Final Exam Problem 1

Yuancheng Xu

Dec 17, 2020

Code link:
`https://github.com/Yuancheng-Xu/AMSC-808N/tree/master/Final%20Exam/Problem1/Code`

Q1: Analysis of the objective function

Denote $f_j(a,b) = [\text{ReLU}(ax_j - b) - g(x_j)]^2$ and $y_j = g(x_j)$. Therefore

$$\nabla f_j = 2(ax_j - b - y_j)\mathbb{1}(ax_j - b > 0)\begin{bmatrix} x_j \\ -1 \end{bmatrix} \tag{1}$$

The stationary points of f corresponds to $\sum_{j=0}^{5} \nabla f_j = 0$. In this problem, $0 = x_0 < x_1 < ... < x_5$ and $0 = y_0 < y_1 < ... < y_5 = 1$. x_j is called activate if $ax_j - b > 0$, and non-active otherwise.

Also, the Hessian $Hf = \frac{1}{12}\sum_{j=0}^{5} Hf_j$, where Hf_j is given by

$$Hf_j = 2\mathbb{1}(ax_j - b > 0)\begin{bmatrix} x_j^2 & -x_j \\ -x_j & 1 \end{bmatrix} \tag{2}$$

In this question, I would like to know the set of stationary points of f. First, I decompose the parameter space into several components, each of which corresponds to 0 to 6 active points. This is visualize in Figure 1. The stationary points are obtained as follows.

- If (a,b) satisfies that none of x_j is active, it is obviously stationary and therefore the flat region (where all points are non-active) is stationary (f is constant in this region). Analytically, this flat region is given by $\{b > 0, b > x_5 a\}$ with $x_5 = \frac{\pi}{2}$.
- If only one of x_j is active (it can only be x_5 or x_0; note that since $x_0 = 0$ and $y_0 = 0$, b must be zero in order for the gradient to be zero, which corresponds to the boundary of the flat region and it is not interesting), let $\nabla f_j = 0$ (with $j = 5$) in Equation 1 and obtain $b = ax_5 - y_5$. Note that we need the constraint that $a > 0, b > ax_4$ so that other points are not active. Analytically I obtain that the point p in Figure 1 is $(\pi, \frac{\pi^2}{2} - 1)$ and therefore the corresponding line of stationary points is $\{b = ax_5 - y_5, a > \pi\}$.

- If n ($n \geq 2$) of x_j are active, the stationary point is either unique or does not exist. To see this, let $\nabla f = 0$ and obtain the following equation (least squares in one-dimension)

$$\begin{bmatrix} \sum x_j^2 & -\sum x_j \\ -\sum x_j & n \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum x_j y_j \\ -\sum y_j \end{bmatrix} \tag{3}$$

  where the sum is over the n activated j. Note that this linear system has an unique solution when $n \geq 2$, with $a = \frac{n\sum x_j y_j - (\sum x_j)(\sum y_j)}{n\sum x_j^2 - (\sum x_j)^2}$ and $b = a\frac{\sum x_j}{n} - \frac{\sum y_j}{n}$. However, the solution must be inside the region where those n activated points are indeed activated, in order to be a stationary point of f. For each component in Figure 1, I solve the corresponding Equation 3 and check whether it is in that component. It turns out that there is only one isolated stationary point x^*, which is in the region where x_2, x_3, x_4 and x_5 are active. I can analytically obtain x^* by substituting $\{x_j, y_j\}_{j=2}^5$ and $n = 4$ into the above formula for a, b. Since the exact formula is tedious, I obtain it numerically and $x^* = (a^*, b^*) = (0.8613, 0.3735)$. The Hessian at x^* is positive definite and by comparing the f value with other stationary points, I conclude that it is the global minimizer with the global minimum $f(x^*) = 3.6e - 4$.

Q2: Gradient descent with constant stepsize

First let us visualize the vector field of $-\nabla f$ in Figure 2.

