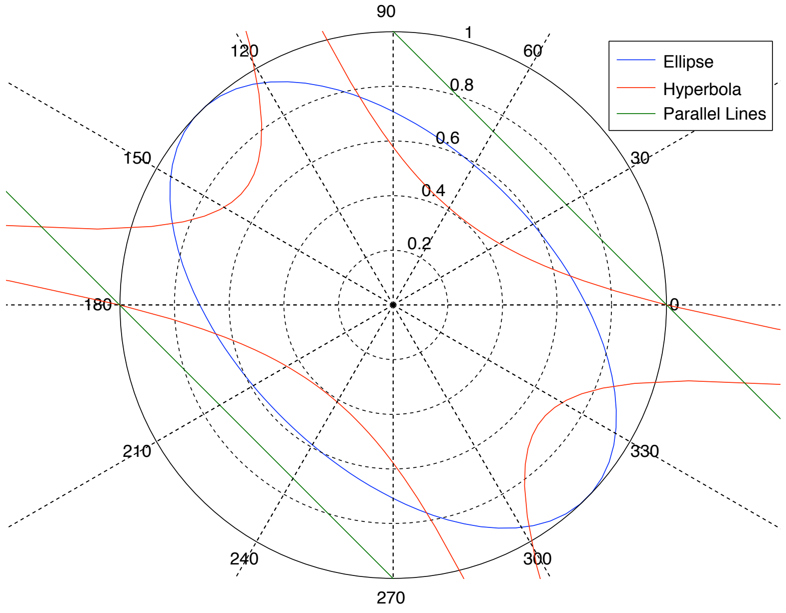
PLOTTING CONICS

STEVEN F. BELLENOT

ABSTRACT. Ways to plot implicit functions of the form $ax^2+2bxy+cy^2 = \pm 1$ are discussed. An implementation in Scilab is also given.



CONTENTS

1. Introduction

Our goal is to be able to use Scilab to plot implicit functions of the form:

$$ax^2 + 2bxy + cy^2 = \pm 1$$

The reason for the 2 in the $2b$ is so the symmetric matrix A does not have fractions:

$$A = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

Note that if $X = \begin{bmatrix} x & y \end{bmatrix}$ then

$$ax^2 + 2bxy + cy^2 = XAX^T$$

Let's check it:

$$(XA)X^T = (\begin{bmatrix} ax+by & bx+cy \end{bmatrix})\begin{bmatrix} x \\ y \end{bmatrix} = ax^2 + bxy + bxy + cy^2$$

This observation could be avoided, but it is used in the implementation.

2. Classification of $ax^2 + 2bxy + cy^2 = 1$

If you search your old analytic geometry book you will discover that one can classify this curve according to $\Delta = ac - b^2$ (Which interestingly enough is the determinant of A, usually written as $\det A$.) Δ is called the discriminate. The following table is helpful

Sign of Δ	Classification of $ax^2 + 2bxy + cy^2 = 1$
$\Delta > 0$	Ellipse centered at (0,0)
$\Delta < 0$	Hyperbola centered at (0,0)
$\Delta = 0$	2 parallel lines centered at (0,0) (degenerate Parabola)

(The next statement is not important for us, but given for completeness. The general second order quadratics has other first order terms so if

$$Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F = 0$$

has $\Delta = 0$ usually one of $D \neq 0$ or $E \neq 0$ and this will make the curve a true parabola and not our degenerate two parallel lines case.)

3. Implicit plotting of $ax^2 + 2bxy + cy^2 = 1$

What is is the easiest way graph our conic? This section is not it, jump to the polar section. This is here for completeness. One way to solve this is to pick values for one unknown and then solve the equation for the other. Solve for y in terms of x and we get

$$cy^2 + (2bx)y + (ax^2 - 1) = 0$$

and the quadratic formula gives

$$y = \frac{-2bx \pm \sqrt{4b^2 - 4c(ax^2 - 1)}}{2c} = \frac{-bx \pm \sqrt{b^2 - c(ax^2 - 1)}}{c}$$

Not a pretty sight. Even if you wanted to use it, you would need to be careful never to have a negative number inside the square root.

A better way, (but still not the best) is to use knowledge of Calculus 3 and do a contour plot on the function of two variable

$$z = f(x,y) = ax^2 + 2bxy + cy^2$$

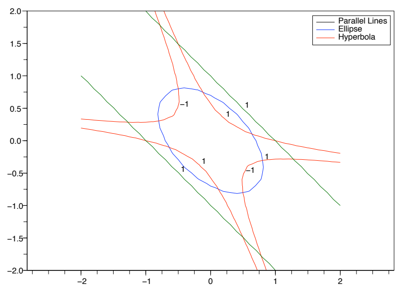


FIGURE 1. Output of contour.in

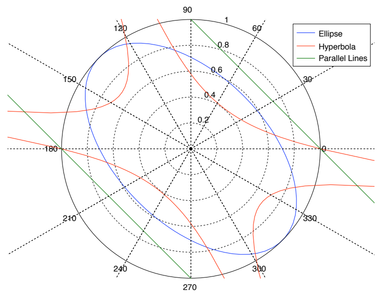


FIGURE 2. Output of polar.in

with contour $z = 1$. You should (or will) do contours plots in Calculus 3. Many mathematical computer programs have this feature so it can be done automatically. These graphs are often not as smooth as one would like without tons of data points. Scilab has such commands, and if our curves were not centered at the origin, it might be the best way to graph. There is a contour.in in scilab directory whose output is plotted in Figure 1. (Actually it is plotting $ax^2 + 2bxy + cy^2 = \pm 1$.)

