Drawing Implicit Functions

David Tall

Mathematics Education Research Centre
University of Warwick, Coventry CV4 7AL

A common problem is to translate an implicit relationship such as the ellipse:

$$x^2+2y^2+4x+6y=5$$

into graphical form. By manipulating the algebra it is possible to solve for y in terms of x (using positive and negative square roots) and then to plot points (x,y) on the curve. But this approach does not generalise easily and proves utterly impracticable for a more complicated relation such as

$$y\sin x + x\cos y = 1.$$

Fortunately it is a relatively simple matter to program numerical methods to draw a more general implicit function

$$f(x,y) = \text{constant}$$

without solving the relationship explicitly. This will be demonstrated using BBC BASIC (a language which beautifully expresses the mathematical structure) though the techniques may be translated to other computers with user-definable functions and high resolution graphics.

The approach advocated may be interpreted as a combination of the Newton-Raphson approximation technique and partial derivatives. But calculus is not necessary to carry out the calculations or to gain insight into the theory. This speaks volumes for the possibility of introducing numerical methods on the computer in parallel with the calculus or even before it. I am certain that this is the direction that will eventually be taken by mathematics in schools. I shall therefore write the article without using any calculus at all.

The idea

The solution of an equation

$$f(x,y) = c$$

is better seen in a wider context. Imagine the values of the function

$$z = f(x,y)$$

are represented by plotting a point height $z=f(x,y)$ above the point (x,y) in the plane. The result is a surface like a range of hills. For example figure 1 shows the graph of

$$z = y\sin x + x\cos y$$

Published in *Mathematics in School*, **15**, 2, pp. 33–37, (1986).

drawn using the program “Super3D” from the *Supergraph* package [1] (with the *z*-scale reduced by a third to give a decent picture).

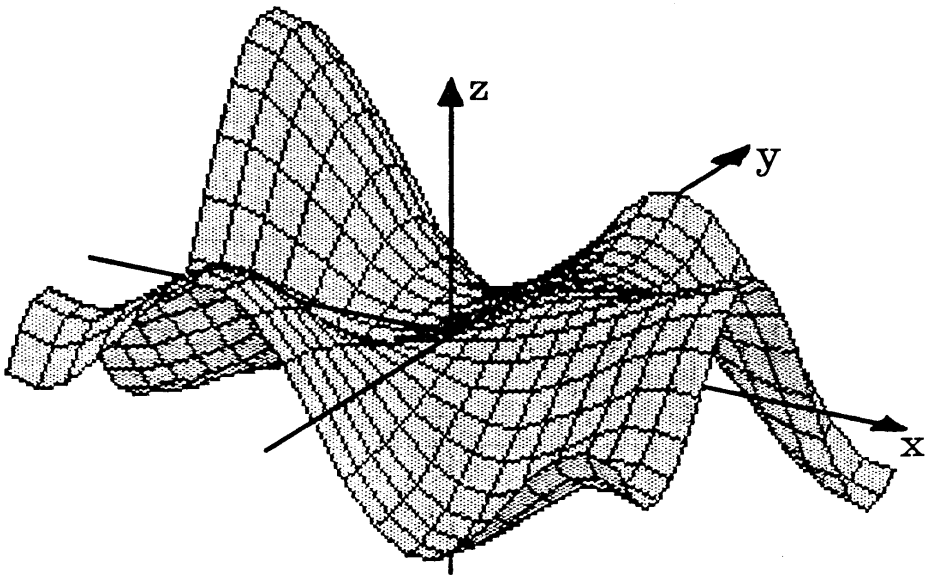

Figure 1 : the surface $z = y\sin x + x\cos y$

The lines drawn on the surface are those of the form x=constant and y=constant.

The implicit relation

$$y\sin x + x\cos y = 1$$

consists of all the points on contour lines height $f(x,y)$=l.

We could get an idea where the contour lines lie by calculating

$$z = y\sin x + x\cos y$$

at an array of points in the plane and mark the points to distinguish between those satisfying $z<1$, $z=1$ and $z>$l. If a point where $z=1$ occurs, this gives a point on a contour but, if not, there will be a contour somewhere between the places where $z>1$ and $z<$l. Figure 2 has a “–” marked at (x,y) wherever $z <$ l and a “+” where $z>1$, with the signs occurring at intervals of 0.5, If this is compared with the original surface, the valleys below a height $z=1$ correspond to the “–” signs and the hills to “+”.

