CMSC828V Project 1 Report

Yuelin Liu & Konstantinos Pantazis

November 5, 2020

Note: Code for each student can be found at:

Yuelin Liu: https://github.com/liuy0421/828v-project1

Konstantinos Pantazis: https://github.com/kpantazis/AMSC-808N-Project-1

1 Support Vector Machine (SVM) with Soft Margins

1.1 Results on Subset of 58 CA Counties

First, only looking at the subset of data from 58 CA counties. We obtain the dividing plane (Figure 1) by running the inexact Newton's method for $k = 1000$ iterations. Rates of convergence are shown in Figure 2. We see from run for $k = 1000$ iterations that step size decreased exponentially, and the results converged around $k = 100$.

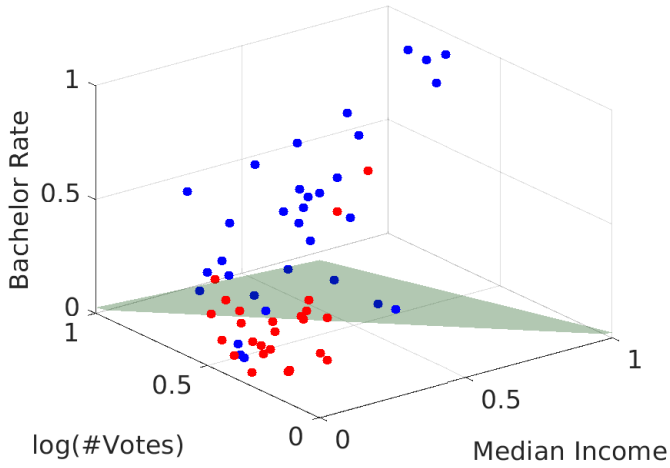

Figure 1: Dividing plane obtained by running the inexact Newton's method for $k = 1000$ iterations on a subset of data from 58 CA counties.

Then, using the dividing plane in Figure 1 as initial guess, we set up soft margin SVM constrained minimization problem after acquiring the initial guess for ξ_i using the Recified Linear Unit (ReLU)

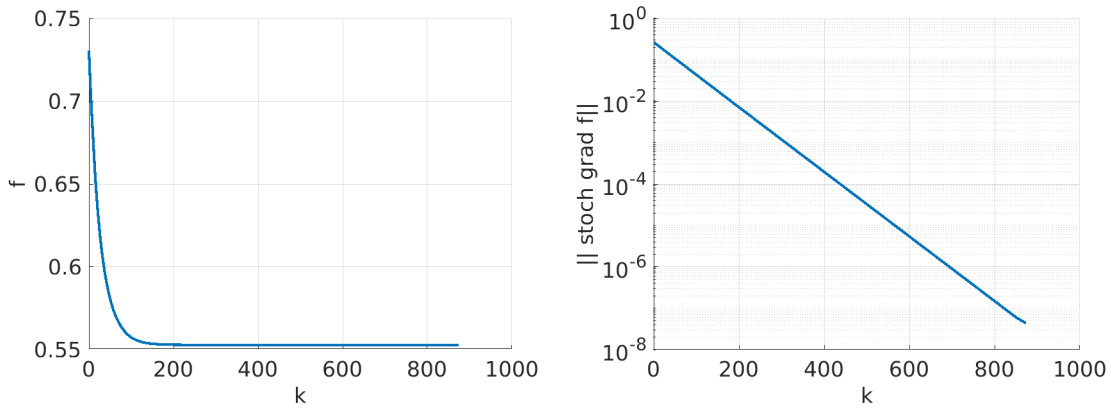

Figure 2: Rates of convergence for the inexact Newton's method on data from 58 CA counties.

loss function. We solve it using the active set method, acquiring a different dividing plane after $k = 1000$ iterations (Figure 3). With the good initial guess provided by the inexact Newton's method, the active set method seems to converge after $k = 92$ iterations (Figure 4).

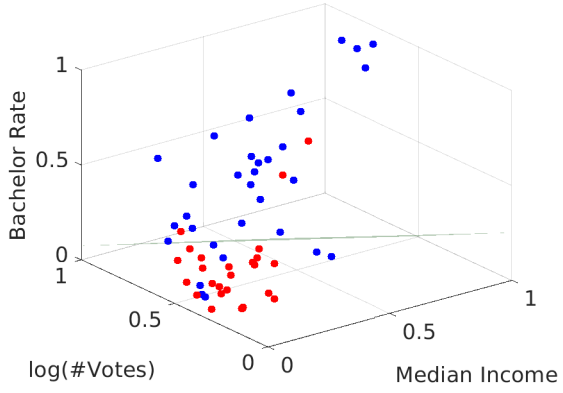

Figure 3: Dividing plane obtained by running the active set method for soft margin SVM constrained minimization problem for $k = 1000$.

Comparing the two different dividing planes from a rotated perspective (Figure 5), we observe that the one obtained by soft margin SVM seems more reasonable. The plane shifted slightly towards the two red points that are wrongly classified in the top right, suggesting that it reduced the "over-fitting" behavior of the model.

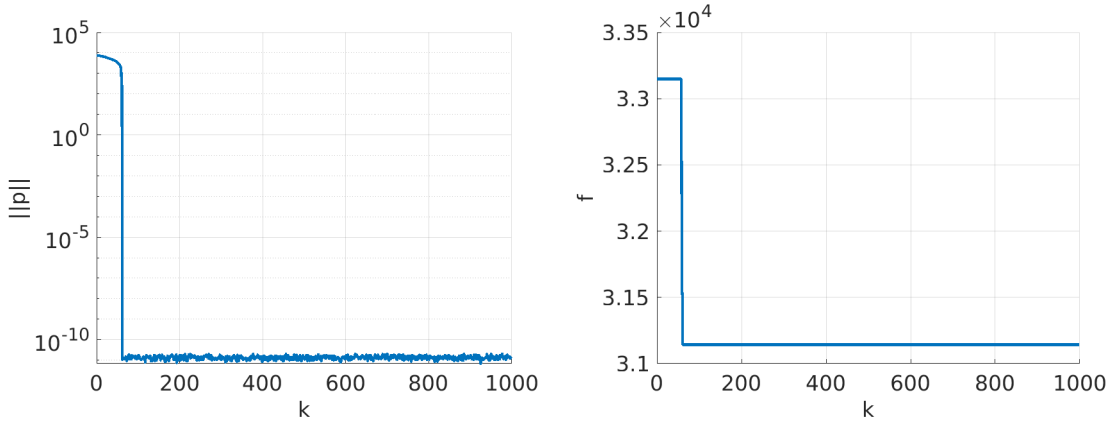

Figure 4: Rates of convergence for the active set method for soft margin SVM constrained minimization problem for $k = 1000$ on data from 58 CA counties.