Minimal stepsize α^* that the iterates end up in the flat region As we can see from Figure 2, if the iterates were to enter the flat region, they would enter through the boundary $\{b > 0, b = x_5 a\}$. On the outer part of the boundary, we have $-\nabla f = (\pi, -2)$, pointing outwards of the flat region. Therefore, in order to enter the flat region, the iterates should hit the boundary **at the first step**, which corresponds to the minimal stepsize α^*. If the actual step size is $\alpha < \alpha^*$, it will bounce away from the boundary and never enter the flat region. Therefore, I obtain α^* by letting the first step of the iterates (with direction $-\nabla f(1,0)$) hit $\{b > 0, b = x_5 a\}$ (that is, the point$(a, b) = (1, 0) - \alpha^* \nabla f(1, 0)$ satisfies $b = x_5 a$). Numerically, I obtain $\alpha^* = 1.51$, which corresponds to slightly overshooting the boundary at the first step).

$\alpha < \alpha^*$ does not necessarily results in convergence In order to see this, I take $\alpha = 0.99\alpha^*$ and the iterates are shown in Figure 3, where they bounce between the two sides of the global minimizer without ever convergence to it. As a side-note, in MatLab I can actually see the iterates step-by-step by plotting each iterate in every iteration and setting breakpoints. To explain why $\alpha = 0.99\alpha^*$ does not lead to convergence, note that in theory, the iterates need to enter a region where f is (strongly) convex in order to converge to x^*.

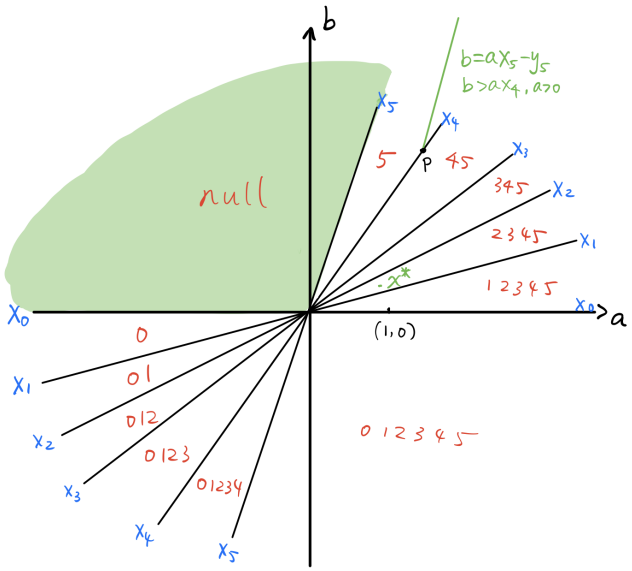
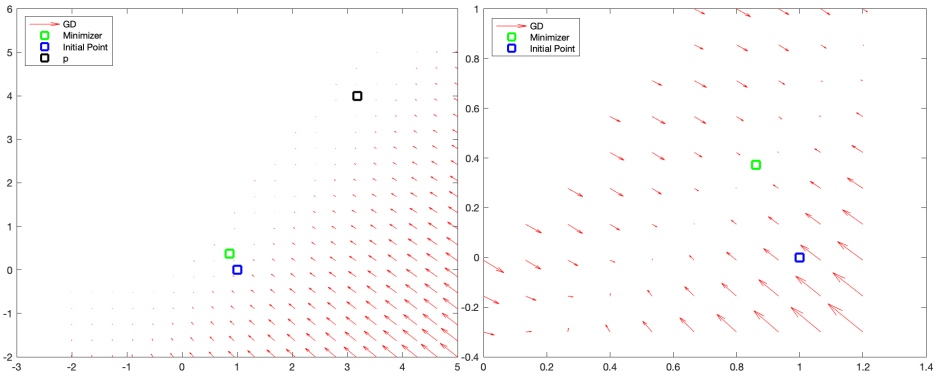


Figure 1: Decomposition of the parameter space. In blue, the line with x_j means the line $b = x_j a$. In red, the number indicates which points are activated, meaning that $ax_j - b > 0$. For example, '2345' means x_2, x_3, x_4, x_5 are activated. Finally, stationary points are in green. Note that they consist of a flat region, a line (with $b = ax_5 - y_5$ where $a > \pi$) and the global minimizer x^*.



