Numerical Optimization - Hand-In 7

Isabela Blucher

April 8, 2018

Introduction

For this week's assignment we will be dealing with variants of the Logistic Regression problem, which was introduced in week 1. Two algorithms have been implemented to estimate the true model solution for the Logistic Regression problem. Both algorithms use ADMM (alternating direction method of multipliers), which means that it is possible break down our complex optimization problem into smaller, easier to handle pieces.

The first algorithm that is going to be discussed is a trust-region Newton algorithm based on Algorithm 4.1 from the book with some changes to the trust region shape and the subproblem function.

In the second algorithm we use an adaptation of the Lasso algorithm to deal with the non-smooth 1-norm. Our objective here is to minimize the penalized Logistic Regression problem.

Trust-region Newton algorithm

As mentioned in the Absalon page, the application we are using this algorithm for finding genes associated with a sickness. For this problem, we usually have a much smaller number of subject than the number of genes considered in the analysis, which makes finding the correct gene combination very hard.

We generate a dataset with 20-dimensional points based from a standard gaussian distribution, and we generate the labels from a given true model $\theta^* = (0.5, 10, 3, 1, 0, ..., 0)$. We can add noise to the label generation, thus making it harder to predict correct values. Changing the size of the dataset can also alter the difficulty of the problem. For the following tests, we will use a dataset with 10 20-dimensional datapoints.

Description of the algorithm

We use Algorithm 4.1 from the book as a base for the implementation of our modified trust-region. To find our step direction p we solve a different subproblem, where our model function is $\min_{p,q} f_k + g_k^T p + \frac{1}{2} p^T B_k p + \sum_{i=0}^{2d+2} (p_i - q_i)^2 - \alpha_i (p_i - q_i)$ s.t. $\max\{-x_k i, -\Delta\} \leq q_i \leq \Delta$.

On the subproblem solver we first calculate a new p_{k+1} by taking a newton step, in which the gradient and the hessian are in relation to the modified model function. After that, we calculate q_{k+1} with the following closed formula $q_{k+1} = p_{k+1} - \frac{\alpha_k}{\mu}$. And for the final step we compute the new Lagrange multiplier with the formula $\alpha_{k+1} = \alpha_k - \mu(p_{k+1} - q_{k+1})$. We run this process until the values of p_k and q_k are very close, i.e. $\|p_k - q_k\| < 10^{-6}$.

After having computed the p value on the subproblem solver, we use it for the same process as all the other trust-region algorithms implemented in the past weeks. We compute and evaluate ρ and then decide what to do about our trust-region radius size and our next step. We run the outer loop until the step taken is very small (again, the tolerance is set at 10^{-6}, which means that we aren't moving significantly anymore.

Convergence of the subproblem solver

For the subproblem solver the stopping criteria is that if p and q are very close to one another the algorithm stops, which means that at every iteration we compute $\|p_k - q_k\|$ and see if it's smaller than 10^{-6}. To produce Figure 1, the trust-region algorithm was run with 30 different starting θ randomly sampled from a gaussian distribution. Since for each subproblem call we get a different number of iterations to converge, what is being plotted is the mean value of $\|p_k - q_k\|$ for all subproblem calls of one trust-region algorithm call, which is why we have 30 lines plotted. The algorithm was run with the same parameters for all starting iterates. The parameters were q_0 and α_0 as zero vectors, $\mu = 1$, $\Delta_0 = 1$, $\Delta_{max} = 10$, $\lambda = 1$ and the same dataset was used on all 30 function calls.