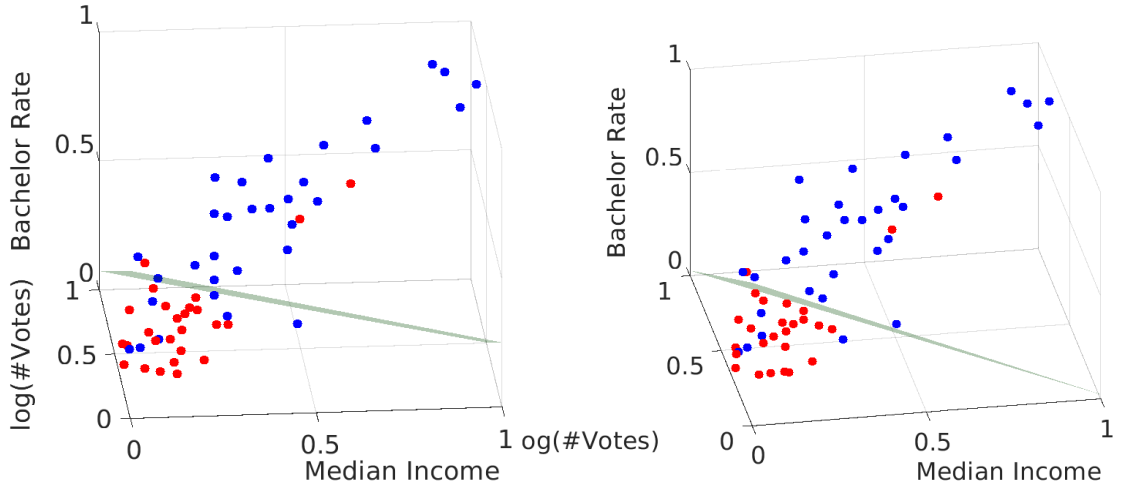

Figure 5: Comparison between the resulting dividing plane using inexact Newton's method (left) and using constrained minimization SVM formulation (right) on subset of data from 58 CA counties.

1.2 Results Incorporating other States

As instructed, we added data points in other states besides CA: WA, NJ, NY, and OR. From Figure 6, we can see that the planes are largely similar. Looking at the rates of convergence, however, we observe that, while the inexact Newton's method still converged relatively quickly around $k = 100$ iterations (Figure 7), the constrained soft margin SVM did not reach convergence until $k > 200$ iterations (Figure 8).

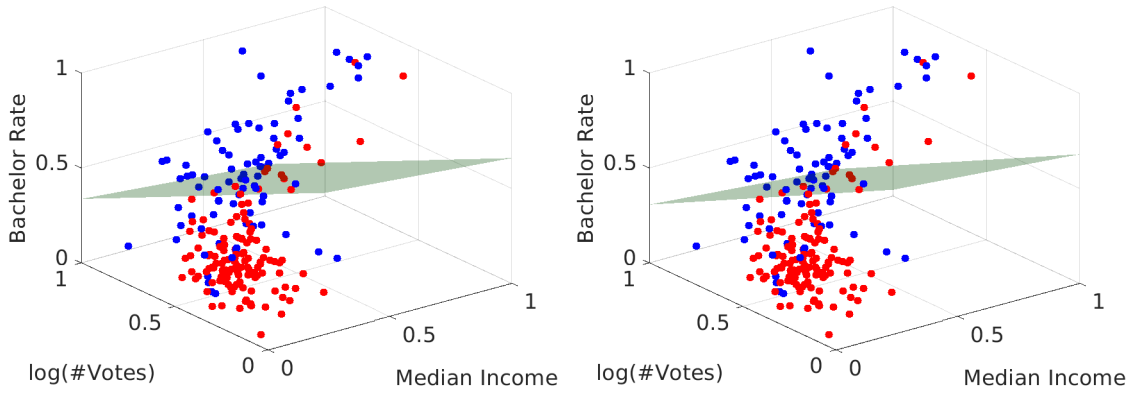

Figure 6: Comparison between the resulting dividing plane using inexact Newton's method (left) and using constrained minimization SVM formulation (right.) on data from CA, WA, NJ, NY, and OR.

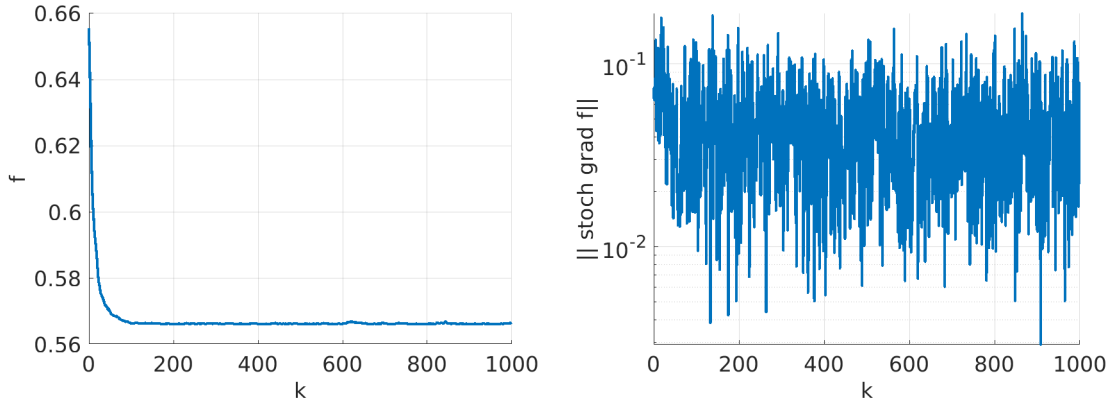

Figure 7: Rates of convergence for the inexact Newton's method on data from CA, WA, NJ, NY, and OR.

The fact that constrained minimization formulation needed an increasing amount of iterations to reach convergence as the number of data points increase motivates us to switch to unconstrained minimization of a reasonable loss function.

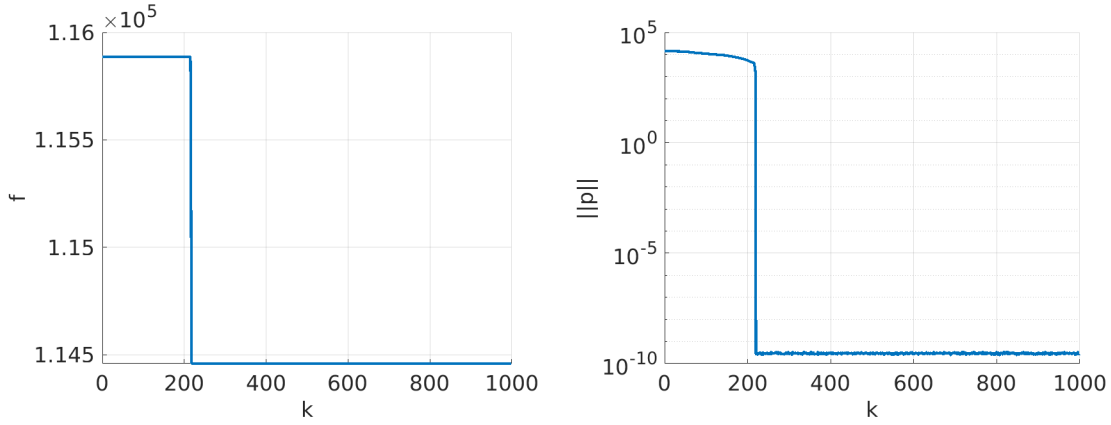

Figure 8: Rates of convergence for the active set method for soft margin SVM constrained minimization problem for $k = 1000$ on data from CA, WA, NJ, NY, and OR.

