Project for Math 2605: Following level curves

In this project, you will use the tangent line approximation to graph in implicitly defined curve. Recall that implicitly defined curves are given by an equation of the form

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

For example, consider the equation for the circle of radius 2 centered at the origin. To write an equation for it in this form, we can use $f(x,y) = x^2+y^2$, and $c = 4$. This equation express the fact that the circle is a "level curve" of this function f.

The thing that we are going to take advantage of is that it is easy to write down an explicit parameterization of the tangent line to an implicitly defined curve. If $\mathbf{x}_0$ is any point in the plane, and if $\nabla f(\mathbf{x}_0) \neq 0$, then define

$$\mathbf{v} = (\nabla f(\mathbf{x}_0))^{\perp} \tag{1}$$

where by definition $\begin{bmatrix} a \\ b \end{bmatrix}^{\perp} = \begin{bmatrix} -b \\ a \end{bmatrix}$. Then a parametric form of the equation for the tangent line is

$$\mathbf{x}(t) = \mathbf{x}_0 + t\mathbf{v}\ .$$

The speed at which this is traced out depends on $|\mathbf{v}|$. The larger this number is, the greater the speed at which the tangent line is traced out. We can arrange to trace it out at unit speed if we replace $\mathbf{v}$ by the unit vector

$$\mathbf{u} = \frac{1}{|\mathbf{v}|}\mathbf{v}\ . \tag{2}$$

Our new parameterization is

$$\mathbf{x}(s) = \mathbf{x}_0 + s\mathbf{u}\ .$$

If we pick some small number $\epsilon > 0$, and then evaluate this at $s = \epsilon$, we get a point along the tangent line that is ϵ units of distance from $\mathbf{x}_0$. Call this point $\mathbf{x}_1$. That is

$$\mathbf{x}_1 = \mathbf{x}_0 + \epsilon\mathbf{u}$$

where $\mathbf{u}$ is computed in terms of f and $\mathbf{x}_0$ using (1) and (2).

Here is the algorithm for drawing the level curve through $\mathbf{x}_0$: We pick a short time interval $\epsilon > 0$. (How short this should be is one of the issues we are going to investigate here). We compute $\mathbf{v} = (\nabla f(\mathbf{x}_0))^{\perp}$, and then $\mathbf{u}$. We then follow the tangent line a distance ϵ arriving at the point $\mathbf{x}_0 + \epsilon\mathbf{u}$. Call this point $\mathbf{x}_1$. It is a new point that is "almost" on the level curve, since it is on the tangnet line, which is pretty close to the level curve near $\mathbf{x}_0$

Now, starting from $\mathbf{x}_1$, use the same procedure (replacing $\mathbf{x}_0$ in all formulas by $\mathbf{x}_1$) to define $\mathbf{x}_2$, and so on. Iterating this procedure produces a "list of dots". To graph the curve, connect the dots. That is it, but here is a more formal description:

Level Curve Graphing Algorithm

Declare an integer number of steps N, and array of $N+1$ points in the plane, and integer j to count steps, and a time interval ϵ. Set the value of the first entry of the array to be $\mathbf{x}_0$, and initialize $j = 0$. Then:

While $j < N$**:**

(1) Define $\mathbf{x}_{j+1}$ by

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \epsilon \mathbf{u}$$

where

$$\mathbf{u} = \frac{1}{|\mathbf{v}|}\mathbf{v} \qquad \text{and} \qquad \mathbf{v} = \left(\nabla f(\mathbf{x}_j)\right)^{\perp} \ .$$

Assign the value $\mathbf{x}_{j+1}$ to the $(j+1)$st entry in the array of points.

(2) Increment j; that is over write j with $j+1$.

When the while loop terminates, you have a list of dots. “Connecting the dots” gives the curve. To graph is, use another while loop. Reset $j = 0$, and then:

While $j < N$**:** Call on some paint method to draw the line segment connecting $\mathbf{x}_j$ and $\mathbf{x}_{j+1}$, and then increment j.

This algorithm draws an approximation to the level curve of f through $\mathbf{x}_0$. Actually, it draws a piece of it passing through $\mathbf{x}_0$ in one direction. You could get the other direction by changing the sign of ϵ, and repeating the procedure.

We now ask:

- *What can go wrong?*
- *How well does this work, when it does work?*

One obvious thing that can go wrong is that it might happen that $\nabla f(\mathbf{x}_j) = 0$ for some j. At this point, $\mathbf{u}$ is undefined, and the algorithm breaks down. A good implementation would have to incorporate some exception handling to deal with this contingency.

How well it works depends on how closely the approximation

$$f(\mathbf{x}_j) \approx f(\mathbf{x}_0)$$

holds true. The point is that since we are following the tangent line to the level curve, and not the level curve itself, even at the first step

$$f(\mathbf{x}_1) \neq f(\mathbf{x}_0) \ .$$

However, since the tangent line is a good fit – the best linear fit – the difference will be small. So in each step, we make a small error. But if N is large, we will be making many small errors. Could they add up to a large error?

The situation might seem pretty dire. If we want to follow the level curve a unit distance making steps of length ϵ, we will have to chose $\frac{N=1}{\epsilon}$ (or the first integer larger than this value). So as the step size goes to zero, the number of steps increases. Does it even help to take the step size smaller and smaller?