(a) Global view. (b) Local view

Figure 2: The vector field around the minimizer and the initial point. At each point, the slope corresponds to the minus gradient of f.

I find numerically that the smallest eigenvalue of the Hessian Hf at the global minimizer x^* is 0.03, which is small. This indicates that it is not so easy to enter that basin (which is indeed the case as we can see from Figure 3) and explains why $\alpha = 0.99\alpha^*$ doesn't lead to convergence.

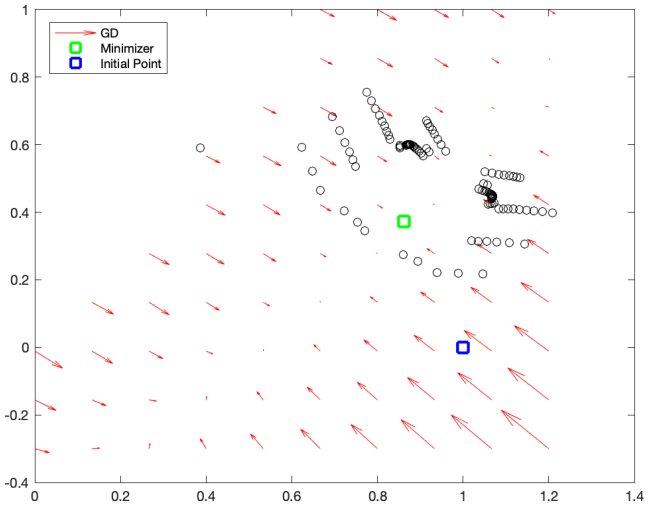
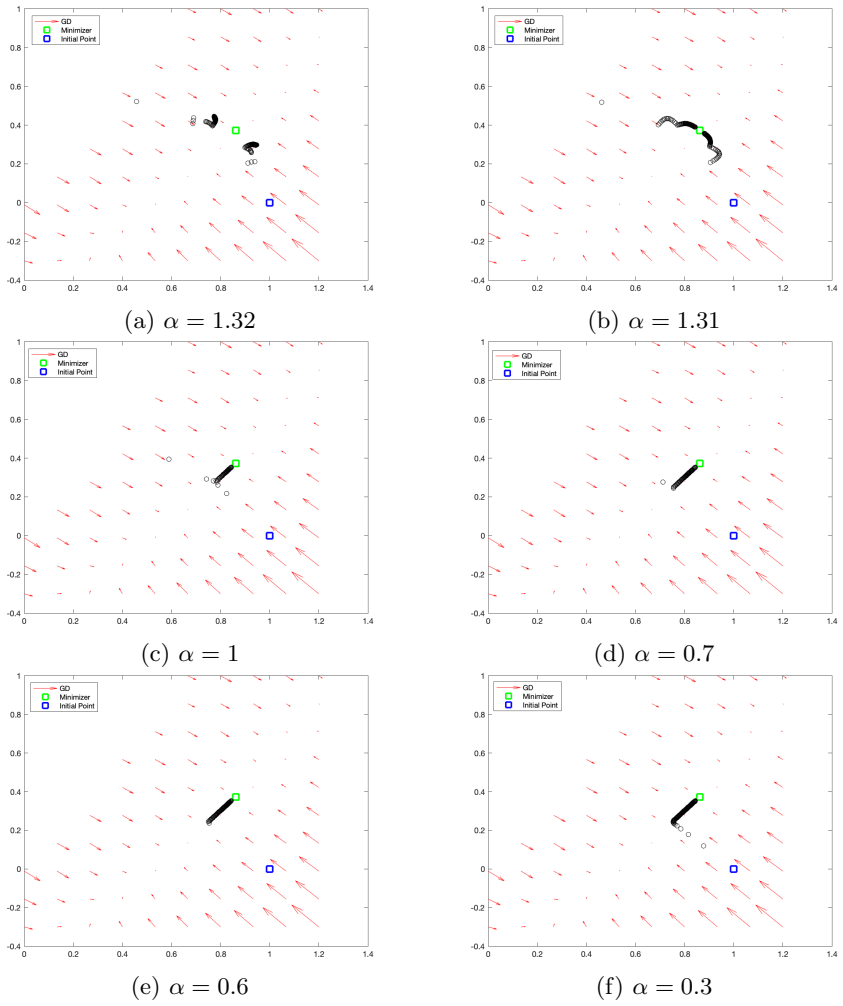


