Homework Assignment 4

Abhay Gupta (andrew id: abhayg)

November 18, 2018

1 Question 5

Consider a rectangle of perimeter of k, then the perimeter of the rectangle with length x and breadth y is given by $2x + 2y = k \implies 2x + 2y - k = 0$, which is the constraint $h(x) = 2x + 2y - k = 0$. We have to find the rectangle of maximum area $a(x, y) = xy$, which is equivalent to minimizing the area $a(x, y) = -xy$. Thus, applying the constraints, we get,

$$\begin{aligned}
F(x, y, \lambda) &= -xy + \lambda(2x + 2y - k) \\
\nabla F(x, y, \lambda) &= \begin{pmatrix} -y + 2\lambda \\ -x + 2\lambda \\ 2x + 2y - k \end{pmatrix} \text{ taking derivative to compute maximum} \\
\nabla F(x, y, \lambda) &= 0 \\
&\implies y = 2\lambda \\
&\implies x = 2\lambda \\
\text{Putting this in } 2x + 2y = k, \text{ we get } 8\lambda = k &\implies \lambda = \frac{k}{8} \\
&\implies y = \frac{k}{4}, x = \frac{k}{4}
\end{aligned}$$

Thus the area is given by $a(x, y) = k^2/16$ and it is achieved at $x^* = \begin{pmatrix} k/4 & k/4 & k/8 \end{pmatrix}^T$

For the second-order sufficiency conditions, we first need to compute the Hessian of $a(x, y) = -xy$. Doing this, we get,

$$\begin{aligned}
\nabla a(x, y) &= \begin{pmatrix} -y \\ -x \end{pmatrix} \\
\nabla^2 a(x, y) &= \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}
\end{aligned}$$

Now considering the hessian of $h(x, y) = 2x + 2y - k$, we get,

$$\begin{aligned}
\nabla h(x, y) &= \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
\nabla^2 h(x, y) &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}
\end{aligned}$$

Now consider the vector $d = \begin{pmatrix} d_1 & d_2 \end{pmatrix}^T$ to be such that $d^T \nabla h(x, y) = 0$ at $x = x^*$.

$$\begin{aligned}
d^T \nabla h(x, y)\Big|_{x=x^*} &= \begin{pmatrix} d_1 & d_2 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\
&= 2d_1 + 2d_2 = 0 \implies d_1 = -d_2
\end{aligned}$$

Now taking the inner product $d^T \left(\nabla^2 a(x,y) + d^T \nabla^2 h(x,y) \right) d$, and substituting the above results, we get,

$$\begin{aligned} d^T \left(\nabla^2 a(x,y) + d^T \nabla^2 h(x,y) \right) d \Big|_{x=x^*} &= d^T \left(\nabla^2 a(x,y) \right) d \Big|_{x=x^*} \\ &= \begin{pmatrix} d_1 & d_2 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \Bigg|_{x=x^*} \\ &= \begin{pmatrix} d_1 & d_2 \end{pmatrix} \begin{pmatrix} -d_2 \\ -d_1 \end{pmatrix} \Bigg|_{x=x^*} \\ &= -2d_1 d_2 \Big|_{x=x*} \\ &= 2d_1^2 \end{aligned}$$

Thus the quantity $d^T \left(\nabla^2 a(x,y) + d^T \nabla^2 h(x,y) \right) d$ is always greater than 0 implies it is a positive definite matrix, at the minima x^*. This proves the second-order sufficiency conditions.

2 Question 6

(a) The path after 1 iteration is given by Figure 1. To run the code, `trajectory_optimization(1,1)`. The path after convergence is given by Figure 2. To run the code, `trajectory_optimization(1,500)`. For convergence, I checked the norm of the path and compared it with the norm of the prev path and if the difference was lesser than a tolerance of `1e-3`, I stopped the descent.

