Math 128a - Program 1 - Due March 14

Your assignment is to implement a program which computes and plots a curve in the x, y plane, defined as the solution of a implicit equation $f(x, y) = 0$. Your inputs will be

1. A function that evaluates $f(x, y)$, $\partial f(x, y)/\partial x$ and $\partial f(x, y)/\partial y$.
2. A starting point $[x_1, y_1]$ that you may assume satisfies $f(x_1, y_1) = 0$.
3. A suggested value *stepD* for the maximum distance between adjacent points on the curve.
4. A *stepcontrol* flag which is 1 to perform stepsize control (explained below) and 0 not to.
5. A $range = [xmin, xmax, ymin, ymax]$ outside of which you will not solve for or plot the curve.
6. A tolerance *tol* for computing each point on the curve (see below).
7. The maximum number $maxN$ of points on the curve to compute.
8. The maximum number $maxM$ of Newton steps to perform.

Your output should be

1. The number of points N found on the curve.
2. The total number M of Newton steps actually used.
3. The average number of Newton steps per point $avgMpt = M/N$.
4. The maximum number of Newton steps $maxMpt$ needed at any point.
5. Two arrays of points on the curve, $x(1 : n)$ and $y(1 : n)$, in their order along the curve.
6. A plot of the curve. You should plot x's marking the points, as well as "connect the dots" and circle the initial point, via the matlab command `plot(x,y,'k',x,y,'bx',x(1),y(1),'ko')`.
7. The maximum value $maxf$ of $|f|$ at all the points you computed along the curve (should be at most *tol*, as explained below).
8. The maximum and minimum distances $\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$ between any two points on the curve (should be at most about *step*), including $\sqrt{(x_n - x_1)^2 + (y_n - y_1)^2}$ if your algorithm decides that the curve is closed (see below). These maximum and minimum distances are called $maxD$ and $minD$, respectively.
9. A plot of the actual stepsize used to get the starting point for each (x_i, y_i) (see below).
10. As a check, to get a crude idea of what the answer should be you can try making the following (generally much more expensive) plot: compute $f()$ at each point in a 50-by-50 grid in $[xmin, xmax, ymin, ymax]$, and plot a green dot at each point where f is negative, and a red dot at each point where f is positive or zero. The borders between the red and green regions roughly show where the curve is. The following matlab code will do this:

```
[X,Y]=meshgrid( xmin:(xmax-xmin)/50:xmax , ymax:(ymin-ymax)/50:ymin );
% The next lines assumes you have written func to accept vector and matrix
% inputs, and evaluate the function and its derivatives at each point
[f,dfdx,dfdy] = func(X,Y);
hold off, clf, spy(f<0,'g'); hold on, spy(f >= 0, 'r')
```

Try this for `xmin=ymin=-1`, `xmax=ymax=1`, and $f(x,y) = x^2 + y^2 - .5^2$. It will not always give you the right curve, but it is useful to look at.

This is an interesting problem with several solutions. Here are some suggestions.

Start by showing that the tangent line to the curve defined by $f(x,y) = 0$ at a point $(\hat{x}, \hat{y})$ has slope

$$t = -\frac{\frac{\partial f}{\partial x}(\hat{x}, \hat{y})}{\frac{\partial f}{\partial y}(\hat{x}, \hat{y})}$$

1. Using the slope t to approximate the curve at (x_1, y_1) by a straight line, compute an approximate point $(\hat{x}_2, \hat{y}_2)$ on the curve a distance *step* away from (x_1, y_1). In other words, $\sqrt{(\hat{x}_2 - x_1)^2 + (\hat{y}_2 - y_1)^2} = step$ and $(\hat{x}_2, \hat{y}_2)$ lies on the straight line.

2. Using $(\hat{x}_2, \hat{y}_2)$ as a starting guess for Newton's method, find the next point (x_2, y_2) on the curve. To apply Newton's method, you need a function of a single variable, whereas f depends on two variables. Here is the idea. Since a straight line with slope t is tangent to the curve, a straight line with slope $t' \equiv -1/t$ is perpendicular to it, and moving along that perpendicular will be nearly the shortest way to get from $(\hat{x}_2, \hat{y}_2)$ to (x_2, y_2). In other words, we want to find a zero of $f(\hat{x}_2 + s \cdot q, \hat{y}_2 + s \cdot r)$ as a function of s, where $r/q = t'$. You should take $r = t'/\sqrt{1 + t'^2}$ and $q = 1/\sqrt{1 + t'^2}$, which better deals with the case where the slope is vertical (t' is ∞) or nearly vertical. Stop iterating when $|f| \leq tol$.

3. Keep repeating the last 2 steps to trace out the curve.

Here are some issues to consider in writing your code.

- If the curve has a vertical tangent, then the slope is ∞ at that point; this is *not* an error condition (consider a circle).