4. Our Conic in Polar coordinates

Let's plug in $r\cos\theta$ for x and $r\sin\theta$ for y into our conic:

$$ax^2 + 2bxy + cy^2 = \pm 1$$

$$ar^2\cos^2\theta + 2br^2\cos\theta\sin\theta + cr^2\sin^2\theta = \pm 1$$

Note that each term on the LHS has an r^2, so we can solve for r by taking square roots. The ± 1 comes to the rescue, we can pick the sign so that the square is always of something positive. We get

$$r = \sqrt{1/|a\cos^2\theta + 2b\cos\theta\sin\theta + c\sin^2\theta|}$$

which looks ugly but is actually simply a function of the form $r = f(\theta)$ and easy to plot. There is a polar.in in scilab directory whose output is plotted in Figure 2. (The polar.in file executes very slowly). There is still a better way, instead of using polar coordinates, we can convert back to rectangular coordinates.

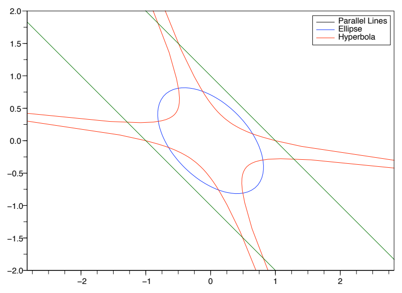


Figure 3. Output of plot.in

For the next figure we plotted $(r \cos\theta, r \sin\theta)$ using the normal plot function in Scilab. The graph is contained in Figure 3 And this is the implementation of choice.

5. Scilab implementation

Basically we need to know 2 new Scilab things to construct our implementation: the plot command and the for loop. (Well three π is %pi in Scilab.)

The plot command:
is 'plot(x,y,ColorMarkerStyle)' where x and y are vectors of the same size and ColorMarkerStyle is a string in single quotes like 'bo-'. The points (x_1, y_1), (x_2, y_2) ... (x_n, y_n) are plotted (where n is the number of entries in each vector). The string is optional but it gives more control on the graph. For example 'bo-' tells scilab to color it (b)lue with circles(o) at each data point and to connect the points with a solid line(-). Other examples are 'r:', for (r)ed dotted(:) line with no markers, 'k+' for blac(k) plus(+) markers and no line and 'g–' for a (g)reen dashed(–) line and no markers.

The for loop:
has the form 'for i=1:n, S_1; S_2: ... S_m; end;' where the S_i are scilab commands. The sequence of commands is n times, the first time with $i = 1$, the second time with $i = 2$, ..., the last time with $i = n$. There are variations, which will not be used here. Often for loops are written (this is a good idea) with the commands S_i on separate lines shifted a few spaces to the right like

```
for  i = 1:n,
     S1;
     S2;
     ...
     Sm;
end;
```

The implementation:
this is broken into 3 parts. The setup, the for loop and the plot. For the setup, we need to define the matrix A and we will divide $0 \le \theta \le 2\pi$ into equal parts and create the vectors x and y that will be used to plot. n will be the number elements in the vector theta (101 in this example). The listing is in Figure 4

Unfortunately the aspect ratio of the resulting graph is not 1:1 by default. One unit in the x direction is more than one unit in the y direction. This can make circles seem like

```
A = [2 1; 1 2];
theta = 0:%pi/50:2*%pi; // %pi is scilab for the constant pi
n = size(theta,2); // theta is a 1 x n row vector
x = zeros(theta); // x and y start as zero vectors
y = zeros(theta);
for i = 1:n,
    angle = theta(i); // this angle
    c = cos(angle);
    s = sin(angle);
    X = [ c, s ];
    r = sqrt(1/abs(X*A*X')); // formula from section 4
    x(i) = r*c; // r cos(theta)
    y(i) = r*s;
end;
plot(x,y,'r*-');
xtitle('Plot of 2x^2 + 2xy + 2y^2 = +/- 1','x','y');
```

Figure 4. Scilab Implementation

ellipses and make perpendicular lines appear to be not perpendicular. This can be fixed using the scilab commands: ‘a = gca(); a.isoview=”on”;’ after the plot command.

The example above is of an ellipse. Hyperbola’s will have to be limited to the area of interest. ‘a.data_bounds=[xmin, ymin; xmax, ymax]’ can be used for this (note comma, semi-colon, comma) after the ‘a =gca()’ command.