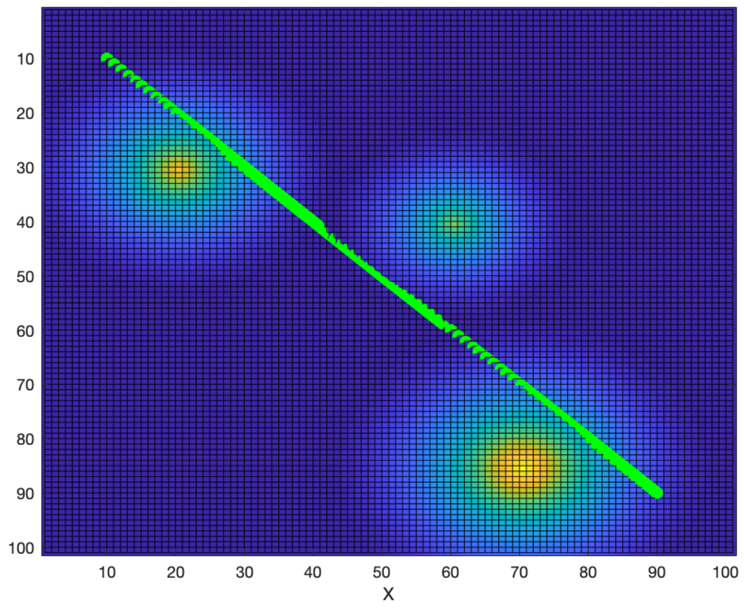

Figure 1: Line after 1 iteration

After convergence, we see that the points that are near the start and goal move away from them as we have not accounted for the fact that this should be a path and not individual points moving in space. Hence, the robot would never actually reach the goal position after vanilla optimization.

(b) The path after 100 iterations is given by Figure 3. To run the code for 100 iterations, `trajectory_optimization(2,100)`. The path after 200 iterations is given by Figure 4. To run the code for 200 iterations, `trajectory_optimization(2,200)`. The path after 500 iterations is given by Figure 5. To run the code for 500 iterations, `trajectory_optimization(2,500)`.

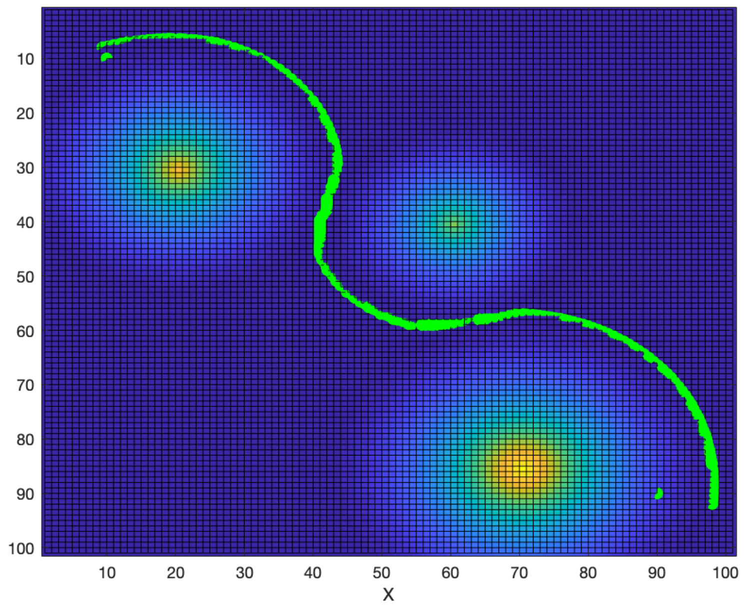

Figure 2: Line after convergence

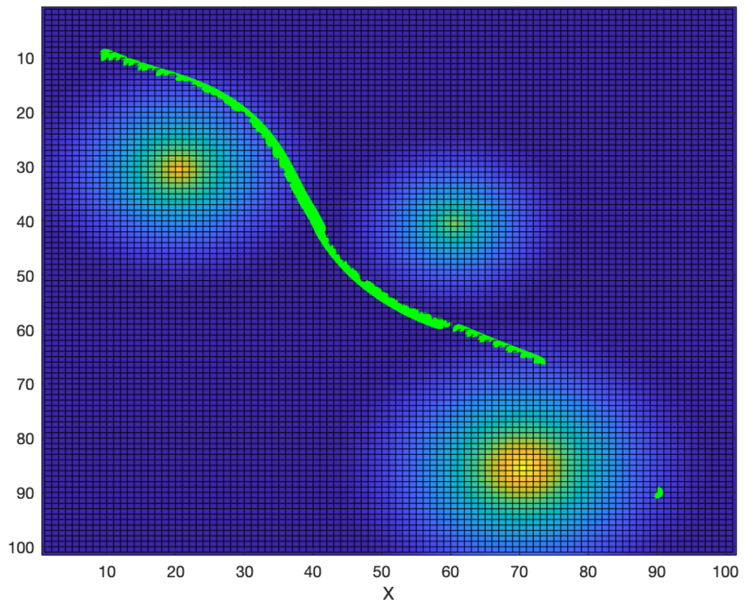

Figure 3: Path after 100 iterations

Here, in a given iteration, we tell the ξ_i point to be dependent on the ξ_{i-1} point value only and we are scaling the difference in their values by a weight factor of 4. But in any given iteration, the ξ_{i-1} is already optimized for the difference with its previous point ξ_{i-2} and hence, in essence, we end up pulling the ξ_i point to be close to the ξ_{i-2} point. When we see this effect for every point, essentially, we are asking the points to come closer to the start point and not account for whether we reach the goal. If we run this for multiple iterations, we compound this effect multiple times and as we can see as the number of iterations increases, the trajectory starts to compress to the start value.

