Sam Wisnoski and Cooper Penkava

Quantitative Engineering Analysis 3: Flatlands

Matlab Code:
https://drive.google.com/drive/folders/1QyroWR8EvV3r1QdIjo3dWdfqfSIeeHKv?usp=sharing

I. Methodology

We decided to take steps, taking an approach where we find the gradient at the start point, rotate the neato to face in the same direction as that vector, and move along part of that vector a distance equal to the length of the gradient*lambda (which is a variable that approaches zero over time, as it gets multiplied by a constant, sigma, which is less than one). Then, at the point where the neato stops, we find the new gradient vector, rotate the neato to face in the same direction as that new gradient vector, and move along part of it a distance equal to the length of the gradient*lambda (a smaller lambda, since it was again multiplied by sigma). We continue this process until we reach a point close enough to the maximum, also known as the tolerance. We know we will eventually reach this point since with each step, the length of the gradient vector*lambda gets smaller, so as you approach the maximum, the gradient vector approaches zero.

The function that the contour map is based upon is:

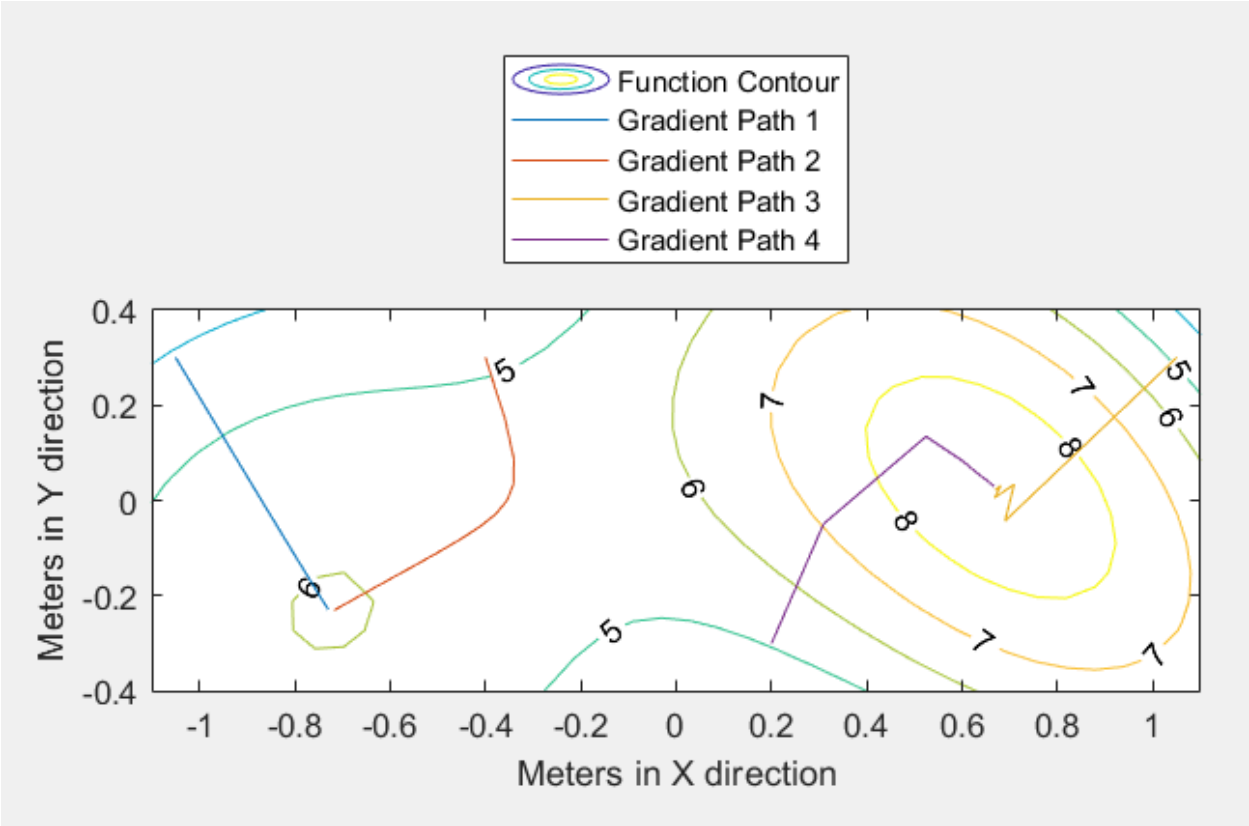
z = 8*exp(-1.5.*(x + y - .75).^2 - .5.*(x - y - .75).^2) + 6.*exp(-1.*(x + .75).^2 - (y + .25).^2)

