Project for Math 2605

This project implements an adaptive algorithm for drawing level curves; i.e., "contour lines". It improves over the approach of "following $(\nabla f)^{\perp}$.

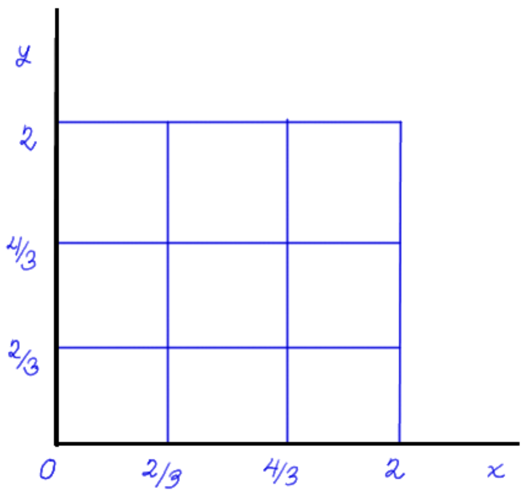
First, let us consider a simple non adaptive approach that we have discussed in class for graphing the contour curve defined implicitly by

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

Non adaptive method, using the tangent plane approximation: *Chop the region up into 'little square "tiles" of side length h. In each tile, replace f be its tangent plane approximation. This has the form $Ax+By+C$ for some values of A, B and C depending on the tile. The graph of $(Ax+By+C) = c$ is either empty, or is a line segment passing through the tile, or the whole tile. Draw all of the line segments that you fine this way.*

The problem with this approach is that you choose the value of h at the beginning. As we will see, that leads to trouble near critical points, where we will have to adapt the value of h.

Just to be concrete in what follows, take $f(x,y) = x^2+y^2$, $h = 1$, and take $a = c = 0$ and $b = d = 2$. Let's take a 3 by 3 grid, and so we divide up our region into 9 "tiles" with length and height 2/3.



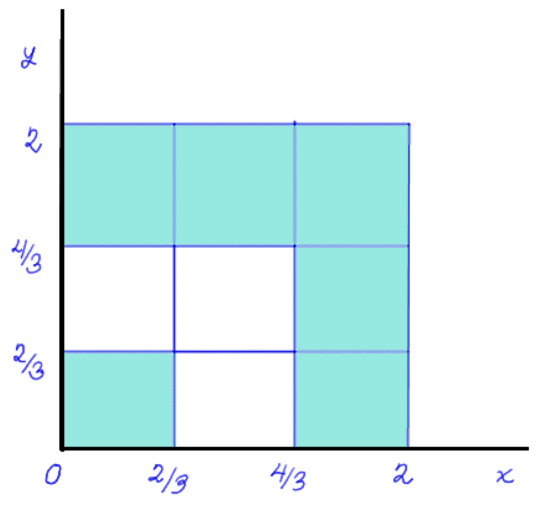
In each of these tiles, we could compute the tangent plane approximation to f at the center, and then work out the line segments.

This would involve computing derivatives, and we can sidestep this by *implicitly* using the tangent plane approximation.

- *If $h(x,y)$ is linear, you can draw the line segment given by $h(x,y) = c$ in a tile just using the values of h at the corners of the tile.*

Here is how this goes:

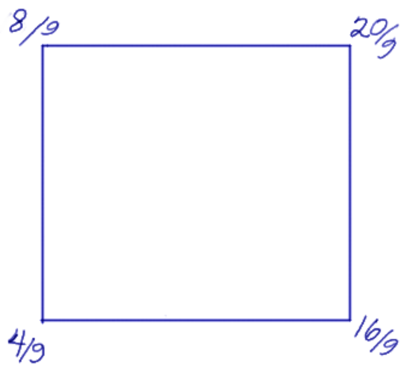
Step 1: Compute $f(\mathbf{x})$ at each $\mathbf{x}$ at the corners of the tile. If $f(\mathbf{x}) < c$ at *each* of the four corners, or if $f(\mathbf{x}) > c$ at *each* of the four corners, assume that the level curve misses that tile entirely. In our example, this happens in each of the 6 shaded squares below.



The level curve cuts through each of the three unshaded squares.

Step 2: In each tile for which If $f(\mathbf{x}) < c$ at *some* of the four corners and $f(\mathbf{x}) > c$ at *some other* of the four corners, draw a line segment separating the "too high" and "too low" corners. To get it in the right place, use linear interpolation on the edges.

To explain this, consider the unshaded tile in the bottom row. Here is a graph showing the values of f at each of its corners.