2 Stochastic Gradient (SG) Method

For the second part of the project, we implemented the stochastic gradient descent method. Unlike the regular gradient descent method, at each step, SG evaluates the gradient for a subsampled batch of data points, and use the average of that to guide the step in that iteration.

First, we set $\lambda = 0.01$. With a random initial guess $w_0 = [0.5; 0.5; 0.5; 0.5]$, we examined the loss function value, gradient function norm, and run time across iterations. Due to the stochasticity of the algorithm, for each set of parameters, we run SG 1000 times, and the reported values per iteration are the average across the 1000 runs. In particular, we examined the effect of different step size decreasing strategy, batch size, and Tikhonov regularization parameter have on the behavior of the algorithm.

2.1 Effect of step size decreasing strategies

With batch size = 100, max number of steps = 1000, we set the step size decreasing strategies to: none ($\alpha_k = 0.001$), slow decay ($\alpha_k = 0.1/k$), and exponential ($\alpha_k = 0.1/1.1^k$). Figure 9 shows the value of the loss function and the norm of the gradient at each iteration, averaged across 1000 different runs.

As we can see, the rate of decrease of the value of the loss function roughly correlates to the rate of decrease of the step size (Figure 9, left column) and roughly correlates to the rate of decrease of the norm of the gradient. The norm of the gradient function has much higher variance, as it is evaluated for a subsample of data points across different runs for each iteration. The average run time for each method is 0.0647, 0.0758, and 0.0728, respectively. It is worth noting that, without having fine tuned the step size decrease strategy for this problem, we fail to converge to a stationary point within 1000 iterations: the result from constant step size shows the function value is still on a continuous decrease at the 1000^{th} iteration, and the slow decay and exponential decrease strategy seemed to reach a low step size too quickly to descend to the minimum.

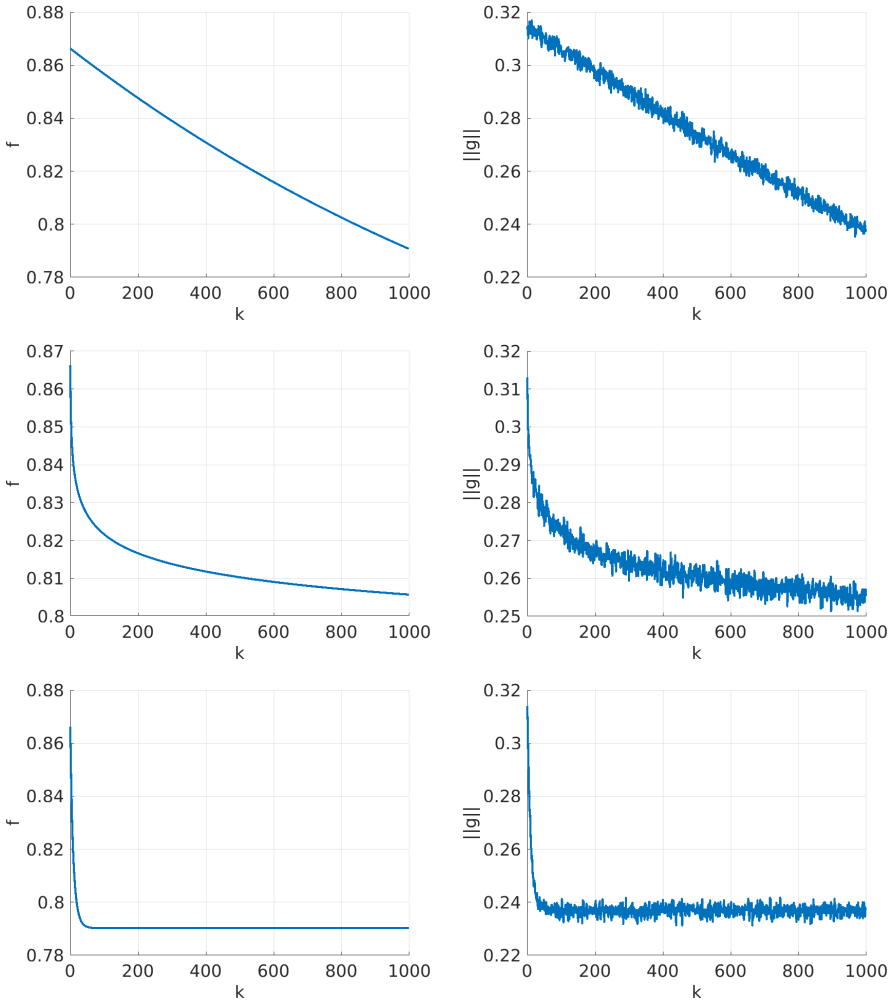

Figure 9: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration, averaged across 1000 different runs for step size decreasing strategies: (1) none ($\alpha_k = 0.001$), (2) slow decay ($\alpha_k = 0.1/k$), and (3) exponential ($\alpha_k = 0.1/1.1^k$). The average run time for each method is 0.0647, 0.0758, and 0.0728, respectively.

2.2 Effect of batch size

Similarly, we examined the effect batch size has on the loss function value, gradient function norm, and run time across iterations. We set the step size decrease strategy to slow decay ($\alpha_k = 0.1/k$), and set batch size to $m = 100, 250, 500$.

As we can see in Figure 10, while the rate of decrease of the loss function and that of the gradient norm are nearly identical for all three batch sizes, the average gradient norm has more variation between iterations for low batch sizes. This makes intuitive sense because the variance is higher when approximating function values using small subsets of data points.