- Suppose that input $stepcontrol = 1$. If Newton does not converge in a reasonable number of steps (say 3) starting at $(\hat{x}_{i+1}, \hat{y}_{i+1})$, or if it converges to a point (x_{i+1}, y_{i+1}) much farther away from (x_i, y_i) than distance *stepD*, then replace *stepD* by *stepD*/2 and try again with a new $(\hat{x}_{i+1}, \hat{y}_{i+1})$. For the next iteration, try starting with *stepD* double the previous value (but not exceeding the original value supplied by the user). This way *stepD* will decrease to follow the curve more closely when it turns sharply, and then increase gradually so it can take larger steps in regions where the curve is smooth, and so be more efficient. If input $stepcontrol = 0$, leave *stepD* constant, but allow as many Newton steps to converge as required.

- Stop when either 1) the curve goes outside the window $[xmin, xmax, ymin, ymax]$, 2) you exceed N points on the curve, 3) you exceed M total Newton steps, or 4) the curve is closed and comes back "close enough" to its starting point (x_1, y_1). The first three conditions are to assure that you do not spend too much time, or go off to infinity. Condition 4 is not unusual (consider a circle).

- If the curves hits a boundary defined by $[xmin, xmax, ymin, ymax]$, go back to (x_1, y_1) and follow the curve in the other direction, to make sure you get all of it.

It is also possible that the curve might *bifurcate*, i.e. split like a "Y", or *self-intersect*, i.e. cross itself like an "8", or even just stop. Dealing with all these possibilities in full generality is difficult, and we do not expect you to do it. Bifurction or self-intersection points can occur where $\partial f/\partial x = \partial f/\partial y = 0$ (these are called *singular points*) which is where we know that Newton's method also has problems. Figuring out exactly the direction to follow the curves (or curves) after hitting such a singular point requires knowing higher derivative of f than just the first derivative (or estimating them numerically). We will ask you to try tracing several curves like this, just so you can see what happens, but will not expect you to trace them out successfully.

Try your code on at least the following examples (matlab code for these functions is available on the web page). Try all examples with $stepcontrol = 0$ and $stepcontrol = 1$, and report on the difference in what happens. (if there is little difference – your judgement – only give details for $stepcontrol = 1$). If your algorithm cannot handle a particular curve, try to diagnose why (you do not necessarily have to fix it, as described above).

For each example, turn in all the output listed above (for stepcontrol=1 and possibly stepcontrol=0, if it is interestingly different), but excluding the table of values of (x_i, y_i) and the red/green plots code for which is given above. In particular, turn in

1. values of N, M, $avgMpt$, $maxMpt$, $maxF$, $maxD$, $minD$
2. plot of curve
3. plot of stepD used to obtain each point on curve, when stepcontrol=1

You will be graded based on

1. Ease of reading and clarity of explanation of how your program works (or is supposed to work). This includes the mathematical derivation and justification of your algorithm. You can use Matlab code on the class web page as a model (or lower bound) on how well-documented your code should be. See also the material on Programming Standards on the class web page. Your detailed mathematical derivations should be turned in separately, and do not have to be in comments in the code!
2. Whether you turn in all your plots and outputs as requested,
3. Your explanation of the output. Does your plot match the real curve, at least in part, and if not why not? As mentioned above, some curves have features, like self-intersections, that make them hard to plot, and we do not expect you to suceeed completely on all of them, but you should at least get part of each plot. What is the impact of stepcontrol=1? If your program does not succeed in plotting the whole curve, try to explain what feature of the curve (or point of the curve the program can't get past) is the problem for your numerical method.
4. How efficient your algorithm is, as measured by the number of points N and Newton steps M.

In all examples below, try $range = [xmin, xmax, ymin, ymax] = [-3, 3, -3, 3]$, $stepD = .1$, $tol = 10^{-12}$, $N = 1000$ and $M = 3000$. You might want to change the range to focus on an interesting part of the curve.

1. Straight line (`lin.m`): $f(x, y) = y - x - 1$, $(x1, y1) = (0, 1)$
2. Circle (`circle.m`): $f(x, y) = x^2 + y^2 - 1$, $(x1, y1) = (1, 0)$
3. Cubic curve (`cubic.m`): $f(x, y) = y - (x + 1) \cdot x \cdot (x - 1)$, $(x1, y1) = (1, 0)$
4. Rotated sine curve (`rotsin.m`): $f(x, y) = x - \sin(3 \cdot y)$, $(x1, y1) = (\sin(3), 1)$
5. Compressed sine curve (`compsin.m`): $f(x, y) = y - \sin(1/x)$, $(x1, y1) = (1, \sin(1))$ (try also range = [0,.1,-1.5,1.5] to see the interesting part).
6. Single point (`point.m`): $f(x, y) = (x - 1)^2 + (y - 1)^2$, $(x1, y1) = (1, 1)$
7. Figure 8 (`fig8.m`): $f(x, y) = (x^2 + y^2)^2 - (x^2 - y^2)$, $(x1, y1) = (1, 0)$
8. Cusp (`cusp.m`): $f(x, y) = y^2 - x^3$, $(x1, y1) = (1, 1)$