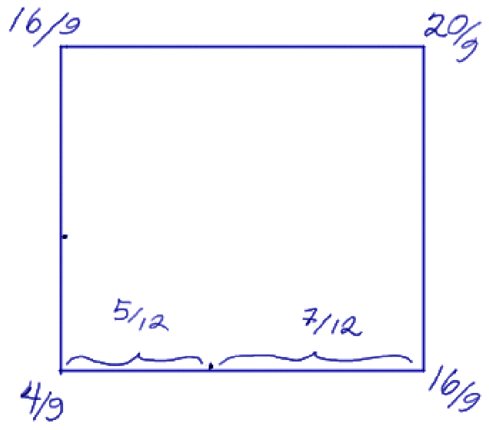
The values are "too low" on the left, and "too high" on the right. So we will draw in a line segment that runs vertically, top to bottom, dividing them.

But where exactly should the endpoints fall? To answer this for the bottom end, assume that f is varying linearly on the bottom edge of the tile, which would be the case if the tangent plane approximation were exact.

Then write 1, the level value as a weighed average of the corner values:

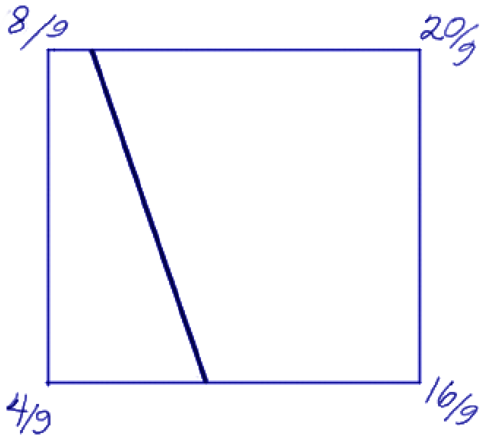
$$1 = a(4/9) + (1 - a)(16/9) \ .$$

Solving for a, we find $a = 5/12$, which means that the bottom end of the segment should fall 5/12ths of the way across the bottom edge of the tile:

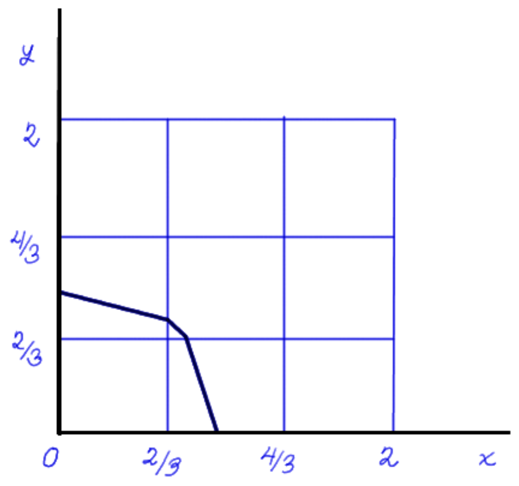


Repeating this for the top edge, we find that the upper end of the segment is just 1/12th of the way across the edge. Indeed, 8/9 is a lot closer to 1 than is 20/9.

Here is the line segment cutting across this tile, connecting the points we found, whose coordinate work out to be $(0, 34/36)$ and $(2/3, 26/36)$:



Dealing with the other three tiles across which the level curve cuts, we produce the graph



Of course, the "true" picture is a semicircular arc running from $(1,0)$ to $(0,1)$. But with such a coarse grid, we can be happy with this not *so* coarse depiction.

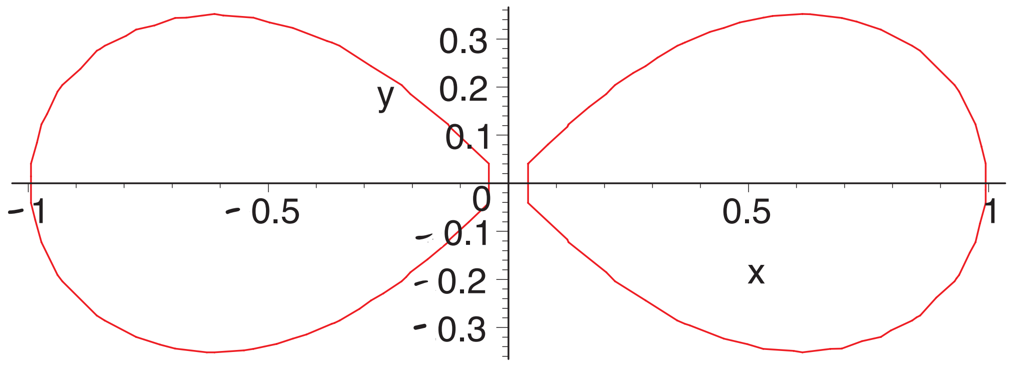
Many programs, including Maple, use a similar scheme based on a fixed grid. This would be fine for $x^2 + y^2 = 1$, but that is because there is no critical point of f on this curve.

