The Electronic Journal of Mathematics and Technology, Volume 13, Number 3, ISSN 1933-2823

Investigating the Reflections of Light Rays Inside Ellipses with GeoGebra, Maxima and Maple

Guillermo Dávila-Rascón
davila@mat.uson.mx
Departamento de Matemáticas
Universidad de Sonora
Hermosillo, SON 83000
MEXICO

Wei-Chi Yang
wyang@radford.edu
Department of Mathematics and Statistics
Radford University
Radford, VA 24142
USA

Abstract

In this paper, we investigate the problem on how light rays are reflected inside a given ellipse by using technology tools. We first solve the problem in a quite general setting with GeoGebra and next we investigate some particular cases by means of the computer algebra systems Maxima and Maple®. We study several cases through some computer programs done with these CASs, with the purpose of answering questions about the periodicity of the light trajectories generated by the successive reflections of a beam of light on the ellipse.

1 Introduction

The problems addressed in this paper can be thought as problems in the theory of elliptical billiards, and therefore, we will be using some well known results from this subject in order to answer the questions we are interested in. Let us say that billiards are an important class of dynamical systems that were introduced by Birkoff around 1927 and can be thought as systems in which a particle moves freely inside a bounded region of the plane (two dimensional billiards) in such a way that every time the particle hits the boundary, it bounces elastically in the well known sense of geometrical optics: the angle of incidence is equal to the angle of reflection. Here, we assume that the billiard region is a convex domain of the plane and the billiard boundary is an ellipse, hence the name elliptical billiards [5].The purpose of this paper is to introduce some computational programs in order to investigate several specific problems about how light rays are reflected inside ellipses, according to the law of light reflection on smooth surfaces, the so called specular reflection. This law states that the angle of incidence is equal to the angle of reflection, where this angle can be measured relative to the reflecting surface, as shown in Figure 1.

Notice that if we measure the angles of the incident and reflected rays with respect to the normal line to the reflecting surface, then the law of reflection remains the same.

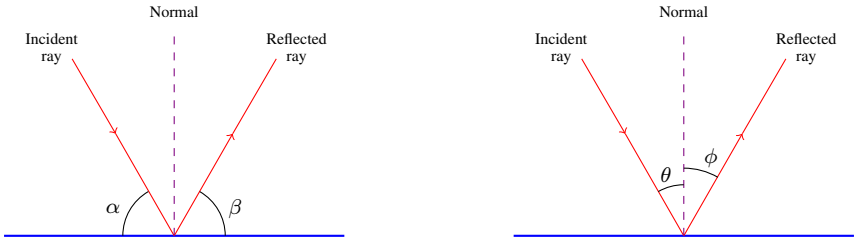


Figure 1: Reflection law: $\angle\alpha = \angle\beta$ and $\angle\theta = \angle\phi$.

In our case, the reflecting surface is an ellipse and the incidence and reflected rays at any point of the ellipse will be measured relative to the normal of the tangent line to the curve at that point, as illustrated in Figure 2.

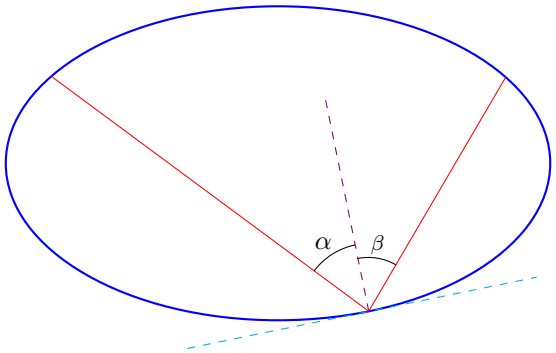


Figure 2: Reflection law on the ellipse: $\angle\alpha = \angle\beta$.

Thus, starting with an initial line representing a beam of light aimed to an arbitrary point on the ellipse, our computational program should be able to do the following:

1. calculate the coordinates of the intersection point on the ellipse and the line representing the incident ray;
2. calculate the equation of the tangent line at that point and the equation of the corresponding normal;
3. determine the equation of the line representing the reflected ray (this is done by reflecting the incident line through the normal line);
4. calculate the coordinates of the point of intersection of the reflected line and the ellipse;
5. repeat the process until some stopping condition is achieved;
6. verify that the last calculated point determines a line that coincides with the initial ray.
7. plot all the objects of interest;

Here, we shall recall that if a straight line in the plane has equation given by $ax + by + c = 0$ and if $(u, v) \in \mathbb{R}^2$ is an arbitrary point that is reflected through this line, then the coordinates (u', v') of the reflected point are given by [9]:

$$u' = u - \frac{2a(au + bv + c)}{a^2 + b^2}, \tag{1}$$

$$v' = v - \frac{2b(au + bv + c)}{a^2 + b^2}. \tag{2}$$

These formulae are extensively used in our computational programs. See Figure 3.

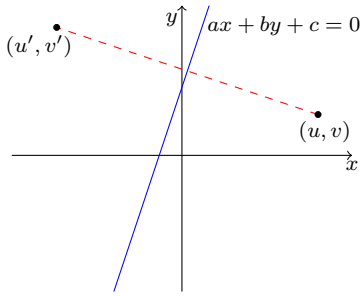


Figure 3: Reflecting a point through a straigth line.

The computational tasks described above will be accomplished by using two computer algebra systems, namely Maxima (http://maxima.sourceforge.net/) and Maple® (www.maplesoft.com), so the interested reader can have different options for performing her/his own computations and comparing the results reported in this paper. Nevertheless, because of the readiness and user's friendly environment provided by GeoGebra (www.geogebra.org), we first solve the problem in a quite general setting by using this dynamic geometry software.

In the case of elliptical billiards, trajectories have a well known behaviour: they always remain tangent to a confocal conic, that can be another ellipse or an hyperbola. More precisely, suppose the boundary Γ of our billiard is an ellipse in $\mathbb{R}^2$ with foci at points F_1 and F_2. If some segment of the billiard trajectory does not intersect the segment $\overline{F_1F_2}$, then no other segment of this trajectory intersects $\overline{F_1F_2}$ and all segments of the billiard trajectory are tangent to an ellipse γ with foci at F_1 and F_2. On the other case, if some segment of the trajectory does intersects $\overline{F_1F_2}$, then all segments of the billiard trajectory intersect $\overline{F_1F_2}$, and all of them are tangent to an hyperbola h with foci at F_1 and F_2 (See Figure 4). Elementary proofs of these assertions can be found in [3, 8].

The conics γ and h are called *caustics*. More generally, caustic curves in a billiard are curves that have the following property: if a segment of a billiard trajectory is tangent to such a curve, then the reflected segment is also tangent to the curve [8].

An important question concerning elliptical billiards is if a trajectory closes after a finite number of bounces. That is, let us suppose that a billiard trajectory starts at point P_0 on the ellipse and continues through points $P_1, P_2, \ldots$, also on the curve. We would like to figure out if the given trajectory eventually closes. For this to happen, it is necessary that after reaching a point P_{n-1} on the ellipse, the next point of the curve touched by the trajectory should be P_0. Thus, $P_n = P_0$, and the next point of reflection, namely P_{n+1} should satisfy $P_{n+1} = P_1$, and therefore, it should happen that $\overline{P_0P_1} = \overline{P_nP_{n+1}}$, so $P_{n+2} = P_2$, and so on. Note that in this case the trajectory is periodic.

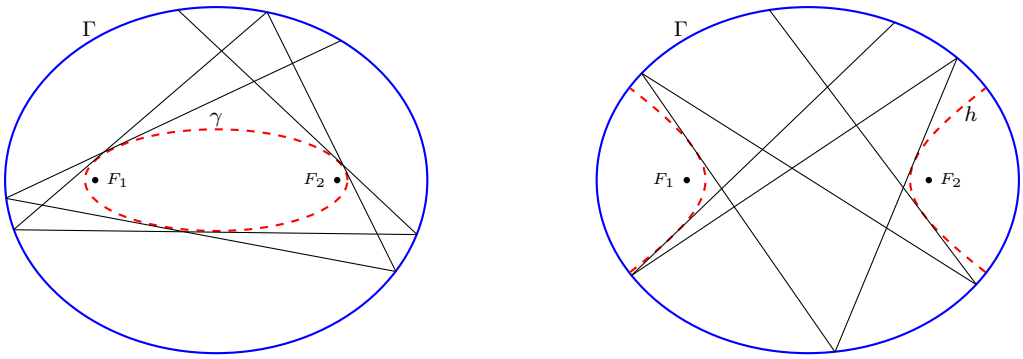


Figure 4: Trajectories in elliptical billiards.

Conditions for periodicity of trajectories inside conics were investigated by Cayley, as early as 1853, in connection with the very famous Poncelet's Theorem, also named Poncelet's Porism or Poncelet Closure Theorem [3, 4, 5, 6, 8]. In the case of ellipses, this theorem can be stated in the following terms: Suppose two ellipses in the plane are given, in such a way that there is a closed polygon inscribed in one ellipse and circumscribing the other. Then there exist infinitely many closed polygons behaving like the original one. In fact, every point of the ellipse is a vertex for one of those polygons [4, 5].

Although the history of Poncelet's Theorem is very interesting and there are many deep mathematical results in connection with this theorem, the conditions for periodicity obtained by Cayley are far beyond the scope of this paper. Instead, we can say that if we think of a two dimensional elliptical billiard as a dynamical system, then it is a completely integrable Hamiltonian system with two degrees of freedom and two independent constants of motion (first integrals), namely, the total energy of the system (say H, in our case, kinetic energy), and the the other one being the dot product of the two focal angular momenta, $L = L_1 \cdot L_2$, which can be calculated explicitly [2, 3, 7, 10].

Therefore, according to the Liouville-Arnold Theorem [1], the dynamics of the system takes place in a two dimensional invariant torus (Liouville torus), $\mathbb{T}^2 = \{(\theta_1, \theta_2) \bmod 2\pi\}$, and we can calculate *action-angle coordinates* $(I_1, I_2, \theta_1, \theta_2)$, where the action variables $I_j = I_j(H, L)$ depend only on the constants of motion, and are again, first integrals of the system. Thus, the dynamics of the billiard can be described by the two frequencies (ω_1, ω_2) which are related to the angle coordinates θ_i by $\omega_i = \mathrm{d}\theta_i/\mathrm{d}t$. In this situation, the condition for periodic motion is that ω_1/ω_2 be a rational number.

