Ellipse to a Line via Non-Isometric Scaling

Andrew Paul

7/12/2017

We know how to use analytic geometry in order to find the minimal distance between a circle and a line. However, how do we find the minimal distance between an *ellipse* and a line? Clearly the same approach does not work.

Let us analytically find the minimum distance between the line $x + y = 4$ and the ellipse $\frac{x^2}{4} + y^2 = 1$. The key is to *scale the horizontal axis* such that $u = \frac{x}{2}$. Then, the ellipse equation becomes:

$$u^2 + y^2 = 1$$

Likewise, the equation of the line becomes:

$$2u + y = 4$$

Now we simply must find the closest distance between that line and the unit circle. This can be done fairly easily using analytic geometry. First find the equation of a line that goes through the origin and is perpendicular to the other line:

$$y = \frac{1}{2}u$$

Now find the intersection point of the two lines. I.e., solve the system:

$$2u + y = 4$$

$$y = \frac{1}{2}u$$

The solutions are $u = \frac{8}{5}$ and $y = \frac{4}{5}$. The second line intercepts the unit circle at $u = \frac{2}{\sqrt{5}}$ and $y = \frac{1}{\sqrt{5}}$ (trigonometry not even required—use simple substitution). So the shortest distance between the circle and the line is the distance between $(\frac{8}{5}, \frac{4}{5})$ and $(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}})$. If we rescale our coordinate plane by substituting back in: $u = \frac{x}{2}$, then we will find the shortest distance between our ellipse and line as being simply the distance between $(\frac{16}{5}, \frac{4}{5})$ and $(\frac{4}{\sqrt{5}}, \frac{1}{\sqrt{5}})$. This is simply given by:

$$\sqrt{\left(\frac{16}{5} - \frac{4}{\sqrt{5}}\right)^2 + \left(\frac{4}{5} - \frac{1}{\sqrt{5}}\right)^2} = \boxed{\frac{1}{5}\sqrt{17(21 - 8\sqrt{5})}}$$

No calculus required! In general, we can translate and dilate the ellipse in such a way that it becomes a unit circle. If we apply the same transformations to the line, and then consider to be a circle-to-line problem on a dilated plane, then we can find the intersection points, and then reverse our dilation to find the corresponding points on the ellipse and line.

