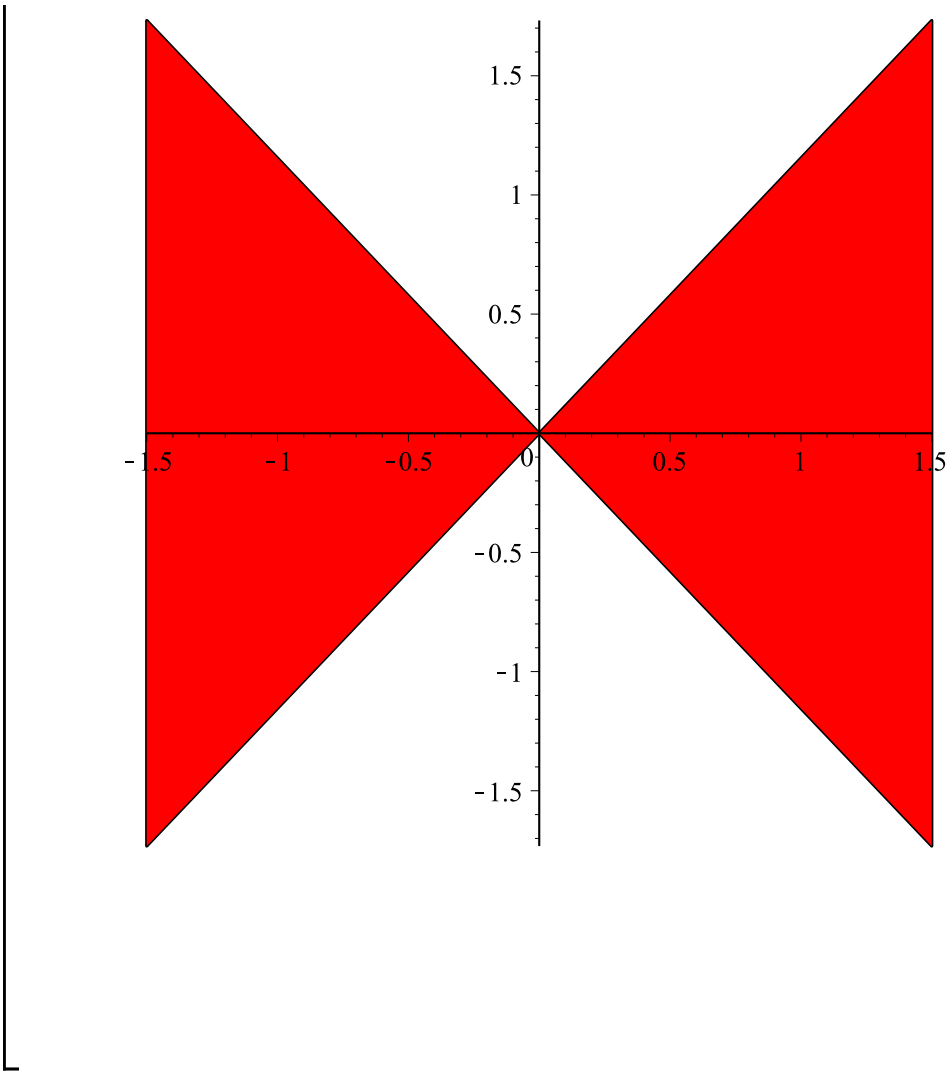
```
> polygonplot(T, color = red, title = "Téglalap")
```



We also have to mention the **polygonplot** procedure from the **plots** package. It waits as a first parameter for a list of two-element lists the way we prepared that in the variable *T*. Each of the two-element lists describes the coordinates of a vertex of the polygon to be drawn. The other parameters are the usual options of the **plot** procedure.

Let's have a look at the result of the next **polygonplot** procedure call.

```
> polygonplot([[x1, y1], [-x1, -y1], [-x1, y1], [x1, -y1]], color = red)
```



The graph has become a little bit crisscrossed although we did nothing else but to exchange the two middle elements of the *T* list. The **polygonplot** supposes that the two-element lists which are next to each other in the list are the coordinates of adjacent vertices thus it connects these with a line. It matters, therefore, in which order the vertices of the polygon to be visualized are written down.