We will not go further in this direction nor calculate explicit conditions for periodicity since for doing so we need to write down the equations of motion for our billiard in, for example, elliptical coordinates (as in [2, 3, 7]), and then to perform all the necessary computations which takes considerable space. Instead, we recommend the interested reader to take a look at the cited references.

Thus, our goal is limited in the sense that we will only investigate, by means of some computer programs in GeoGebra, Maxima and Maple, if a given particular trajectory, defined by points P_1, P_2, $\ldots$, P_n, $\ldots$, in an elliptical billiard, closes after a finite number of bounces.

2 Setting of the problem

Consider an ellipse written in canonical form as

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \tag{3}$$

with semi-axes of lengths a and b, satisfying $\frac{a}{b} = \sqrt{3}$, as shown in Figure 5. Here, we have to say that there is no particular reason for choosing these settings for our ellipse, and since this is an exploratory paper any lengths for a and b can be chosen. In Section 3 we do not impose this constraint.

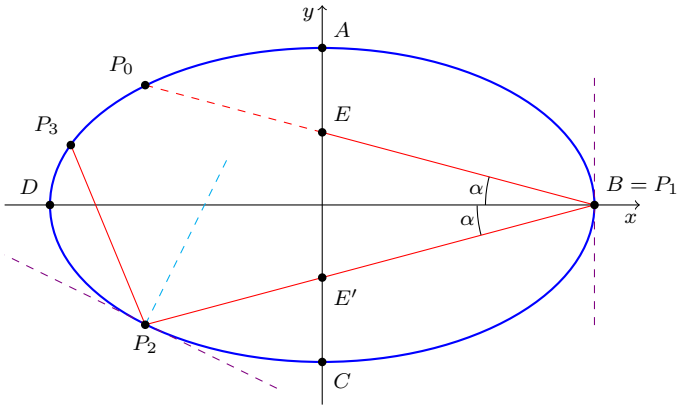


Figure 5: Ellipse corresponding to equation (3).

The interior domain bounded by the simple closed curve represented by equation (3) is a simply connected region and we choose an arbitrary point on this domain as the starting point from which a light ray is aimed into a point on the curve, which it is reflected according to reflection's law, and the reflected ray reaches another point on the ellipse where it is reflected again and so on.

More specifically, in the several situations we address here, the coordinates of an initial point E are given in the form $E = \big(0, s\big)$, for different values of parameter s, and the beam of light starts from here and it is aimed into point $B = P_1 = (a, 0)$ on the ellipse where, after reaching this point, it is reflected again. Here, we represent this ray by line segment $\overline{EB}$, as can be seen in Figure 5. The law of reflection tells us that the angle of incidence equals the angle of reflection, which are represented by α in Figure 5, both measured with respect to the normal line through B, that in this case turns out to be the x-axis. After being reflected, the light beam touches the ellipse at point P_2, represented by line segment $\overline{P_1P_2}$, and it is reflected again, reaching the ellipse at point P_3 (line segment $\overline{P_2P_3}$), and so on. The problem here is to determine if after a number n of successive reflections we get a point P_n on the ellipse in such a way that the reflected ray from this point coincides with line segment $\overline{P_nP_1}$ and thus, this ray coincides with the original one $\overline{EB}$, as depicted in Figure 5. Once again, we have to say that there is no particular reeason for choosing points E and B as we did. Moreover, in Section 3 we eliminate this restricion.

Notice that in order to solve this problem, the point P_n should coincide with point P_0 which is defined by the initial light ray from point E since it is one of the two intersection points on the ellipse determined by the straight line passing through points E and B. That is, $P_n = P_0$, but additionally, one should verify that $\overline{P_0P_1} = \overline{P_nP_{n+1}}$ (see Figure 5).

Thus, we need to run a computer program that calculates every one of the points $P_0, P_1, P_2, P_3, \ldots$ on the ellipse that are touched by the reflected rays, as well as the lines representing those rays, namely, segments $\overline{P_0P_1}$, $\overline{P_1P_2}$, $\overline{P_2P_3}, \ldots$, and, in some way, to figure out if we can reach the situation described above.

On the other hand, because of floating point calculations, it is not easy to determine computationally when a pair of points on the ellipse will coincide. Thus, we need to specify in advance when two points will be numerically considered as (practically) the same point. Therefore, we have to set the tolerance error we will be willing to accept. For example, if we set this numerical error to be ε, with $0 < \varepsilon \ll 1$, and for points $x, y \in \mathbb{R}$, we have $\mid x - y \mid < \varepsilon$, then we will say that $x \cong y$. Similarly, for points $p = (x_1, y_1), q = (x_2, y_2) \in \mathbb{R}^2$ satisfying $\parallel p - q \parallel = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)} < \varepsilon$, we are able to say that $p \cong q$. Notice that we cannot go beyond this, unless it is specifyied another smaller value for the numerical error. The symbol "$\cong$" can be read as "approximately equal".

Thus, if we set $\varepsilon = 0.0001$, then $\mid x - y \mid < 0.0001$ means that x and y coincide, at least, up to the third digit after the decimal point and both will be treated, for practical considerations, as the same point, as is the case for $x = 7.123456789$ and $y = 7.123546789$, for example. For points $p, q \in \mathbb{R}^2$, saying that $\parallel p - q \parallel < 0.0001$ means that the Euclidean distance between these points is less than the specified tolerance. As for example, for $p = (7.123456789, -11.987654321)$ and $q = (7.123446789, -11.987664321)$ we have $\parallel p - q \parallel \approx 0.0000141421356 < 0.0001$.

We mention this because we are faced with that situation in our computer programs since we have to compare, for closeness, each one of the points $P_1, P_2, P_3, \ldots$ with point P_0 and need to determine if for some n, the point P_n is close enough to P_0 and the reflected rays $\overline{P_0P_1}$ and $\overline{P_nP_{n+1}}$ can be considered as the same.

3 Solving the problem with GeoGebra

Taking advantage on the readiness and easy of use of the dynamic mathematics software GeoGebra, we originally started out with some explorations of the problem but later we were able to solve it in a more general setting than that presented in Section 2, in the sense that we are not constrained by condition $a/b = \sqrt{3}$, and we can vary the lengths a and b of the ellipse's semi-axes, as well as points E and P_1 (see Figure 5).

In what follows, we will describe how to use GeoGebra in order to get some good answers and we achieve this with a relatively simple program consisting of very few lines which show us the powerfulness of this dynamic geometry software.

First, after launching GeoGebra, we define two sliders of numeric type, named **a** and **b**, for the ellipse's semi-axes, both with minimum and maximum values at 0.01 and 5.0, respectively, and incrementing them by a step of 0.01. Next, introduce the equation of the ellipse directly at the input bar:

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

Notice that the label assigned by GeoGebra to this conic is **c** and will be used later. Also, in order to have a better picture, we remove the grid, and both axes `x` and `y` on the graphics view of our GeoGebra window. Now, define another numeric type slider, say **t**, that will be used to control the first incidence point; this slider's minimum and maximum values are equal to 0 and 2π, respectively, incrementing it by 0.01. Next we introduce a point P at the input bar:

```
P = (a*cos(t),b*sin(t))
```

This point can travel along the entire curve and will be the point on the ellipse where the beam of light is reflected for the first time.

Now we set another two sliders of numeric type, that we denote by `Ex` and `Ey`, that will control the x and y coordinates of point E, recalling that this is the point where the light ray starts, and it is defined by

```
E = (Ex,Ey)
```

The sliders, `Ex` and `Ey`, have minimum and maximum values equal to -5 and 5, respectively, incrementing them by a step of 0.01. Notice that so defined, point E runs inside the square $[-5,5]\times[-5,5]$ and can be used to represent, virtually, any point inside the ellipse.

The line through points E and P intersects the conic **c** at another point, that we denote by P_0, and which is gotten by typing directly in the commands bar:

```
P_0 = Intersect(c,Line(E,P),1)
```

where the number 1 in this command line stands for the first point of intersection. The second point of intersection is P, but for iterative purposes we give it another name by setting:

```
P_1 = Intersect(c,Line(E,P),2)
```

Points P_0 and P_1 will be the first two entries of a list that will consists of point P_0 and the points on the ellipse where the beam of light is reflected.

Next, we calculate the tangent line to the ellipse at point P_1:

```
Tg1=Tangent(P_1,c)
```

Afterwards, we get the line through point P_1 that is perpendicular to line `Tg1`:

```
Perp1=PerpendicularLine(P_1,Tg1)
```

Now, we are in position to reflect point P_0 over line `Perp1`, in order to get the second point where the beam touches the curve:

```
Pr1=Reflect(P_0,Perp1)
```

Notice that for $t=0$, the reflected point `Pr1` is on the ellipse, but varying the slider **t** the situation changes and the reflected point falls outside the curve. This has to be taken into account in what follows.

We have all the elements that are necessary to define a list of points by using the iterative capabilities of GeoGebra. For that purpose, define a numeric type slider, say **n**, with 1 and 200 as its minimum and maximum values, respectively, incrementing in steps of value 1, and thus the values of **n** are the positive integers from 1 through 200. Clearly, the reader can set these values as she/he prefers.
Now we type in the input bar the following commands:

```
IterationList(Intersect(c,Reflect(Line(A,B),
              PerpendicularLine(B,Tangent(B,c))),2),
              A,B,{P_0,P_1},n)
```

Some explanation is needed here. The GeoGebra command

```
IterationList(<Expression>, <Variable Name>,...,<Start Values>,
              <Number of Iterations>)
```

give us a list of length $n+1$, where n is the number of iterations. The elements of this list are gotten by iterating the defining expression, starting at the given values.

In our case the expression being iterated is defined by the `Intersect` command and calculates the second (`2`) point of intersection of the ellipse **c** with a line. Thus, we can think of this expression as one of the form `Intersect(c, Line L, 2)`, where `Line L` is the object that changes, and with each iteration, the result obtained is a point that is appended to a list. Notice that the varying arguments in our expression are A and B, and represent points. The first value for A is the point P_0 and the first value for B is the point P_1, as settled down with the `'start value' {P_0,P_1}` (Also notice that we have used brackets to define these starting values.)

