10-725 Optimization, Spring 2010: Homework 3

Due: Wednesday, March 17, beginning of class

**Instructions**  There are 4 questions on this assignment. The last question involves coding. Do not attach your code to the writeup. Instead, copy your implementation to 

/afs/andrew.cmu.edu/course/10/725/Submit/your_andrew_id/HW3

To write in this directory, you need a kerberos instance for andrew, or you can log into, for example, unix.andrew.cmu.edu. Please submit each problem separately with your name and userid on each problem. Refer to the webpage for policies regarding collaboration, due dates, and extensions.

1 **Convexity [Sivaraman, 15 points]**

1.1 Linear Maps between convex sets

Assume $C_1 \subset \mathbb{R}^n$ and $C_2 \subset \mathbb{R}^m$ are both convex for $n, m \in \mathbb{Z}^+$. Define $S$ to be the set of all matrices which correspond to a linear map from $C_1$ to $C_2$, i.e. $S = \{ A \in \mathbb{R}^{m \times n} : \forall x \in C_1, Ax \in C_2 \}$

1. [2 pts] Prove that $S$ is a convex set.

2. [5 pts] Let $H$ be a separation oracle for the set $C_2$. Thus, for a point $y' \in \mathbb{R}^m$, $y' \notin C_2$, $H(y')$ returns a hyperplane $\{ z \mid h(z) = 0 \}$, where the linear function $h(z)$ is defined as $h(z) = a^T z + b$, where $a, z \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The returned hyperplane $H(y')$ is such that $h(y') < 0$ and $h(y) \geq 0, \forall y \in C_2$. Note that if $y \in C_2$, $H(y) = \emptyset$.

Let $C_1$ be the convex hull of $k$ points $\{x_1, \ldots, x_k\}$ in $\mathbb{R}^n$, where $k \geq 3$. Using this definition of $C_1$ and the separation oracle $H$ for the set $C_2$, explicitly construct a separation oracle for the convex set $S$.

1.2 Convex Sets and Convex Functions

In the following, you will be asked to prove certain functions or sets are convex

1. [2 pts] Let $C \subset \mathbb{R}^n$ be a convex set, with $x_1, \ldots, x_k \in C$, and let $\theta_1, \ldots, \theta_k \in \{0\} \cup \mathbb{R}_+$ satisfy $\theta_1 + \ldots + \theta_k = 1$. Show that $\theta_1 x_1 + \ldots + \theta_k x_k \in C$. (Hint: use mathematical induction). Note that in this question, we can only use the line segment definition of convex sets as known (i.e., a set is convex iff convex combinations of any two points are still in the set.)

2. [3 pts] Define $x = (x_1, \ldots, x_n)$, show that for $p > 1$,

$$f(x, t) = \frac{|x_1|^p + \ldots + |x_n|^p}{t^{p-1}}$$

is convex on $\{(x, t) | t > 0\}$.

3. [3 pts] Suppose that $x \in \mathbb{R}^n$, show that

$$f(x) = \frac{\|Ax + b\|^2}{c^T x + d}$$

is convex on $\{x | c^T x + d > 0\}$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$.
2 Burglary via Constraint Generation [Yi, 30+3 points]

Say we’re burglars looking to rob a department store at night. There are several safes we’d like to steal money from, but we don’t want to be spotted by the cameras which have been placed around the store. We’d like to choose a clever path through the store which will minimize our chance of being spotted, and since we were CMU students before beginning our career of evil, we decide to apply our optimization skills to this problem.

We’ll phrase this path planning problem as follows: Let’s assume all paths through the store may be drawn on a grid so that there is a finite set \( S \) of states in our process. Say we have a discrete set \( A \) of actions (directions in which we can move). We’ll define a matrix \( E \) with a row for each pair (state \( s \), action \( a \)) and a column for each state \( s' \); we construct \( E \) as follows:

- Set all entries to 0.
- Set \( E((s,a),s') = 1 \) if \( a \) takes us from \( s \) to \( s' \)
- Subtract -1 from \( E(s,a),s \) for all \( s, a \)

Figure 1 shows an example; the matrix \( E \) for it is as follows:

\[
\begin{bmatrix}
  -1 & 0 \\
  -1 & 1 \\
  1 & -1 \\
  1 & -1 
\end{bmatrix}
\]

Figure 1: Example to illustrate our burglary problem.

To represent the penalty for being caught on camera during our burglary, we can write a cost vector \( c \) which has the penalty for taking action \( a \) from state \( s \) for each pair \((s,a)\). Note \( c \) is determined by where the cameras are positioned; it represents a strategy for the store security. Note: we are only concerned with minimizing the penalty from being caught on camera as we go through the store. One way to think about this is that we assume any feasible path will lead us to all safes, and thus we will get all the money anyway and the optimization problem has no concern of maximizing any other rewards. Like in Figure (1), we can imagine that there is a “finish” state with no outgoing edges, and we will finally end at this state (with money from all safes). Note that also like in Figure (1), this finish state does not correspond to a column in \( E \): we imagine such a state just for understanding that our only concern is to minimize the penalty along the path.

\[ \pi(s_1, a_2) = \frac{3}{4} \]

\[ \pi(s_2, a_1) = 1, \pi(s_2, a_2) = 0 \]

\[ \pi(s_1, a_1) = 1/4 \]

start state

Finish
Continuing with the example in Figure 1, we could have a cost vector $c$ which is e.g. $c = (2, 1, 1, 1)$, i.e., $c_{(s_1,a_1)} = 2$ and cost 1 for other three possible state-action pairs. Here we are penalized 2 for doing $a_1$ from state $s_1$, because, e.g., our entire face will likely be caught by a camera if we take action $a_1$ at state $s_1$, according to the placement of cameras in the store.

