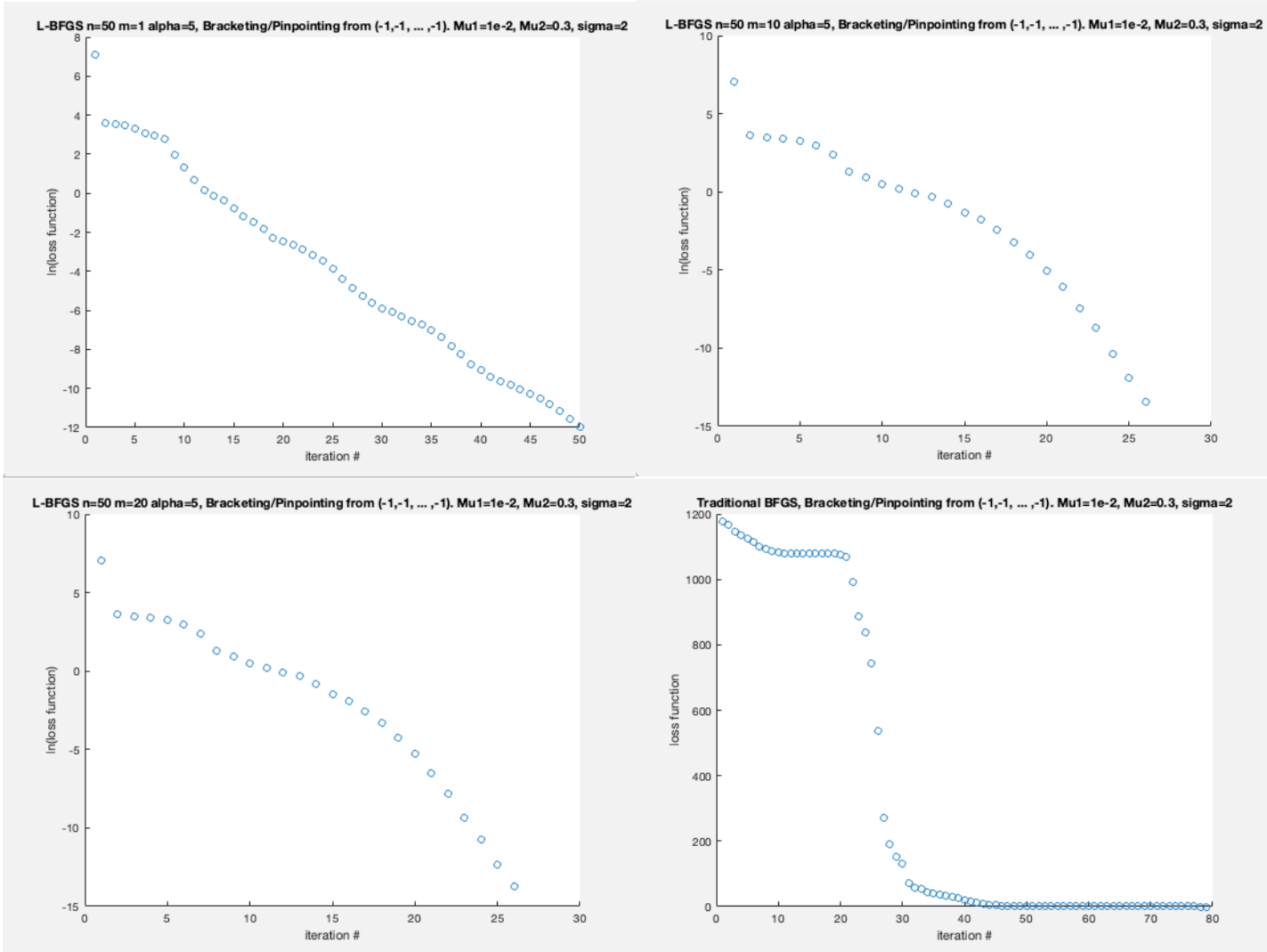
Jacob Snyder

ACM213 Set 2

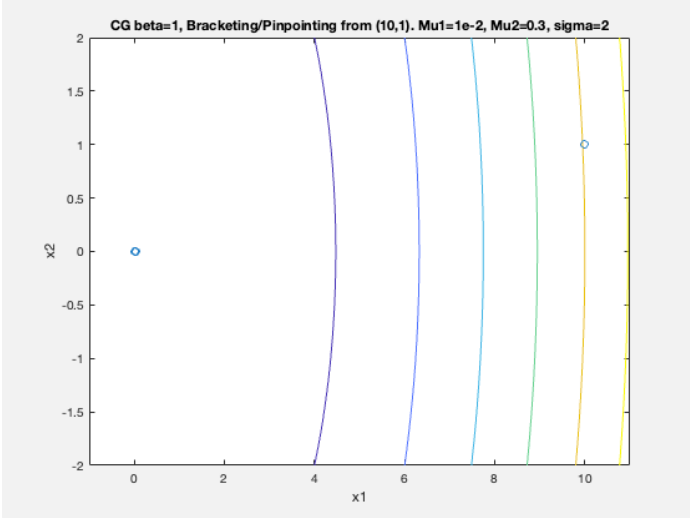
1.a)



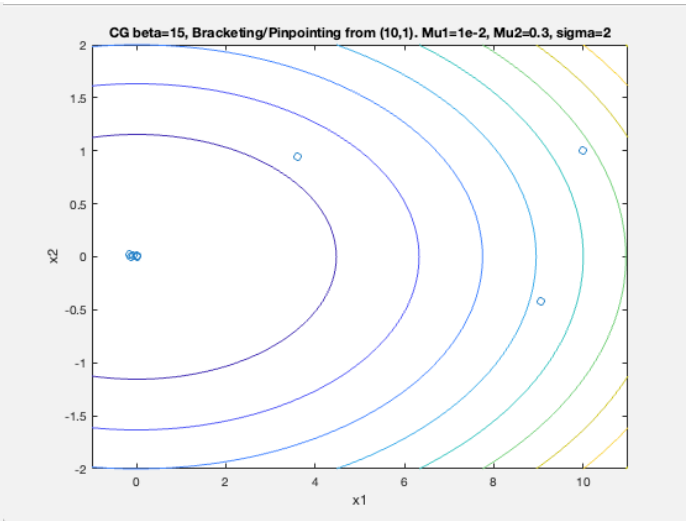
We can see the impact of the memory parameter m: when m=1 (effectively no memory other than the most recent iteration), we see a very constant reduction of the loss function (in log terms). But when we add a non-zero memory parameter