We have a blue ellipse and a red rectangle and still we are not very satisfied because we would like to see these shapes in one graph. We have already drawn functions in the same coordinate system with the **plot** procedure. But now we want to display the result of two different procedures (**implicitplot** and **polygonplot**) in one coordinate system.

The creators of Maple were aware of this need and they choose a pretty creative solution. The drawing procedures of Maple work in two steps. As the first step of the drawing the procedures create a so-called **plot structure** the representation of which is executed in an independent second step. Since the creators paid attention to the fact that the different drawing procedures should generate the same structured plot structures, it was enough to introduce a procedure (**display** procedure) which provides the display of plot objects of arbitrary numbers.

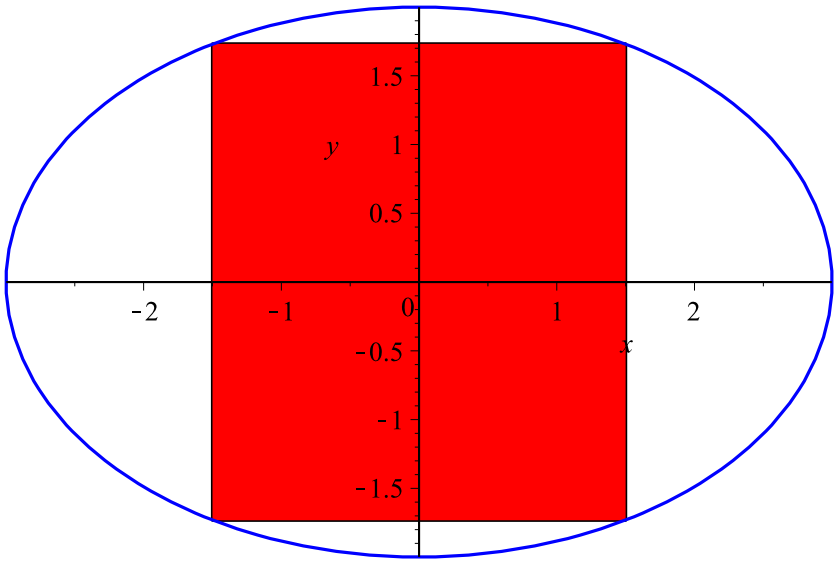
According to this, both the **implicitplot** and the **polygonplot** procedures create a plot object each of which we can put in a variable and then we can display the content of these variables with the **display** procedure. The **display** procedure can also be found in the **plots** package. If we had not given the *with*(*plots*) command, then we would have to use the full name of **plots[display]** instead of the short form **display** in the following instructions.

```
> pe := implicitplot(ellipszis, x = -3 ..3, y = -2 ..2, color = blue)
                                  pe := PLOT(...)                                  (7)

> pt := polygonplot([[x1, y1], [-x1, y1], [-x1, -y1], [x1, -y1]], color = red):

> display([pt, pe], scaling = constrained, title = `Ellipszisbe rajzolt teglalap`);
```

Ellipszisbe rajzolt teglalap



We have received what we wanted to see.

This applied technique is so typical of Maple that we constantly meet it. That's why we highly recommend our readers to understand this method fully. Besides we would like to draw your attention to the output of the **plot** procedures. The system does not show the whole plot structure. It only indicates with the *pe* := *INTERFACE_PLOT*(...) output that the plot structure was created in the variable *pe* as a result of the given command. Since it is not too informative, the display of the output was disabled in the second command.

The options (**color**, **scaling**) of the **display** procedure can easily confront with the options given in the plot objects to be displayed. For example, what would have happened as a result of *display*([*pt*, *pe*], *color* = *green*); ? Would the graph have been green? Or what would happen if we give the *display*([*pt*, *pe*], *scaling* = *unconstrained*) command while we wrote the *scaling* = *constrained* option in the pe plot object? Again, we highly recommend our readers to experiment and gain experience. Spend time on it. The acquirement of Maple is beneficial.

We have strayed a little bit far from our task. So the task is to determine the rectangle of the biggest area

that can be drawn in the $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ ellipse. We have our tools. Let's do the task.

