Homework 3

Convex Optimization 10-725

Due Friday October 11 at 11:59pm

Submit your work as a single PDF on Gradescope. Make sure to prepare your solution to each problem on a separate page. (Gradescope will ask you select the pages which contain the solution to each problem.)

Total: 75 points

1 Duality in linear programs (18 points)

(a, 3 pts) Derive the dual of

$$\begin{aligned} \min_{x_1,x_2} \quad & -4x_1 + 2x_2 \\ \text{subject to} \quad & -x_1 + x_2 \geq 2 \\ & x_1 - x_2 \geq 1 \\ & x_1, x_2 \geq 0. \end{aligned}$$

What are the primal optimal value and the dual optimal value? What is the duality gap?

(b) Suppose both Ryan and the TAs want many students to attend their office hours. However, the TAs have noticed that students are less likely to go to their office hours if they attend Ryan's, so the TAs decide to sabotage Ryan's office hours. The TAs will block the paths between class in Baker A51 and Ryan's office in Baker 229B.

To simplify, we'll think of the ways to get landscape between class and Ryan's office as directed graph $G = (V, E, C)$. Here, vertices $v_i, v_j \in V$ correspond to the locations i, j, the directed edge $(i, j) \in E$ is the directed path from v_i to v_j, and the capacity $c_{ij} \in C$ is the maximum number of convex optimization students that can pass through (i, j). Students start from v_s, our classroom in Baker A51, and move along the directed edges towards v_t, Ryan's office in Baker 229B. We assume there are no edges that end in v_s or originate in v_t.

The TAs decide to block paths by building barricades. However, they want to do as little physical labor as possible, so they only want to block the tightest path (i.e., smallest total capacity) in a way that still prevents every student from reaching Ryan's office.

In other words, the TAs want to find a partition, or cut, $C = (S, T)$ of V, such that $v_s \in S$ and $v_t \in T$ and it has minimum capacity. The capacity of a cut is defined as:

$$c(S, T) = \sum_{(i,j)\in E} b_{ij} c_{ij},$$

where $b_{ij} = 1$ if $v_i \in S$ and $v_j \in T$, and $b_{ij} = 0$ otherwise. Thus the The TAs min-cut problem can

be formulated as follows:

$$\begin{aligned} \min_{b\in\mathbb{R}^{|E|},\, x\in\mathbb{R}^{|V|}} \quad & \sum_{(i,j)\in E} b_{ij}c_{ij} \\ \text{subject to} \quad & x_s = 1,\ x_t = 0 \\ & b_{ij} \geq x_i - x_j,\ b_{ij}, x_i, x_j \in \{0,1\},\ \text{for all } (i,j)\in E. \end{aligned} \tag{1}$$

Now the questions.

(i, 1 pt) Explain what the variables x_i and x_j for all $(i,j) \in E$ mean and why the introduction of these variables is necessary (what would happen if the x_i, x_j variables weren't introduced?)

(ii, 2 pt) The problem in (1) is an integer linear program (ILP), because its variables take integer values. Because ILPs are generally difficult to solve, they are often relaxed to LPs. Consider the following relaxation of the integer constraints in (1):

$$\begin{aligned} \min_{b\in\mathbb{R}^{|E|},\, x\in\mathbb{R}^{|V|}} \quad & \sum_{(i,j)\in E} b_{ij}c_{ij} \\ \text{subject to} \quad & x_s - x_t \geq 1,\ b \geq 0 \\ & b_{ij} \geq x_i - x_j,\ \text{for all } (i,j)\in E. \end{aligned} \tag{2}$$

Explain why we can view this as a relaxation. How does the optimal value of the original ILP, $f^\star_{\text{ILP}}$, compare to the optimal value of the relaxed LP, $f^\star_{\text{LP}}$?

(iii, 6 pts) Next, derive the dual of (2). Use the following dual variables: $f \in \mathbb{R}^{|E|}$, $y \in \mathbb{R}^{|E|}$, $w \in \mathbb{R}$, corresponding to the constraints in the order they appear in (2).

(iv, 2 pts) What does each constraint of the dual you derived in part (iii) mean in the setting of our path-blocking problem? Hint: the dual of the relaxed min-cut problem is called max-flow.

(v, 1 pt) Finally, how does the optimal value of the relaxed LP, $f^\star_{\text{LP}}$, compare to the optimal value of the dual, $f^\star_{\text{dual}}$?

(vi, 1 pt) Interestingly, a well-known theorem (the max-flow min-cut theorem) tells us is that the original ILP and the max flow problem have equal optimal criterion values. What does this result imply about the tightness of the convex relaxation of the ILP?

(vii, 2 pts) Consider the setting of our path-blocking problem in Figure 1. The capacities of all edges are shown in the figure, and the min-cut has been drawn. Which paths will the TAs barricade? What is the value of the max flow in this problem?