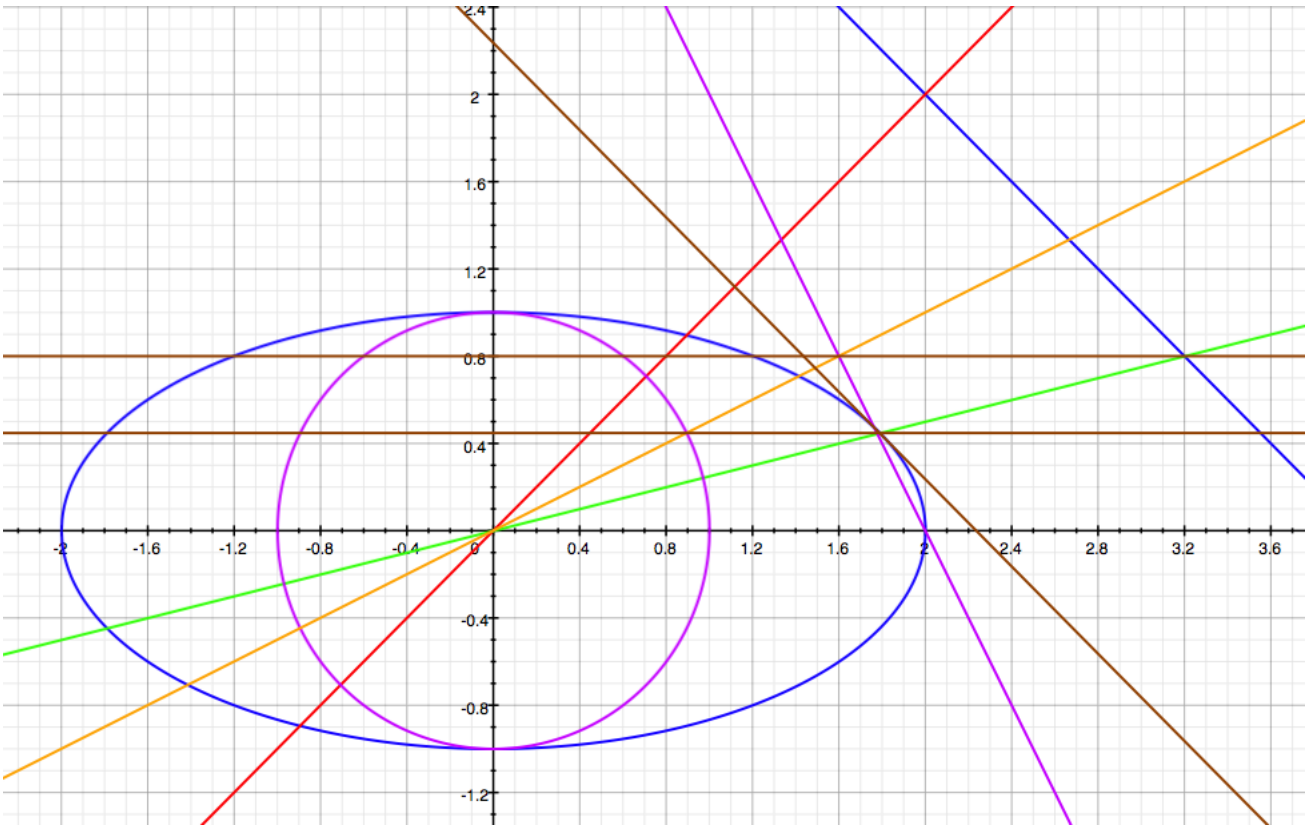

Figure 1: The given curves, intermediate projections, and final curves.

In the Fig 1, the given curves are the blue ellipse and the blue line. To find the minimal distance between them, the incorrect approach is to use the line that passes through the center of the ellipse and is perpendicular to the given line (which is the red line). Upon scaling the axis, we arrive at the pink circle and line (which have been projected onto the original unscaled graph for perspective), and the orange line passes through the center of the circle and is perpendicular to the pink line.

When we readjust, the coordinates of where the orange line intercepts the circumference of the pink circle and the pink line become the coordinates where the green line intercepts the original blue ellipse and line. It is between these points that the minimal distance is found. Two horizontal brown lines prove that the readjustment of the coordinates after dilation only changes the x-coordinate of the points (as we only scaled the x-axis in the problem). A third brown line is a line tangent to the ellipse that is also parallel to the original blue line. The traditional calculus approach first finds this tangent line and calculates the distance between that tangent line and the original blue line. The fact that the point of tangency is is precisely the point on the ellipse that was found to

be on the green line after readjustment serves as further justification to the analytic approach.

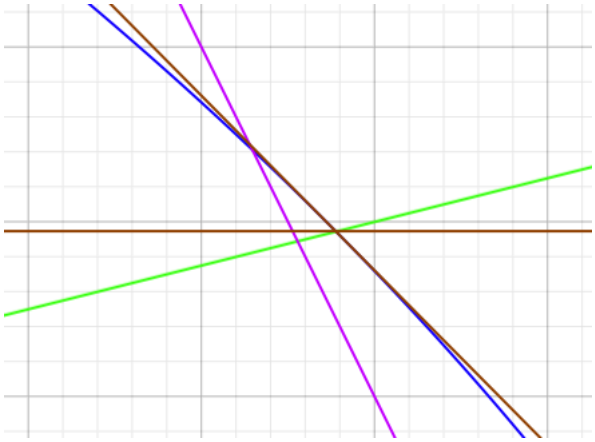

Figure 2: This is not an intersection with the pink line.