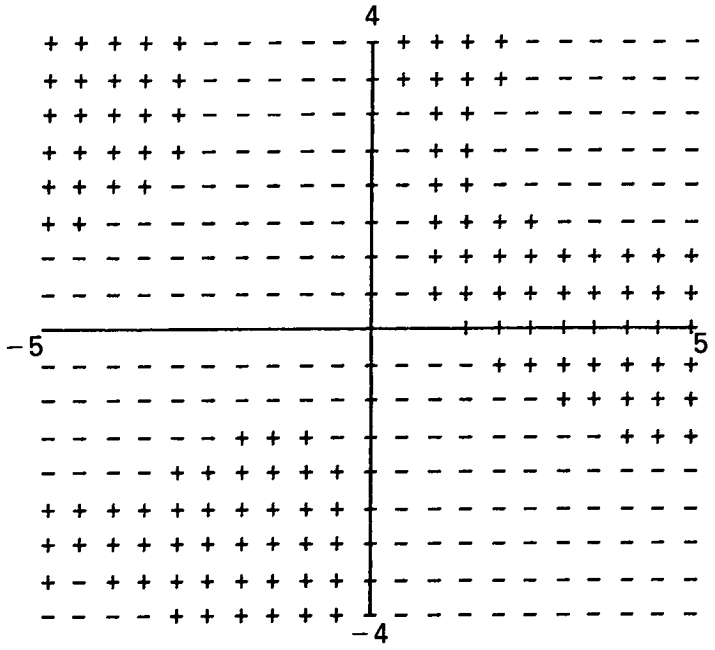

Figure 2 : The sign of the expression $y\sin x + x\cos y - 1$

The contour between them is the set of points (x,y) satisfying the implicit relation $y\sin x + x\cos y = 1$. It is in several parts. To find a point on the part in the top left quadrant, looking along the line $y=2$, reveals a change in sign between $x= -3.5$ and -3. The accurate position could be found by putting $y=2$ in the original equation:

$$2\sin x + x\cos(2) = 1$$

and solving it to find a root for x between $x= -3.5$ and -3, we shall see how this is done in the next two sections before attacking the main problem of tracing round the contour.

The gradient of a graph

The gradient between two points $(a, f(a))$, $(b, f(b))$ on a graph $y=f(x)$ can always be calculated as

$$\frac{f(b)-f(a)}{b-a}$$

but on a curved graph this will depend on the values of a, b. However, if a small segment of the graph highly magnified looks like a straight line then the calculation of the gradient using any two nearby points $(a, f(a))$, $(b, f(b))$ in the segment will give approximately the same value.

You may not believe that curved graph such as a circle can ever be nearly straight. In one sense you would be right. But imagine that you are a dedicated athlete (a strain on me but, I hope, not on you) and that you are running round a circular track a kilometre in circumference. It happens to be foggy so that you can only see ten metres ahead. The track

turns through 360 degrees in a kilometre, which is one degree in approximately 27 metres; in the limited visibility the track will look virtually straight, although in the large it turns full circle.

The same happens for a circle say $x^2+y^2=1$ drawn on A4 paper. It is clearly curved, but if a tiny part were redrawn to a very high magnification it would look virtually straight. Most graphs in terms of ordinary functions have this "locally straight" property when highly magnified. It is this property that is fundamental for differentiable functions in the calculus. Our purpose is to use this property *without* the calculus.

To calculate the gradient near a point x on a locally straight graph $y=f(x)$, we take a suitable tiny value, say e=0.000l, and work out the gradient from $(x, f(x))$ to $(x+e, f(x+e))$. The gradient is (approximately)

$$\frac{f(x+e)-f(x)}{e}$$

On a computer this can be done in two stages: first express the function in a user-defined form, say as a program line in BASIC. For instance, if the function is $f(x)=x^2-2$ then write

```
10 DEF FNf(x)=x^2-2
```

and define the gradient function $g(x)$ as

```
20 DEF FNg(x)=(f(x+e)-f(x))/e
```

Before using this expression you will need to type in a value of e, say

```
LET e=0.001
```

The (approximate) gradient of the graph near x=1 is then obtained by typing

```
PRINT FNg(1)
```

to give the value 2.0010001.