Let us describe more explicitly this process (we refer the reader to Figure 5):

1. Determine `Line L` as follows:
   (a) Calculate the line passing through points A and B (`Line(A,B)`). Thus, the first line calculated is the one defined by points P_0 and P_1.
   (b) Calculate the tangent line to the ellipse at point B (`Tangent(B,c)`). The first of these tangents is the one that is tangent to ellipse **c** at point P_1.
   (c) Calculate the perpendicular line to `Tangent(B,c)` at point B. In the first iteration this will be the perpendicular line to `Tangent(`P_1`,c)` at point P_1.
   (d) Reflect the line obtained in 1a onto the line calculated in 1c (`Reflect(Line(A,B),PerpendicularLine(B,Tangent(B,c)))`.)
2. Calculate the second point of intersection of the ellipse **c** with the line obtained in 1d (`Intersect(c, Line L, 2)`.) The first point thus obtained is P_2.
3. Next, A and B take the values P_1 and P_2, respectively, giving point P_3 as the result, and so on. This process give us a list of points $\{P_0, P_1, P_3, \ldots\}$ (Remark that if we do not specify a name for the list obtained with the command `IterationList`, GeoGebra automatically assigns a label to it. In our case, it was L_1 and we will use this label in what follows.)

Finally, we define a sequence of segments by joining consecutive points of the list L_1 by typing directly in the input bar:

```
Sequence(Segment(Element(L_1,i),Element(L_1,i+1)),i,1,n)
```

The `Sequence` command produces a list L_2 whose elements are the lengths of the segments $\overline{P_0P_1}$, $\overline{P_1P_2}$, $\overline{P_2P_3}$, ... Each one of these segments represents the reflection on the ellipse of a beam of light that started at point E and was aimed into point P_1 on the curve. Those reflected rays can be fully appreciated in Figure 6, which summarizes all we have described here, but moreover, we can see that under the given conditions, we have a closed trajectory.

Remark that Figure 6 was produced by using the facilities within GeoGebra for exporting our work, and we have done so by rendering the figure gotten in GeoGebra's graphic view into a figure in

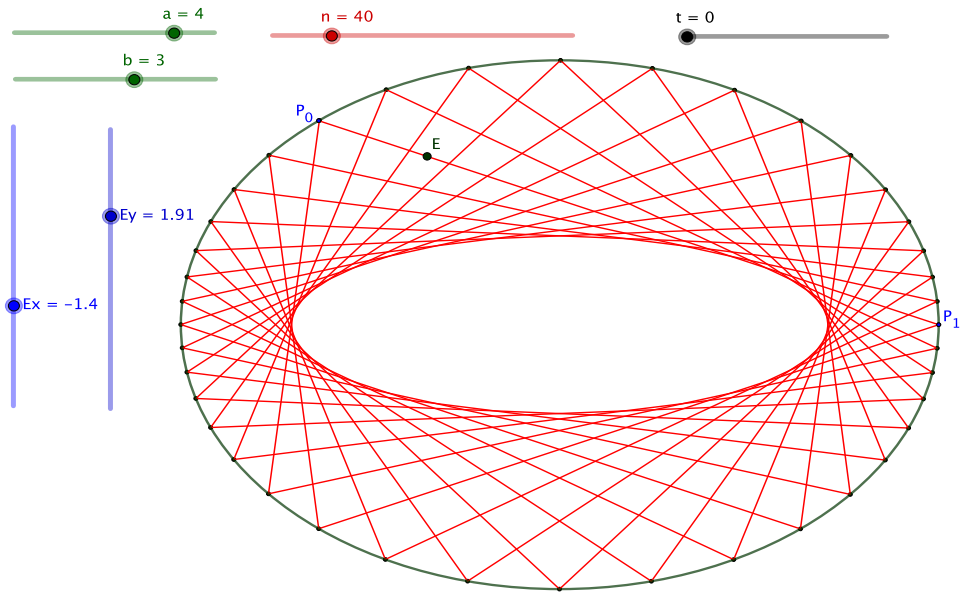


Figure 6: Reflection points inside an ellipse, with GeoGebra.

PDF format. Notice also how is clearly depicted in Figure 6 a caustic curve that is the tangent curve to the reflected rays represented by the line segments. In this case, the caustic is a confocal ellipse, but varying the slider **t**, and consequently the point P_1, we are able to appreciate the other kind of caustic in the case of elliptic billiards, namely, a confocal hyperbola (See Figure 7b.)

On the other hand, the decorations in Figure 6 (color and size of points; color, lengths and positions of sliders; color and line width for segments, etcetera) are up to the reader and we remark that in order to have a more clear image in our GeoGebra's graphic view, many of the objects constructed in the process leading us to solve our problem are not shown. But again, the reader can easily change the properties of all the involved objects.

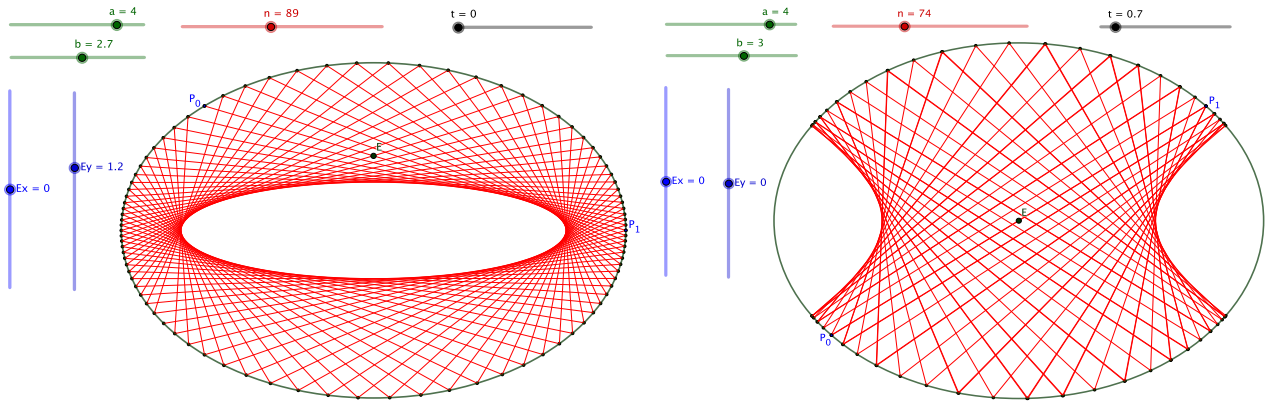
Finally, in Figure 7 we show two other cases for illustrative purposes in which occur the two types of caustic curves.

4 Solving the problem with the CAS Maxima

In this section we document several computer programs, done with the CAS Maxima, in order to solve some particular problems concerning the reflections of light rays inside ellipses, we described above. We investigate different cases and get some interesting results that can be used for educational purposes in the classroom.

4.1 Case $E = \big(0, b\tan(\pi/12)\big)$

We document here one of the computer programs we have writen in order to solve our problem for the case $E = \big(0, b\tan(\pi/12)\big)$, where a and b are related by $a = b\sqrt{3}$, and $a = 1.5$ is the length of



(a) Do we have a closed trajectory in this case? (b) One case of a caustic curve being an hyperbola.

Figure 7: Two other cases with GeoGebra

the major semi-axis of the ellipse given by (3), shown in Figure 5.
First, we reset all values assigned by Maxima to any variable in the program and load the drawing machine. The `kill` command is very useful every time we need to start all over again.

```
(%i1) kill(all);
(%i2) load(draw)$
```

Now, we define a Maxima function (procedure) for calculating the coordinates (xr, yr) of a point (p, q) after being reflected through a straight line having equation $Ax + By + C = 0$, according to formulae (1)-(2):

```
(%i3)Reflex(p,q,A,B,C) := ([xr,yr],
       xr : p - 2*A*(A*p + B*q + C)/(A^2 + B^2),
       yr : q - 2*B*(A*p + B*q + C)/(A^2 + B^2),
       Preflx : [xr,yr])$
```

Here, the variables p, q, A, B, C, are local and do not affect any other appeareance afterwards. We will be using this function extensively in our program. Next, introduce the lenghts of the ellipse's semi-axes and set the equation of the ellipse in Maxima's implicit form in order to plot it later:

```
(%i4) a:1.5$ b:a/sqrt(3.0)$
(%i5) Ellipse : implicit(x^2/a^2 + y^2/b^2 = 1, x,-a-0.5,a+0.5,
                         y,-b-0.5,b+0.5)$
```

By implicit differentiation of the canonical equation of the ellipse (3), we can calculate the slope of the tangent line at any point of the ellipse and set it as a function that will be repeteadly called:

```
(%i6)Slope(x,y) := - (b^2/a^2)*(x/y)$
```

As was said above, we start with an initial point $E = (x_0, y_0)$ (point E in Figure 5), so let us introduce it to Maxima:

```
(%i7) x0:0.0$ y0:ev(b*tan(%pi/12),numer)$
(%i8) E:[x0,y0]$
```

The function `ev(..., numer)` is used here in order to get a floating point value for `y0`. Now, introduce point P_1 which is the point where the initial light ray touches the ellipse for the first time

(point $B = P_1$ in Figure 5):

```
(%i9) P[1]:[a,0.0]$
```

We use points E and P_1 to compute the slope of the straight line passing through them and the equation for this line is set in Maxima's implicit form:

```
(%i10) m[0]: (P[1][2] - E[2])/(P[1][1] - E[1])$
(%i11) line[0] : implicit(y - m[0]*x + m[0]*P[1][1] - P[1][2] = 0,
                  x, -a-0.5,a+0.5, y,-b-0.5,b+0.5)$
```

Notice that `line[0]` contains the light ray $\overline{EP_1}$. Next, we calculate the intersection points of `line[0]` with the ellipse in order to determine point P_0, as depicted in Figure 5. For performing these calculations we use the function `solve`, requiring the output to be in numerical form by means of the command `ev(..., numer)`. We also use the function `ratprint` with the value `false` since we do not want to see in the computer screen the messages of Maxima informing us about the many conversions of floating point format into fractions.

```
(%i12) ratprint :  false$
(%i13) Pellips : ev(solve([x^2/a^2 + y^2/b^2=1, y-m[0]*x
                  + m[0]*P[1][1]-P[1][2]=0],[x,y]),numer);
```

Usually, the Maxima output we get is of the form

```
[[x=1.5,y=0],[x=-1.299038105676657,y=0.4330127018922199]]
```

Notice that the first term in this list is the point `[x=1.5,y=0]` which we already have (it is $B = P_1$). Nevertheless, sometimes the output could be in the form

```
[[x=-1.299038105676657,y=0.4330127018922199],[x=1.5,y=0]]
```

and we have to swap for getting the correct point. We choose it through the following commands:

```
(%i14) if abs(rhs(Pellips[1][1]) - P[1][1]) < 0.0001 then
         P[0]:[rhs(Pellips[2][1]),rhs(Pellips[2][2])]
         else
         P[0]:[rhs(Pellips[1][1]),rhs(Pellips[1][2])]$
```

Here, `rhs(Pellips[i][1])` refers to the right hand side of `first` entry of the i-th term in the list.