we can see that the loss function decreases faster and faster as we add more data to the memory to be used. The difference between m=10 and m=20 is hard to see but the difference between m=10 and m=1 is clear. The difference between the memory algorithms and the traditional BFGS is also clear.

1.b) Spent about 3 hours on this problem.

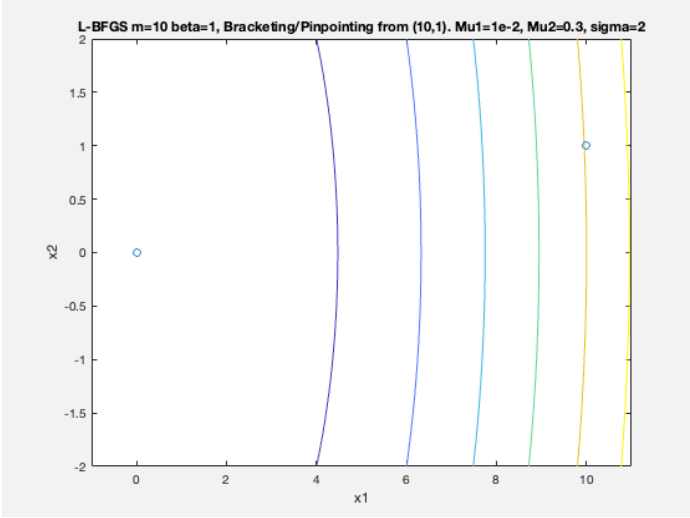
2.a)



