Convex Optimization 10-725/36-725
Homework 4, due Oct 31

Instructions:

- You must complete Problems 1–3 and **either Problem 4 or Problem 5** (your choice between the two).
- When you submit the homework, upload a single PDF (e.g., produced by LaTeX, or scanned handwritten exercises) for the solution of each problem separately, to blackboard. You should your name at the top of each file, except for the first problem. **Your solution to Problem 1 (mastery set) should appear completely anonymous to a reader.**

N.B. A few problems may appear quite long, but really, there is just a lot of text setting up and motivating the problem. This does not mean more work for you! It's simply trying to give you an idea of where the problem is coming from, and why you'd want to solve it in the first place.

1 Mastery set [20 points] (Yifei)

A [5 points]. Prove that the dual norm of the dual norm is the original norm.

B [5 points]. Another derivation for the prox operator for the L_1 penalty $\min_x \frac{1}{2}\|y - x\|_2^2 + \lambda\|x\|_1$. What is its dual problem? What is an obvious explicit solution to that dual problem? What is the explicit solution to the primal problem using KKT conditions?

C [5 points]. Consider the linear programming problem

$$\begin{aligned}
\max_{x_1,x_2} \quad & x_1 + x_2 \\
\text{s.t.} \quad & x_1 + \frac{x_2}{2} \le 1 && (\text{ constraint } u_1) \\
& \frac{x_1}{3} + x_2 \le 1 && (\text{ constraint } u_2) \\
& x_1, x_2 \ge 0 && (\text{ constraints } u_3, u_4)
\end{aligned}$$

What is its dual problem paramerized by $(u_1, \ldots, u_4)$? Do we have strong duality? What is the KKT correspondences between the primal optimizer and the dual optimizer? How can you change this problem to a feasibility problem that ellipsoid method uses?

D [5 points]. Prove that the dual problem of the dual problem on page 24 of lecture 14 is the primal. Here, you may assume that f is closed and convex, which implies $f^{**} = f$.

2 Safe rules for the LASSO [25 points] (Adona)

By now you should be pretty familiar with the LASSO. As we've seen in class, the LASSO is an algorithm for performing sparse linear regression, based on solving the following L$-$1 regularized optimization problem:

$$\min_{\beta} \frac{1}{2}||Y - X\beta||_2^2 + \lambda||\beta||_1$$

We can solve the above problem using standard convex optimization methods such as generalized gradient descent, and we are guaranteed to obtain a solution β with a large number of zeros. Moreover, we can prove that the solution is stable: with probability 1, perturbing y slightly will generate a β with the same support (set of zero entries).

This exercise explores a very surprising result about the LASSO, and $L - 1$ regularized problems in general: the fact that you can create *a priori tests* which can identify some entries of β *guaranteed* to be zero, without needing to solve the optimization problem. The tests likely won't identify *all* of the zero entries, but could potentially identify a large number of them, and thus greatly reduce the size of the optimization problem you need to actually solve. This seems like it shouldn't work, but it does, and it's a great example of the power of dual theory. Let's see how it goes!

Consider a slightly more general setting than the LASSO, in which we are trying to optimize the following L-1 problem:

$$\min_{\beta \in R^p} f(X\beta) + \lambda||\beta||_1 \tag{1}$$

where f is a convex function, $X \in R^{n \times p}$, $\lambda > 0$.

It can easily be shown (and you will show below!) that the dual problem is:

$$\begin{aligned} &\max_{u \in R^n} -f^*(-u) \\ &\text{subject to } ||X^T u||_\infty \leq \lambda \end{aligned} \tag{2}$$

and that the stationarity KKT condition for β gives:

$$X^T u \in \lambda \partial ||\beta||_1 = \lambda \begin{cases} \{sign(\beta_i)\}, & \text{if } \beta_i \neq 0 \\ [-1, 1], & \text{if } \beta_i = 0 \end{cases} \tag{3}$$

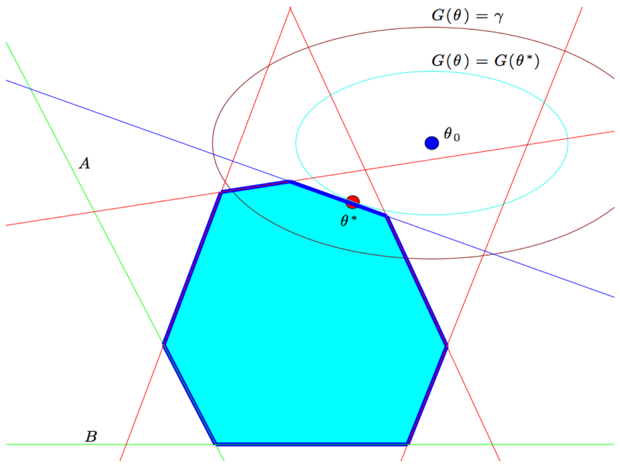


Figure 1:

Visualizing these results is very insightful. Consider Figure 1, taken from the original safe rules paper[1]. We know that the dual objective $-f^*(-u)$ must be concave, so we can represent it via contour lines (the ellipses in the upper-right corner) around the unconstrained maximum point (denoted θ_0 instead of u_0 in the figure). The constraints set of the dual problem is the intersection of $2n$ half-planes, $-\lambda \leq X_k^T u \leq \lambda$, so it is a polytope (the shaded polytope in Figure 1). Optimizing problem (2) corresponds to finding the tightest contour of the objective which still intersects the polytope, in other words finding the contour of the objective *tangent* to the polytope. Further, from the KKT condition, we know that if any of the constraints $X_k^T u$ is not tight ($X_k^T u < \lambda$), then that entry of β_k is zero. So the face of the polytope the contour is tangent to *determines our support set.* This is key!

We can use this insight to come up with a clever way to generate a safe rule. Finding the actual tangent contour line involves solving the optimization problem, and is a lot of work. What if instead we use an approximation, a wider contour line corresponding to a lower value of the objective $-f^*(-u)$? Such a contour line might intersect more of the constraints than the optimal one, so it might be suspicious of entries β_k which are in reality zero. However, any constraints it does not intersect correspond to entries β_k which are *guaranteed* to be zero.

Mathematically, assume we are given γ, a lower bound on our dual objective at the optimum: $-f^*(-u^*) \geq \gamma$ (the objective calculated at any feasible point u will do for now!). Then, we can calculate:

$$T_k = \max_{u \in R^n} |X_k^T u|$$
$$\text{subject to } -f^*(-u) \geq \gamma$$

[1]http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-126.html

and test $T_k < \lambda$ (have we intersected the constraint or not). If k passes the test (not intersected), then β_k is guaranteed to be zero. This new optimization problem is in many cases much simpler than the original problem in (1) or (2), since it no longer involves non-smooth norms. We will see that for the LASSO, we can dualize it (!), and then solve it in closed form. This will give us a direct, closed form test for whether a entry β_k is guaranteed to be zero. Ok, to work!

(a) [8 points] Show that the dual problem of (1) is (2), and derive the KKT condition in (3).

(b) [7 points] Show that the dual of the (positive part) of the test $T_{k,+}$:

$$\begin{aligned} T_{k,+} = \max_{u \in R^n} & \; X_k^T u \\ \text{subject to} & \; -f^*(-u) \geq \gamma \end{aligned}$$

is:

$$T_{k,+} = \min_{\mu > 0} -\mu\gamma + \mu f(-\frac{X_k}{\mu})$$

(c) [5 points] Substitute the LASSO function $f(X\beta) = \frac{1}{2}||Y - X\beta||_2^2$, and minimize to show that:

$$T_{k,+} = \sqrt{Y^TY - 2\gamma} + Y^T X$$

Assume that X has normalized columns: $X_k^T X_k = 1$ for all k.

(d) [2 points] Let

$$\begin{aligned} T_{k,-} = \max_{u \in R^n} & \; -X_k^T u \\ \text{subject to} & \; -f^*(-u) \geq \gamma. \end{aligned}$$

Show that

$$\lambda > T_k = \max(T_{k,+}, T_{k,-})$$

is equivalent to:

$$\lambda > \sqrt{Y^TY - 2\gamma} + |Y^T X_k|.$$

Notice that we've just derived a safe rule which is trivial to evaluate; in other words, if the above holds, then we can pre-determine that $\beta_k = 0$ at the solution!

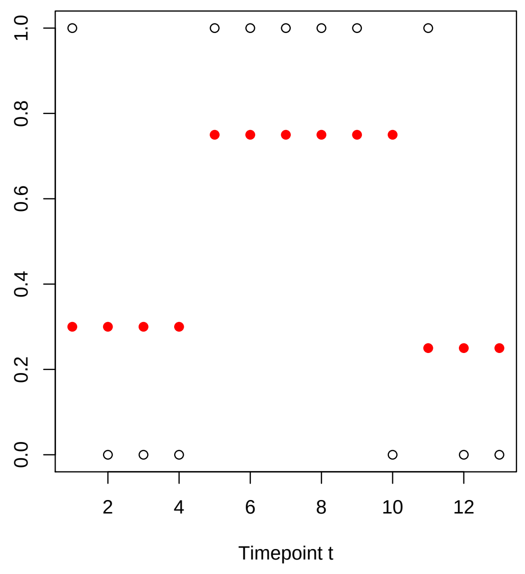
(e) [3 points] Let's complete the rule; all that is left is we need a way to find a lower bound γ. Show that you can obtain a dual feasible point by just scaling Y by a constant, i.e., $u = sY$ for a constant s, and give a corresponding (explicit) form for γ.