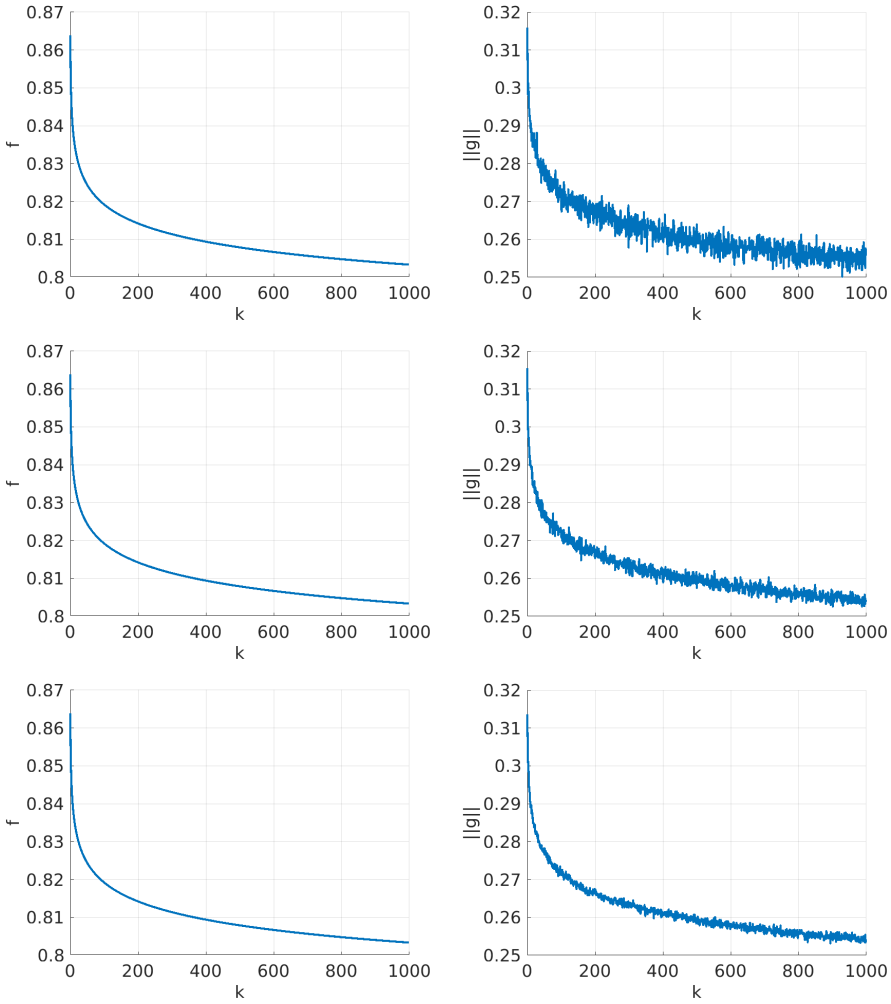

Figure 10: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration, averaged across 1000 different runs, using the slow decay ($\alpha_k = 0.1/k$) step size decrease strategy, varying batch size equal to (1) 100, (2) 250, and (3) 500.

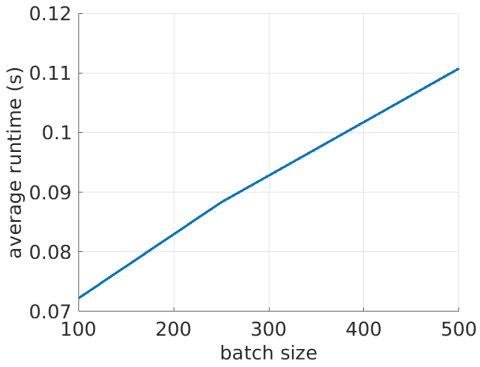

Figure 11: The effect of batch size on run time for stochastic gradient descent.

Furthermore, we examined the effect of batch size on algorithm run time. As in Figure 11, we observe that run time increases linearly with batch size, holding everything else constant. It makes intuitive sense, as batch size dictates the number of gradient evaluations we need to perform in computing the average in each iteration of the algorithm.

2.3 Effect of Tikhonov regularization parameter λ

Finally, we examine the effect the Tikhonov regularization parameter λ has on the loss function value, gradient function norm, and run time across iterations. We set the Tikhonov regularization parameter $\lambda = 0.01, 0.05, 0.1$.

From Figure 12, we observe that smaller values of λ reached lower loss function value at the end of the 1000 iterations, and have a lower net decrease of gradient norm. The run times are 0.0764, 0.0755, and 0.0955, respectively, for the three settings of λ.

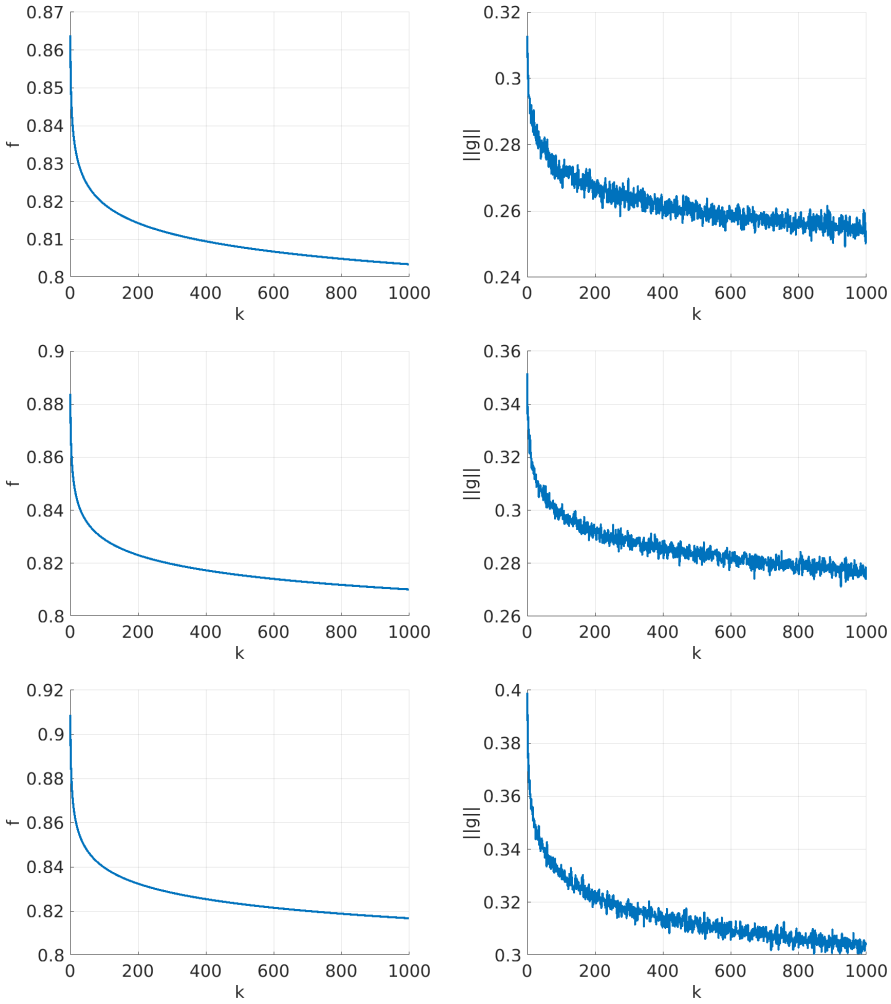

Figure 12: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration, averaged across 1000 different runs, using the slow decay ($\alpha_k = 0.1/k$) step size decrease strategy, batch size of 100, and Tikhonov regularization parameter λ of (1) 0.01, (2) 0.05, and (3) 0.1.