We will use point P_0 as a comparison point by measuring the Euclidean distance from this point to each and all of the incidence points $P_1, P_2, P_3, \ldots$, in order to see if we can get a closed trajectory inside the ellipse. That is, if there is a point P_n such that $P_0 \cong P_n$, and then calculate incidence point P_{n+1}, so that $\overline{P_0P_1} = \overline{P_nP_{n+1}}$, for which it is necessary that $P_{n+1} = P_1$.

Since we are interested in the points of the ellipse touched by the reflected rays, we collect those points and the straight lines representing those reflected rays, into a list of points and a list of lines. We initiate these lists as follows:

```
(%i15) Points:[P[1]]$
(%i16) Lines:[line[0]]$
```

The first point to be reflected on the normal line through point P_1 is point E, and the image of this reflection is point E' (see Figure 5). Afterwards, we calculate the equation of the line passing through this reflected point and point P_1 which contains segment $\overline{P_1P_2}$ representing the reflected ray from point P_1, as shown in Figure 5. The equation for this normal line (x-axis) is $y = 0$ and hence we introduce the parameters values $a = 0$, $b = 1$, and $c = 0$ of the straight line $ax + by + c = 0$,

according to formulae (1)-(2). See also Figure 3.

```
(%i17) a0:0.0$ b0:1.0$ c0:0.0$
```

Now, the most important part of the Maxima's computer program is the loop we present next.

```
(%i18) for i:1 step 1 while (sqrt((P[0][1]-P[i][1])^2
                              + (P[0][2]-P[i][2])^2) > 0.01)
       do(
          Pr[i]:Reflex(x0,y0,a0,b0,c0),
          m[i]:(Pr[i][2] - P[i][2])/(Pr[i][1]- P[i][1]),
          line[i]:implicit(y - m[i]*x + m[i]*P[i][1]-P[i][2]=0,
                           x, -a-0.5,a+0.5, y,-b-0.5,b+0.5),
          Lines:append(Lines, [line[i]]),
          Pinters[i]:ev(solve([x^2/a^2 + y^2/b^2=1,y-m[i]*x
                        + m[i]*P[i][1]-P[i][2]=0],[x,y]),numer),
          if abs(rhs(Pinters[i][1][1]) - P[i][1]) < 0.00001 then
          Pnew[i]:[rhs(Pinters[i][2][1]), rhs(Pinters[i][2][2])]
          else
          Pnew[i]:[rhs(Pinters[i][1][1]), rhs(Pinters[i][1][2])],
          Points:append(Points, [Pnew[i]]),
          mtg[i]:Slope(Pnew[i][1],Pnew[i][2]),
          TgLine[i]:implicit(y-mtg[i]*x+mtg[i]*Pnew[i][1]
                        - Pnew[i][2]=0, x,-a-0.5,a+0.5,
                                        y,-b-0.5,b+0.5),
          mperp[i]:-(1/mtg[i]),
          LinePerp[i]:implicit(y-mperp[i]*x+mperp[i]*Pnew[i][1]
                         - Pnew[i][2]=0, x, -a-0.5,a+0.5,
                                         y,-b-0.5,b+0.5),
          kill(a0),
          kill(b0),
          kill(c0),
          a0 : -mperp[i],
          b0 : 1,
          c0 : mperp[i]*Pnew[i][1] - Pnew[i][2],
          P[i+1] : Pnew[i],
          kill(x0),
          kill(y0),
          x0 : Pr[i][1],
          y0 : Pr[i][2]
          );
```

Several remarks are in order in relation with this code.

1. The loop starts with $i = 1$, advancing in whole steps $2, 3, 4, \ldots$ and stops at step n, when the (euclidean) distance between a (to be calculated) point P_n and P_0 is less than 0.01. This is done with the `while` operator. (There is no diference if we set the tollerance to be 0.001, as can be checked out.) Notice also that the first comparison is between points P_0 and P_1, that are already

defined.

2. `Pr[i]` denotes the point obtained by reflecting the point (x_0, y_0) over the normal line through the (to be calculated) point P_i on the ellipse. This normal line has equation $a_0x + b_0y + c_0 = 0$, and it is labeled as `LinePerp[i]` since it is perpendicular to `TgLine[i]`, the tangent line to the ellipse at point P_i. The point we reflect first is E, as pointed out above, and thus $Pr_1 = E'$ (see Figure 5).

3. Next, use Pr_i and P_i to compute slope `m[i]` in order to determine `line[i]`, passing through them. After this, calculate `Pnew[i]` as the intersection point of the ellipse and `line[i]`, and use it to compute `mtg[i]`, the slope of `TgLine[i]` which is the tangent line to the ellipse at `Pnew[i]`. After setting the equation in Maxima's implicit form for `TgLine[i]`, we can determine the line perpendicular to it, `LinePerp[i]`, at `Pnew[i]`. Note that `Pnew[i]` becomes point P_{i+1}, and so on.

4. After using (x_0, y_0) in step i, the new values for x_0 and y_0 in step $i + 1$ are the coordinates of point `Pr[i]`. Thus, `Pr[i+1]` uses these new values to get the reflection of `Pr[i]` over the normal line `LinePerp[i]` passing through point P_{i+1}. Notice also how the values of the parameters `a0`, `b0`, `c0` are changed to agree with those of `LinePerp[i]`.

Continuing with our program, the following command give us the time required to perform the calculation on the loop until it finishes:

```
(%i19) time(%);
(%o19) [25.978]
```

As we pointed out above, we collect the points on the ellipse touched by the reflected rays in the list labeled `Points`. The number of elements of this list is gotten as follows:

```
(%i20) length(Points);
(%o20) 47
```

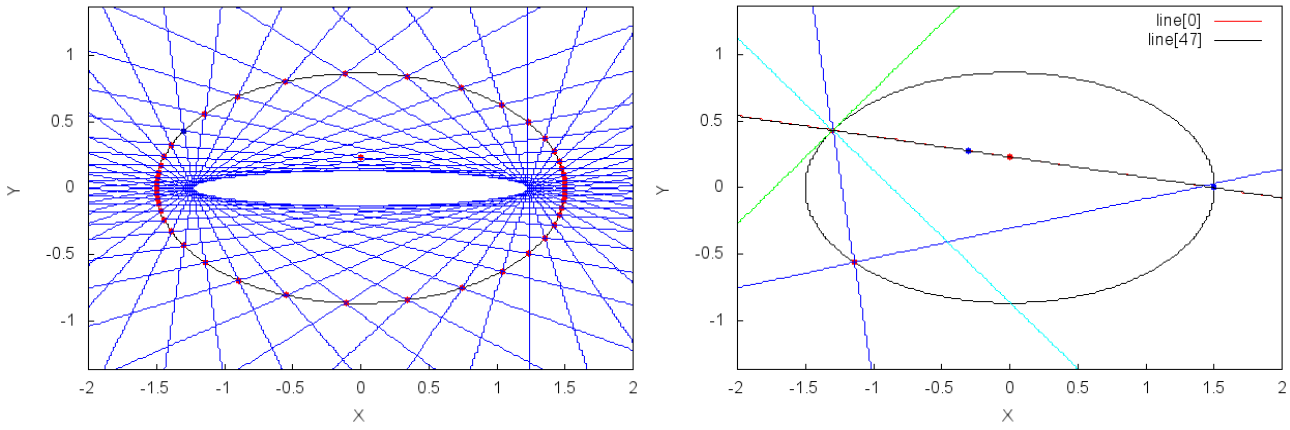
Now we can plot all objects of interest as shown in Figure 8a:

```
(%i21) wxplot_size:[650,500]$
(%i22) wxdraw2d(color=black, Ellipse,
          color=red,point_type=filled_circle, points([E]),
          points(Points),
          color=dark-blue,point_type=filled_circle,
          points([P[0]]),
          color=blue, [Lines],
          proportional_axes=xy
          );
```

Let us make some comments on what we have obtained.

1. The graph (Figure 8a) clearly suggests that there is a confocal ellipse, tangent to each of the line segments inside the original ellipse, namely, that a caustic exists for the conditions of our problem.

2. We also have a good hint that the trajectory closes after 47 iterations.



(a) Reflection of a light ray in the ellipse given by (3).

(b) Segment $\overline{P_0P_1}$ in `line[0]` is "equal" to segment $\overline{P_{47}P_{48}}$ in `line[47]`.

Figure 8: Outcomes of Maxima for the case $E = \big(0, b\tan(\pi/12)\big)$.

3. In fact, comparing points P_0 and P_{47}, we get the following:

```
(%i23) P[0]; P[46]; P[47];
(%o23) [-1.299038105676657,0.4330127018922199]
(%o24) [-1.142678264684355,-0.561036654601091]
(%o25) [-1.300770762793338,0.4312754041452849]
```

   Notice how "close" are to each other:

```
(%i26) sqrt((P[0][1]-P[47][1])^2 + (P[0][2]-P[47][2])^2);
(%o26) 0.00245363080871805
```

   That is, the Euclidean distance between them satisfies $\mathrm{d}(P_0, P_{47}) < 0.0025$.

4. Since the `while` loop stops at $i = 47$, the last line calculated was `line[46]`. Thus, unless we modify the code of the program, we cannot go further than this. However, in what follows we make some more calculations.

In order to get a better picture, let us compare the straight lines containing the segments $\overline{P_{47}P_1}$ and $\overline{P_0P_1}$.
First, we need to calculate the equations of the tangent line to the ellipse at point P_{47}, as well as that of the normal line at that point; thus, the slopes are, respectively:

```
(%i27) mtg[47] : Slope(P[47][1],P[47][2])$
(%i28) mperp[47] : -(1/mtg[47])$
```

Therefore, the equations of the referred lines are, respectively:

```
(%i29) TgLine[47]:implicit(y - mtg[47]*x + mtg[47]*P[47][1]
                      - P[47][2]=0,x, -a-0.5,a+0.5, y,-b-0.5,b+0.5)$
(%i30) LinePerp[47]:implicit(y - mperp[47]*x + mperp[47]*P[47][1]
                     - P[47][2]=0,x, -a-0.5,a+0.5, y,-b-0.5,b+0.5)$
```

Both of these lines are depicted in Figure 8b, above.