Consider the function

$$f(x,y) = (x^2 + y^2)^2 - x^2 + y^2 \ .$$

The level curve given by $f(x,y) = 0$ is the *Bernouli lemiscate*, better known as the infinity symbol.

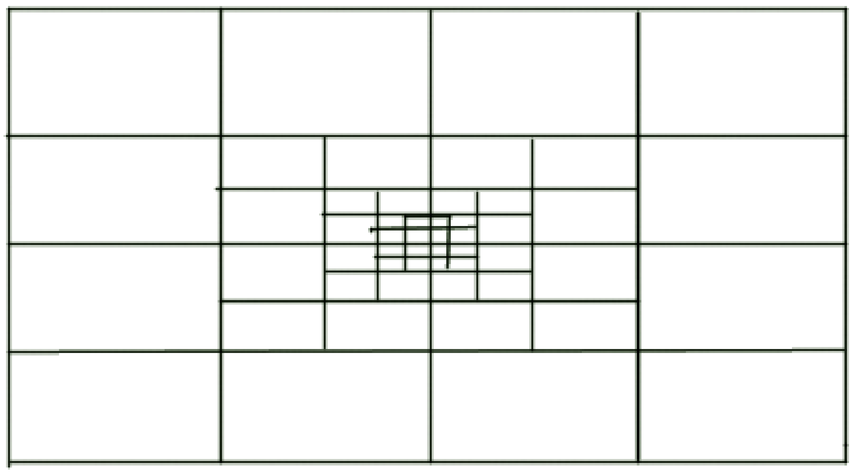
Here is a graph of the level curve $f(x,y) = 0$ using a 50×50 grid; i.e., with 2500 tiles drawn by this method.



Using this method, we didn't quite get an infinity symbol. There is a problem near $(0,0)$, though apart from there, things look pretty good. Indeed, a 50×50 grid is appropriate except near $(0,0)$.

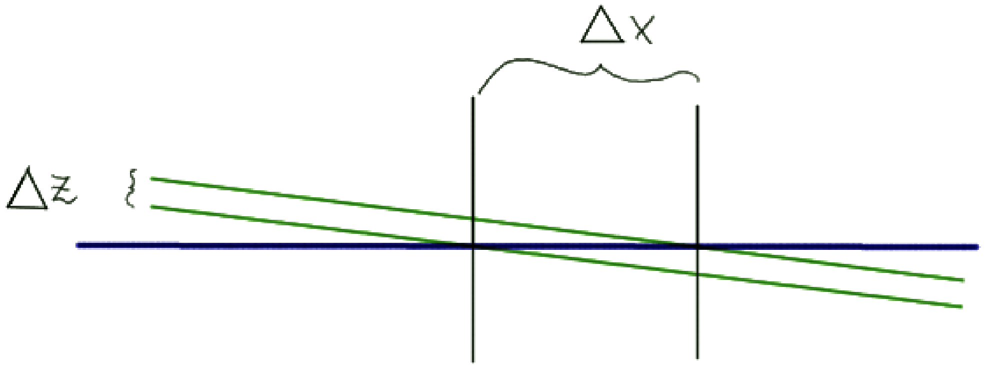
What goes wrong at $(0,0)$? It is a critical point of f, that's what. To deal with this, we use an adaptive approach.

An adaptive approach *We proceed as before, but if a tile is "problematic", we subdivide it into four smaller tiles. If these are "non problematic", use them. Otherwise, if any of these tiles are problematic, subdivide again. Keep going until all tiles are "non problematic", or "problematic" get cut down a single pixel,* For the Bernoulli lemiscate, using the adaptive approach, you you might arrive at a grid like



To decide whether a tile is problematic or not, we have to analyze what can go wrong. The errors in the tangent plane approximation are measures by the size of the Hessian. The bigger the Hessian, the more the height given by the tangent plane will be off.

