Eric Withrow
gtg333g@mail.gatech.edu
GTid 901665961
February 25, 2005
MATH 2605-A1

Project 1: Following Level Curves

Overview

This project implements a tangent line algorithm for drawing accurate graphs of level curves. The algorithm takes in a function, an initial point X_0 for where to start the curve and an epsilon value that determines how 'close' the next point will be drawn. The algorithm looks like this:

Set the number of steps to be taken as N

While i < N
    $X_{j+1} = X_j + epsilon*u$
    Where u = (1/norm(v)) * v, v = perpendicular(gradient of $f(X_j)$)
While i < N
    Plot each X.

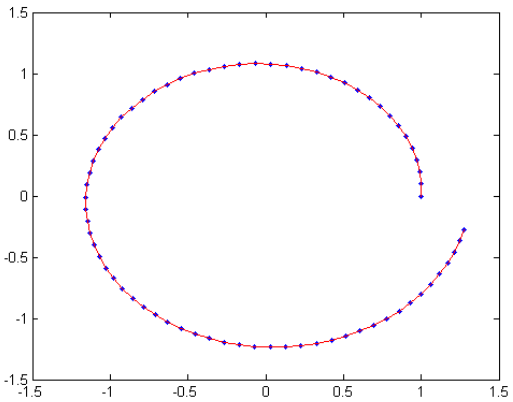
First Function

The first function used to test the program will be:
f(x,y) = x^2 + y^2 $x_0 = (1, 0)$

The first step is to enter the initial point into project1.m, the function into f.m, and the gradient of f ([2x, 2y]) into gradf.m.

The procedure will be to try lower epsilon value until finding a value that will draw a very accurate level curve.

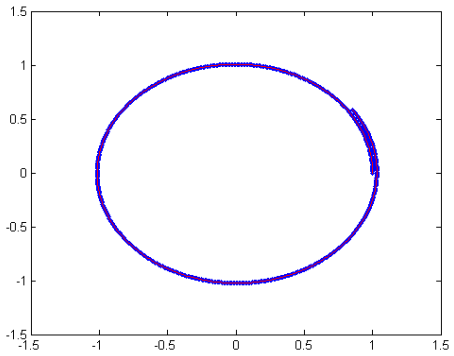
epsilon = 0.1, N = 7 / epsilon



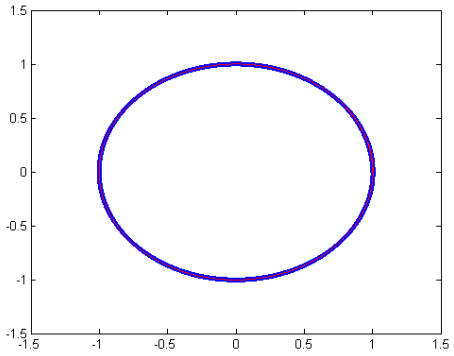
It is clear that the epsilon value is not small enough. As each step is taken around the circle, the new tangent line gets farther and farther away from the 'exact' tangent line.

The next step will be to try lower epsilon values in order to try to decrease the amount of error introduced into the graph.

epsilon = 0.01, N = 7 / epsilon



epsilon = 0.001, N = 7 / epsilon



It is clear that this epsilon value produces a very accurate graph, and the level curve is a perfect circle at $x_0 = (1, 0)$.

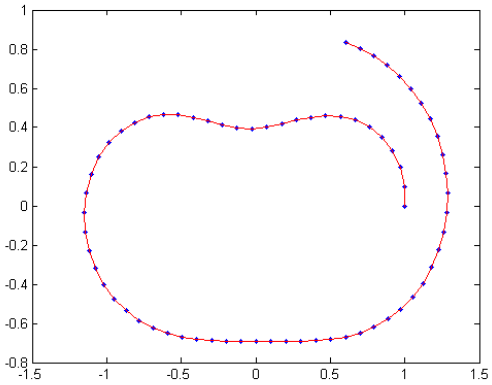
Second Function

The second function used to test the program will be:
f(x,y) = (x^2 + y^2)^2 − x^2 + y^2 $x_0 = (1, 0)$

The first step is to enter the initial point into project1.m, the function into f.m, and the gradient of f ([4X^3 + 2X(2Y^2 - 1), 4Y^3 + 2Y(2X^2 + 1)]) into gradf.m.