2 Conjugate fundamentals (14 points)

Recall that for any function f, convex or not, we can define its conjugate f^* as

$$f^*(y) = \max_x\ y^T x - f(x).$$

An immediate property of the conjugate is that $f(x) + f^*(y) \geq x^T y$ for all x, y. This will be helpful for the coming parts. Assume that f is closed; or for simplicity, just replace this by the assumption that $\text{dom}(f) = \mathbb{R}^n$.

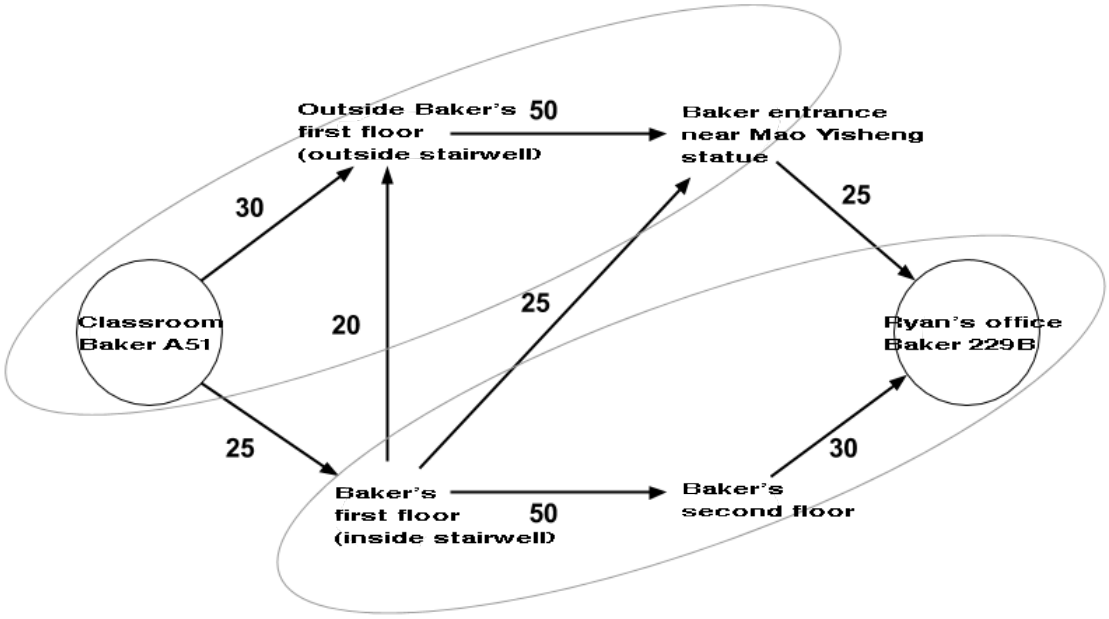

Figure 1: Min-cut of the path-blocking problem in Q1(b).

(a, 2 pts) Show that f^{**}, the conjugate of f^*, satisfies $f^{**} \leq f$.

(b, 4 pts) Prove that f^{**} is the pointwise maximum of all affine functions that underestimate f, i.e.,

$$f^{**}(x) = \max\{g(x) : g \text{ is affine}, g \leq f\}.$$

(c, 5 pts) Assuming f is convex, show that $f^{**} = f$. Hint: note that from part (b) it suffices to find, at each x, an affine underestimator g of f such that $g(x) = f(x)$. To find such a function, use the fact that f is convex, and so its epigraph $\text{epi}(f)$ is a convex set. Therefore it has a supporting hyperplane at $(x, f(x))$: there is some $(a, b) \neq 0$ such that

$$a^T x + bf(x) \leq a^T z + bt,$$

for all $(z, t) \in \text{epi}(f)$. Use this to find the desired affine underestimator with $g(x) = f(x)$.

(d, 3 pts) Again assuming that f is convex, show that

$$x \in \partial f^*(y) \iff y \in \partial f(x).$$

Hint: one direction is straightforward, recalling the max rule for subgradients. For the other direction, apply part (c).

3 Practice with conjugates and duality (16 points)

Below we go through an example of the use of conjugates for deriving to the dual of an optimization problem. Please specify the domain of the conjugate function, where appropriate.

(a, 4 pts) Derive the conjugate function of $f(x) = \log(1 + e^{-x})$.

(b, 3 pts) Derive the conjugate function of $f(\theta) = \sum_{i=1}^n \log(1 + e^{-y_i\theta_i})$, where $y_i \in \{-1, 1\}$.