Next, we reflect point P_{46} over line `LinePerp[47]` to get a point that we label as `Pr[47]`:

```
(%i31) Pr[47]:Reflex(P[46][1],P[46][2],-mperp[47],1,
                     mperp[47]*P[47][1]-P[47][2])$
```

The slope of the line passing through points P_{47} and Pr_{47} is the following:

```
(%i32) m[47] : (Pr[47][2] - P[47][2])/(Pr[47][1]- P[47][1])$
```

We are now in position to calculate the equation for `line[47]` and to find the intersection point of this line with the ellipse:

```
(%i33) line[47]:implicit(y - m[47]*x + m[47]*P[47][1]-P[47][2]=0,
              x, -a-0.5,a+0.5, y,-b-0.5,b+0.5);
(%i34) Pinters[48]:ev(solve([x^2/a^2 + y^2/b^2=1,y-m[47]*x
            + m[47]*P[47][1]-P[47][2]=0],[x,y]),numer);
```

Once again, we need to choose the correct point:

```
(%i35) if abs(rhs(Pinters[48][1][1]) - P[47][1]) < 0.0001 then
        Pnew[48]:[rhs(Pinters[48][2][1]),rhs(Pinters[48][2][2])]
        else
        Pnew[48]:[rhs(Pinters[48][1][1]),rhs(Pinters[1][1][2])];
```

We get the following answer:

$$\text{Pnew}[48] = [1.499999819831534, 4.244625489607216 * 10^{-4}]$$

As pointed out before, `Pnew[48]` becomes point P_{48}, and if everything goes as expected, P_1 and P_{48} would be practically the same. Recall that $P_1 = (1.5, 0)$. Thus, let us compute the distance between them:

```
(%i36) dist_P48_P1 : sqrt((Pnew[48][1]-P[1][1])^2
            + (Pnew[48][2]-P[1][2])^2);
```

The output gotten can be written as

$$\text{dist\_P48\_P1} = [1.499999819831534, 4.244625489607216 * 10^{-4}]$$

That is, we can say that

$$\mathrm{d}(P_{48}, P_1) = 4.244625871981054 * 10^{-4} = 0.0004244625871981054 < 0.0005,$$

which tell us that $P_{48} \cong P_1$. Therefore one could expect that `line[47]` would be "equal" to `line[0]`. We illustrate this in Figure 8b by plotting the objects of our interest:

```
(%i37) wxdraw2d(color=black, Ellipse,
                color=red,point_type=filled_circle,
                points([E,P[1],P[46],P[47]]),
                color=blue, points([Pr[47],Pnew[48]]),
                color=blue, [line[45],line[46]],
                color=green,TgLine[47],
                color=cyan,LinePerp[47],
                key="line[0]",color=red, line[0],
                key="line[47]",color=black,line[47],
```

```
                proportional_axes=xy
                );
```

Therefore, the conclusion is that starting with a light ray at point $E = \big(0, b\tan(\pi/12)\big)$ inside the ellipse given by equation (3) and reflecting it into point $P_1 = (1.5, 0)$, the successive reflection make that the trajectory of the ray closes after 47 reflections.

4.2 Case $E = \big(0, a\tan(\pi/12)\big)$

It is clear that this case is completely similar to the previous one regarding our Maxima program and the only thing that needs to be changed in that program is the initial point E. Nevertheless, we will see how different the overall behaviour turns out to be.
Thus, we will only document here the parts that are different in the Maxima code. Let us starting by introducing the initial point $E = (x_0, y_0)$:

```
(%i7) x0:0.0$ y0:ev(a*tan(%pi/12),numer)$
(%i8) E:[x0,y0]$
```

The light ray starts from this point and it is aimed at point $P_1 = (a, 0)$ which is introduced the same way as in input `(%i9)` at previuos Maxima code. After that, we calculate the slope of the line passing through points E and P_1 and use it to get coordinates of the point P_0 on the ellipse (see Figure 5 and inputs `(%i13)` and `(%i14)` in Section 4.1). We get the following answer:

$$P_0 = [-0.9683428667784539, 0.6613904777964087]$$

Again, in order to obtain the equation for the line passing through points P_1 and (the not yet calculated) point P_2, we first reflect point E over the normal line through P_1, that is, over the x-axis, getting the point E' as depicted in Figure 5, but corresponding to point `Pr[1]` in the `while loop` of the Maxima code at input `(%i18)`. The line by points P_1 and E' is then computed. Point P_2 is the point of intersection of this line and the ellipse, which is obtained through the `solve` function in the `while loop`. After this, the program computes the tangent and normal lines at P_2 in order to reflect point `Pr[1]` over this normal line, thus obtaining point `Pr[2]`, which is used to calculate the line through it and P_2. This line intersects the ellipse at point P_3, and so on.
As in the prevoius case, the while loop stops at step n when the distance between points P_n and P_0 is less or equal to 0.01. Here we notice the first differences between the two cases:

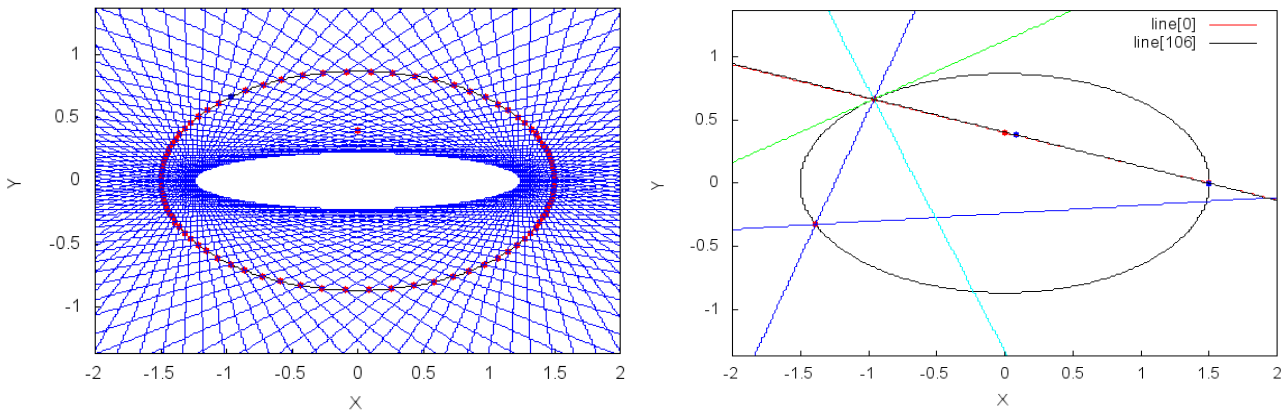
```
(%i19) time(%);
(%o19) [60.562]
```

and also the number of points calculated:

```
(%i20) length(Points);
(%o20) 106
```

Plotting the objects of interests give us a very nice picture as shown in Figure 9a.
Now let us continue and verify if everything goes as expected. We proceed first by comparing the points P_0 and P_{106}, the last point computed:

```
(%i23) P[0]; P[105]; P[106];
(%o23) [-0.9683428667784539,0.6613904777964087]
(%o24) [-1.389666606461468,-0.3259993470780111]
(%o25) [-0.9617684510713951,0.6645804306788179]
```



(a) Reflection of a light ray in the ellipse given by (3).

(b) Segment $\overline{P_0P_1}$ in `line[0]` is "equal" to segment $\overline{P_{106}P_{107}}$ in `line[106]`.

Figure 9: Outcomes of Maxima for the case $E = \big(0, a\tan(\pi/12)\big)$.

We can observe the "closeness" between points P_0 and P_{106} is less than 0.0075:

```
(%i26) sqrt((P[0][1]-P[106][1])^2 + (P[0][2]-P[106][2])^2);
(%o26) 0.007307444237297534
```

Next, let us compute the straight line `line[106]`, since the last line computed in the `while loop` was `line[105]`. First, we calculate the slope of tangent and normal lines at point P_{106}:

```
(%i27) mtg[106] : Slope(P[106][1],P[106][2])$
(%i28) mperp[106] : -(1/mtg[106])$
```

Thus, the equations for these lines are, respectively:

```
(%i29) TgLine[106]:implicit(y - mtg[106]*x + mtg[106]*P[106][1]
            -P[106][2]=0,x,-a-0.5,a+0.5, y,-b-0.5,b+0.5)$
(%i30) LinePerp[106]:implicit(y-mperp[106]*x+mperp[106]*P[106][1]
            -P[106][2]=0,x,-a-0.5,a+0.5, y,-b-0.5,b+0.5)$
```

Now we reflect point P_{105} on the line `LinePerp[106]` in order to calculate `line[106]`. Call it `Pr[106]`:

```
(%i31) Pr[106]:Reflex(P[105][1],P[105][2],-mperp[106],1,
              mperp[106]*P[106][1] - P[106][2])$
```

The slope of `LinePerp[106]` can be computed next:

```
(%i32) m[106] : (Pr[106][2] - P[106][2])/(Pr[106][1]- P[106][1])$
```

Hence, the equation for `line[106]` is straightforward and we calculate the intersections points of this line with the ellipse and choose the right one:

```
(%i33) line[106]:implicit(y - m[106]*x + m[106]*P[106][1]
       -P[106][2]=0, x,-a-0.5,a+0.5, y,-b-0.5,b+0.5)$
(%i34) Pinters[107]:ev(solve([x^2/a^2 + y^2/b^2=1,y-m[106]*x
       + m[106]*P[106][1]-P[106][2]=0],[x,y]),numer);
```

```
(%o34) [[x=-0.9617684510713956,y=0.6645804306788178],
       [x=1.499998670984924,y=-0.001152828906202703]]
(%i35) if abs(rhs(Pinters[107][1][1]) - P[106][1]) < 0.0001 then
        Pnew[107]:[rhs(Pinters[107][2][1]),rhs(Pinters[107][2][2])]
        else
        Pnew[107]:[rhs(Pinters[107][1][1]),rhs(Pinters[107][1][2])];
(%o35) [1.499998670984924,-0.001152828906202703]
```

Comparing the distance between the points `Pnew[107]` and `P[1]` we get:

```
(%i36) dist_P107_P1 : sqrt((Pnew[107][1]-P[1][1])^2
             + (Pnew[107][2]-P[1][2])^2);
(%o36) 0.001152829672266286
```

That is, we can say that

$$\mathrm{d}(P_{107}, P_1) = 0.001152829672266286 < 0.0015,$$

which tell us that $P_{107} \cong P_1$. Therefore one could expect that `line[106]` would be "equal" to `line[0]`. We illustrate this in Figure 9b by plotting the objects of our interest:
Thus, we can conclude that reflecting a light ray from point $E = \big(0, a\tan(\pi/12)\big)$ into point $P_1 = (1.5, 0)$ on the ellipse given by equation (3), the trajectory inside this ellipse of the successive reflections closes after 106 reflections.