3 Binary sequences of piecewise constant expectation [30 points]

Suppose that we observe a sequence of binary variables, $z_t \in \{0,1\}$ across timepoints $t = 1, 2, \dots n$, and we believe that these variables are generated according to

$$z_t \overset{\text{ind}}{\sim} \text{Bin}(p_t), \quad t = 1, \dots n,$$

where the underlying probabilities $p_t \in [0,1]$ are piecewise constant across $t = 1, \dots n$. An illustration looks something like this:



where the black hollow points represent the observed data points z_t and the red solid points the underlying probabilities p_t.

As motivation, suppose that we are examining products that are being shipped to our company, one at a time, across timepoints $t = 1, \dots n$, and z_t is the indicator that the product t has a particular defect. We believe that the manufacturing process is such that there is a given (constant) probability of defect across some unknown period of time, and then due to a change in the process, this probability changes to some other value and remains there for another unknown period of time, with another change potentially happening after that, etc.

To construct an estimate of the underlying probabilities from the observed binary data, we assume the model

$$p_t = \frac{e^{\beta_t}}{1 + e^{\beta_t}},$$

and compute estimates $\hat{\beta}_t$, $t = 1, \dots n$ by solving the following minimization problem:

$$\min_{\beta \in \mathbb{R}^n} \sum_{t=1}^{n} \Big(- z_t \cdot \beta_t + \log(1 + e^{\beta_t}) \Big) + \lambda \sum_{t=1}^{n-1} |\beta_t - \beta_{t+1}|. \tag{4}$$

This is a regularized logistic regression problem, with a fused lasso penalty. The tuning parameter is $\lambda \geq 0$.

(a) [2 points] Show that (4) fits into the setup considered by Problem 4 of Homework 3; in particular, look at the setup considered in part (b) of this problem. What is the matrix $D \in \mathbb{R}^{m \times n}$ in the present case, and what is m?

Now that we have established this, we may use the result of Problem 4 of Homework 3 and write the dual of (4) as

$$\begin{aligned} \max_{u \in \mathbb{R}^m} \; g(u) := & - \sum_{t=1}^{n} \Big(y_t(D^T u)_t \log \big(y_t(D^T u)_t \big) + \big(1 - y_t(D^T u)_t \big) \log \big(1 - y_t(D^T u)_t \big) \Big) \quad (5) \\ \text{subject to} \quad & 0 \leq y_t(D^T u)_t \leq 1, \;\; t = 1, \dots n \\ & -\lambda \leq u_i \leq \lambda, \;\; i = 1, \dots m, \end{aligned}$$

where $y_t = 2z_t - 1 \in \{-1, 1\}$, $t = 1, \dots n$ is a transformation of our binary sequence, and $D \in \mathbb{R}^{m \times n}$ the particular matrix that you described in (a). The dual variable u here is m-dimensional, and from a dual solution $\hat{u}$ we can compute a primal solution via

$$\hat{\beta}_t = -y_t \log \big(y_t(D^T \hat{u})_t \big) + y_t \log \big(1 - y_t(D^T \hat{u})_t \big), \quad t = 1, \dots n. \tag{6}$$

(b) [2+2+2+2+2 points] Show that $\nabla g(u) = Dc(u)$ for some vector c. Show that $\nabla^2 g(u) = DW(u)D^T$ for some diagonal matrix W. Define $\phi(u)$ to be the log barrier function. Show that $\nabla \phi(u) = a(u) + Db(u)$ for some vectors a, b. Show that $\nabla^2 \phi(u) = U(u) + DV(u)D^T$ for some diagonal matrices U, V. Hence for some barrier constant τ, when solving $\tau g(u) + \phi(u)$, write down the Newton step direction.

(c) [8 points] Implement the barrier method using Newton's method for the inner loops. Your function for the barrier method should take as inputs (besides the obvious inputs z, D, λ): an initial barrier parameter $\tau^{(0)} > 0$, an update parameter $\mu > 1$ for the barrier parameter, an initial step size $s^{(0)}$ before backtracking in Newton's method, parameters $\gamma_1, \gamma_2 > 0$ for the backtracking in Newton's method, a tolerance $\epsilon_{\text{inner}} > 0$ for the inner loop, and a tolerance $\epsilon_{\text{outer}} > 0$ for the outer loop.