3 Subsampled Inexact Newton's Method

We further experimented with the effect of different batch size on subsampled inexact Newton's method. We recorded the average loss function value and gradient function norm per iteration across 1000 different runs of the algorithm. In our experiment, similar to what we did in Section 2.2, we set the batch size to $m = 100, 250, 500$.

As we can see in Figure 13, while all 3 batch sizes were able to converge to the same stationary point, the smaller batch size $m = 100$ provided the fastest decrease. A similar trend is observed

in the norm of the gradient. This is quite interesting result. Recall in Figure 10, we observed that batch size has little effect on the rate of convergence in stochastic gradient descent. However, subsampled inexact Newton's method seems to be very sensitive to batch size. To further investigate, we also ran experiments with batch size $m = 2, 10$ (Figure 14). With too small batch size, $m = 2$, the method is unable to well approximate the Hessian times vector, and therefore cannot effectively decrease the loss function value. However, once the batch size is large enough, $m = 10$ for this example, it is able to rapidly decrease the loss function value and the gradient norm (Figure 14).

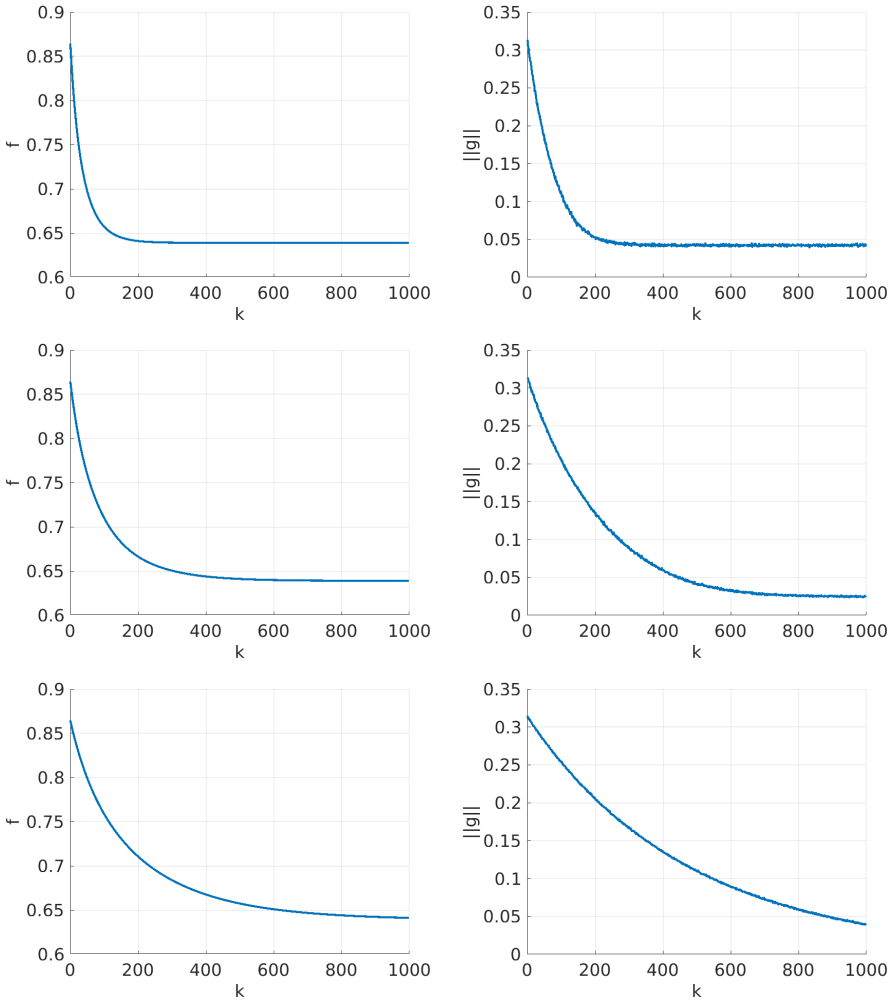

Figure 13: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration using subsampled inexact Newton's method, averaged across 1000 different runs, with batch size: (1) 100, (2) 250, and (3) 500.

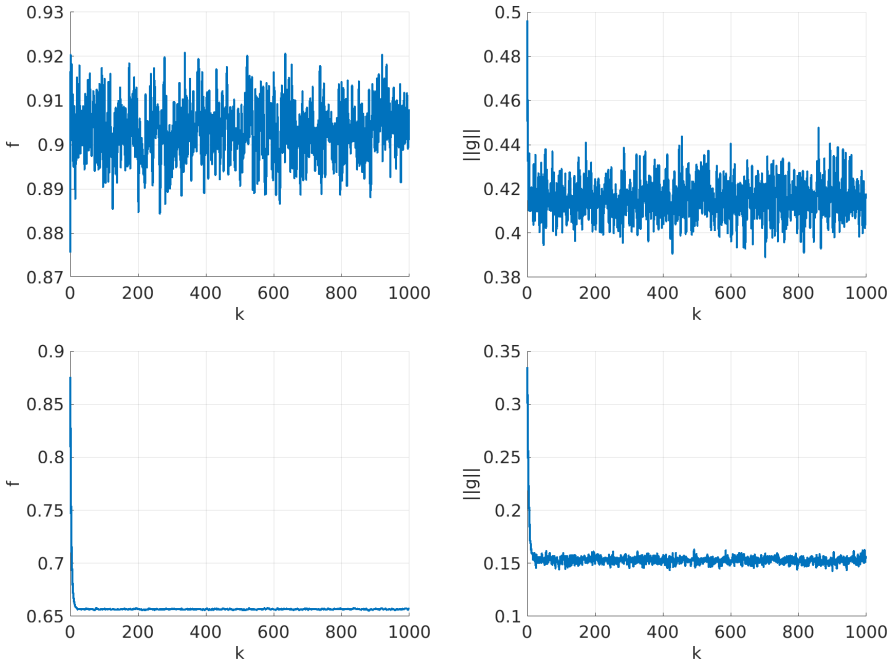

Figure 14: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration using subsampled inexact Newton's method, averaged across 1000 different runs, with batch size: (1) 2, (2) 10.

The results from Figure 13 and 14 together suggest that choosing a small batch size that is sufficient to approximate the Hessian is critical. This is further underlined by the run time result in Figure 15. We see that, like the batch size-run time relations observed in stochastic gradient descent (Figure 11), the run time of the algorithm scales linearly with respect to the batch size used. It further motivates the use of smaller batch size, as long as it can be used to approximate the Hessian sufficiently well. It is also worth noting that, compared to stochastic gradient descent, the run time for the subsampled inexact Newton's method is slower, and increases with a steeper slope.