Figure 3: Trajectories when $\alpha = 0.99\alpha^*$. The first step misses the boundary of the flat region and the iterates bounce between the two sides of the global minimizer without ever convergence to it.

Finding the largest stepsize α for convergence In order to find the stepsizes that result in convergence, I plot the trajectories using various α in Figure 4. Observe that $\alpha_c = 1.31$ is the critical value for convergence (largest stepsize for convergence). When $0.7 < \alpha < 1.31$, the iterates will first hit the other side of the minimizer but eventually they converge. When $\alpha < 0.7$, the iterates never hit the other side and converge to the minimizer directly.

But what is the best stepsize? Since larger stepsize may lead to the oscillating behavior in this problem, and thus waste a lot of effort towards convergence, it is not necessary that larger stepsize will lead to faster convergence. Therefore, here I compare, for each α, the number of iterations needed for convergence (up to tolerance $||\nabla f|| < 1e-6$). The result is shown in Figure 5. Observe how the number of iterations first drop gently (due to the increase in stepsize) and then increase abruptly (due to the waste of effort in too much oscillating). The best stepsize is 1.2652. To understand the increase in the number of iterations, let us visualize the trajectories with $\alpha = 1.2652$ and $\alpha = 1.2988$ in Figure 6. Although oscillating behavior appears in both cases, notice that when $\alpha = 1.2652$, two oscillating curves converge in the same direction so there is not a lot waste in oscillating (and since $\alpha = 1.2652$ is large, convergence is



(a) $\alpha = 1.32$ (b) $\alpha = 1.31$

(c) $\alpha = 1$ (d) $\alpha = 0.7$

(e) $\alpha = 0.6$ (f) $\alpha = 0.3$

Figure 4: Various convergence modes. Except for the first case ($\alpha = 1.32$) which doesn't converge, several convergence modes (with or without oscillating) are shown.

fast). However, when $\alpha = 1.2988$, the two oscillating curves are converging in the opposite directions (so they are always oscillating to each other, wasteful!), which explains the sudden drop in the convergence speed.

To sum up, the best $\alpha = 1.2652$, such that it is relatively large, but not too large (it does not waste too much effort on oscillating).

Remark This final choice of α is obtained through careful examination by hand. In the future it maybe desirable to automate the process by proposing better algorithm. For example, exploiting the "phase transition" in Figure 6 might be helpful.