Please note that the pink line **does not** intercept the ellipse at the aforementioned point of tangency (Fig 1.2). Its position relative to that point holds no significance.

The algorithm that the program runs is as follows:

The user will input eight different parameters, given the equations of the line and ellipse:

$$a_l x + b_l y = c_l$$

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1 \text{ or } \frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$$

for $a > b$. These are:

$$a \rightarrow \text{the semi-major axis length of the ellipse}$$

$$b \rightarrow \text{the semi-minor axis length of the ellipse}$$

$$a_l \rightarrow \text{line coefficient}$$

$$b_l \rightarrow \text{line coefficient}$$

$$c_l \rightarrow \text{line coefficient}$$

$$h \rightarrow x\text{-coordinate of the ellipse center}$$

$$k \rightarrow y\text{-coordinate of the ellipse center}$$

$$\omega \rightarrow \text{the orientation parameter of the ellipse}$$

The ω parameter determines which direction the ellipse is most stretched in. If $\omega = 1$, then we have an ellipse that is more horizontally stretched (i.e. a^2 appears under $(x-h)^2$). If $\omega = 2$, then

the ellipse is more vertically stretched (i.e. a^2 appears under $(y-k)^2$). If the ellipse is equally stretched in both dimensions, (i.e. it is a circle), then we are free to choose either 1 or 2 for the value of ω.

First, the program checks for nonsensical inputs. For instance, if $ab = 0$, or if $a_l = b_l = 0$, then the program outputs an error. The rest of this description beyond this point will assume that none of the error conditions have been met.

Next, the program checks the ω value. This parameter is relevant because the algorithm assumes the equation of an ellipse to be:

$$\frac{(x-h)^2}{a_e^2} + \frac{(y-k)^2}{b_e^2} = 1$$

The key difference with the algorithm's assumption and the standard form of an ellipse is that in an ellipse, the values a and b are defined as the major and minor axes respectively. However, the algorithm's "standard form" lets a_e and b_e to simply be the horizontal and vertical stretch factors respectively. In other words, a_e is always the horizontal stretch factor, under $(x-h)^2$, and b_e is always the vertical stretch factor, under $(y-k)^2$, regardless of whether it is true that $a_e \geq b_e$ or $a_e \leq b_e$.

Therefore, to clarify if the user's a value corresponds to the algorithm's a_e variable and the b value corresponds to the b_e variable (i.e. if the ellipse is stretched more horizontally than it is vertically) we let $\omega = 1$. If the ellipse is actually stretched more vertically than it is horizontally, then we let $\omega = 2$ which tells the program to assign $a = b_e$ and $b = a_e$.

This description of the algorithm beyond this point will assume $\omega = 1$. It is evident that if $\omega = 2$, the program will follow the exact same steps, except it will always switch the values of a_e and b_e relative to what it would assign if $\omega = 1$.

The program checks to see if either $a_l = 0$ or $b_l = 0$ (but not both because this would result in an error as mentioned above). If $a_l = 0$, then the given line is horizontal (with equation $y = \frac{c_l}{b_l}$). Hence the closest distance between the ellipse and the line will trivially be the distance from the center of the ellipse to the line minus the length of the vertical axis of the ellipse. In terms of our parameters, we see that this is given by:

$$\left|\frac{c_l}{b_l} - k\right| - b_e$$

But if this value is less than 0, then it is implied that the line intersects the ellipse. If that value is equal to 0, then it is implied that the line is tangential to the ellipse. In either of those cases, the shortest distance between the ellipse and the line is 0 because there is at least one shared point between the ellipse and the line. Therefore:

$$a_l = 0 \rightarrow \left|\frac{c_l}{b_l} - k\right| - b_e \geq 0 \rightarrow \text{output} = \left|\frac{c_l}{b_l} - k\right| - b_e$$

$$a_l = 0 \rightarrow \left|\frac{c_l}{b_l} - k\right| - b_e < 0 \rightarrow \text{output} = 0$$