Let's start with the fact that we can forget those variables and other Maple objects which we have created so far. We will use only the gained experience for the rest of the worksheet. The **restart** procedure behaves the way it has to. It makes Maple behave as if we had restarted it. The variables forget their previous values and the system resets itself. Naturally the effect of the earlier given *with*(*plots*) command has also become invalid thus we have to execute it again.

```
> restart
> with(plots):
```

We also have to enter the general ellipse. At least you can practise the creation of the exponents and the fractions in **2-D Math**. Just a quick reminder: the exponent can be created with **ALTGR+3** and the input of the fraction should be started with the / (**SHIFT + 6**) characters as a result of which we can give the numerator and the denominator separately. We can switch between them with the **up** and **down arrow** keys.

```
> x^2/a^2 + y^2/b^2 = 1
```

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad \textbf{(8)}$$

Our aim is to determine the area function of the rectangle drawn in the ellipse. Due to the symmetries typical of the ellipse this area is the fourth time of the area of the rectangle belonging to the first quadrant. Assume that x is that side of the rectangle belonging to the first quadrant which lies on the x axis. We can get the other side of the rectangle by determining the second coordinate of the x abscissa point of the ellipse. But there are two of these points as we can see in the output of the command below.

```
> solve((8), y)
```

$$\frac{\sqrt{-x^2+a^2}\,b}{a}, -\frac{\sqrt{-x^2+a^2}\,b}{a} \qquad \textbf{(9)}$$

Anyway, now we can easily enter the area function which we were looking for. Notice that we can index the instruction labels just like any other names. The creation has two steps. First, open the dialogue window with **CTRL+L** and there give the value of the label in the **LabelValue** field. After the system inserted the label, we can switch the cursor into subscript with the **underscore** and there we can give the preferred index.

```
> (9)[1]
```

$$\frac{\sqrt{-x^2+a^2}\,b}{a} \qquad \textbf{(10)}$$

```
> 4 · x · (10)
```

$$\frac{4x\sqrt{-x^2+a^2}\,b}{a} \qquad \textbf{(11)}$$

The domain of the area function is the]0 ..*a*[open interval.

```
> ter := (11)
```

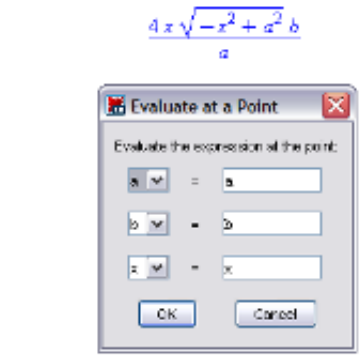
>

$$ter := \frac{4\,x\,\sqrt{-x^2+a^2}\,b}{a} \qquad \textbf{(12)}$$

>

Naturally we can only display the function if we give values to the a and b parameters. We could do this with the usage of the $subs(a=3, b=2, ter)$ command but now let's practise the usage of the **context menus**.

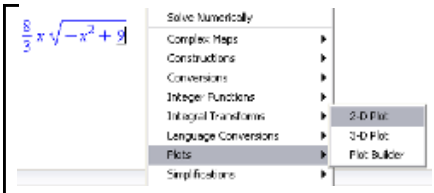
Open the context menu on the output line and choose the **Evaluate at a Point** command. Overwrite a with 3 and b with 2 in the input fields of the pop-up dialogue window.



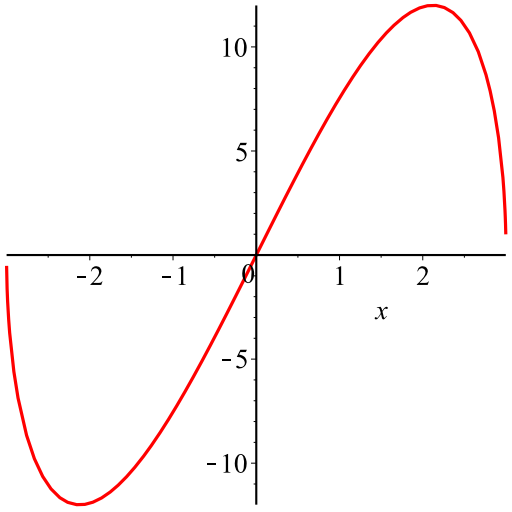
> $eval(\ \textbf{(12)},\ [a=3, b=2, x=x])$

$$\frac{8}{3}\,x\,\sqrt{-x^2+9} \qquad \textbf{(13)}$$

Context menu supports the drawing of the expressions. To do this, choose the **2-D Plot** command of the **Plots** menu. As a result, the $smartplot(\ \textbf{(12)}\ )$ command is generated and we get the curve of the area function.