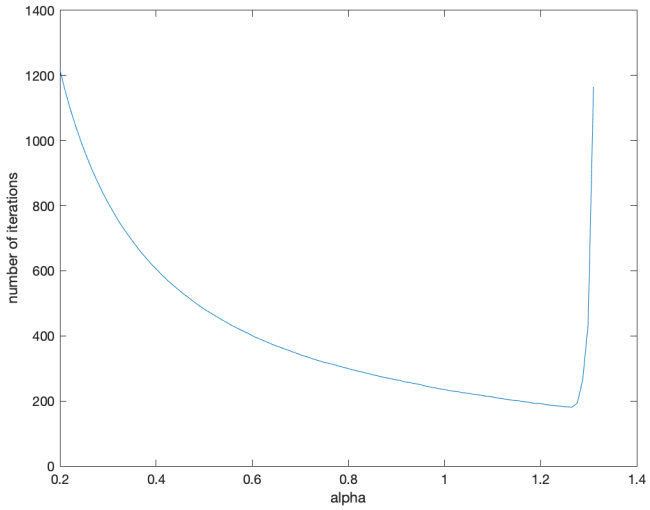
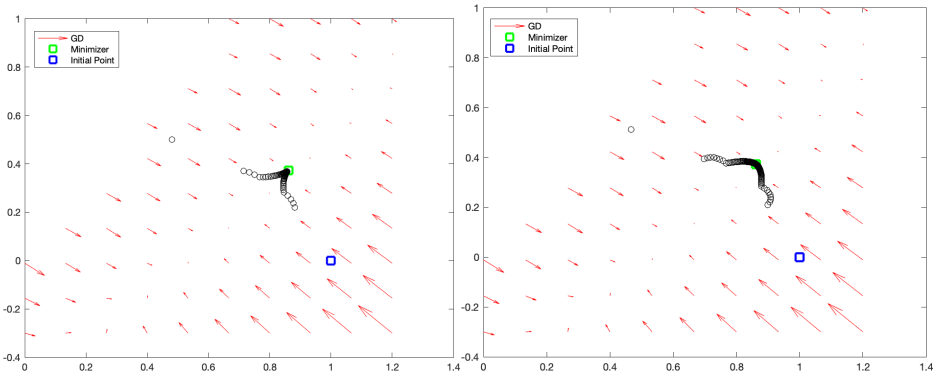


Figure 5: The number of iterations needed to achieve $||\nabla f|| < 1e-6$ for various α. Observe how the number of iterations first drop gently and then increase abruptly. $\alpha = 1.2652$ needs 181 iterations, which is fastest.

Q3: Stochastic Gradient Descent

In this question, I use SGD with batch size 1 and I would like to find a strategy for stepsize reduction such that SGD will converge to the global minimizer. To this end, I decide to use the stepsize scheme where the stepsize at iteration k is given by $\alpha_k = \alpha_0 \frac{M}{M+k}$, where α_0 and M are tuning parameters. When α_0 is fixed, larger M means slower decay of the stepsize. Recall that in class we have the convergence theory of SGD that, informally, if α_0 is small enough (depending on the objective function f), an appropriate stepsize reduction scheme is used (an appropriate M in our case) and f satisfies some regularity conditions (such as convexity; not necessarily hold here), then SGD converges in expectation. In the following I will empirically look into different strategies. A good strategy



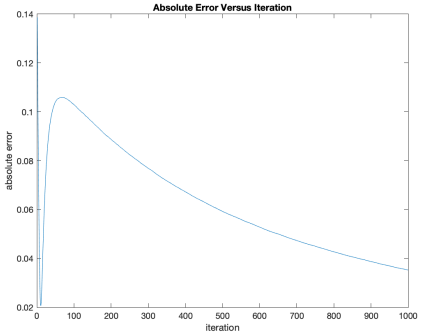
(a) $\alpha = 1.2652$ (b) $\alpha = 1.2988$

Figure 6: The trajectories for the critical α that results in an abrupt increase in the number of iterations needed for convergence. Observe that there is a **phase transition** in the geometry of the trajectories. Although oscillating behavior appears in both cases, notice that when $\alpha = 1.2652$, two oscillating curves converge in the same direction so there is not a lot waste in oscillating (and since $\alpha = 1.2652$ is large, convergence is fast). However, when $\alpha = 1.2988$, the two oscillating curves are converging in the opposite directions (so they are always oscillating to each other, wasteful!), which explains the sudden drop in the convergence speed.