If e is changed to e =0.000567 then the gradient is given as 2.0005663, agreeing with the previous value to 4 significant figures. Likewise any other small value of e gives a gradient approximately equal to 2 near x=1. Sensible values of e are around 1E– 4 or so. Anything much smaller leads to loss in accuracy due to dividing one tiny number by another. The approximate gradient anywhere else may be obtained in the same way.

The Newton Raphson Technique

Suppose that $y=f(x)$ crosses the x-axis at $x=c$, so that $f(c)=0$ and that x_1 is near enough to c for the graph to be almost straight. (Figure 3.)

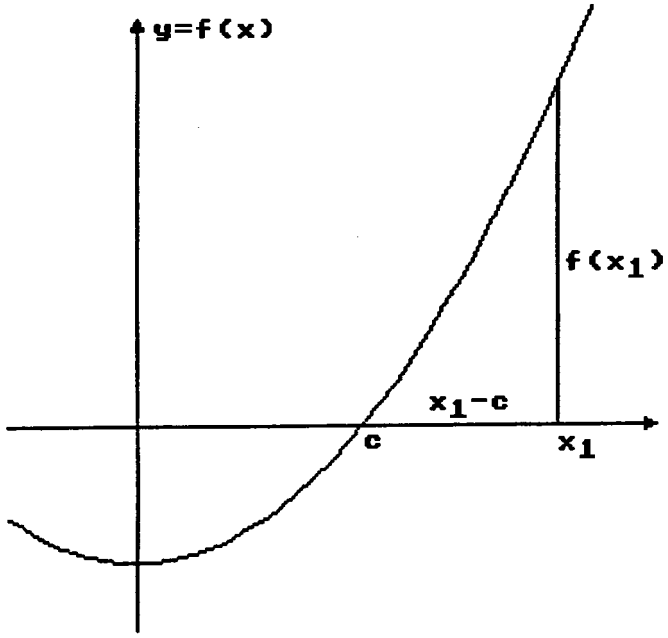

Figure 3 : The graph $y=f(x)$ with x_1 near where $f(c)=0$

The gradient of the graph from the point $(c,0)$ (where it crosses the x-axis) to $(x_1, f(x,))$ is just

$$\frac{f(x_1)}{x_1 - c}.$$

But the gradient near $x=x\ 1$, is also approximately

$$g(x_1) = \frac{f(x_1 + e) - f(x_1)}{e}$$

so

$$\frac{f(x_1)}{x_1 - c} \approx g(x_1)$$

and

$$c \approx x_1 - \frac{f(x_1)}{g(x_1)}.$$

If we let

$$x_2 = x_1 - \frac{f(x_1)}{g(x_1)}$$

then x_2 may be a better approximation and a repetition of the process using x_2 in place of x_1 may continue to get closer to the true value.

In a computer program we drop the suffix and simply say that if x is a fair approximation to a root, then

$$x–f(x)/g(x)$$

is usually a better one. We can then replace x by $x–f(x)/g(x)$ and repeat the process.

When $f(x) = x^2–2$, we may use the following program to print successive approximations to a root of $c^2–2=0$ until the result is accurate within 1E–5:

```
 5  p=0.001
10  DEF FNf(x)=x^2-2
20  DEF FNg(x)=(FNf(x+e)-FNf(x))/e
30  INPUT "x=" x
40  REPEAT
50   x=x-FNf(x)/FNg(x)
60   PRINT x
70  UNTIL FNf(x)<1E-5
```

For a computer language without the REPEAT command, delete line 40 and replace line 70 by

```
70  IF FNf(x)>1E-5 THEN 30
```

Starting from x= l the BBC computer gives the sequence:

```
1.4999764
1.41666808
1.41421537
```

The error tolerance 1E–5 maybe reduced: e=1E–5 on a BBC computer gives $\sqrt{2}$=1.41421356 to the maximum accuracy of the machine in only one more iteration. On a BBC computer we can replace line 10 by

```
10  INPUT "f(x)=" f$
15  DEF FNf(x)=EVALf$
```

to allow the function to be input as a BASIC string when the program is run. This technique will be used in the main program.