The answer is yes. The reason comes from the formula

$$f(\mathbf{x}_1) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\mathbf{x}_1 - \mathbf{x}_0) \pm C|(\mathbf{x}_1 - \mathbf{x}_0)|^2$$

with a constant C depending on the size of the second derivatives of f. In our case

$$\mathbf{x}_1 - \mathbf{x}_0 = \epsilon \mathbf{u}$$

where $\mathbf{u}$ is a unit vector orthogonal to $\nabla f(\mathbf{x}_0)$, so this reduces to

$$|f(\mathbf{x}_1) - f(\mathbf{x}_0)| \le C\epsilon^2 \ .$$

That is, we shift the value of f by at most $C\epsilon^2$ with each step. The worst case scenario is that every time the shift is by this much and with the same sign, so that in $N = 1/\epsilon$ steps the total shift has added up to

$$\frac{1}{\epsilon} C\epsilon^2 = C\epsilon \ .$$

This is very small if ϵ is very small!.

The conclusion is that after N steps, when we have gone a unit distance along the curve, we will be on a level curve not through $c_0 = f(\mathbf{x}_0)$ but through some nearby value $c_N = f(\mathbf{x}_0)$. How far have we wandered from the true level curve?

The question is: How far do we have to go from $\mathbf{x}_N$ to reach a point $\mathbf{x}$ at which $\mathbf{x} = c_0$?

To be concrete, suppose that the value c_N is too low:

$$c_N = c_0 - C\epsilon \ .$$

To increase our altitude as quickly as possible, we would proceed in the direction of steepest ascent, and "climb back up" to the level c_0. This direction is given by $\nabla f(\mathbf{x}_N)$, and the slope in this direction is $|\nabla f(\mathbf{x}_N)|$. Slope is rise over run, so to get a rise of $C\epsilon$, we need a run of

$$\frac{C\epsilon}{|\nabla f(\mathbf{x}_N)|} \ . \tag{3}$$

If this is not too small, we are not far away from the original level curve. In fact, if this distance is less than the distance corresponding to the width of one pixel in our graph, the descrepency will be invisible, and for all practical purposes, our graph is exact.

Notice once again that the ratio in (3) is undefined if $\nabla f(\mathbf{x}_N) = 0$, and becomes troublesomely large if $\nabla f(\mathbf{x}_N)|$ is small. Clearly, our algorithm can have trouble if the level curve wanders into a region in which $|\nabla f(\mathbf{x})|$ is small.

What to do

First write a program that implements the algorithm. You can do it as a java applet, and in that case, you may want to use the function parser package (provided on the project page) to incorporate text fields in which an expression for f can be entered.

Then try your program out. First take $f(x,y) = x^2 + y^2$ and $\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Let $N = 7/\epsilon$, which will be enough to "wrap around" the circle (since $7 > 2\pi$). Start with $\epsilon = 10^{-1}$, and draw the graph. This value of ϵ is no small enough for the path to "close" up into a circle. Try again for $\epsilon = 10^{-2}$, $\epsilon = 10^{-3}$ and so on until you find a value that is small enough for the circle to close. Notice that further decreasing ϵ from this point has no effect – try it out. This gives a good rule of thumb for when ϵ is small enough: If you repalce ϵ by $\epsilon/10$, and the graph does not change, then ϵ was small enough.

If you look at your graphs in which epsilon was too large, you will see that the curves "spiral out". That is, all of the errors are of the same sign, and tend to increase the radius. can you explain this?

Next, consider $f(x,y) = (x^2 + y^2)^2 - x^2 + y^2$ and $\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Try graphing this for the values of N and *epsilon* used for the circle. Do you ever run into points where the gradient of f is nearly zero? (Notice that $\nabla f(0,0) = 0$). What does you program do then?

How small do you have to take ϵ to get a good graph in this case? Discuss your reasoning.

Extra credit

We can improve the algorithm by incorporating a "corrector" at each step. At the jth step, define $\mathbf{y}_{j+1}$ using the old rule for $\mathbf{x}_{j+1}$:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \epsilon \mathbf{u}$$

where

$$\mathbf{u} = \frac{1}{|\mathbf{v}|}\mathbf{v} \qquad \text{and} \qquad \mathbf{v} = (\nabla f(\mathbf{x}_j))^{\perp} \ .$$

Now compute $f(\mathbf{y}_{j+1})$, which we would like to be equal to $f(\mathbf{x}_j)$. If it isn't, then go straight up or down hill as necessary to adjust the value. Since the slope at $\mathbf{y}_{j+1}$ in the directions of steepest ascent/descent is $\pm|\nabla f(\mathbf{y}_{j+1}|$, and since slope is rise over run, we can correct our altitude by moving to

$$\mathbf{x}_{j+1} = \mathbf{y}_{j+1} + \frac{f(\mathbf{x}_j) - f(\mathbf{y}_{j+1})}{|\nabla f(\mathbf{y}_{j+1})|^2}\nabla f(\mathbf{y}_{j+1}) \ .$$

For the extra credit:

(1) Explain the formula

(2) Write a program implementing the improved algorithm. Can you get away with larger values of ϵ (and therefore smaller values of N) using the improved program? If so, how much larger can ϵ be to get a graph of the same quality? Discuss the efficency of the new algorithm: There are fewer steps, but each one takes more computation. Does it really pay off?