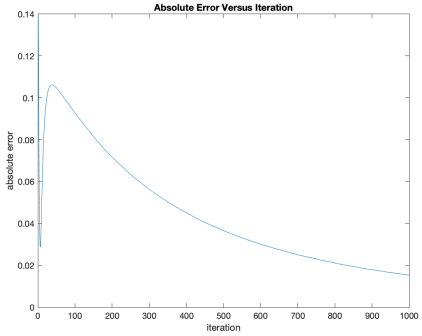
should satisfy (a) convergence in expectation (b) converge with high probability (most trajectories converge) (c) fast convergence (d) small error.

I found that $M = 1000$ performs well, so I tune α_0, the initial stepsize. The average absolute error (with respect to x^*) versus iteration is shown in Figure 7, which can be used to verify whether SGD converge in expectation. It turns out that SGD converges (in expectation) when $\alpha \leq 3$. Interestingly, (deterministic) gradient descent will diverge with $\alpha_0 = 3$ as shown previously, even with the same stepsize reduction strategy (since it is messed up at the first step). Another observation is that, when $\alpha_0 \leq 2$, there is a "double descent" phenomenon where the error first drop at the very beginning, then increase and finally decrease again. When $\alpha_0 < 1$, the first decrease will arrive at an error even smaller than the final error (the error at a reasonably large iteration). Lastly, when $\alpha_0 > 4.65$, SGD fails to converge. This is further verified in Figure 8, where there is a small fraction of trajectories that does not converge when $\alpha_0 = 4$, and a large fraction when $\alpha_0 = 4.65$.

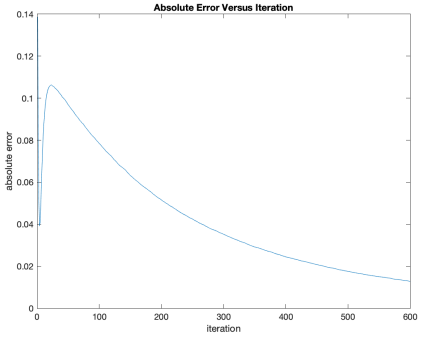
Which strategy should be used? I find that $\alpha_0 = 2$ should be used (that is, $\alpha_k = 2\frac{1000}{1000+k}$). This is obtained by carefully observe Figure 7 and choose the one with fastest convergence and lowest error. Also, in Figure 8 we can see that with almost probability 1, the iterates converge to x^*. This choice is further justified in Figure 9, where it is observed that the iterates tend to be stuck at