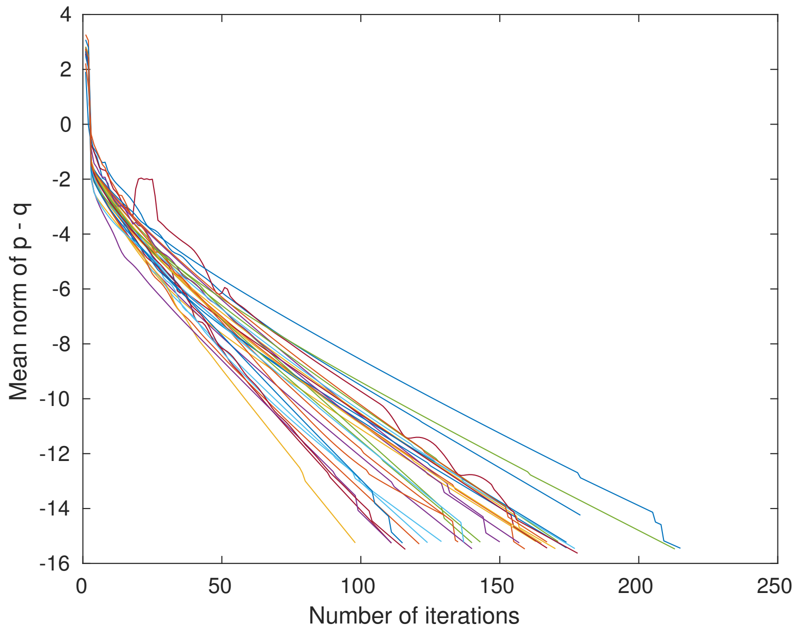

Figure 1: Convergence log-plot of mean of $\|p - q\|$ for 30 different starting iterates

Figure 1 shows that the subproblem algorithm converges in the interval of 100 to 210 iterations and even with oscillations in the y-axis values, it does end up in a place where p and q are almost equal. The statistics for the number of iterations until convergence for the subproblem solver are shown in Table 1.

Mean	Median	Standard Deviation
151.9667	154.5	28.7187

Table 1: Statistics on the number of iterations until subproblem convergence

Importance of μ on the performance of the algorithm

To analyze the importance of the parameter μ, we will run the algorithm with the same parameter values as for the subproblem convergence, except the starting iterate this time is a zero-vector. We run the algorithm for μ values of $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3, 10^4$ and get the number of iterations and also the run time needed for the full algorithm to converge.

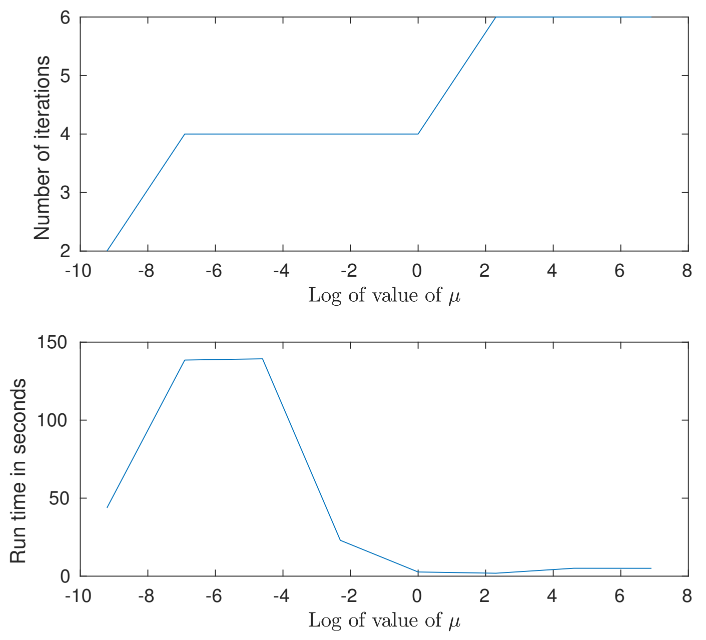

Figure 2: Number of iterations and run time until convergence for different values of μ

From Figure 2 we can conclude that for smaller values of μ, such as 10^{-4} and 10^{-3} it takes less iterations but the run time is so much longer than for larger values of μ. From testing, we can see that the optimal value for μ is 1 and that the parameter selection is very important for the algorithm's performance.

Convergence of the complete solver

The projected gradient wasn't converging really well, so the criteria used here is that the previous step length and the new step length are very similar to one another, which implies that we aren't moving enough to leave the current θ. Since the starting value for p is a zero-vector, the graph goes up before converging down. The plot was done with 30 different starting iterates, randomly sampled from a gaussian distribution.