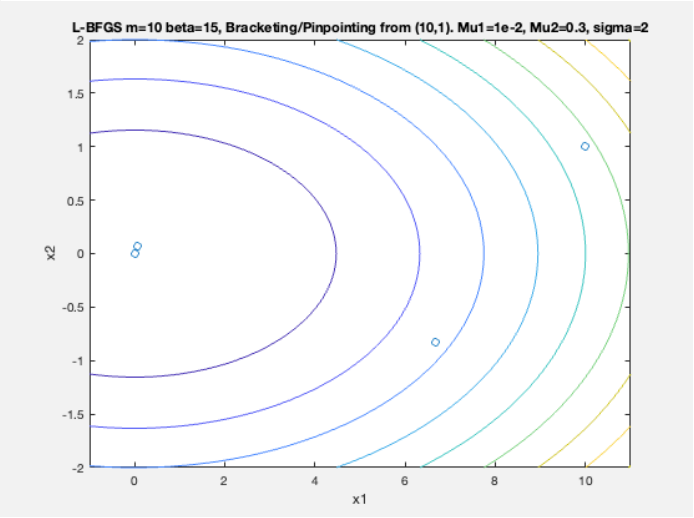
3 iterations



10 iterations



2 iterations



4 iterations

2.b) Comparing to the plots from the previous homework (steepest descent, newton, BFGS), we can see that L-BFGS outperforms BFGS for both beta=1 and beta=15 (this makes sense since L-BFGS should be strictly better). It appears that CG requires a few more iterations to run than BFGS, especially for higher beta. Newton's method outperforms both L-BFGS and CG on this function. Steepest descent is far outperformed by all other algorithms on this function.

2.c) 2 hours

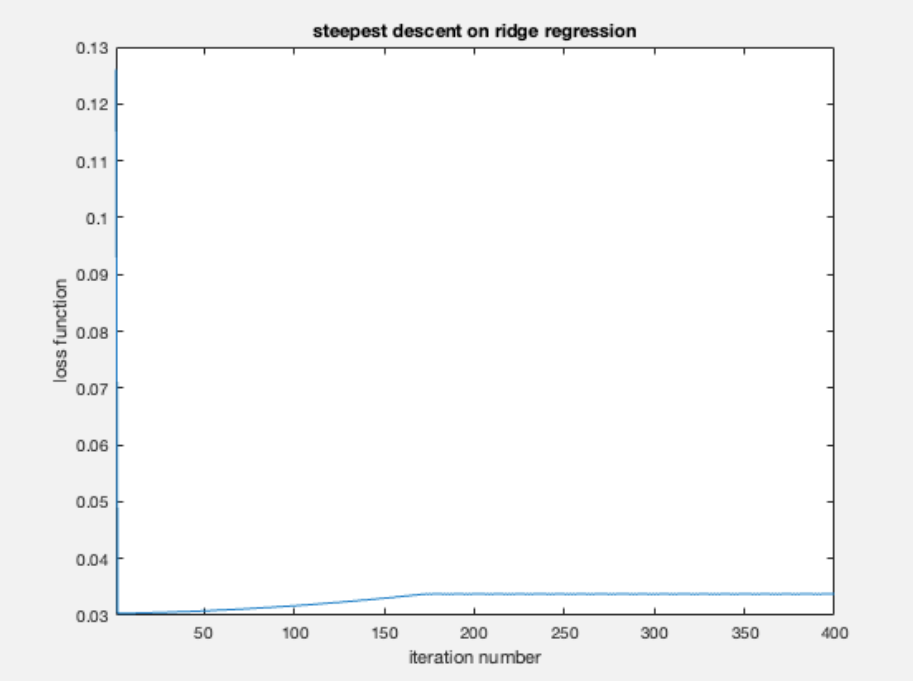
3.a)