> $smartplot(\ \textbf{(13)}\ )$

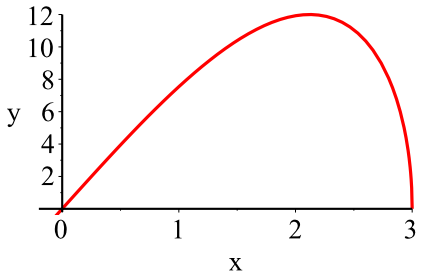


The **smartplot** did not inquire much about the domain. It drew the function in the -3..3 interval. But we are only interested in the 0..3 interval which we can also set with the usage of the context menu.

The context menu opened from the output of the **smartplot** command supports the settings of the different attributes of

the plot object. Choose the **Ranges** command of the **Axis** menu.	

In the pop-up dialogue window, set the [-0.2 ..3] domain to the x-axis and the [-0.2 ..12] domain to the y-axis. After the confirmation of the settings the system overwrites the previous graph and we can see the following on the screen.



The graph beautifully illustrates the behaviour of the area function. Calculate the critical point of the area function and the extremum itself.

```
> d/dx ter
```

$$\frac{4\sqrt{-x^2+a^2}\,b}{a} - \frac{4x^2 b}{\sqrt{-x^2+a^2}\,a} \qquad \textbf{(14)}$$

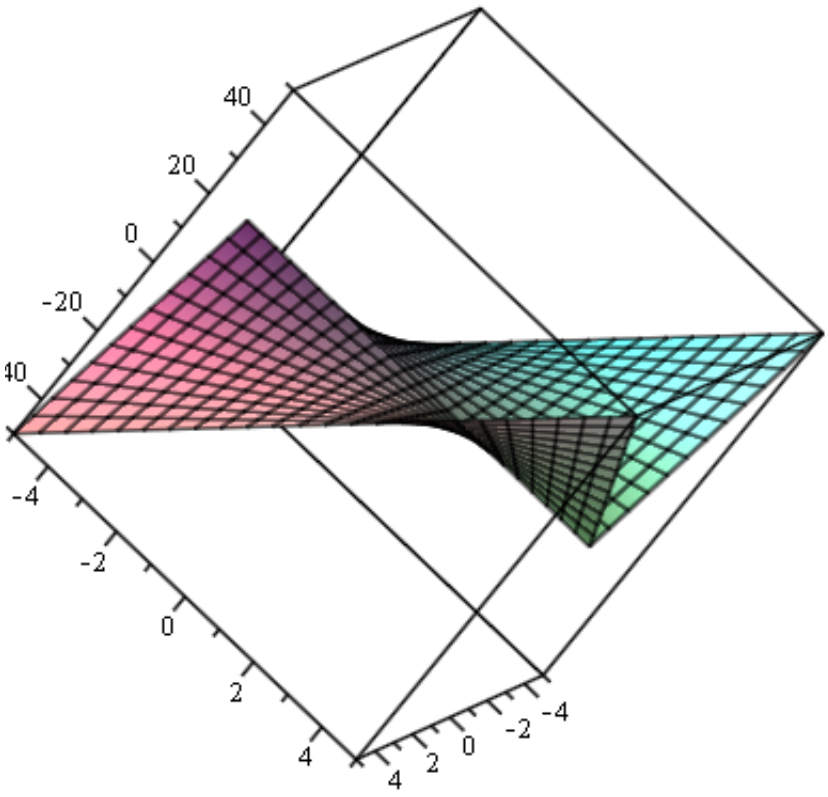
```
> solve(%, x)
```

$$\frac{1}{2}\sqrt{2}\,a,\ -\frac{1}{2}\sqrt{2}\,a \qquad \textbf{(15)}$$

```
> maximum := subs(x = (15)[1], ter);
```

$$maximum := 2\sqrt{a^2}\,b \qquad \textbf{(16)}$$

```
> smartplot3d[a, b]( (16) )
```



So the maximum area is the function of two variables of the *a* and *b* parameters. The **smartplot3d** command draws the surface of this function which we created with the usage of the context menu.