(Hint 1: for the barrier method, you must begin with a strictly feasible point for (5); this is not a trivial computation, but it's one that can be solved with an LP.)

(Hint 2: to make your algorithm run faster, you can take advantage of the Hessian, i.e., the structure of D, to efficiently solve the linear systems at each iteration of Newton's method. In R or Matlab, this is just done by making sure the Hessian is stored as a sparse matrix.)

(d) [10 points] Run your algorithm on the binary sequence data "binseq.txt" linked from the class website, to solve the problem (4) with $\lambda = 20$. You can set $\tau^{(0)} = 5, \mu = 10, s^{(0)} = 1, \gamma_1 = 0.1, \gamma_2 = 0.8, \epsilon_{\text{inner}} = 1e-8, \epsilon_{\text{outer}} = 1e-6$. Note: you are actually solving the dual (5); to get the primal solution, use the transformation (6).

Plot the estimated probabilities, $\hat{p}_t = \exp(\hat{\beta}_t)/(1+\exp(\hat{\beta}_t))$, $t = 1, \ldots n$, as a function of t. How many iterations in total of Newton's method were performed (i.e., how many linear systems in total were solved) before convergence? What are the estimated times at which the underlying probability of success changes?

4 Statistical estimation and penalty methods [25 points] (Sashank)

A [KL divergence estimation]. For the first part, we will explore the use of optimization problems for statistical estimation problems. In particular, we focus on estimation of KL divergence between multivariate probability distributions $\mathbb{P}$ and $\mathbb{Q}$. As input, we are given set of samples $\{X_1, \ldots, X_n\}$ and $\{Y_1, \ldots, Y_n\}$ from distributions $\mathbb{P}$ and $\mathbb{Q}$, with densities p and q respectively, over $\mathcal{X} \subset \mathbb{R}^d$. Our goal is to estimate KL divergence, which is given by the following:

$$\mathrm{KL}(\mathbb{P}, \mathbb{Q}) = \int p(x) \log \frac{p(x)}{q(x)} dx.$$

Assume that the support of densities p and q is $\mathcal{X}$. One way to estimate the KL divergence is to first estimate the densities p and q, and then calculate the KL divergence using these estimators. This procedure is computational and statistically inefficient. We will take a different route to estimate the KL divergence. Answer the following questions as concisely as possible.

(a) [5 points] Prove that $\mathrm{KL}(\mathbb{P}, \mathbb{Q}) = \sup_{g>0} \int \log g(x) p(x) dx - \int g(x) q(x) dx + 1$. (Hint: use the conjugate of log). What value of $g(x)$ achieves the maximum?

(b) [2 points] Assuming that the empirical average is a good estimator of expected value, e.g., $\frac{1}{n}\sum_i f(X_i)$ is a good enough approximation of $\int f(x)p(x)dx$, write down an estimator for KL divergence.

(c) [5 points] Assume that the ratio $g(x) = \frac{p(x)}{q(x)}$ is of the form $\exp w^\top x$. Write a dual for the optimization problem above. How will you solve this optimization problem?

It should be noted that while we used this procedure for KL divergence, it can be extended to estimate a more general class of functionals.

B [Penalty methods]. Suppose that f, h_i are convex differentiable functions, and let (P) denote the following convex program:

$$\begin{gathered}\min_{x\in\mathbb{R}^n} f(x)\\ \text{subject to } h(x) \le 0,\end{gathered}$$

where $h(x) = (h_1(x), \ldots, h_m(x))$.

Let $h_i^+(x) = \max(0, h_i(x))$, and construct the following penalty function:

$$p(x) = \sum_{i=1}^{m} h_i^+(x)$$

Using this penalty p, for any $c > 0$, we define the penalty problem $(P(c))$ as

$$\min_{x\in\mathbb{R}^n} f(x) + cp(x).$$

The goal of this exercise is to show that for this particular penalty function p if the penalty parameter c is large enough, then (P) and $(P(c))$ are equivalent. Let $u^* \in \mathbb{R}^m$ be an optimal solution of the dual of (P), and let $c > \|u^*\|_\infty$. Let x^* be a primal solution.

(i) [3 points] Write down all the KKT conditions involving x^*, u^*, f, h, p.