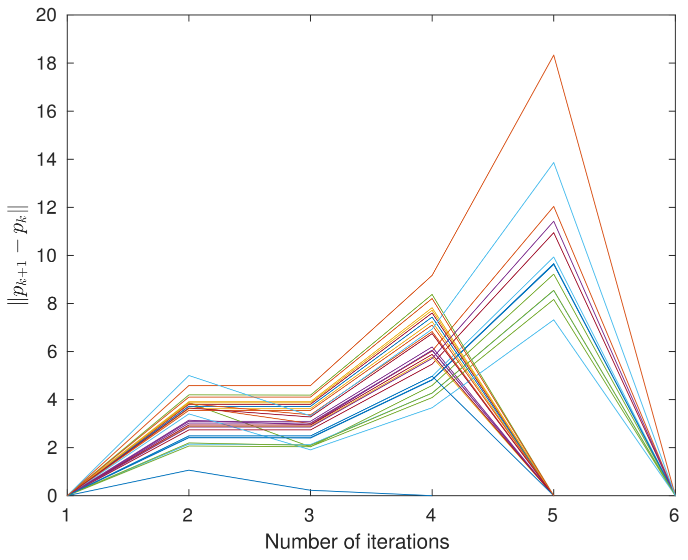

Figure 3: Convergence plot for the complete solver

Table 2 shows the statistics on the number of iterations for the complete solver.

Mean	Median	Standard Deviation
5.8889	6	1.95

Table 2: Statistics on the number of iterations until full solver convergence

Experiment with noise rate and regularization parameter: λ. Can you obtain a parameter vector close to the correct one?

For the noise rate $\sigma > 0$, the greater the value, the harder it is to obtain a parameter vector close to the correct one. When the noise is 1 the final θ is already very different from the true θ^*. The greater the noise, the sparser the final result.

For the regularization parameter λ, the algorithm wouldn't even converge for very small values such as $10^{-4}, 10^{-3}$ and 10^{-2}. As for the values 1, 10, 100 and 1000 the following statistics were found

	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$	$\lambda = 1000$
Number of iterations	6	8	9	10
Run time in seconds	2.4020	1.2653	1.7807	1.2916

Table 3: Statistics on the regularization parameter λ

Adaptation of Lasso algorithm

For this algorithm, the same description of the dataset generation holds. We generate a dataset with 20-dimensional points based from a standard gaussian distribution, and we generate the labels from a given true model $\theta^* = (0.5, 10, 3, 1, 0, ..., 0)$.

Description of the algorithm

For the alternative ADMM algorithm, we want to stop running when θ_k and φ_k are very close to one another, which means that we evaluate if $\|\theta_k - \varphi_k\|$ is smaller than a tolerance, in this case, 10^{-6} (violation of equality constraint).

The loop structure involves computing θ_{k+1} by minimizing a function with Newton's method, then we compute φ_{k+1} with a closed formula obtained by 1D soft shrinkage, and finally we compute the new Lagrange multiplier λ_{k+1} with the same formula given in (17.39).

Convergence studies

The stopping criteria for this algorithm is that θ and φ are very close in value to one another and that the number of iterations doesn't exceed a fixed maximum of 10^4. So for our convergence analysis we will be plotting $\|\theta_k - \varphi_k\|$ over the number of iterations needed for convergence. We do that for 30 different starting iterates randomly sampled from a gaussian distribution.

The parameter values used for testing here are $\rho = 1$, $\mu = 1$ and the initial φ and λ are both zero-vectors. Again our dataset has 10 20-dimensional datapoints, and the noise parameter σ for the tests is 0.1.

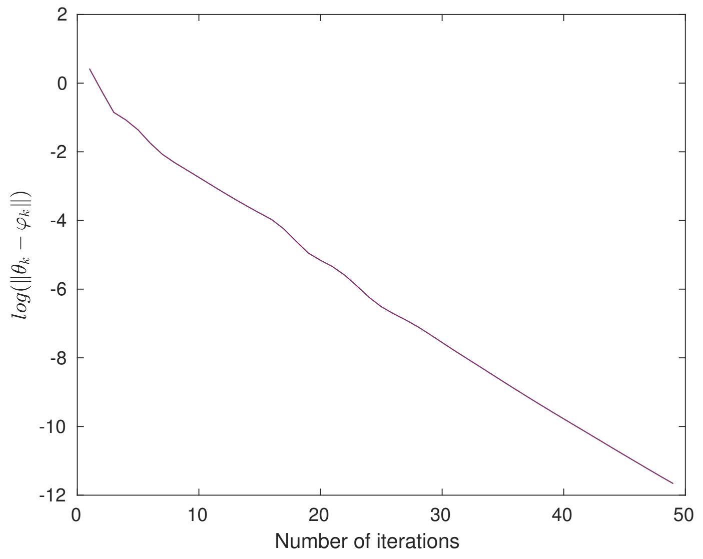

Figure 4: Convergence plot for the complete solver

Figure 3 has 30 lines but the decrease of $\|\theta_k - \varphi_k\|$ was so similar for all of the 30 points that the plots are aligned. The average number of iterations needed to converge is 49.