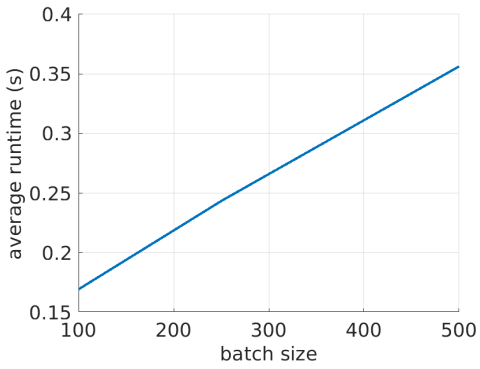

Figure 15: The effect of batch size on run time for subsampled inexact Newton's method.

4 Stochastic L-BFGS

4.1 Effect of batch size

In stochastic L-BFGS, there are two batch sizes to consider: Ng, batch size for evaluating stochastic gradient, and Nh, batch size for computing inverse Hessian. In this section, we first examine the effect of them on loss function value, gradient norm, and run time separately, then together. We default our step size decreasing strategy to slow decay ($\alpha(k) = 0.1/k$), memory size $m = 5$, and frequency to update $M = 10$. As usual, we iterate

4.1.1 Effect of batch size for gradient (Ng)

As we can see in Figure 17, the trend we see is similar to what we've seen in stochastic gradient descent (Figure 10): change in Ng leads to no noticeable change in the decay of loss function value or gradient norm across iterations, although smaller batch size leads to more variations of gradient norm between iterations.

4.1.2 Effect of batch size for inverse Hessian (Nh)

We conducted similar experiment fixing $Ng = 20$, varying $Nh = 50, 60, 70$. We observe similar trend as that of varying Ng as described in Section 4.1.1 (Figure 18).

4.1.3 Effect of both Ng and Nh batch sizes

Lastly, we conducted the experiment increasing both Ng and Nh at the same time. We observe similar trend as that of varying Ng as described in Section 4.1.1 and Section 4.1.2 (Figure 19).

4.1.4 Effect of batch size on run time

We also examine the effect of varying Ng, Nh, or both on the run time of stochastic L-BFGS. The run times from the parameter combinations from the previous sections are recorded. From the limited batch size combination we experimented with, we observe an overall (super?) linear relationship between batch size and program run time. This observation is consistent with that of stochastic gradient descent (Figure 11) and subsampled inexact Newton's method (Figure 15).

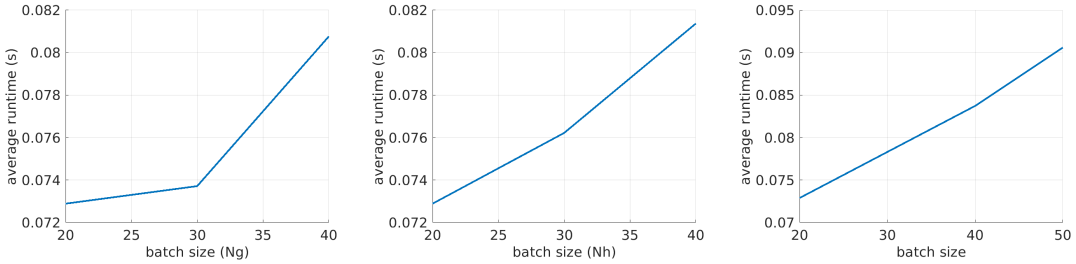

Figure 16: Change in program run time as we vary (1) Ng, (2) Nh, or (3) both.

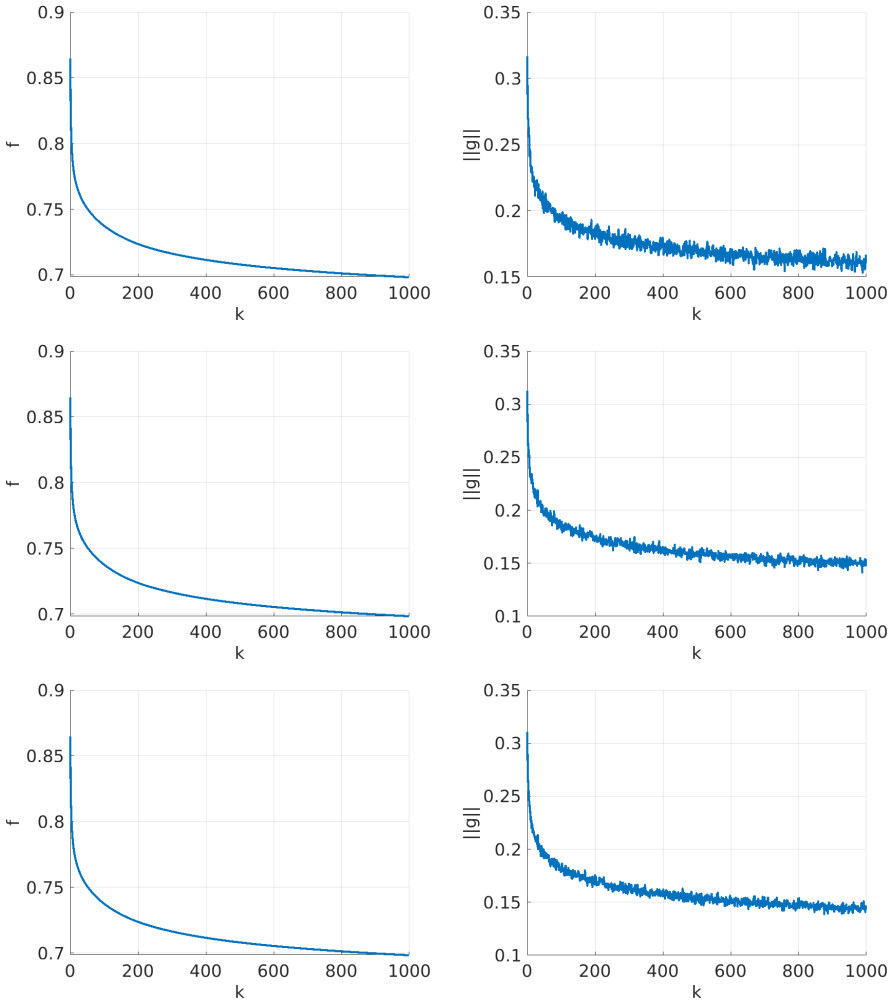

Figure 17: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration using stochastic L-BFGS, averaged across 1000 different runs, with batch size $Nh = 50$, and varying Ng: (1) 20, (2) 30, and (3) 40.