Tracing round a contour

The initial move in tracing an implicit curve $f(x,y)=c$ is to find a point on the graph from which to begin the drawing routine. The technique is to estimate a point (x,y) near the curve then to fix one of x or y and to move the other onto the curve using the Newton Raphson method. The main task is to trace round the contour. Walking in the Peak District recently I found that there were often several paths crossing each other on a mountain. It was often possible to start at a certain height and to move to another place at the same height by going down one path and turning onto

another and walking up. The surface $z=f(x,y)$ in the three-dimensional picture drawn earlier using Super3D is traversed by paths x=constant and y= constant. To move from one point at a certain height to another at the same height nearby, a good plan of attack is to walk down a line x=constant on the surface, then turn through 90 degrees and walk up a line y=constant an appropriate distance to return to the original height. If x varies and y stays constant, the gradient of the path on the surface above (x,y) is approximately

$$g_1(x,y)=(f(x+e,y)-f(x,y))/e.$$

Similarly, if y varies and x stays constant, the gradient of the path on the surface above (x,y) is approximately

$$g_2(x,y)=(f(x,y+e)-f(x,y))/e.$$

Moving an x-step h from (x,y) to $(x+h,y)$ therefore causes a rise in the z-direction by an amount

$$hg1(x,y).$$

At this point, keeping $x+h$ fixed and taking a step k in the y-direction to reach $(x+h,y+k)$ will further change the height by

$$kg_2(x+h,y).$$

The total change in height from (x, y) to $(x+h, y+k)$ is therefore approximately

$$hg_1(x,y)+kg_2(x+h,y).$$

If this returns to the former height then

$$hg_1(x,y)+kg_2(x+h,y).$$

This equation allows us to determine k in terms of h as

$$k=hg_1(x,y)/g_2(x+h,y).$$

(provided that $g_2(x+h,y)\neq 0$.)

However, an attempt to write h in terms of k gives

$$h=kg_2(x+h,y)/g_1(x,y).$$

which involves h on the right hand side. If $g_2(x+h,y)$ and $g_2(x,y)$ are fairly close, then the simplest way out of this dilemma is to replace $g_2(x+h,y)$ by $g_2(x,y)$ to get a simpler (but possibly worse) approximation

$$hg1(x,y)+kg_2(x,y)=0.$$

This single approximation can then be used to determine h in terms of k or k in terns of h, whichever proves to be more appropriate.

To keep h and k a moderate size, a neat trick is to specify a step and take the larger of k, h to be this size. First let

$H=g_2(x,y)$, $K=g_1(x,y)$

so that

$Hg_1(x,y)+Kg_2(x,y)=0$.

If $|H|>|K|$ (which, in particular, means $H\neq 0$) then we take the x-step to be size s. To make sure we go in the same direction as H, we actually take

$h=s*SGN(H)$,

and then put $k=h*K/H$.

The full rule (ii) in BASIC is:

```
IF ABS(H)>ABS(K) THEN h=s*SGN(H) : k=h*K/H
ELSE IF K<>0 THEN k=s*SGN(K) : h=k*H/K
```

The values of x, y are then replaced by $x+h$ and $y+h$ to give a new point further along the contour. Any inaccuracy nay be improved by a further use of the Newton-Raphson technique before taking the next step.

A program to sketch an implicit function

We are now in a position to introduce the main program. The axes are drawn by `PROCdraw_axes` and, after the input of the function $f(x, y)$ (as a BASIC string) and the constant c, the procedure `PROCplot_colour` plots an array of points (x,y):

in red if $f(x, y)<c$, yellow if $f(x, y)=0$ and white if $f(x, y)>c$.

(If you cannot easily distinguish yellow and white points on your monitor, add the line:

```
2015 VDU 19,3,6;0;
```

to change the white colour to cyan.)

The main REPEAT loop allows values of x,y to be input and the value of y to be improved to get closer to the contour line required. The step s is then requested for the program to use as it traces round the graph of the implicit function. The graph is drawn until the SHIFT KEY is touched (INKEY(–l)) and returns to allow a move somewhere else to draw another part of the contour. To draw a contour in the opposite direction, change the sign of the step.