The same logic applies if the line is vertical:

$$b_l = 0 \rightarrow \left|\frac{c_l}{a_l} - h\right| - a_e \geq 0 \rightarrow \text{output} = \left|\frac{c_l}{a_l} - h\right| - a_e$$

$$b_l = 0 \rightarrow \left|\frac{c_l}{a_l} - h\right| - a_e < 0 \rightarrow \text{output} = 0$$

But if both a_l and b_l are nonzero, then the line has a finite nonzero slope. These are the nontrivial cases, and that brings us to the heart of the program.

When $a_l, b_l > 0$, then we have an oblique line. First, the program will translate both the ellipse and line such that the center of the ellipse coincides with the origin. While this is really just a superficial step, it makes the next steps mathematically cleaner and easier to follow (and also computationally more efficient). When this is done, the equations for our ellipse and line are:

$$\frac{x^2}{a_e^2} + \frac{y^2}{b_e^2} = 1$$

$$a_l(x + h) + b_l(y + k) = c_l$$

At this point, the program scales both axes according to the equations:

$$x = a_e x_s$$

$$y = b_e y_s$$

Where (x_s, y_s) are the scaled coordinates. The line and ellipse equations become:

$$x_s^2 + y_s^2 = 1$$

$$a_l a_e(x_s + h) + b_l b_e(y_s + k) = c_l \Rightarrow a_l a_e x_s + b_l b_e y_s = c_l - a_l a_e h - b_l b_e k$$

The program then assigns the substitutions $a_{l_2} = a_l a_e$, $b_{l_2} = b_l b_e$, and $c_{l_2} = c_l - a_l a_e h - b_l b_e k$. The translated and scaled line thus has equation:

$$a_{l_2} x_s + b_{l_2} y_s = c_{l2}$$

It is at this point that the program makes a determination as to whether or not the original ellipse and oblique line had intersected (or were tangential) which would imply that their minimal distance apart was 0. It does this in a rather crafty manner.

Suppose the original line and ellipse had intersected or were tangential. This means they share at least one common ordered pair. The key insight is to realize that if the original line and ellipse had intersected or were tangential, then the same would be true for the new transformed line and ellipse (which has effectively become a unit circle)!

For the transformed line to intersect or be tangential to the unit circle, its distance from the

origin must be lesser than or equal to 1 (if the distance is equal to 1, then the transformed line and unit circle are tangential and so are the original line and ellipse, and if the distance is lesser than 1, then the transformed line and unit circle intersect and so do the original line and ellipse). The output should clearly be 0 in both cases.

The distance between the origin and the transformed line ($a_{l_2}x + b_{l_2}y = c_{l_2}$) is easily given by the point-to-line equation. Upon simplification, it is:

$$\frac{|c_{l_2}|}{\sqrt{a_{l_2}^2 + b_{l_2}^2}}$$

We have already deduced above that if this expression is less than or equal to 1, then the original line and ellipse must share at least one common point. This means:

$$\frac{|c_{l_2}|}{\sqrt{a_{l_2}^2 + b_{l_2}^2}} \leq 1 \rightarrow \text{output} = 0$$

If this value is however greater than 1, then the original line is completely outside the ellipse. If this is the case, then the program continues.