(c, 3 pts) Prove that the dual of

$$\min_\theta \sum_{i=1}^n \log(1 + e^{-y_i\theta_i}) + \lambda\|D\theta\|_1, \tag{3}$$

where $D \in \mathbb{R}^{m\times n}$ is an arbitrary matrix and $y_i \in \{-1, 1\}$, is

$$\begin{aligned} \min_u \quad & \sum_{i=1}^n \Big(y_i(D^T u)_i \log(y_i(D^T u)_i) + (1 - y_i(D^T u)_i) \log(1 - y_i(D^T u)_i) \Big) \\ \text{subject to} \quad & 0 \le y_i(D^T u)_i \le 1,\ i = 1, \ldots, n,\ \|u\|_\infty \le \lambda. \end{aligned} \tag{4}$$

Hint: use the relationship between duals and conjugates, and the "shifting linear transformations" property, as covered in lecture.

(d, 3 pts) For the pair of primal and dual to the problems in part (c), write down the primal solution θ in terms of dual solution u. Write down u in terms of θ, or explain why an analytical solution is unavailable.

(e, 3 pts) Explain (for a generic D) which of the first-order methods you learned in lectures can be applied to solve (3), and separately, to solve the dual (4). Here "can be applied" means that it is implemented in practice, not in principle (meaning, if there is not a known closed-form for a subgradient or projection or proximal operator, then we will say it is not implementable).

4 Binary sequence denoising (27 points)

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

$$z_i \sim \text{Bin}(p_i), \quad \text{independently, for } i = 1, \ldots, n,$$

where the underlying probabilities $p_i \in [0, 1]$ are piecewise constant across $i = 1, \ldots, n$. An illustration is given in Figure 2.

As motivation, suppose that we work at a hospital, and are examining boxes of medicine that are being shipped to us, one at a time, across timepoints $i = 1, \ldots, n$. Let z_i be the indicator that the box at time i is defective. 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 logistic model

$$p_i = \frac{e^{\theta_i}}{1 + e^{\theta_i}}, \quad i = 1, \ldots, n,$$

and compute estimates $\hat{\theta}_i$, $i = 1, \ldots, n$ by solving:

$$\min_\theta \sum_{i=1}^n \big(- z_i\theta_i + \log(1 + e^{\theta_i}) \big) + \lambda \sum_{i=1}^{n-1} |\theta_i - \theta_{i+1}|. \tag{5}$$

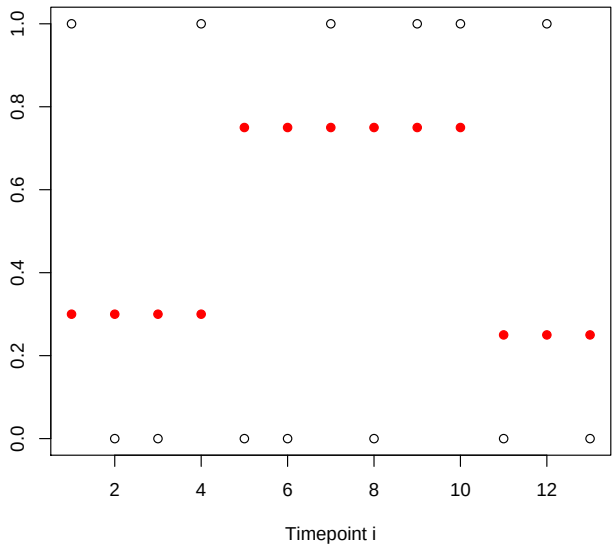

Figure 2: An illustration of binary data for Q4, where the black hollow points represent the observed data points z_i and the red solid points the underlying probabilities p_i.

This is called a logistic fused lasso problem.

(a, 10 pts) What happens if we were to try to solve (5) directly using proximal gradient descent? To do this, we would need to be able to evaluate the prox function

$$\text{prox}_t(x) = \underset{z}{\text{argmin}} \ \frac{1}{2t}\|x - z\|_2^2 + \lambda \sum_{i=1}^{n-1} |z_i - z_{i+1}|. \tag{6}$$

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, there is a beautiful dynamic programming algorithm by Nicholas Johnson (see Johnson (2013), "A dynamic programming algorithm for the fused lasso and L_0-segmentation") that gives us a direct (noniterative) method for (6), and runs in linear-time. An implementation of this algorithm in C++ is given on the course website in `prox.cpp`.

Implement proximal gradient descent to solve (5), using the prox function as provided to you in `prox.cpp`. Use backtracking to determine the step size at each iteration, and stop when the difference in criterion value across successive iterations is less than a user-specified tolerance level ϵ. Note: it might require some effort/fiddling to be able to call the C++ function in `prox.cpp` from your programming language of choice (that you are using to write the solution); but essentially every high-level programming language (e.g., Python, R, Matlab, Julia) should allow you to do this directly. For example, in R, you can first compile the C++ code by running `R CMD SHLIB prox.cpp` from your command line. This will give you the file `prox.so`. Then use the provided code in `prox.R` on the course website, to access the prox function from R. In Python, you can use the provided code in `prox.py` on the course website and call `from prox import prox_dp` (please see the doc string for usage).