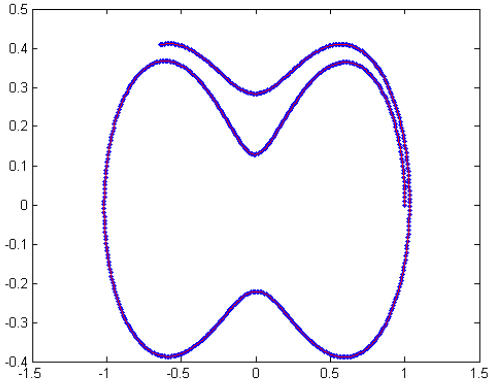
The procedure will be to try lower epsilon value until finding a value that will draw a very accurate level curve.

epsilon = 0.1, N = 7 / epsilon

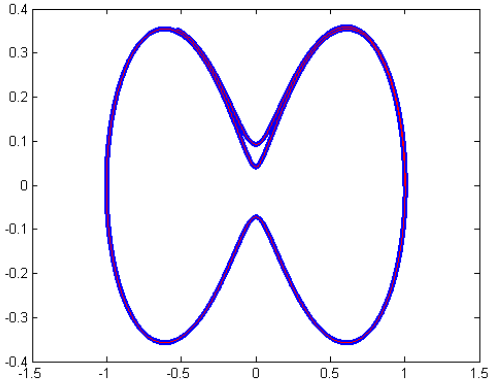


It is very difficult to see what is happening with such a large step size, so the epsilon value should be decreased

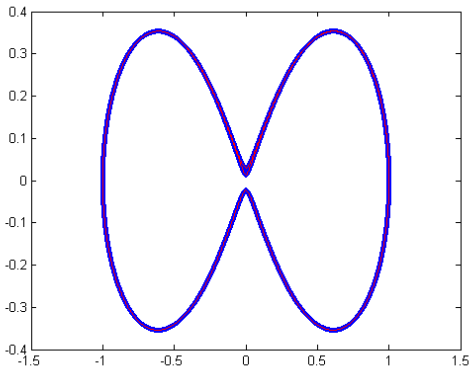
epsilon = 0.01, N = 7 / epsilon



epsilon = 0.001, N = 7 / epsilon



epsilon = 0.0001, N = 7 / epsilon



The plotted points in the graph with epsilon = .0001 overlap after going around more than once, so this is a good epsilon value. From this graph hyperbola-shaped graph we can easily conclude the function is some sort of saddle.

When the gradient of f is zero

It is possible to run into trouble in the algorithm if the gradient of f is computed to be zero at the point we are currently at. This would cause the norm to be zero, and the unitary vector in direction of v would be undefined due to a divide by zero error. In this case I choose to simply skip the point and move on to the next. This can be done in Matlab with a simple if statement:

```
if nrm == 0
    u = 0;
else
    u = (1/nrm)*v';
end
```

When the next point is calculated, epsilon*u will be zero so the undefined point will be replaced by the current point.

Moving forwards and backwards

It is possible to move backwards instead of forwards by simply using a negative value of epsilon. In my implementation, I simple chose to run the algorithm twice: once moving forwards, and once moving forwards from the same starting point. This produces a more symmetric graph but of course twice as many steps (2N) will be needed since half are taken up by going one way, and half by the other.

First I set the middle plot point to be

```
X(N/2,:) = X0;      % first point on the curve is X0
```

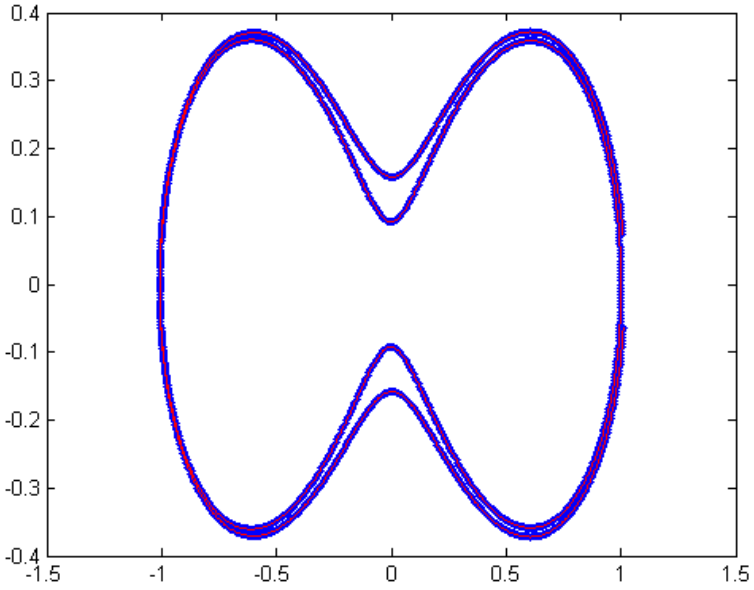
Then the algorithm runs forward from N/2 to N

```
% RUN FORWARDS for N/2 steps
for i=N/2:N
...
end
```

Then the algorithm runs backwards from N/2 to 1

```
% RUN BACKWARDS for N/2 steps
for i=[(N/2):-1:2]
...
end
```

This produces a symmetric graph:



Note that the epsilon value was purposely set a little too high to see the difference of moving both forwards and backwards.