4.3 Some other cases with Maxima

We have investigated with Maxima several other cases by changing the initial point E or the the first point of reflection on the ellipse, namely P_1. We will not repeat what was done in the previous two cases and we have just plotted the obtained result and then compare the several cases in Table 1. Nevertheless, in Section 7 of Supplementary Electronic Materials the reader can find the complete code for each of the cases we have worked out.

In Table 1, we summarize some of the data we have gotten from all the different examples addressed in the paper with the CAS Maxima, and we do it for comparison purposes in order to have a better idea of the behaviour of this computer algebra system in relation with the problems we have investigated.

On the other hand, the last two examples included in Table 1 were not included in the paper but the reader can easily analyze both cases by implementing a Maxima or a Maple program, similar to those documented here for the various examples we have discussed.

4.4 Some remarks

In view of the results we have gotten, we think it will be quite illustrative for the reader if instead of performing the calculations within a `while` loop, as we have done in the computer programs documented in this paper, she/he tries to make the calculations with another iterative procedure, for example, by using definite initial and final numeric values for starting and finishing the number of iterations. In Maxima we can do something like the following:

```
(%i13) ratprint:false$
```

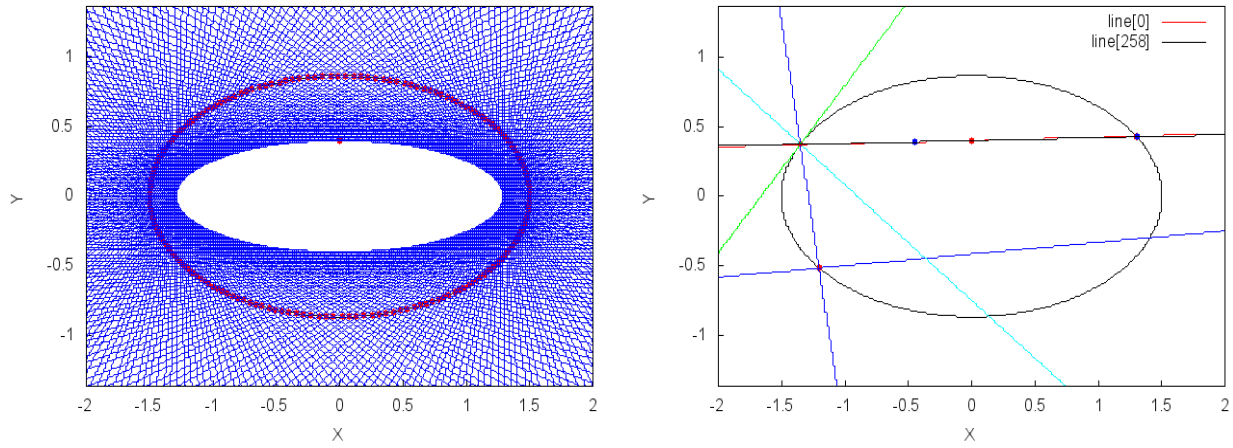


Figure 10: Case $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.3, 0.4320493798938573)$.

Point E	Point P_1	Time (in seconds)	Number of points
$(0, b\tan(\pi/12))$	$(1.5, 0)$	25.978	47
$(0, a\tan(\pi/12))$	$(1.5, 0)$	60.562	106
$(0, a\tan(\pi/12))$	$(1.3, 0.4320493798938573)$	148.006	258
$(0, b\tan(\pi/12))$	$(1.45, 0.2217355782608346)$	166.266	288
$(0, a\tan(\pi/12))$	$(1.45, 0.2217355782608346)$	240.699	420
$(0.0, 0.4)$	$(1.5, 0)$	123.506	215
$(0.0, 0.5)$	$(1.5, 0)$	350.181	610

Table 1: Summary of some data for the various Maxima cases.

```
        ⋮
(%i19) N:100$
(%i20) for i : 1 thru N do(
        Pr[i] : Reflex(x0,y0,a0,b0,c0),
        m[i] : (Pr[i][2] - P[i][2])/(Pr[i][1]- P[i][1]),
        ⋮
        x0 : Pr[i][1],
        y0 : Pr[i][2]
        );
```

Thus, we have control on the number of iterations and this can be useful for plotting a small number of points and lines. In this situation, we can also plot the tangent and normal lines for those few points. Actually, in the `while` loop that we have documented, the command lines

```
TgLine[i]:implicit(y-mtg[i]*x+mtg[i]*Pnew[i][1]-Pnew[i][2]=0,
```

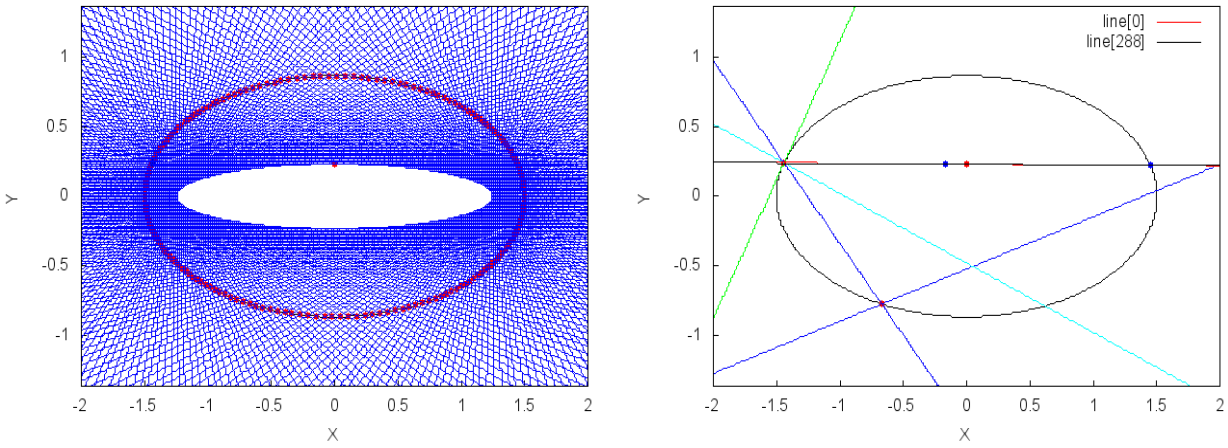


Figure 11: Case $E = \big(0, b\tan(\pi/12)\big)$ and $P_1 = (1.45, 0.2217355782608346)$.

```
                    x, -a-0.5,a+0.5, y,-b-0.5,b+0.5),
    ⋮
LinePerp[i]:implicit(y-mperp[i]*x+mperp[i]*Pnew[i][1]
                  -Pnew[i][2]=0,x,-a-0.5,a+0.5, y,-b-0.5,b+0.5),
```

can be eliminated and the Maxima program runs perfectly. However, those lines are necessary if we want to plot the straight lines they define.

On the other hand, although in all the cases that were investigated apparently we have a positive answer concerning the periodicity of the trajectories in the different examples, a caution remark is necessary. That is to say, we have to be aware that the closeness of points P_n and P_0, for certain n, is not a guarantee for the lines segments $\overline{P_0P_1}$ and $\overline{P_nP_{n+1}}$ to coincide, since it can happen that point P_{n+1} is *far away* from point P_1 on the ellipse. Here we present an example in which the `while` loop is completed because the condition of closeness is fulfilled, however the aforementioned line segments are quite different.

The maxima code is the same as in the discussed examples, thus, we only present some command lines that make up our remark.

Let us start with the point $E = (0, b\tan(\frac{\pi}{12}))$, as in some examples above.

```
(%i7) x0:0.0$ y0:ev(b*tan(%pi/12),numer)$
(%i8) E:[x0,y0]$
```

Now define the first point of reflection on the ellipse, namely P_1, to be

```
(%i15) P[1]:[1.3,0.4320493798938573]
```

As before, it is necessary to calculate the normal line through P_1 in order to calculate the parameters for the `Reflex` procedure, which turn out to be

```
(%i18) a0:-0.9970370305242858$ b0:1.0$ c0:0.8640987597877144$
```

The `while` loop stops since the closeness condition was achieved and we get the following data:

```
(%i28) time(%);
(%o28) [28.532]
```

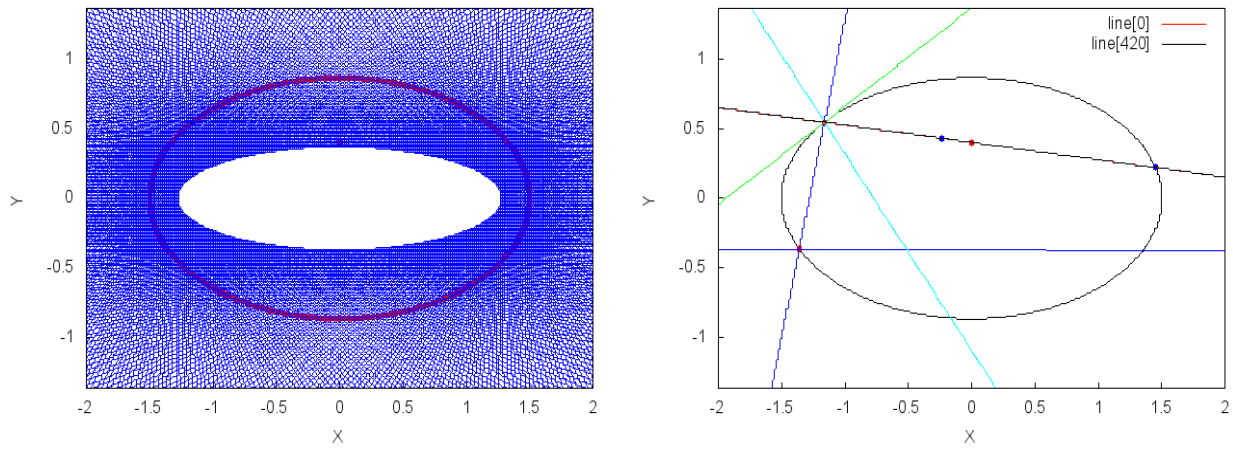


Figure 12: Case $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.45, 0.2217355782608346)$.

```
(%i29) length(Points);
(%o29) 50
```