$$f_i(\theta) = \frac{1}{2}(h_\theta(x_i) - y_i)^2 + \frac{\lambda}{2m}\theta^T\theta$$

$$\rightarrow f_i(\theta) = \left(\theta^T \begin{bmatrix} x_i \\ 1 \end{bmatrix} - y_i\right)^2 + \frac{\lambda}{2m}\theta^T\theta$$

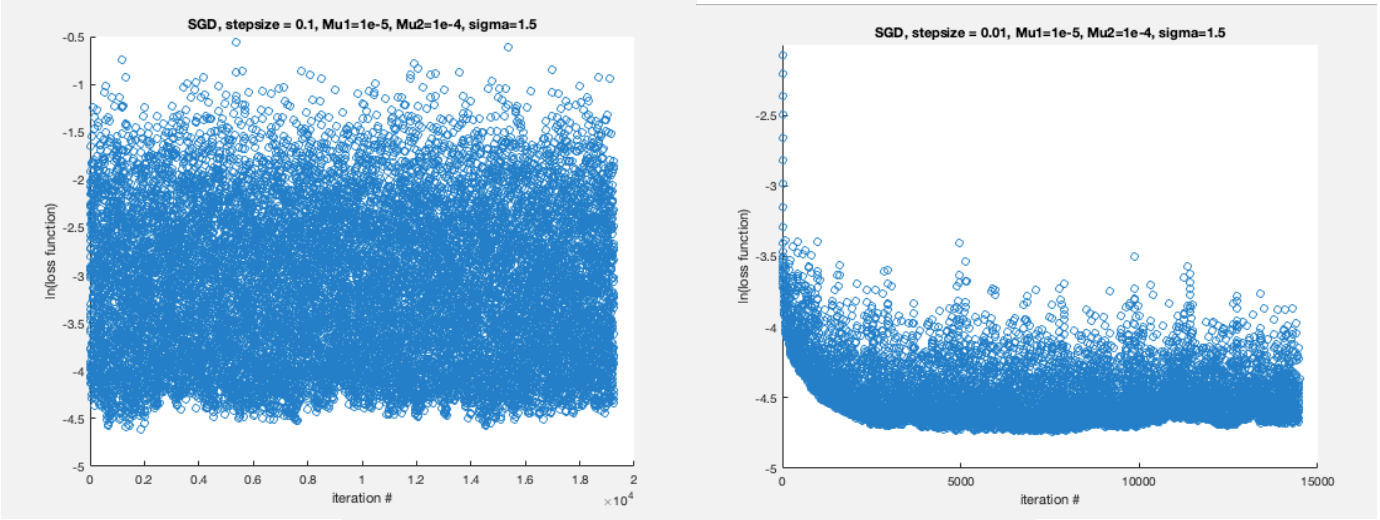
$$\rightarrow \nabla_\theta f_i = 2\begin{bmatrix} x_i & 1 \end{bmatrix}\left(\theta^T \begin{bmatrix} x_i \\ 1 \end{bmatrix} - y_i\right) + \frac{\lambda}{m}\theta^T$$