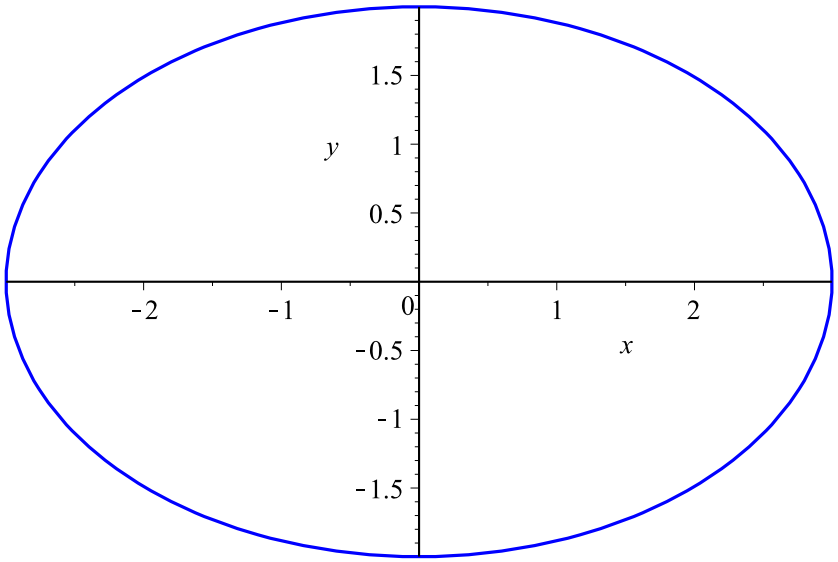
Step by step we have learnt everything about the rectangles that can be drawn in an ellipse but we also needn't disclaim about a fine animation. For this, we have to become familiar with the repetition statement of Maple. Maple, like any other advanced system, provides repetition statements with the help of which we can execute more command sequences without entering them again and again.

Since the **for** (the repetition) statement is the first compound command we have met we have to discuss about typing mode. The most important to emphasise is that compound commands have to be given in **1-D Math**. Maple User's Guide also warns the users about this. While we still acknowledge that here is another limitation of **2-D Math** we can notice that one can switch between **1-D Math** and **2-D Math** with **F5**.

Since the default input mode is **2-D Math** we start the entering of the lines below with pressing **F5**. Remember that in **1-D Math** the instructions must be closed with semicolons (;) or if we do not want to see the output then it must be closed with a colon (:). We can enter the next line with **Shift+Enter**. **Enter** makes all the commands of the created execution group be executed.

```
> ell:=x^2/a^2+y^2/b^2 = 1:
  Ell:=eval(ell,{a=3,b=2}):
  pe:=plots[implicitplot](Ell,x=-3..3,y=-2..2,color=blue,scaling=
```

```
constrained):
pe;
```

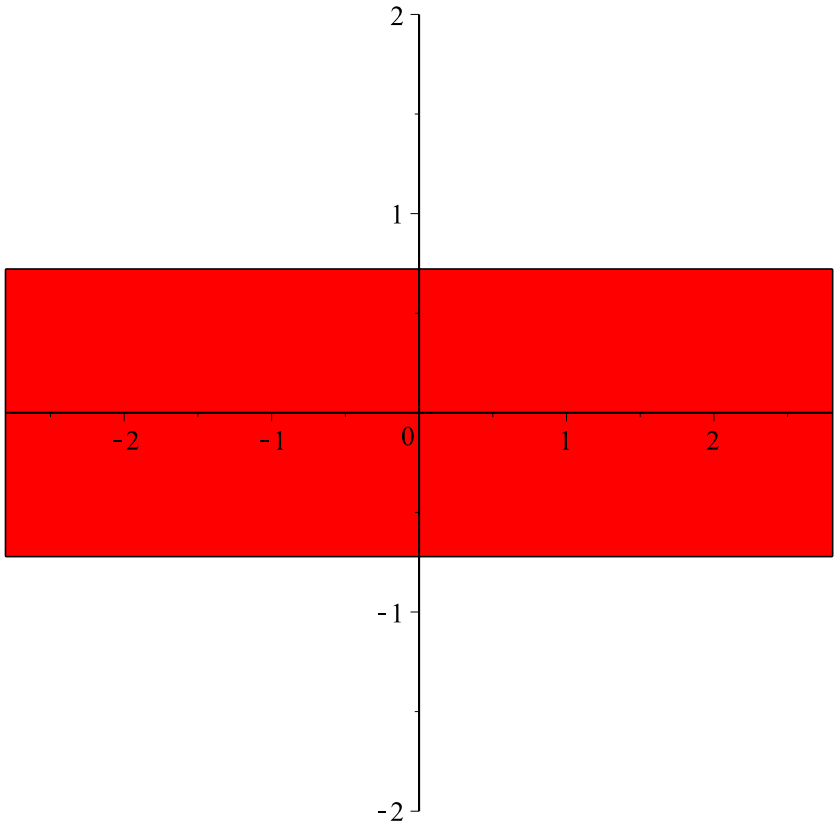



```
> elly := solve(ell, y)[1]; Elly := eval(elly, {a = 3, b = 2})
```