Also we have

```
(%i32) P[0]; P[49]; P[50];
(%o32) [-1.499998352684478,0.001283477548325892]
(%o33) [1.313791688592809,-0.4179120318059193]
(%o34) [-1.499976831562785,-0.004813341696730361]
(%i35) sqrt((P[0][1]-P[50][1])^2 + (P[0][2]-P[50][2])^2);
(%o35) 0.00609685722857008
```

Thus we can see that points P_0 and P_{50} are quite close to each other:

$$\mathrm{d}(P_0, P_{50}) < 0.0061$$

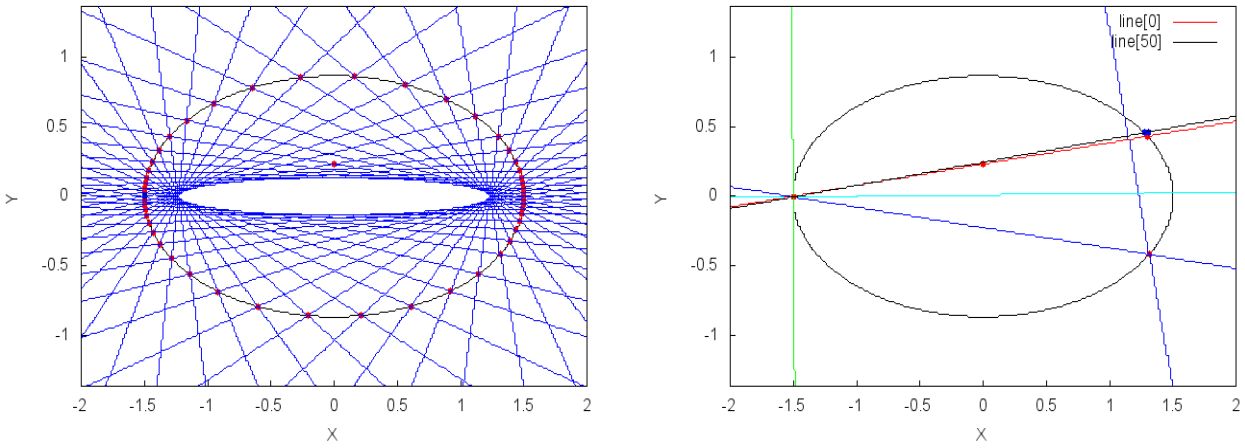
Performing the remaining calculations, we get point P_{51} to be

```
(%i45) if abs(rhs(Pinters[51][1][1]) - P[50][1]) < 0.0001 then
        Pnew[51]:[rhs(Pinters[51][2][1]),rhs(Pinters[51][2][2])]
        else
        Pnew[51]:[rhs(Pinters[51][1][1]),rhs(Pinters[51][1][2])];
(%o45) [1.273976188060564,0.4571595900247074]
```

However, notice that

$$\mathrm{d}(P_1, P_{51}) = 0.03616298439944626$$

which is tell us that points P_1 and P_{51} are not close enough. This *big* difference can be appreciated in Figure 13b, between the segment lines $\overline{P_0P_1}$ and $\overline{P_{50}P_{51}}$.



(a) Initial points $E = \big(0, b\tan(\pi/12)\big)$ and $P_1 = (1.3, 0.4320493798938573)$

(b) Segment lines $\overline{P_0P_1}$ and $\overline{P_{50}P_{51}}$ do not coincide

Figure 13: Results for Maxima case $E = \big(0, b\tan(\pi/12)\big)$ and $P_1 = (1.3, 0.4320493798938573)$.

5 Solving the problem with Maple

As it is well known, if a computational task is achieved by either using the CAS Maxima or Maple, then the same task can be done with the other, and viceversa. In what follows, we solve with the CAS Maple all the cases done with Maxima in the previous section. However, for reason of space, we do not document those Maple program and we urge the reader to consult Section 7 of Supplementary Electronic Materials for the complete programs. Here, an important remark is that we got the same results as those with Maxima for all the worked examples.

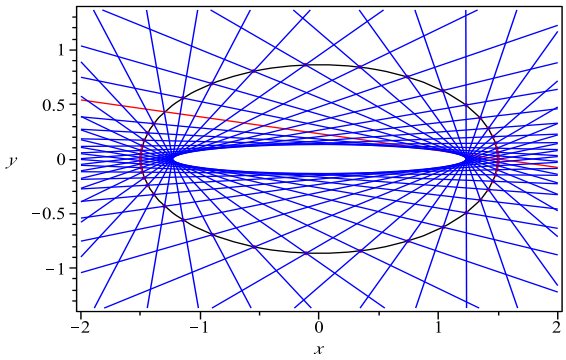
On the other hand, we will have the oportunity to realize how similar is the syntax for these two computer algebra systems and also how easily is to transform one code into the other.

5.1 Case $E = \big(0, b\tan(\pi/12)\big)$

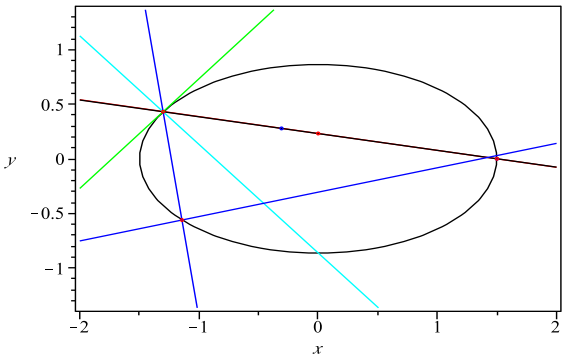
In this part, we redo in Maple what was done in subsection 4.1. We leave the detailed computations for this Maple worksheet in [S10].
Now, we plot the objects of interest that are shown in Figure 14a (compare with Figure 8a):

We can proceed as in the previous cases and calculate some more lines and points in order to make some comparisons.

The results of this last discussion can be seen in Figure 14b (compare with Figure 8b).



(a) Reflections inside ellipse given by (3) with the CAS Maple



(b) Segment $\overline{P_0P_1}$ in `line[0]` is "equal" to segment $\overline{P_{47}P_{48}}$ in `line[48]`.

Figure 14: Maple outcomes for case $E = \big(0, b\tan(\pi/12)\big)$

5.2 Case $E = \big(0, a\tan(\pi/12)\big)$

This case is parallel to that in subsection 4.2 and here, as we did before, we leave the detailed discussions using Maple in [S11].
We plot what it is of our interest and proceed to complete the picture and make some comparisons, as for example with Figures 9a and 9b:

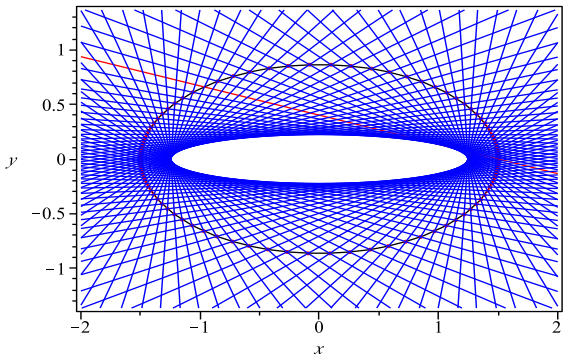
This result shows that the closeness between points P_0 and P_{106} and now we are able to calculate the straight line joining the points P_{106} and P_1, namely `line[106]`, in order to compare it with `line[0]` that is the line passing through P_0 and P_1. So, we proceed as previously and show the results in Figure 15.

5.3 Some other cases with Maple

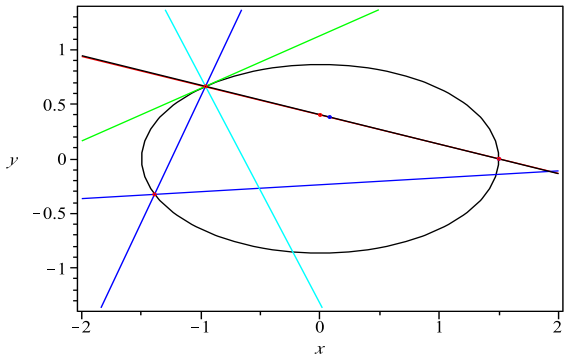
We have analyzed the same cases for Maple as those of presented in subsection 4.3 and it is not necessary that we repeat the Maple code for those examples. Instead we refer the reader to the Supplementary Electronic Materials in Section 7.

1. Case $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.3, 0.4320493800)$
   Results of this example are shown in Figure 16 and detailed Maple worksheet can be found in [S12].

2. Case $E = \big(0, b\tan(\pi/12)\big)$ and $P_1 = (1.45, 0.2217355782608346)$
   A summary of the results for this case are illustrated in Figure 17. The Maple worksheet can be found in [S13].



(a) Reflections inside ellipse given by (3) with the CAS Maple



(b) Segment $\overline{P_0P_1}$ in `line[0]` is "equal" to segment $\overline{P_{106}P_{107}}$ in `line[106]`.

Figure 15: Maple results for case $E = \big(0, a\tan(\pi/12)\big)$

3. Case $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.45, 0.2217355782608346)$
   The results for this example are ahown in Figure 18. The Maple worksheet can be found in [S14].

5.4 Maple comparisons for the different cases

As in the Maxima case, we now compare the different cases we addressed with Maple and make a summary in Table 2 and Table 3. Nevertheless, the Maxima's default float point precision FPPrec) if of 16 significative figures and that of Maple is 10 significative decimals, although we set that explicitly with Maple's command

[> *Digits*:=10:

Thus, we have included two comparison tables for Maple: one with the Maple's default float point precision and the other one for

[> *Digits*:=16:

This give us a good idea about Maple's performance with our problems and the reader can have a point of comparison between the two CASs we have used in this paper. We leave the Maple worksheets for the last two examples included in Table 2 in [S15] and [S16] respectively for readers to explore further.

We note that the examples in Table 3 are not included here, but again, the reader can run the corresponding Maple's computer programs with no difficulty.