Even if the true graph is relatively planar, a small error in the height of the plane can result in a large error in where it cuts across $z = h$ for any given h if the plane is nearly horizontal. The next drawing shows this:



As we know, the error Δz in the height using the tangent plane approximation at the center of a square tile of width h is no more that

$$\frac{1}{2}\|H_f\|_{\mathrm{HS}}(h/\sqrt{2})^2\ .$$

Using $|\nabla f|$ as a measure of the slope, we can expect the horizontal error Δx to be

$$|\Delta x| \approx \frac{\|H_f\|_{\mathrm{HS}}^2}{4|\nabla f|} h^2 \ .$$

• *Now, using the values of H_f and ∇f computed at the center of the tile, we decide that the tile is "non problematic" if the quantity $\Delta x|$ amounts to no more than one pixel width.*

The problem with this criterion is of course that it requires us to compute the Hessian at the center of the tile. There is discussion of how to do this, and of a way to avoid it, at the ned of the project.

Here is a pseudo code description of the algorithm:

Adaptive algorithm for drawing level curves

We are given a region

$$a \le x \le b \qquad \text{and} \qquad c \le y \le d \ . \tag{0.0}$$

in which we will draw our graph, a function $f(x, y)$, and a height h for the level curve. The goal is to graph the level curve given by $f(x, y) = h$.

The algorithm requires a fixed constant ϵ, representing the width of one pixel. It will also use two variable length array varaiables*: The *tile array*, and the *line segment array.* The tile array will store a list of "rectangle objects". The data in each such object will specify the corners of of the rectangle, and the object will have methods for reporting the coordinates of the corners, its height and its width. The line segment array will store line segment objects. These will have a method for drawing the segment to the screen.

Initially, we break up the region specified by (0.0) into a regular 10 by 10 grid of tiles, and initialize the tile array by list these in it. Initially the line segment array is empty. Then:

While the tile array is not empty, "pop" the first tile in the array off. Compute to see if it is "good" or not.

If the tile is good, compute the corresponding line segment, and push it onto the array of line segments.

Else, if the rectangle is not good, first check to see if its height and width are both larger than ϵ. **If** so, subdivide it by bisection into four smaller tiles, and "push" them onto the tile array. **Else**, discard the tile.

When the while loop terminates, the tile array is empty, and the line segment array has all of the line segment in it (if any) that we shall draw:

While the line segment array is not empty, "pop" the first line segment in the array off, and draw it to the screen.

* These are often called stacks.

What to do: The project is to write a program that draws level curves using the adaptive algorithm. As a litmus test, it should do a good job with the Bernoulli lemiscate at the critical point $(0,0)$. Produce some graphs for other functions with critical points.

for purposes of comparison, disable the adaptive feature of your program so that it just uses a fixed default mesh size h. Try this for different values of the default mesh size, and compare your results with the graphs produced by the adaptive algorithm.

Notes on the implementation. You will need to compute derivatives to compute the gradients and Hessians. Use the approximations

$$g'(x) \approx \frac{g(x+\delta)-g(x)}{\delta} \qquad \text{and} \qquad g''(x) \approx \frac{g(x+\delta)+g(x-\delta)-2g(x)}{\delta^2}$$

where δ is very small – say 10^{-10}. Starting from here, you should be able to deduce formulae for all the derivatives you need.

Alternatively, solve for A through F in

$$f(x,y) \approx C + A(x-x_0) + B(y-y_0) + D\frac{(x-x_0)^2}{2} + E(x-x_0)(y-y_0) + F\frac{(y-y_0)^2}{2}$$

by solving a least squares problem given by trying to match the two sides above that the four corners of the tile, the midpoints of each edge, and the center. This gives us 9 equations, which are linear, for 6 unknowns. More specifically, the least squares problem has the form

$$M\mathbf{a} = \mathbf{b}$$

where $\mathbf{a} = \begin{bmatrix} C \\ A \\ \vdots \\ F \end{bmatrix}$, where M is the 9×6 matrix whose ith row is

$$[\,1 \quad (x-x_0) \quad (y-y_0) \quad (x-x_0)/2 \quad (x-x_0)(y-y_0) \quad (y-y_0)^2/2\,]$$

evaluated at one of the 9 points mentioned above, and where the ith entry of $\mathbf{b}$ is the value of f at this same point.

Actually, the mean residual $|M\mathbf{a}-\mathbf{b}|^2/9$, where $\mathbf{a}$ is the solution, is a good proxy for $(\Delta z)^2$, and of course $\sqrt{A^2+B^2}$ is a good proxy for the slope. So a very good way to implement this is avoid computing derivative altogether, and just use the size

$$h^2\sqrt{\frac{|M\mathbf{a}-\mathbf{b}|^2}{9(A^2+B^2)}}$$

in the criterion for non problematic tiles. This is the recommended way.