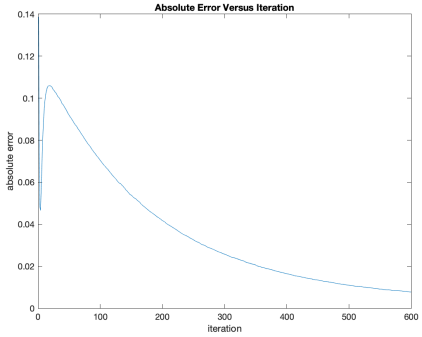
(a) $\alpha_0 = 0.3$



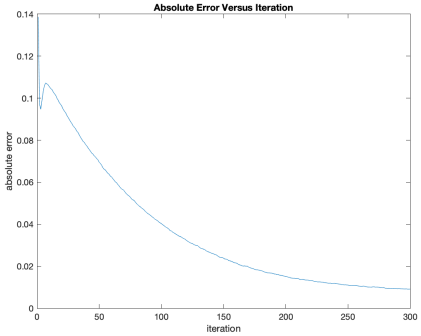
(b) $\alpha_0 = 0.6$



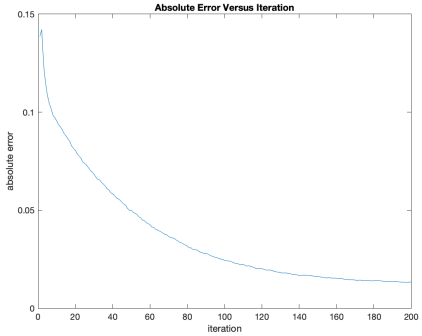
(c) $\alpha_0 = 0.8$



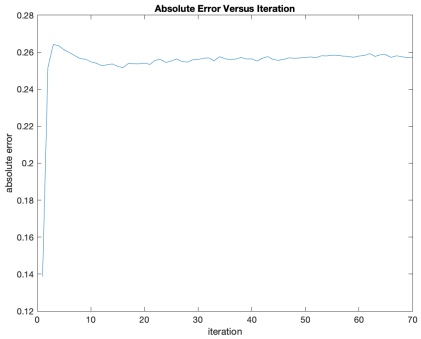
(d) $\alpha_0 = 1$



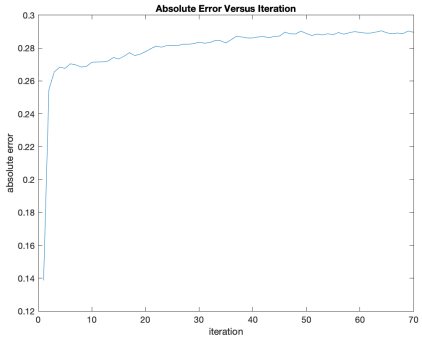
(e) $\alpha_0 = 2$



(f) $\alpha_0 = 3$

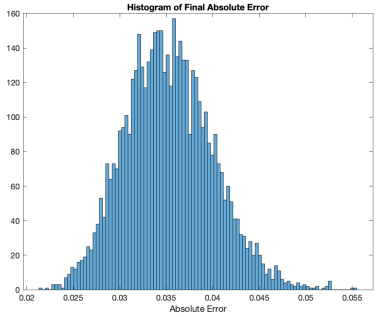


(g) $\alpha_0 = 4.65$

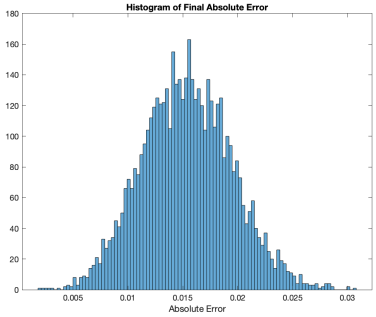


(h) $\alpha_0 = 4.70$

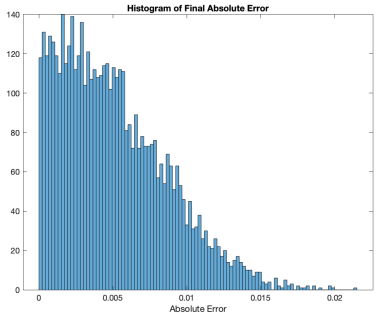
Figure 7: Absolute error (with respect to the global minimizer x^*) versus iterations, when $M = 1000$. "Double descent" occurs when $\alpha \leq 2$. The plot is averaged over 1000 trajectories.