(c) The path after 100 iterations is shown in Figure 6. To run the code for 100 iterations, `trajectory_optimization(3,100)`. The path after 5000 iterations is shown in Figure 7. To run the code for 5000 iterations, `trajectory_optimization(3,5000)`.

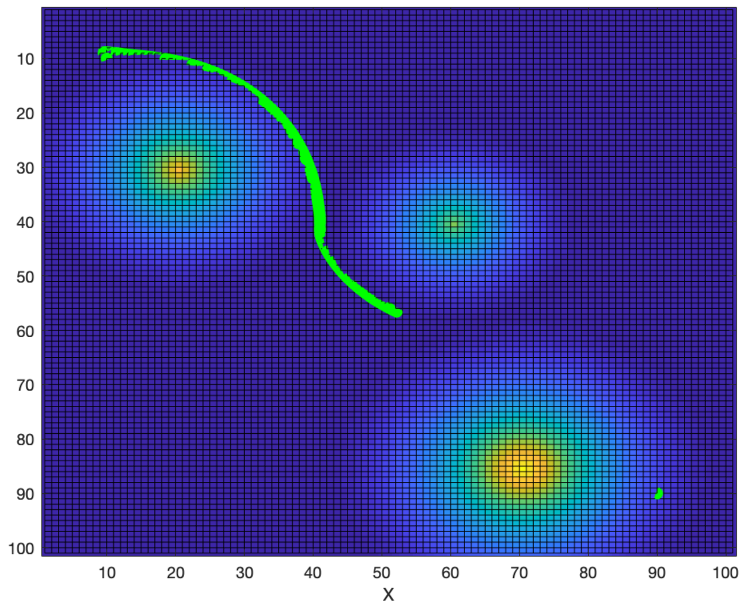

Figure 4: Path after 200 iterations

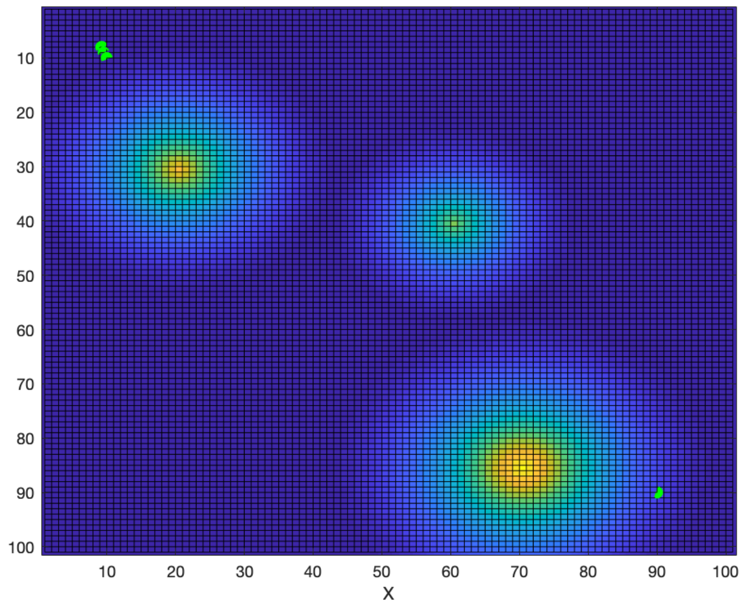

Figure 5: Path after 500 iterations

Here, in a given iteration, we tell the ξ_i point to be dependent not only on the ξ_{i-1} point value but also the ξ_{i+1}, thus asking a given point to be connected to both its predecessor and its successor. When we take the gradient, we are saying that the both the difference of $\xi_i - \xi_{i-1}$ and $\xi_{i+1} - \xi_i$ matter and thus when we optimize at a given iteration, we are using an optimized value of ξ_{i-1} but an un-optimized value of ξ_{i+1}, thus ensuring that the gradients move in favor of reaching the goal while utilizing the connected behavior from the previous question. Hence, this avoids the problem of faced in part (b) and we can successfully reach the goal while following an optimized path around high-obstacle values.

(d) The initial trajectory determines the actual path we achieve after optimization. For example, I took an initial trajectory that first goes only along Y and after reaching the `GY` value, it traverses only along X until it reaches