To stop the program entirely, either hold down the CTRL KEY (to operate the final INKEY(–2) condition) at the same time as the SHIFT KEY, or press ESCAPE.

```
  10 MODE 1
  20 PROCdraw_axes
  60 INPUT "f(x,y)=" f$
  70 INPUT "c=" c
  80 PROCplot_colour : REM plot array of coloured dots for +/-
  90 REPEAT : CLS : REM - main program loop
 100  INPUT " exact x=" x
 110  INPUT "approx y=" y
 120  PROCimprove_y : REM get accurate y for given x
 130  PRlNT "exact y=" ;y
 140  PLOT 69,100*x,100*y : REM - plot start point
 150  INPUT "step     s=" s
 160  REPEAT : REM - drawing loop for a solution curve
 170    H=-FNg1(x,y):K=FNy2(x,y)
 180    IF ABS(K)<ABS(H) THEN
          h=s*SGNH:x=x+h:y=y+h*K/H:PROCimprove_y
        ELSE IF   K<>0 THEN
          k=s*SGNK:y=y+k:x=x+k*H/K:PROCimprove_x
 190    DRAW 100*x,100*y : REM - join to next point
 200  UNTIL INKEY(-1) : REM - until SHIFT key pressed
 210  UNTlL INKEY(-2) : REM - until CTRL key pressed
 220  END
1000  DEF  FNf(x,y)=EVALf$
1010  DEF  FNg1(x,y)=(FNf(x+e,y)-FNf(x,y))/e
1020  DEF  FNg2(x,y)=(FNf(x,y+e)-FNf(x,y))/e
2000  DEF  PROCdraw_axes
2010  VDU 29;640;480; : REM - set graph window size
2020  VDU 24,0,3,39,0 : REM - set text window size
2020  DRAW -500,0 : MOVE 500,0 : REM - start to draw axes
2030  DRAW 0,-500 : MOVE 0,500
2040  VDU 5 : MOVE -564,-4 : PRINT "-5"
2050  MOVE 500,-4 : PRINT "5" : VDU4
2060  ENDPROC
3000  DEPROCplot_colour
3010  FOR x=-5 TO 5 STEP 1/2
3020  FOR y=-s TO s STEP 1/2
3030    z = FNf(x,y)
3040    IF z<c THEN GCOL 0,1 ELSE
        IF z=c THEN GCOL 0,2 ELSE GCOL 0,3 :REM - set colour
3050    PLOT 6,100*x,100*y : REM - plot point colour
3060    IF INKEY(-1) THEN x=5 : y=5 :REM touch SHIFT to quit
3070  NEXT
3080  NEXT
3090  GCOL 0,2 : REM - reset colour
3095  ENDPROC
4000  DEF  PROCimprove_y
4010  IF  FNg2(xy)=0  THEN  ENDPROC
4020  REPEAT
4030  y=y-(FNf(x,y)-c)/FNg2(x,y)
4040  UNTIL  ABS(FNf(x,y)-c)<1E-5  OR  FNg2(x,y)=0  OR  INKEY(-1)
4050  ENDPROC
5000  DEF  PROCimprove_x
5010  IF  FNg1(x,y)=0  THEN  ENDPROC
5020  REPEAT
5030  x=x-(FNf(x,y)-c)/FNg1(x,y)
5040  UNTIL  ABS(FNf(x,y)-c)<1E-5  OR  FNg1(x,y)=0  OR  INKEY(-1)
5050  ENDPROC
```

When the program is RUN, input the function

$f(x,y)=x\text{^}2+y\text{^}2$

with constant and start from $(x,y)=(1,0)$ with step $s=0.1$. The implicit curve drawn is the circle

$x^2+y^2=1$.

The picture is satisfactory because the Newton-Raphson approximations in line 180 pull the calculations back onto the curve at every stage. Deleting `PROCimprove_y` and `PROCimprove_x` would cause a great deterioration in the program. With $s=0.1$ the solution would spiral out as in Figure 4.