Point E	Point P_1	Time (in seconds) Default FPPrec	Number of points
$(0, b\tan(\pi/12))$	$(1.5, 0)$	1.314	46
$(0, a\tan(\pi/12))$	$(1.5, 0)$	2.176	105
$(0, a\tan(\pi/12))$	$(1.3, 0.4320493800)$	4.369	257
$(0, b\tan(\pi/12))$	$(1.45, 0.2217355784)$	4.866	287
$(0, a\tan(\pi/12))$	$(1.45, 0.2217355784)$	6.825	419
$(0.0, 0.4)$	$(1.5, 0)$	3.844	214
$(0.0, 0.5)$	$(1.5, 0)$	9.745	609

Table 2: Summary of some data for the various Maple cases (default precision.)

Point E	Point P_1	Time (in seconds) FPPrec: 16	Number of points
$(0, b\tan(\pi/12))$	$(1.5, 0)$	1.363	46
$(0, a\tan(\pi/12))$	$(1.5, 0)$	2.313	105
$(0, a\tan(\pi/12))$	$(1.3, 0.4320493798938573)$	4.457	257
$(0, b\tan(\pi/12))$	$(1.45, 0.2217355782608346)$	5.203	287
$(0, a\tan(\pi/12))$	$(1.45, 0.2217355782608346)$	6.962	419
$(0.0, 0.4)$	$(1.5, 0)$	4.032	214
$(0.0, 0.5)$	$(1.5, 0)$	10.081	609

Table 3: Summary of some data for the various Maple cases (16 significative digits.)

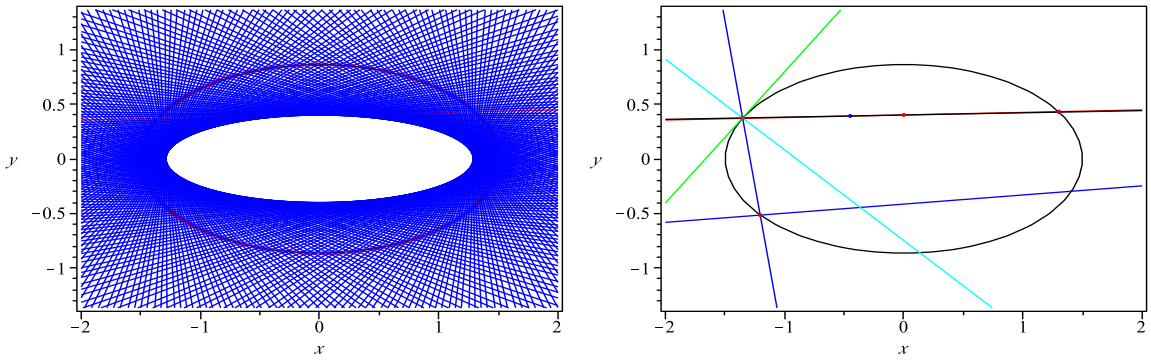


Figure 16: Maple case for $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.3, 0.4320493800)$

6 Conclusions

Evolving technological tools definitely have made mathematics fun and accessible on the one hand, but they also allow the exploration of more challenging and theoretical mathematics. We hope that when mathematics is made more accessible to students, it is possible more students will be inspired to investigate problems ranging from the simple to the more challenging. We do not expect that exam-oriented curricula will change in the short term. However, encouraging a greater interest in mathematics for students, and in particular providing them with the technological tools to solve challenging and intricate problems beyond the reach of pencil-and-paper, is an important step for cultivating creativity and innovation. We have used Geogebra to solve the problem we were faced out and have documented several computer programs in the computer algebra systems Maxima and Maple, with the purpose of investigate how a ray of light is reflected inside a given ellipse. The assumption is that the ray starts from some specific points inside the ellipse, and then it is aimed into some other point on the curve over which it is reflected according to the usual law of reflection. For the cases discussed in this paper, we can say that there is enough computational evidence for the periodicity of the trajectories determined by the succesive reflections on the curve for those particular initial points we have started from.

The problems we have investigated resemble analogous problems in the theory of elliptical billiards, which are also related to the famous Poncelet Porism, and the results obtained in the paper, along with the presented computer programs, can be taught to high school and college students in order to motivate more studies concerning the reflection of light rays on arbitrary curves.

Acknowledgements. The authors want to thank the referees for their useful suggestions and several corrections of this paper.

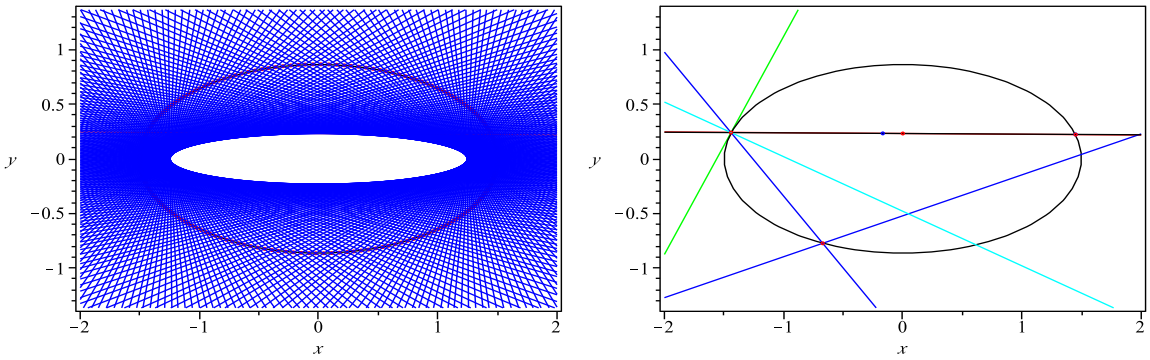


Figure 17: Maple case for $E = (0, b\tan(\pi/12))$ and $P_1 = (1.45, 0.2217355782608346)$

7 Supplementary Electronic Materials

[S1] Explorations with GeoGebra in Section 3.

[S2] Explorations with Maxima in Section 4.1.

[S3] Explorations with Maxima in Section 4.2.

[S4] Explorations with Maxima in Section 4.3.

[S5] Explorations with Maxima in Section 4.3.

[S6] Explorations with Maxima in Section 4.3.

[S7] Explorations with Maxima for point $E = (0, 0.4)$ and $P_1 = (1.5, 0)$.

[S8] Explorations with Maxima for point $E = (0, 0.5)$ and $P_1 = (1.5, 0)$.

[S9] Explorations with Maxima for point $E = (0, b\tan(\pi/12))$ and $P_1 = (1.3, 0.4320493798938573)$.

[S10] Explorations with Maple in Section 5.1.

[S11] Explorations with Maple in Section 5.2.

[S12] Explorations with Maple in Section 5.3.

[S13] Explorations with Maple in Section 5.3.

[S14] Explorations with Maple in Section 5.3.

[S15] Explorations with Maple for point $E = (0, 0.4)$ and $P_1 = (1.5, 0)$.

[S16] Explorations with Maple for point $E = (0, 0.5)$ and $P_1 = (1.5, 0)$.

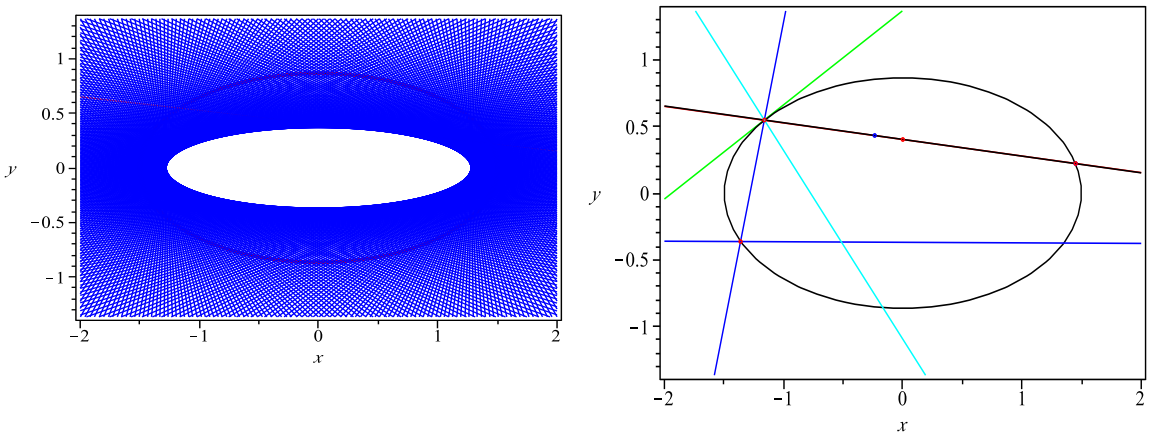


Figure 18: Maple case for $E = \big(0, a\tan(\pi/12)\big)$ and $P_1 = (1.45, 0.2217355782608346)$

References

[1] V. I. Arnold, *Mathematical Methods of Classical Mechanics*, 2nd. Edition, Springer, New York, 1989.

[2] M. V. Berry, Regularity and chaos in classical mechanics, illustrated by three deformations of a circular billiard, *Eur. J. Phys.*, **2** (1981), pp. 91-102.

[3] Shau-Jin Chang, and Richard Friedberg, Elliptic billiards and Poncelet's theorem, *J. Math. Phys.*, **29** (1988), No. 7, pp. 1537–1550.

[4] Andre Del Centina, Poncelet's porism: a long story of renewed discoveries, I, *Arch. Hist. Exact Sci.*, 2015, DOI 10.1007/s00407-015-0163-y.

[5] Vladimir Dragović, and Milena Radnović, *Poncelet Porisms and Beyond. Integrable Billiards, Hyperelliptic Jacobians and Pencils of Quadrics*, Birkäuser, Basel, 2010.

[6] Vladimir Dragović, and Milena Radnović, Bicentennial of the great Poncelet theorem (1813–2013): Current Advances, *Bulletin (New Series) of the American Mathematical Society*, Vol. 53, No. 3 (2014), pp. 373-445.

[7] J. C. Gutiérrez-Vega, S. Chávez-Cerda, R. M. Rodríguez-Dagnino, Probability distributions in classical and quantum elliptic billiards, *Rev. Mex. Fís.*, **47** (5) (2001), pp. 480-488.

[8] Mark Levi, and Serge Tabachnikov, The Poncelet grid and billiards in ellipses, *The American Mathematical Monthly*, **114** (2007) No. 10, pp. 895–908.

[9] George E. Martin, *Transformation Geometry. An Introduction to Symmetry*, Springer, New York, 1982.

[10] J. Zhang, A. C. Merchant and W. D. Rae, Geometric derivations of the second constant of motion for an elliptic 'billiard' and other results, *Eur. J. Phys.* **15** (1994), pp. 133-138.