The equation of the transformed line is $a_{l_2}x_s + b_{l_2}y_s = c_{l_2}$. It follows that the equation of a line that is perpendicular to this transformed line that also passes through the origin is $y_s = \frac{b_{l_2}}{a_{l_2}}x_s$. The smallest distance between the unit circle and the transformed line is simply equivalent to the smallest distance between where this perpendicular line intersects the circle and where it intersects the transformed line. To find the coordinates of the intersection between the transformed line and the perpendicular line, we must solve the system of equations:

$$\begin{cases} a_{l_2}x + b_{l_2}y = c_{l_2} \\ y = \frac{b_{l_2}}{a_{l_2}}x \end{cases}$$

The program assigns the solution to the coordinate (x_{s_1}, y_{s_1}). The solution is:

$$x_{s_1} = \frac{a_{l_2}c_{l_2}}{a_{l_2}^2 + b_{l_2}^2}$$

$$y_{s_1} = \frac{b_{l_2}c_{l_2}}{a_{l_2}^2 + b_{l_2}^2}$$

Similarly, to find the intersection point between the perpendicular line and the circle, the following system of equations must be solved:

$$\begin{cases} x_s^2 + y_s^2 = 1 \\ y_s = \frac{b_{l_2}}{a_{l_2}}x_s \end{cases}$$

The program assigns the solution to the coordinate (x_{s_2}, y_{s_2}). The solution is:

$$x_{s_2} = \pm \frac{a_{l_2}}{\sqrt{a_{l_2}^2 + b_{l_2}^2}}$$

$$y_{s_2} = \pm \frac{b_{l_2}}{\sqrt{a_{l_2}^2 + b_{l_2}^2}}$$

To determine the signs of x_{s_2} and y_{s_2}, the program will take into account the signs of x_{s_1} and y_{s_1} respectively. We can note that the desired intersection point between the perpendicular and the unit circle must reside in the same quadrant as the intersection point between the perpendicular and the transformed line. This implies that if $x_{s_1} > 0$ then $x_{s_2} > 0$ or if $x_{s_1} < 0$ then $x_{s_2} < 0$. Likewise, if $y_{s_1} > 0$ then $y_{s_2} > 0$ or if $y_{s_1} < 0$ then $y_{s_2} < 0$.

Now that the desired intersection points (x_{s_1}, y_{s_1}) and (x_{s_2}, y_{s_2}) are found on the scaled graph, the program undoes the scaling to find the corresponding points (x_3, y_3) and (x_4, y_4) respectively. Essentially, the program has found the closest points on the scaled curves, so it is unscaling to find the closest points on the original curves (the ellipse and the original line). Recall that the plane was originally scaled according to the equations $x = a_e x_s$ and $y = b_e y_s$. Using these equations, the program finds the final coordinates to be:

$$(x_{s_1}, y_{s_1}) \to (a_e x_{s_1}, b_e y_{s_1})$$

$$(x_{s_2}, y_{s_2}) \to (a_e x_{s_2}, b_e y_{s_2})$$

The last step is to find the distance between these two final points. By the distance formula, the program concludes:

$$\text{line is oblique and outside the ellipse} \to \text{output} = \sqrt{(a_e x_{s_1} - a_e x_{s_2})^2 + (b_e y_{s_1} - b_e y_{s_2})^2}$$

Keep in mind that this was all under the assumption that $\omega = 1$. If $\omega = 2$, then the program performs the same steps except it will swap a_e and b_e in every computation. If the original ellipse was already a circle, then $a_e = b_e$, swapping them does not change the final output, hence the ω value is irrelevant.