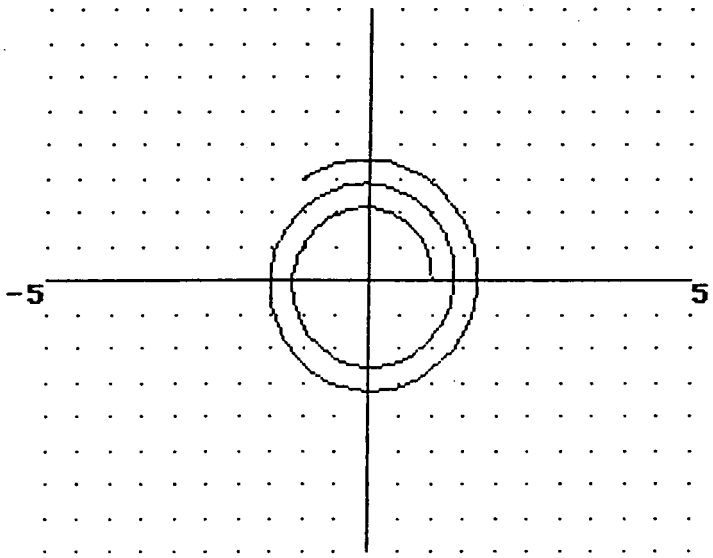

Figure 4 : Drawing without the improvement procedure

To draw the hyperbola

$x^2-y^2=1$,

start from (l,0) with $s=0.2$ then return to the same point with $s=-0.2$ to obtain the curve in the opposite direction, repeating the process from (0,1) to obtain the other branch. (Figure 5.)

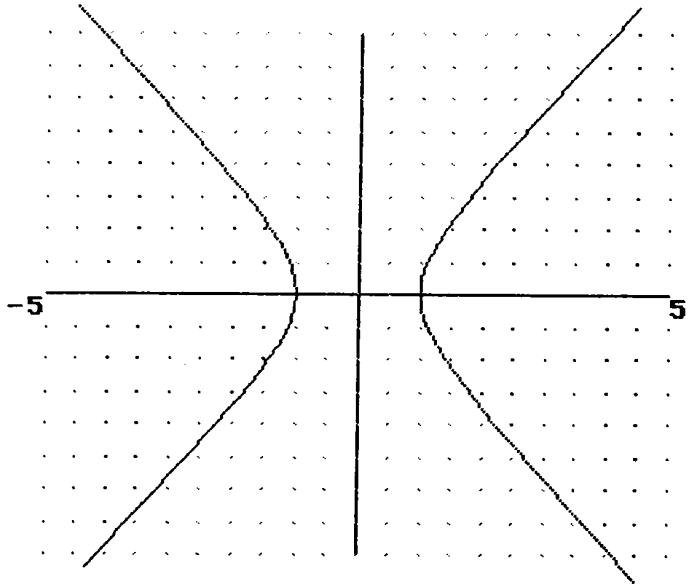

Figure 5 : the hyperbola $x^2–y^2=1$, with each branch drawn separately

The more difficult graph

$$y\sin x + x\cos y = 1.$$

may be drawn by using this technique several times over (Figure 6.)

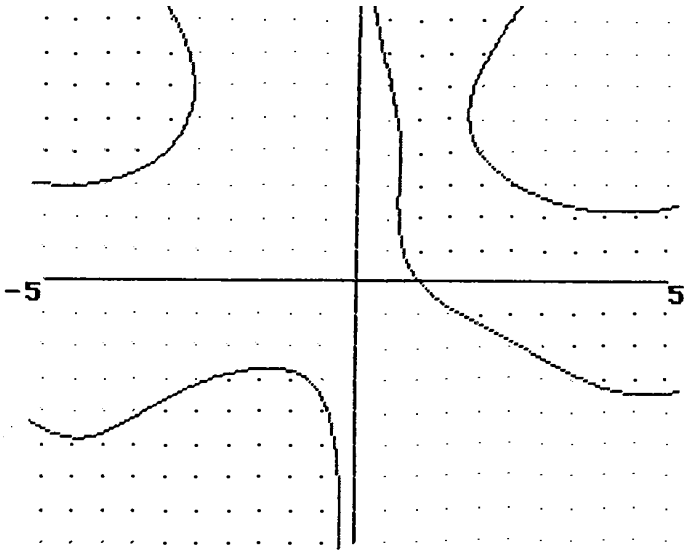

Figure 6 : The graph of the relation $y\sin x + x\cos y = 1$

Notice that the program does more than just draw the equation $f(x,y)=c$. It also shades the regions $f(x,y)<c$, $f(x,y) > c$, so it represents inequalities as well as equalities.