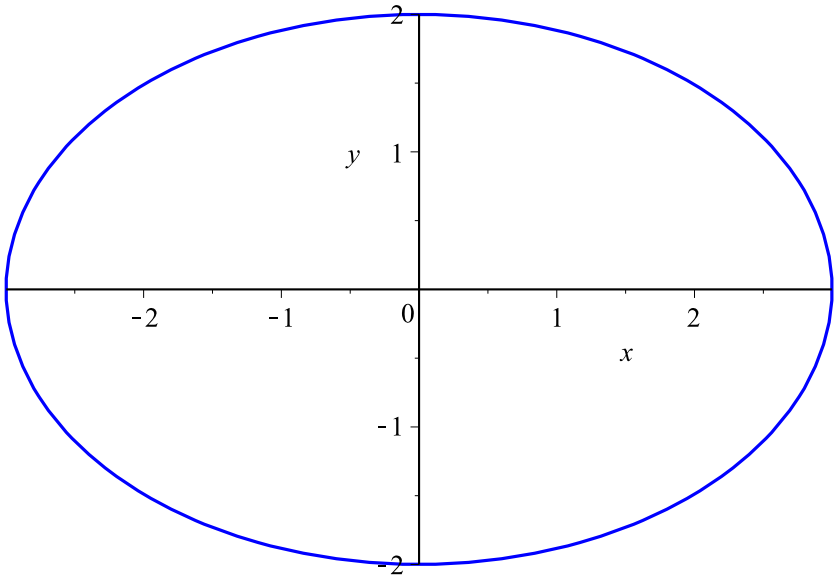
$$elly := \frac{\sqrt{-x^2 + a^2}\, b}{a}$$

$$Elly := \frac{2}{3}\sqrt{-x^2 + 9} \qquad \textbf{(17)}$$

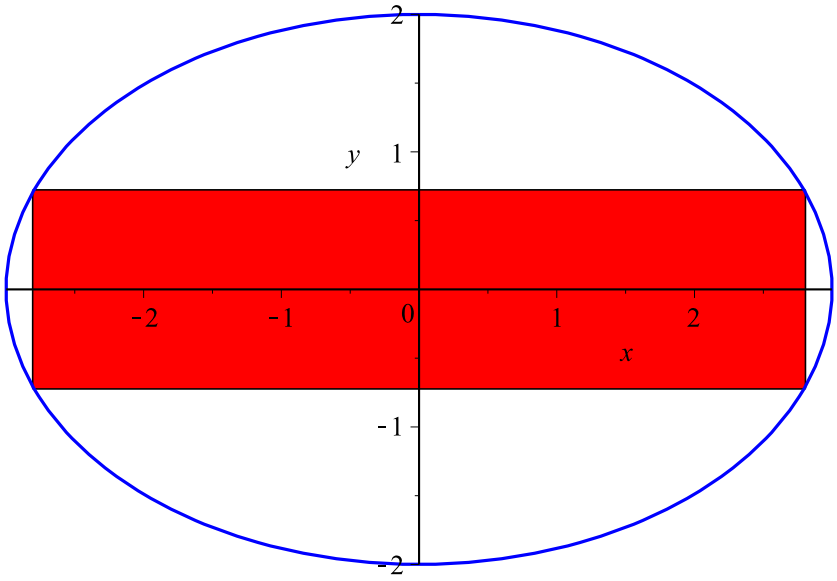
```
> K:=NULL:
  for z from -2.8 to 2.8 by 0.2 do:
   fe:=eval(Elly,x=z):
   p:=plots[polygonplot]([[z,fe],[z,-fe],[-z,-fe],[-z,fe]],color=
  red):
   K:=K,p:
  od:
  plots[display]([K],insequence=true);
```