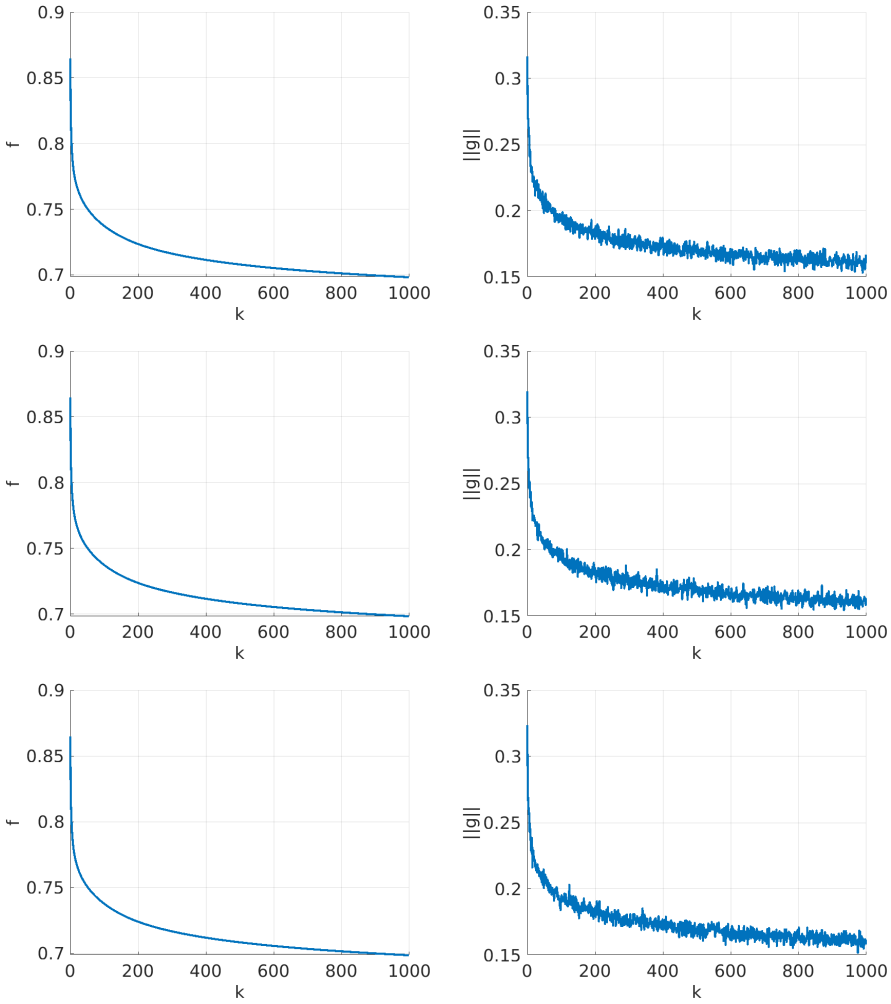

Figure 18: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration using stochastic L-BFGS, averaged across 1000 different runs, with batch size $Ng = 20$, and varying Nh: (1) 50, (2) 60, and (3) 70.

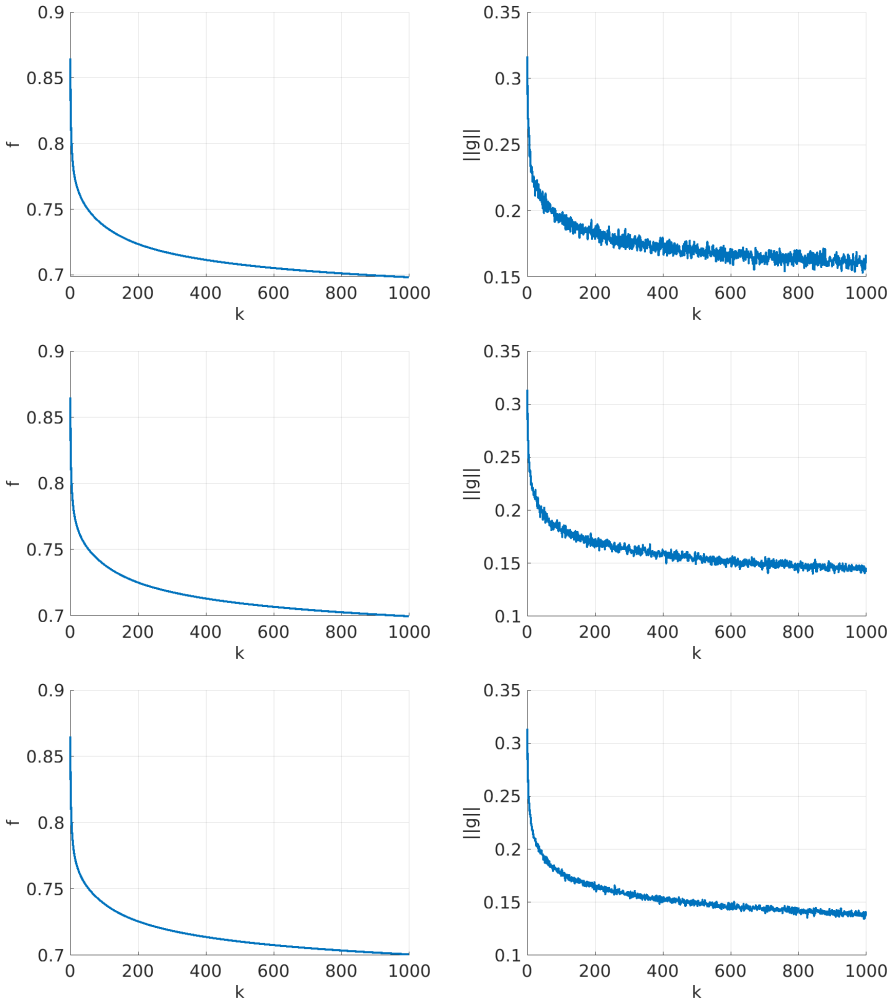

Figure 19: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration using stochastic L-BFGS, averaged across 1000 different runs, with batch size (Ng, Nh) varying: (1) 20-50, (2) 40-100, and (3) 100-200.

4.2 Effect of step size decreasing strategies

Setting $Ng = 20, Nh = 50$, $M = 10$, we examined the effect of step size decreasing strategies have on the performance of stochastic L-BFGS. We vary the step size decreasing strategies on stochastic L-BFGS: (1) none ($\alpha_k = 0.001$), (2) slow decay ($\alpha_k = 0.1/k$), and (3) exponential ($\alpha_k = 0.1/1.1^k$).

Like in stochastic gradient descent (Figure 9), both the loss function values and gradient norms decays in the same trend as the step size (Figure 20).