Using these conditions, and the convexity, differentiability of f, h, prove the following statements:

(ii) [5 points] If a vector $\tilde{x}$ solves (P), u^* is a dual optimal solution of (P), and $c > \|u^*\|_\infty$, then $\tilde{x}$ solves $(P(c))$ as well.

(iii)[5 points] If a vector $\tilde{x}$ solves $(P(c))$, u^* is a dual optimal solution of (P), and $c > \|u^*\|_\infty$, then $\tilde{x}$ solves (P) as well.

5 Binary sequences revisited [25 points]

The binary sequence denoising problem (4) in Problem 3 is not an easy one to solve directly. The typical plan of attack is the one explored in Problem 3, which solves the dual problem using an interior point method. This is appealing because it can be efficient (the inner Newton iterations are quite efficient, due to the structure of the Hessian), but it is perhaps not the simplest approach.

What happens if we were to try to solve (4) directly using generalized gradient descent? To do this, we would need to be able to evaluate the prox function

$$\text{prox}_s(\beta) = \underset{x\in\mathbb{R}^n}{\text{argmin}}\ \frac{1}{2s}\|\beta - x\|_2^2 + \lambda \sum_{t=1}^{n-1} |x_t - x_{t+1}|. \tag{7}$$

Evaluating such a prox function is difficult because the above problem does not have an explicit (closed-form) solution; transparently, we would have to apply another iterative optimization technique to approximate its solution. Fortunately, your instructors and TAs are smart people—actually, it's just that they know other smart people—and are providing you with code to evaluate the prox function in (7) exactly. This code is an implementation of a beautiful algorithm by Nicholas Johnson (see "A dynamic programming algorithm for the fused lasso and L_0-segmentation", published in JCGS in 2013).

(a) [8 points] Implement generalized gradient descent to solve problem (4), using the prox function given to you in C++ code, linked from the course website as "prox_R.cpp" and "prox_matlab.cpp" for use in R and Matlab, respectively. Use backtracking to determine the step size at each iteration, and stop when the difference in criterion value across iterations is less than a user-specified tolerance level. Hence (aside from z, λ) your function should take as inputs: an initial step size $s^{(0)}$ before backtracking, a backtracking update parameter γ, and a tolerance level ϵ.

(Hint: R users, compile this code using by running `R CMD SHLIB prox_R.cpp` from the command line. This will give you the file "prox_R.so". Then use the provided code "prox.R" to access the prox function from R.

Matlab users: compile the code by running `mex prox_matlab.cpp` inside the Matlab command window. Then use the provided code "prox.m" to access the prox function from Matlab.)

(b) [6 points] Run your generalized gradient descent implementation on the binary sequences data, provided in "binseq.txt". Run generalized gradient descent over 80 values of λ between 0.001 and 200, equally spaced on the log scale. (Note: in R, these are given by `lams = exp(seq(log(0.001), log(200), length=80))`, and in Matlab, `lams = exp(linspace( log(0.001), log(200), 80));`.) Starting from the largest λ value to the smallest, run generalized gradient using both warm starts (starting from the previously computed solution), and using cold starts (starting from $\beta = 0$). For the rest of the parameter values, use $s^{(0)} = 1, \gamma = 0.8, \epsilon = 1e-6$.

For each strategy (warm/cold starts), record the total number of prox evaluations performed by the algorithm at each value of λ. Remember that the prox evaluation is more or less the fundamental unit of computation for generalized gradient descent. Plot the number of prox operations taken by the algorithm as a function of λ, overlaying the curves for both warm and cold starts. Do you see a difference? And more broadly, what do the curves portray about the difficulty of solving the problem (4) as a function of λ?

(c) [6 points] Do the same as in (b), but using your barrier method implementation from Problem 3. (For the parameters $\tau^{(0)}, \mu, \ldots$, use the same values as in Problem 3.)

Now, for the barrier method, you will record the number of Newton iterations (i.e., linear system solves) at each λ value, this being its fundamental unit of computation. Also, you will solve the problems starting at the smallest value of λ and working your way up to the largest (note that in the other direction, the warm starts would not be feasible). Produce the same plot and address the same questions as in (b).

(d) [5 points] Make comparisons between the two algorithms (barrier method and generalized gradient). We are leaving this open-ended on purpose; you can make both high-level qualitative, and quantative comparisons; e.g., you might find it useful to look at the criterion values produced by solutions from each method, as a function of λ.

(Bonus) [5 points] Using whatever algorithm you find more efficient/accurate, devise a principled method for selecting an appropriate value of λ for the binary sequence data (it can't involve looking at the solutions by eye!). What value of λ does your method choose? Plot the corresponding underlying probabilities.