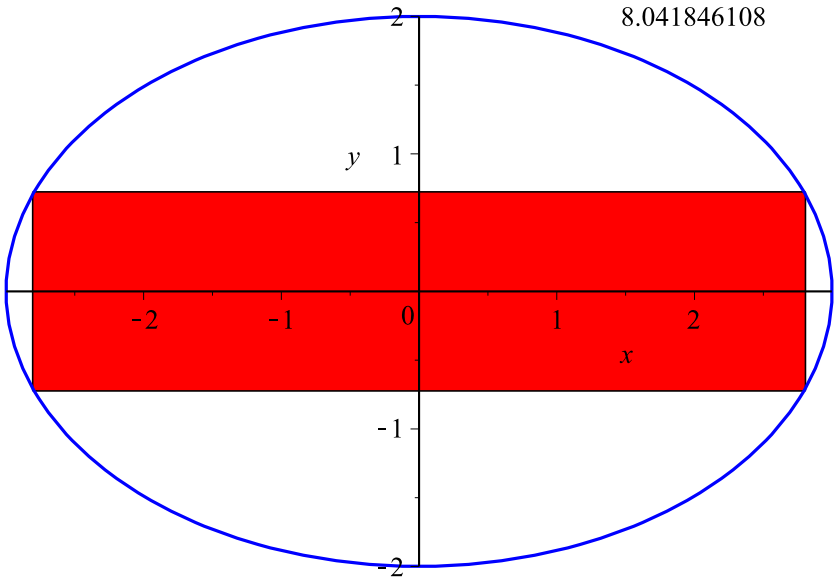

```
> K:=pe:
  for z from -2.8 to 2.8 by 0.2 do:
   fe:=eval(Elly,x=z):
   p:=plots[polygonplot]([[z,fe],[z,-fe],[-z,-fe],[-z,fe]],color=
  red):
   K:=K,p:
  od:
  plots[display]([K],insequence=true);
```




```
> K:=NULL:
  for z from -2.8 to 2.8 by 0.2 do:
   fe:=eval(Elly,x=z):
   p:=plots[polygonplot]([[z,fe],[z,-fe],[-z,-fe],[-z,fe]],color=
  red):
   d:=plots[display]({p,pe}):
   K:=K,d:
  od:
  plots[display]([K],insequence=true);
```




```
> K:=NULL:
  for z from -2.8 to 2.8 by 0.2 do:
   fe:=eval(Elly,x=z):
   p:=plots[polygonplot]([[z,fe],[z,-fe],[-z,-fe],[-z,fe]],color=
  red):
   t:=abs(4*z*fe):
   t:=plots[textplot]([2,2,convert(t,string)]):
   d:=plots[display]({p,pe,t}):
   K:=K,d:
  od:
  plots[display]([K],insequence=true);
```



The syntax of the **for** command which we used:

- **for** *cycle variable* **from** *initial value* **to** *final value* **by** *stepsize* do
- ... the instructions of the body of the loop...
- **od;**

It makes the instructions between the **do** and **od** keywords, which form the body of the loop, be executed to each value of the *cycle variable*. The first value of the cycle value is the *initial value*, its next value is *initial value* + *stepsize* , then *initial value* + 2 · *stepsize* ... etc. This lasts until the *cycle variable*c is smaller than or equal to the *final value* .

The formula *Elly* is the explicit function of the half of the ellipse. While the cycle variable *z* runs from –2.8 to 2.8 by 0.2, each of the [*z*, *fe*] pair is the point of the ellipse, *p* is the plot object of the rectangle determined by the [*z*, *fe*], [*z*, -*fe*], [-*z*, -*fe*], [-*z*, *fe*] ellipse points and *d* is the display object containing *p* and *pe* plot objects.

The area function *t* contains the expression that describes the area function. Thus the value of the variable area is the area of the *t* rectangle determined to three significant digits. We converted this value to a string with the **convert** procedure so that we could display it with the using of the **textplot** at the point with coordinates (2,2).