Importance of μ on the performance of the algorithm

To test the importance of the parameter μ, we run the algorithm for μ values of $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2$, $10^3, 10^4$ and measure the number of iterations and run time in seconds until convergence. Figure 5 shows how the performance changes with the parameter value.

The other parameters were fixed as $\rho = 1$, the starting φ and λ as zero-vectors and the starting θ as a one-vector.

Also from Figure 5, we see that for very small values of μ, like 10^{-4} and 10^{-3}, the algorithm takes less than 10 iterations to converge. In this case parameter selection is very important, as for some values of μ the performance is less than ideal.

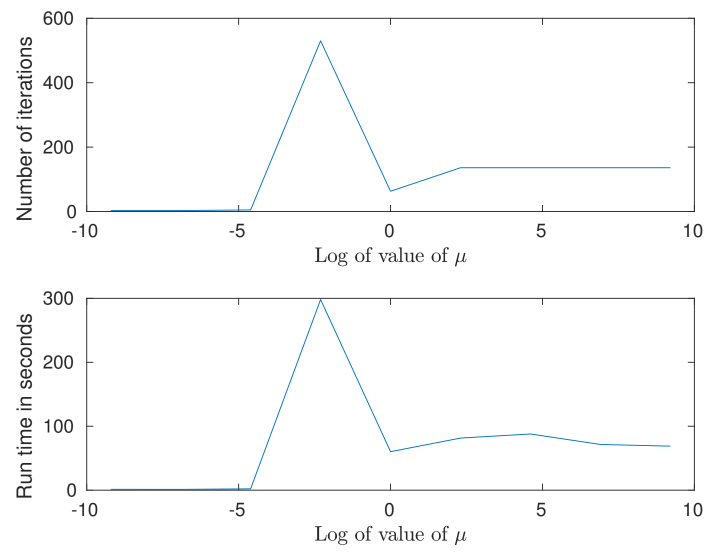

Figure 5: Performance of the algorithm for different values of μ

Importance of ρ on the performance of the algorithm

To test the importance of ρ on the algorithm's performance we run the same tests that we did for μ. The values of ρ being tested are also $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3, 10^4$. The starting φ and λ are zero-vectors and the starting θ is a one-vector. Figure 6 shows how the performance changes with different parameter values.

For values of ρ smaller than 10^{-1}, the algorithm was taking too long to converge, which is why the values aren't presented in the graph below.

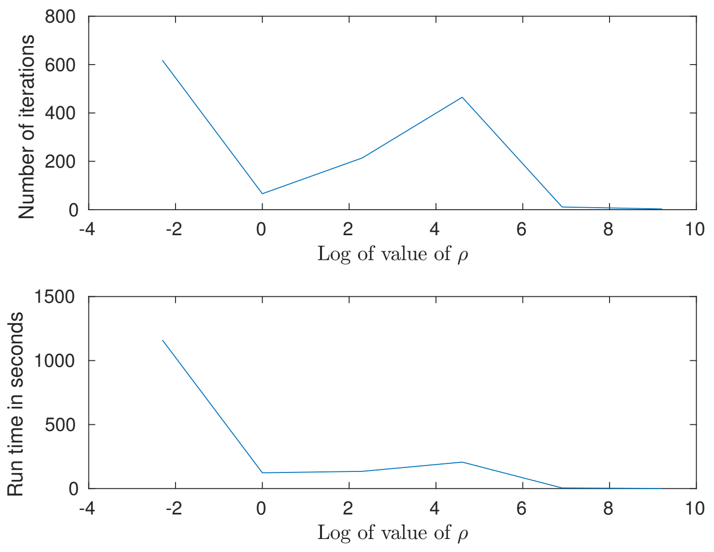

Figure 6: Performance of the algorithm for different values of ρ

From Figure 6 we can see that the optimal parameter value for ρ is 1. For very small values convergence slows down too much and for greater values it makes a bit more computation to converge, even though the run time is relatively similar.

Conclusion

In conclusion, running both algorithms for problems where we have less datapoints than dimensions is a complex process, and it can get easier and harder depending on a number of things, such as, noise in the dataset generation, the regularization parameter for the trust-region algorithm and general optimal parameter values selection, which can make the convergence much faster.