Here is the program in Java:

```
1  //Copyright: ANDREW PAUL
2  import java.util.Scanner;
3  public class EllipseCalculator
4  {
5  public static void main(String[] args)
6  {
7  Scanner in = new Scanner(System.in);
8  System.out.println("Please input the x coefficient of the line.");
```

```
 9  double xLine = in.nextDouble();
10
11  System.out.println("Please input the y coefficient of the line.");
12  double yLine = in.nextDouble();
13
14  System.out.println("Please input the constant term in the standard "
15      + "form of your line.");
16  double cLine = in.nextDouble();
17
18  System.out.println("Please input the x-coordinate of the ellipse center.");
19  double xEllipse = in.nextDouble();
20
21  System.out.println("Please input the y-coordinate of the ellipse center.");
22  double yEllipse = in.nextDouble();
23
24  System.out.println("Please input the semi-major axis length of the ellipse.");
25  double semiMaj = in.nextDouble();
26
27  System.out.println("Please input the semi-minor axis length of the ellipse.");
28  double semiMin = in.nextDouble();
29
30  System.out.println("If the ellipse is more horizontally stretched than "
31      + "vertically stretched, please enter 1. Otherwise, enter 2.");
32  int orientationParameter = in.nextInt();
33
34  in.close();
35
36  double distance;
37
38  if((semiMaj <= 0 || semiMin <= 0) || (orientationParameter != 1 &&
39      orientationParameter != 2) || (semiMin > semiMaj) ||
40      (xLine == 0 && yLine == 0))
41  {
42      System.out.println("Your input is invalid!");
43      System.exit(0);
44  }
45
46  int counter = 0;
47
48  do
49  {
50  if(orientationParameter == 1)
51  {
52     if(xLine == 0)
53    {
54       distance = (Math.abs((cLine/yLine)-yEllipse))-semiMin;
55       if(distance < 0)
56       {
```

```
57        distance = 0;
58      }
59      System.out.println("The shortest distance between the ellipse "
60          + "and the line is: " + distance);
61      System.exit(0);
62    }
63    else if(yLine == 0)
64    {
65      distance = (Math.abs((cLine/xLine)-xEllipse))-semiMaj;
66      if(distance < 0)
67      {
68        distance = 0;
69      }
70      System.out.println("The shortest distance between the ellipse "
71          + "and the line is: " + distance);
72      System.exit(0);
73    }
74
75    double xLine2 = xLine * semiMaj;
76    double yLine2 = yLine * semiMin;
77    double cLine2 = cLine - (xLine2 * xEllipse) - (yLine2 * yEllipse);
78
79    distance = (Math.abs(cLine2))/(Math.sqrt((Math.pow(xLine2, 2)) +
80        (Math.pow(yLine2, 2))));
81    if(distance <= 1)
82    {
83      distance = 0;
84      System.out.println("The shortest distance between the ellipse "
85          + "and the line is: " + distance);
86      System.exit(0);
87    }
88
89    double xOne = (xLine2 * cLine2)/((Math.pow(xLine2, 2)) +
90        (Math.pow(yLine2, 2)));
91    double yOne = (yLine2 * cLine2)/((Math.pow(xLine2, 2)) +
92        (Math.pow(yLine2, 2)));
93    double xTwo;
94    double yTwo;
95    if(xOne > 0)
96    {
97      xTwo = xLine2/Math.sqrt(((Math.pow(xLine2, 2)) +
98          (Math.pow(yLine2, 2))));
99    }
100   else
101   {
102     xTwo = - xLine2/Math.sqrt(((Math.pow(xLine2, 2)) +
103         (Math.pow(yLine2, 2))));
104   }
```

```
105     if(yOne > 0)
106     {
107       yTwo = yLine2/Math.sqrt(((Math.pow(xLine2, 2)) +
108           (Math.pow(yLine2, 2))));          }
109     else
110     {
111       yTwo = - yLine2/Math.sqrt(((Math.pow(xLine2, 2)) +
112           (Math.pow(yLine2, 2))));
113     }
114
115     distance = Math.sqrt((Math.pow(((semiMaj * xOne) - (semiMaj * xTwo)), 2))
116         + (Math.pow(((semiMin * yOne) - (semiMin * yTwo)), 2)));
117     System.out.println("The shortest distance between your ellipse "
118         + "and line is: " + distance);
119     counter = 2;
120     }
121
122   else
123   {
124     double interMin = semiMin;
125     double interMaj = semiMaj;
126     semiMaj = interMin;
127     semiMin = interMaj;
128     orientationParameter = 1;
129     counter++;
130   }
131
132   }
133   while(counter == 1);
134
135   }
136   }
```