After implementing it, run your algorithm on the binary data `binseq.txt` on the course website. Use $\lambda = 20$ as the tuning parameter. Start with an initial step size $t = 1$ before each backtracking loop, with $\beta = 0.8$ as the contraction factor in backtracking. Use a stopping tolerance $\epsilon = 10^{-6}$. Plot

the binary data, and on top of it, the estimated probabilities

$$\hat{p}_i = \frac{e^{\hat{\theta}_i}}{1 + e^{\hat{\theta}_i}}, \quad i = 1, \ldots, n,$$

that come from your solution $\hat{\theta}_i$, $i = 1, \ldots, n$. Report how many total iterations (sum of the number of inner backtracking iterations) your algorithm took.

(b, 2 pts) By appropriately transforming $z \in \{0, 1\}^n$ into a vector $y \in \{-1, 1\}^n$ (you should specify the transformation, but note, this is simply a relabeling), show that the loss in (5) is exactly of the form in (3). Show that the penalty in (5) is also of the form in (3) (specify the matrix D).

(c, 5 pts) Now that we have seen (5) and (3) are the same problem, we consider solving their dual (4). In order to facilitate using first-order algorithms, we will "soften" the constraints and lift them into the criterion, giving us

$$\min_u \sum_{i=1}^n \Big(y_i (D^T u)_i \log(y_i (D^T u)_i) + (1 - y_i (D^T u)_i) \log(1 - y_i (D^T u)_i) \Big) \\ - \frac{1}{\tau} \cdot \sum_{i=1}^n \Big(\log(y_i (D^T u)_i) + \log(1 - y_i (D^T u)_i) \Big) - \frac{1}{\tau} \cdot \sum_{i=1}^{n-1} \Big(\log(\lambda - u_i) + \log(u_i + \lambda)) \Big). \quad (7)$$

Here we will fix τ to be a large but positive constant, and so when the constraints are close to being violated, the second half of the criterion function (the part we added) approaches infinity. Choosing a large τ may seem like a hack, but we will learn a more principled way of doing this later when we talk about the log barrier method.

In order to run gradient descent, you will need to initialize it at a point $u^{(0)}$ that is feasible for the criterion in (7). In fact, it is helpful to find a feasible point that lies inside the domain by some "buffer", say for $\delta > 0$ (and $\delta < \min\{1, \lambda\}$), we would like to find a point u such that

$$\delta \leq y_i (D^T u)_i \leq 1 - \delta, \; i = 1, \ldots, n, \; \|u\|_\infty \leq \lambda - \delta.$$

Formulate a linear program (LP) to find such a point. Then, for y, D as in part (a) (corresponding to the binseq data in `binseq.txt` from the website), $\lambda = 20$, and $\delta = 0.01$, solve the LP (using your favorite built-in solver), and verify that your solution truly meets all the constraints.

(d, 10 pts) Now implement gradient descent with backtracking to solve (7). Again use backtracking to determine the step size at each iteration, and stop when the difference in criterion value across successive iterations is less than a user-specified tolerance level ϵ. An important note: in order to properly backtrack, you of course need to make sure that the proposed update is feasible for the criterion. The easiest way for you to do this is just to define the criterion to be infinite when a point is outside of its domain. If your code reflects this, then the backtracking exit criterion will trivially fail when the proposed update is not feasible (because the left-hand side of the desired inequality, which is the criterion value at the proposed update, will be infinite), and so the inner backtracking loop will continue until a feasible point is reached.

After implementing it, run your algorithm on the binary data `binseq.txt` on the course website. Use $\tau = 10^3$. Again use $\lambda = 20$ as the tuning parameter, $t = 1$ as the initial step size before running each backtracking loop, $\beta = 0.8$ as the contraction factor, and $\epsilon = 10^{-6}$ as the stopping tolerance. Run your algorithm for a maximum of 50,000 iterations. Use the primal-dual relationship you found in Q3(d) to get a primal solution from your dual solution, then plot the estimated probabilities $\hat{p}_i$, $i = 1, \ldots, n$ and compare to those from part (b). Are they close? Also compare the *primal* criterion values from the solutions you computed here and in part (b). Which is lower?

Hint 1: storing D as a sparse matrix should make a big difference in computational speed, since in this case multiplying by D or D^T should be much faster.

Hint 2 (so that you don't spend days trying to debug): the point here is supposed to be that dual gradient descent is very slow to converge. Our implementation reached the maximum number of iterations (it didn't achieve an $\epsilon = 10^{-6}$ difference between successive criterion values before that). This approximate solution from dual gradient descent should look like a "more curved" version of the solution reached via primal proximal gradient. If ran for more iterations, it should start to look more piecewise constant, like the primal proximal gradient solution. Newton's method, as we will see later, should do a lot better when applied to the dual here.