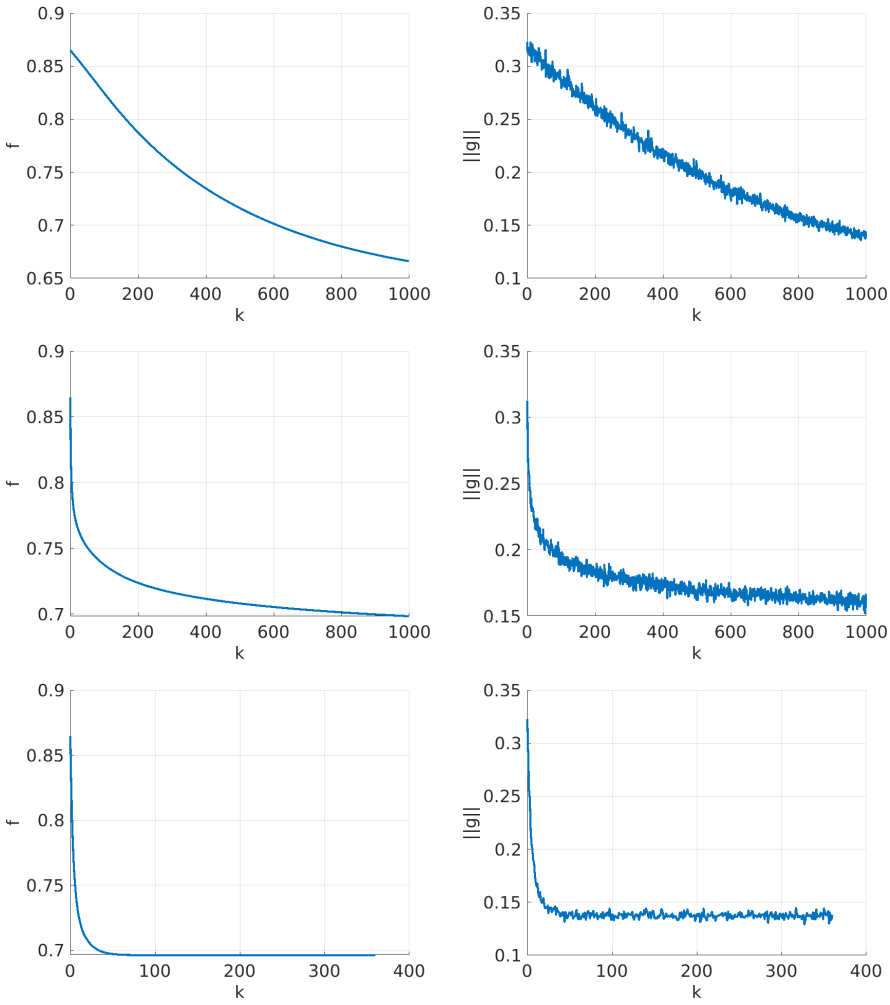

Figure 20: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration, averaged across 1000 different runs for step size decreasing strategies on stochastic L-BFGS: (1) none ($\alpha_k = 0.001$), (2) slow decay ($\alpha_k = 0.1/k$), and (3) exponential ($\alpha_k = 0.1/1.1^k$). The average run time for each method is 0.0734, 0.0891, and 0.0793, respectively.

4.3 Effect of frequency of updating

One special parameter of stochastic L-BFGS is the frequency we update the (s, y) pairs. Fixing $Ng = 20, Nh = 50$ and step size decrease strategy to slow decay $\alpha(k) = 0.1/k$, we vary $M = 10, 20, 30$.

While we can observe no noticeable difference in loss function decay or gradient norm decay (Figure 21), with the limited examples we tested, we see that the run time of the stochastic L-BFGS algorithm increases almost almost exponentially. This is a run time trend unobserved in stochastic gradient descent or stochastic Newton's method (Figure 22).

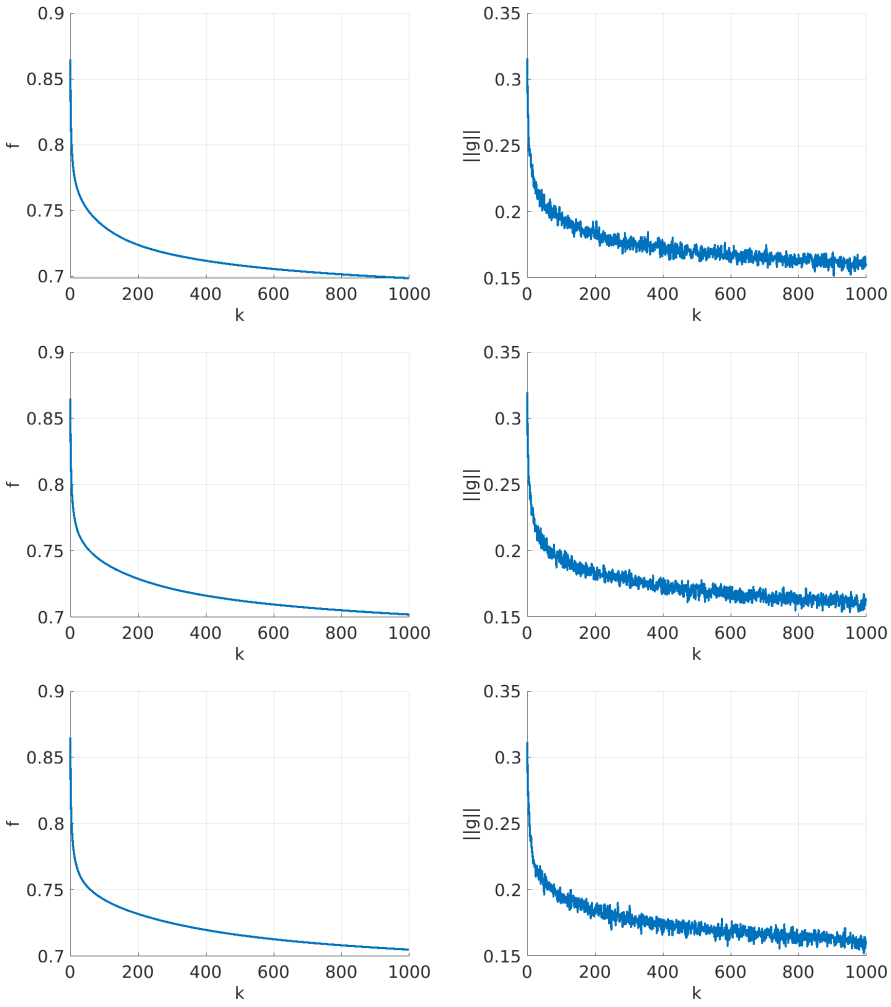

Figure 21: Values of the loss function (left column) and the norm of the gradient (right column) at each iteration, averaged across 1000 different runs for $M = 10, 20, 30$

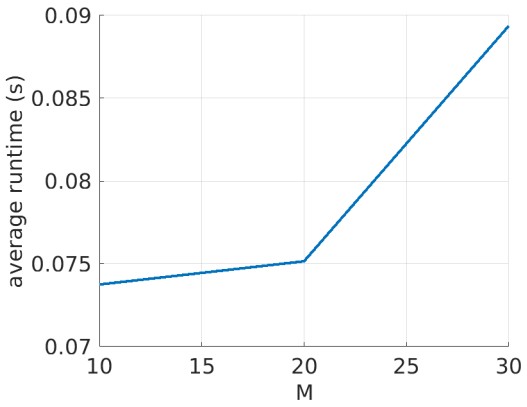

Figure 22: Run time of stochastic L-BFGS as we vary $M = 10, 20, 30$.