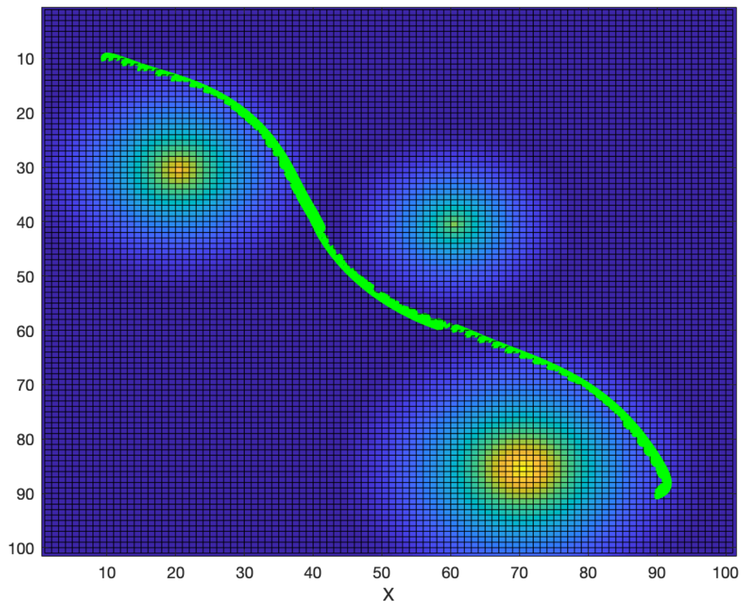

Figure 6: Path after 100 iterations

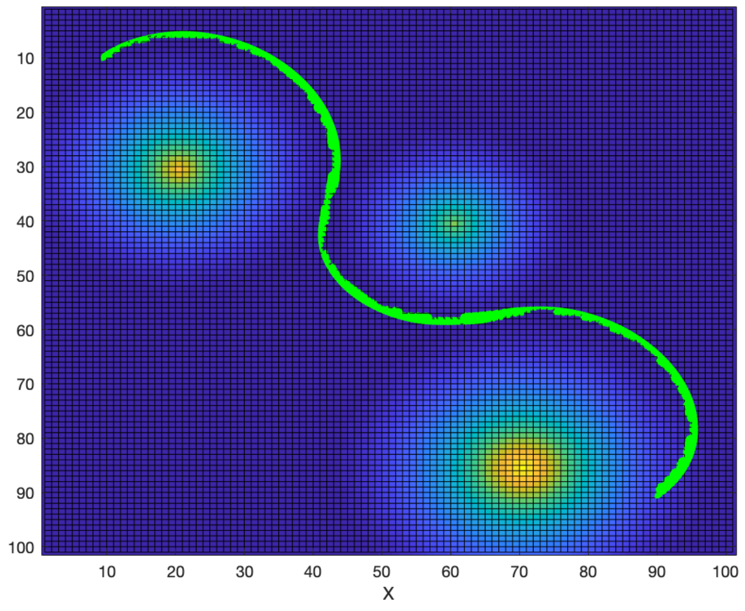

Figure 7: Path after 5000 iterations

the `GX` value, basically an L shaped curve. And after iterating the 3rd algorithm for 100 iterations, we can see the optimized path changes as shown in Figure 8.

This is because at every point of the algorithm we are optimizing the trajectory that we start with to move around the obstacles along the given path and based on the initial trajectory we will get a different path. Hence, the final paths for two different initial trajectories need not be the same.

(e) One simple strategy would be to try and change the weights that we are using. It is possible that because of small weight updates, even with a large number of iterations, we are not able to get a good path - instead is we use appropriate weights for scaling the gradients, we may be able to achieve a better final solution.

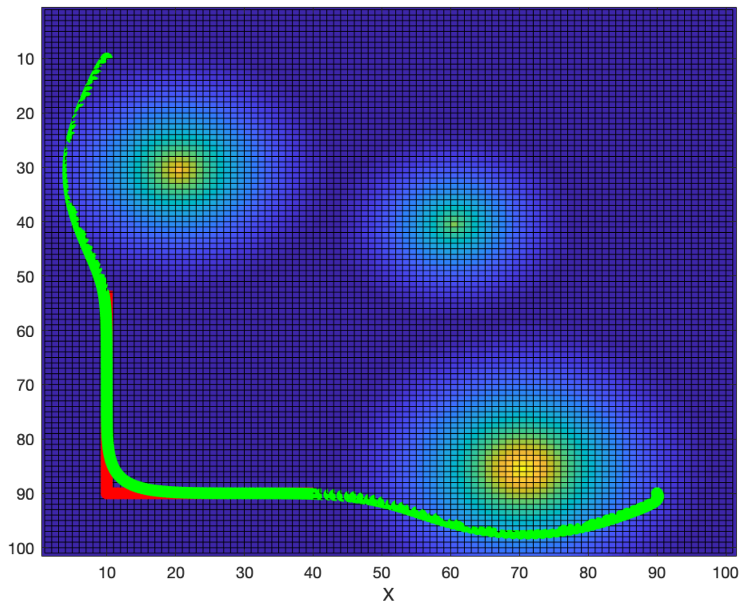

Figure 8: Path after 100 iterations with different initial trajectory

Another strategy would be to try and determine a good initial trajectory. Since we have some knowledge of the cost function around us, we could perform a simple line search to see if a given path is good in some vicinity around the current position and if not, backtrack and try another direction - this can help set a good initial trajectory and then we can use this to obtain a better final trajectory.