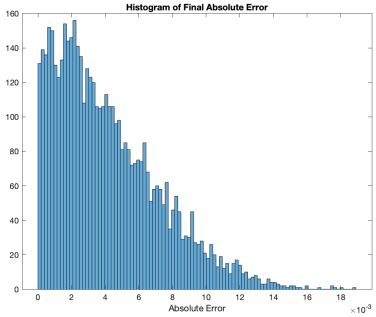
(a) $\alpha_0 = 0.3$



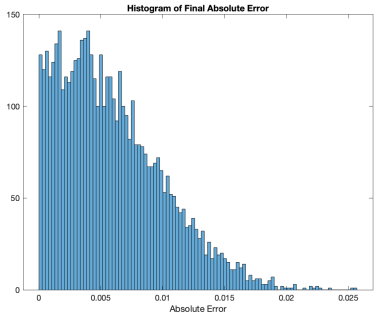
(b) $\alpha_0 = 0.6$



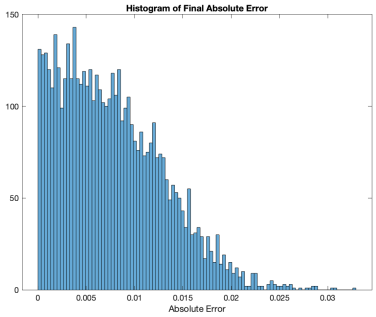
(c) $\alpha_0 = 0.8$



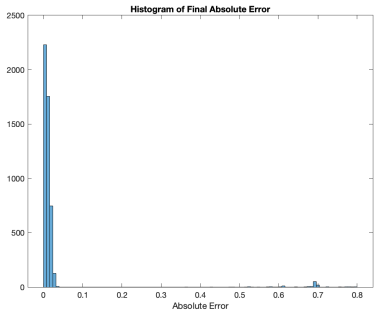
(d) $\alpha_0 = 1$



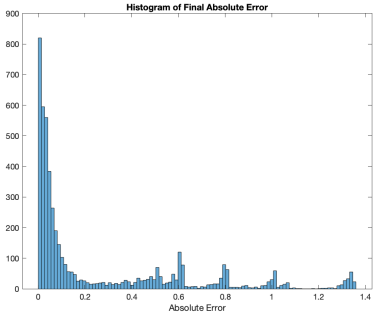
(e) $\alpha_0 = 2$



(f) $\alpha_0 = 3$

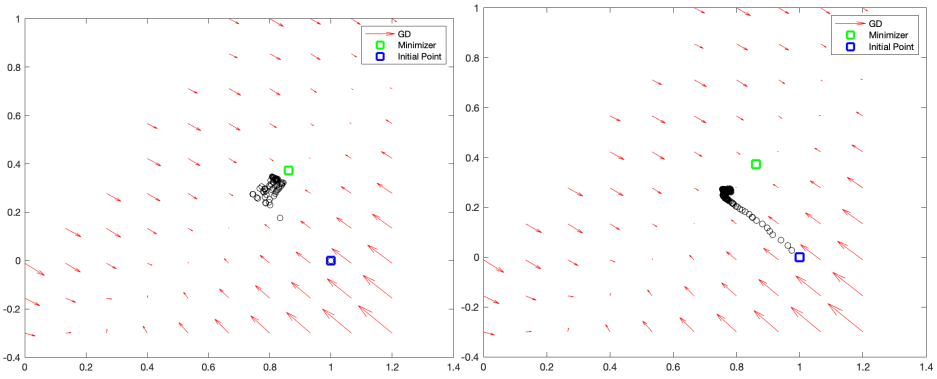


(g) $\alpha_0 = 4$



(h) $\alpha_0 = 4.65$

Figure 8: Histogram of absolute error (with respect to the global minimizer x^*), when $M = 1000$. The plot is averaged over 1000 trajectories. When $\alpha_0 = 4.65$, there is relatively high probability that SGD will not converge to x^*.



(a) $\alpha_0 = 2$ (b) $\alpha_0 = 0.3$

Figure 9: One trajectory for SGD. When $\alpha_0 = 0.3$, the iterates are stuck in the region where the gradient is small and therefore progress extremely slowly. Using larger stepsize such as $\alpha_0 = 2$ speeds up convergence significantly.

a flat region (with small derivatives) and thus larger stepsize helps speed up convergence.

GD works better than SGD in the problem Finally, note that since this is only a two-dimensional problem with only 6 data points, using SGD doesn't really contribute to computational efficiency. Also, GD converges faster and much closer to x^* than SGD: when $\alpha_0 = 1$ with no stepsize reduction, GD achieves absolute error of 0.0027 with 100 iterations, while SGD (with stepsize reduction) gets an error of 0.01 with 600 iterations.