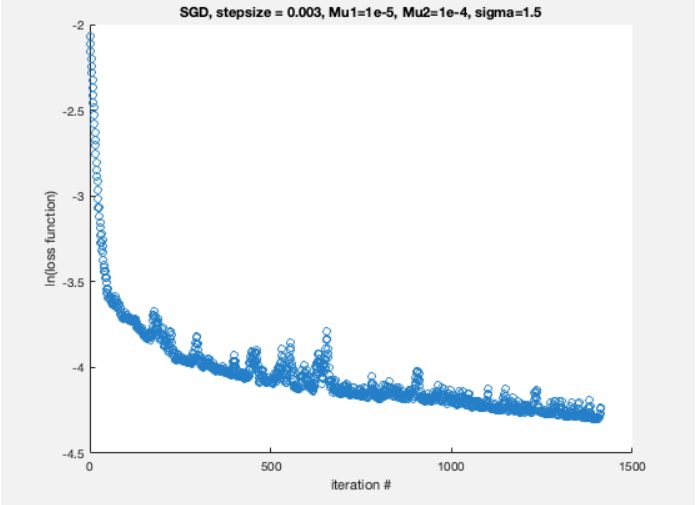
3.c)



It appears that the steepest descent algorithm finds significant improvement within the first few epochs but then never reaches a point where $\|\nabla J_\theta\|_2 \leq 0.01\|\theta\|_2$. The algorithm continues infinitely.

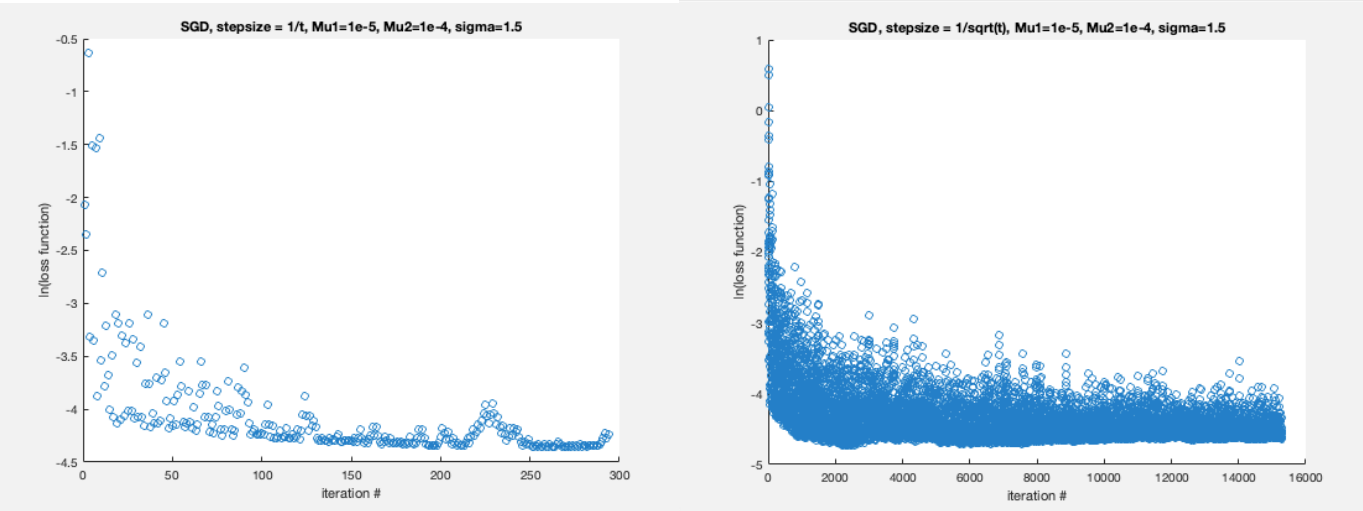
3.d)





Too large of a step size dose not allow the algorithm to gradually approach the minimum; instead, it skips over it resulting in the first plot where even after a large number of epochs the loss function is still responding somewhat randomly. The algorithm does not converge.
The middle step size allows the algorithm to reach a low-error area but there is still some degree of overshoot and the algorithm does not converge.
The lowest step size allows for the algorithm to move slowly and carefully so we see a much more exact descent that eventually terminates.

3.e)



The variable step size of 1/t worked very well; it terminated after a low number of epochs at the minimum. The step size of 1/sqrt(t) appears to be a bit too big as we see some of the overshoot phenomenon and a very large number of epochs.

3.f) 3 hours