The gradient is the vector with the partial derivative with respect to x and the partial derivative with respect to y.

We started with a lambda of 0.2 and multiplied it by 0.95 at each time step. To find out how long it should drive, we just set an arbitrary drive speed and used d = rt to find how long to drive for. To find out how long to rotate for, we used omega = $(V_R-V_L)/0.245$ and then used angle = omega*time to find how long to rotate for.

Note: the lambda and sigma values were determined by trial and error, making reasonable adjustments based on the performance of our algorithm until it worked for each starting point.

II. Contour plot



This is a graph of four separate simulated gradient ascent paths found using the method described in part I. The four starting points are, in order, (-1.05, 0.3, -45°), (-0.4, 0.3, -90°), (1.05, 0.3, -135°), and (0.2, 0.3, 90°).

III. Screen Capture

Our screen capture was submitted with this document to the Canva assignment.

IV. Physical Neato

The video of our physical neato was submitted with this document to the Canva assignment.

V. Matlab Code

https://drive.google.com/drive/folders/1QyroWR8EvV3r1QdIjo3dWdfqfSIeeHKv?usp=sharing

VI. Try your code starting at (−1.05, .3), with the Neato facing −45◦ (a southeast heading). Does it still work? Why or why not? How might you edit your code to work for all points?

It does in fact work for all points. However, earlier in the project, our code would not have worked. The problem with our earlier code was how we determined the angle of the new gradient vector in order to rotate at each step. At first, we had coded a solution that just found how far to turn based on the arctan of the y-component of the gradient divided by the x-component of the gradient. This was an issue because arctan can only return values between -90 and 90 degrees, despite the fact that the neato sometimes needs to turn more than that. We fixed this by implementing an if statement that has the following effect: if x is ever negative, and therefore the neato would need to turn you subtract 180 degrees from the angle to turn (subtracting instead of adding means you turn for less time and implement less neato error, adding 180 degrees would make the neato turn in at least a 360 degree circle before reaching the desired angle).

VII. Edit your code to work for any starting point

As we touched on above, our code did not work for any point at first, but now, it should work for any point. By allowing the angle to be above 90 degrees, any starting orientation should work due.

VIII. Write a brief description of how the behavior of gradient ascent changes based on the current point you are at.

As you get closer and closer to the maximum, the rate of change is nearing zero (based on the definition of a critical point) So, gradient ascent changes as you get closer and closer to the maximum in that the gradient portion gets smaller and smaller, eventually nearing zero. Additionally, the lambda value will continue to get smaller with each step, which also lowers the vector that the neato travels.

Another way the gradient ascent changes based on the current location has to do with the number of nearby maximums. If there are two local maxes right next to each other, the contours will form a saddle point. If the neato were in between the two but

below the saddle point, it would most likely first climb to the saddle point, then decide which gradient is steeper from there and climb to that maximum.

Finally, one last behavior lies on the steepness of the curve. Gradient ascent, implemented in steps, follows the gradient vector in each step. In our approach, the length of the gradient vector influences how far it drives. This means, when the neato is located on a steeper part of the curve (that is, the gradient vector is larger), it will travel much further in that direction than it would in a part of the curve that is not steep.