Observing the $K := K, d;$ command! Here the comma appears as an infix operator and it results in the affixing of two sequences sequentially. Initially K is an empty sequence (NULL). During the first run of the cycle the d plot object is created which contains the ellipse, the rectangle and its area and we append it to K. Thus we have received a one-element sequence. During the second run of the cycle a new d value is created (with the ellipse, a new rectangle and its area) which we append to K again. Now we have received a two-element sequence. Keep on doing this until the value of the cycle variable is bigger than 2.8. At the end of the cycle we get the sequence of plot objects in K and this sequence can be displayed with the display command. Notice that we put K into square brackets ([K]) because the display procedure requires a set or a list as a first parameter. We can get a list from a sequence if we put it into square brackets.

The *insequence = true* option of the **display** procedure describes that the system shows the images sequentially like an animation. Naturally the rectangles can be seen at once with the omitting of the *insequence = true* option, that is, with the input of the $display([K])$ command.

Try out the animations. Have fun!

What Have You Learnt About Maple?

- The general form of the sequential substitution is $subs(x_1 = e_1, x_2 = e_2, expression)$. First, it makes all the appearances of x_1 be substituted with e_1 and in the created *expression* it makes all the appearances of x_2 be substituted with e_2. Naturally the number of the variables to be substituted is not restricted to two.
- The **implicitplot** is a procedure of the **plots** package. It is used to display implicit functions. Its syntax is:

$$implicitplot(equality, x = a..b, y = c..d, options).$$

The *equality* describes the implicit function with the independent variable of which is x and its dependent variable is y. The second and the third parameters are the domains on each axis. The *options* are the ones which we became familiar with at the **plot** procedure. Their usage is not mandatory.

- The **polygonplot** is a procedure of the **plots** package. It is used to draw polygons. Its form is:

$$polygonplot([[x_1, y_1], ..., [x_n, y_n]], options).$$

The two-element lists given as a parameter are the coordinates of the adjacent vertices of an n angle. The options are the ones which we became familiar with at the **plot** procedure. Their usage is not mandatory.

- The system can be restarted with the **restart** procedure.
- The absolute value of the expression can be calculated with the **abs** procedure.
- The **plot3d** is a library procedure. Its syntax is

$$plot3d(f, x = a..b, y = c..d, options),$$

in which f is an expression of two variables in x and y. As a result, a three dimensional surface is created on the chosen domain. The options are the ones which we became familiar with at the **plot** procedure. Their usage is not mandatory.

- The $convert(f, string)$ command converts the f Maple object into a character sequence in every case when this conversion is interpretable.
- The simplest form of the **for** command is:

for *cycle variable* **from** *initial value* **to** *final value* **by** *stepsize* **do**

... commands of the body of the loop...

od

It makes the body of the loop be executed again and again to the values of *cycle variable* for *initial value* , *initial value* + *stepsize*, *initial value* + 2 · *stepsize* etc. This lasts until the *cycle variable* does not exceed the *final value*. Finally notice that instead of the **od-od** key words the **do-end do** key word pair can be used as well.

- The comma (,) infix operator is used to append sequences. So for the s_1 and s_2 sequences s_1, s_2 is that sequence which is created in a way that the elements of the s_2 sequence are written subsequent to the elements of the s_1.
- We can create a **list** or **set** from the *s* sequence by putting *s* between squared or curly brackets. Thus $[s]$ is the list and $\{s\}$ is the set that consists of the elements of the s.

Exercises

1. Draw the following implicit functions.
   - $x - y^2 = 0$
   - $x^2 - y^2 = 0$
   - $(x^2 + y^2 - 1)\ (x^2 + y^2 - 4) = 0$
2. Draw a regular pentagon, hexagon and octagon with the help of the **polygonplot** procedure.
3. The volume of a tank which is open at the top and has a shape of a cuboid is V. Under what size will its surface be the smallest?
4. Which is the rectangle the perimeter of which is 2p and the one that by rotating it around one of its sides then we get the biggest area solid of revolution?
5. Determine the rectangle of the biggest area that can be drawn in the curve of the $y = 2\sqrt{1 - \left(\frac{x}{3}\right)}$ function. Limit ourselves to the first quadrant?
6. Determine the rectangle of the biggest area that can be drawn in the curve of the $y = 2\cos\left(\frac{\mathrm{Pi}\, x}{6}\right)$ function. Let's constrain ourselves to the first quadrant.