Use it to draw other implicit functions of your choice, but be warned that every step in drawing involves several evaluations, so complicated expressions may take much longer to draw. BBC BASIC is a fast version of the language, but it is not speedy enough when many calculations are required.

Modifications

It is possible to modify the program either to get more insight into how it works or to improve the facilities. To gain insight, one idea might be to insert

```
4005  count=0
4025  count=count+l
```

with the same lines at 5005 and 5025 and then add

```
 195  PRINT TAB(20,13);count;"    "
```

to see how many extra iterations are required to get the .implicit curve as accurate as desired. My computer has an Acorn speech chip so I added the line

```
 195  SOUND-1,48+count*100
```

instead, to hear Kenneth Kendall's dulcet tones call out the numbers[1].

The synchronisation is not perfect but it adds a certain cultural quality. It shows that the Newton-Raphson approximation requires very few iterations to reach the required accuracy, taking one or two more steps at tight corners.

An improvement might be to replace the dots in the `plot_colour` procedure by line segments in the direction of the curve using the following extra lines:

```
3052  H=-FNg2(x,y):K=FNg1(xy)
3054  1F H=0 THEN a=0 : b=12 ELSE
      G=K/H : a=12/SQR(1+G^2) : b=a*G
3056  PLOT 0,-a,-b : PLOT 1,a+a,b+b
```

Figure 7 shows the ellipse

$$x^2 + 3y^2 = 7$$

drawn with the technique, revealing the direction lines of other ellipses in the form

$$x^2 + 3y^2 = \text{constant}.$$

[1]The Acorn computer in 1986 had a speech chip with a small dictionary of words spoken by BBC Newsreader Kenneth Kendall. This included number names and was accessed by the SOUND command.

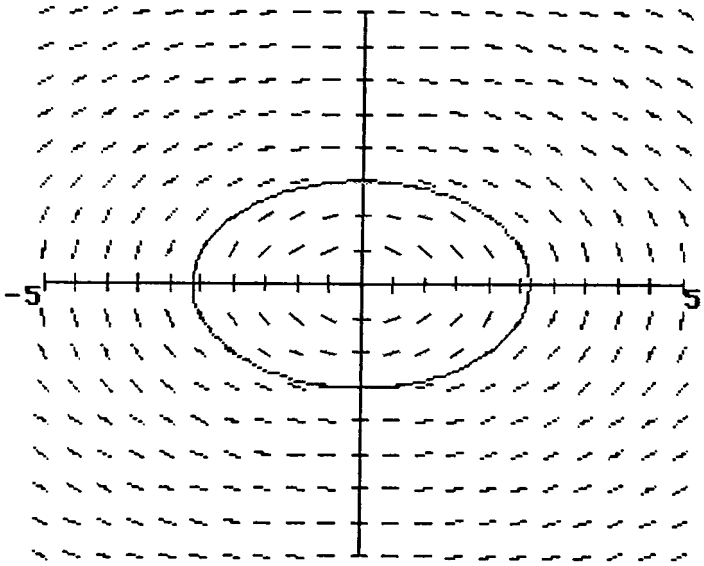

Figure 7 An ellipse with direction lines for other level curves

The opportunities for experiment are up to you, but that is the beauty of modifying a given program. Powerful ideas like this provide practical and useful mathematics. Programming numerical methods and graphing the results in this way should surely become part of the mathematics curriculum in the near future.

Challenge

Can you modify the program so that it draws implicit curves automatically without any intervention from the user once the function is typed in? (It may not be fast, but it should be accurate...)[2]

Reference

l. Tall, David (1985) *SuperGraph*, textbook and disc of interactive graph drawing programs for the BBC computer, Glentop Publishing, D.V.

[2] A program which does this is incorporated in the *Supergraph* suite of programs. It draws an implicit function in less than a minute. Later, when the Archimedes computer was introduced with a fast RISC chip, the same program operated in two or three seconds.