To represent our strategy for sneaking around the store, we can write down a policy $\pi$ which maps each pair (state $s$, action $a$) to the probability $\pi(s,a)$ with which we take action $a$ from state $s$. Of course, $\sum_a \pi(s,a) = 1$ for each $s$. Given $E$ and a fixed start state (e.g., a 2nd-story window with a fire escape), we can, equivalently, write a vector $f$ (indexed by state-action pairs like $\pi$) specifying how frequently we take action $a$ from each state $s$. Note that $f$ is different than $\pi$: $\pi$ tells how we make decisions (choose actions) from each state, but $f$ tells how often we end up taking each action from each state. (Think of a random walk according to $\pi$ in a world described by $E$ with steady state $f$: $f$ can have values greater than 1.) In this problem, we’ll find an optimal $f$, which we could then use to compute an optimal policy $\pi$.

To help the understanding, let’s continue with the example in Figure 1. Say we use the policy $\pi$ described in the figure; e.g. we do $a_1$ from $s_1$ with probability 1/4 (note the optimal policy would really be to always do $a_1$ from $s_1$, but let’s focus on this suboptimal one). For the given policy, we have a steady state $f$ which is: $f(s_1,a_2) = \frac{3}{4} + (\frac{3}{4})^2 + \ldots = \sum_{i=1}^{\infty} (\frac{3}{4})^i = 3$, $f(s_1,a_1) = 1$, $f(s_2,a_1) = 3$, and $f(s_2,a_2) = 0$. (We expect to do $a_2$ from $s_1$ about 3 times and to do $a_1$ from $s_1$ about 1 time.)

Finally, given a policy $\pi$ and a cost vector $c$, we can write a vector $v$ specifying, for each state $s$, the total cost we expect to incur if we start in $s$ and follow the given policy $\pi$ under a cost vector $c$.

In the example from Figure 1 with given $\pi$ in the figure and cost vector as $c = (2, 1, 1, 1)$, our expected costs are $v(s_1) = \frac{1}{2}(2) + \frac{3}{2}(1 + v(s_2))$ and $v(s_2) = 1 + v(s_1)$, which could be solved for the actual values. Note you will not actually need to compute $f$ or $v$ in this problem.

To summarize:

- $E$ (matrix with rows indexed by $(s,a)$, columns by $s'$): transition matrix with -1 added to all entries where $s = s'$
- $c$ (vector indexed by $(s,a)$): penalties
- $f$ (vector indexed by $(s,a)$): expected state-action frequencies, which actually connects to a policy $\pi$
- $v$ (vector indexed by $(s)$): expected cost of each state

2.1 Bad Security

Say the department store has bad security and we know exactly where the cameras are placed. Then we know the cost vector $c$, and we can calculate the value $v_s$ of start state $s$ using the following LP:

$$\max_c v_s \quad \text{s.t.} \quad Ev + c \geq 0 \quad (3)$$

The dual of this problem is:

$$\min_f f \cdot c \quad \text{s.t.} \quad E^T f + \bar{s} = 0, \quad f \geq 0 \quad (4)$$

where $\bar{s}$ in the dual is a vector of all zeros but with a 1 in the position for start state $s$.

1. [4 pts] What does the constraint of the primal LP mean? Write out a short (1-2 sentence) plain-English explanation as well as a mathematical expression using $v_s$ and $c$.

2. [4 pts] In the same way, give a short (1-2 sentence) plain-English and mathematical description of how we can interpret the objective and constraints of the dual LP.
2.2 Good Security

Now suppose store security has a finite (but large) set of possible locations to put cameras and may choose to place a small number of cameras at any of these locations. Each configuration corresponds to a cost vector $c$. As a burglar, we’d like to choose a strategy which will likely do well no matter how store security decides to place the cameras, so we’ll phrase the problem as follows. We have a finite set $F$ of strategies expressed as state-action frequencies for deterministic policies $\pi$. The store has a finite set $C$ of strategies (the set of all possible cost vectors determined by camera placements). Note that $F, C$ have columns $\{f_j\}$ and $\{c_j\}$, respectively. Each column $f_j$ is a vector of state-action frequencies (corresponding to a specific policy $\pi$), and each column $c_j$ is a vector of cost (corresponding to a specific placement of cameras). Like in the dual form of the previous question, for a specific choice of expected state-action frequencies $f$ (of a policy) and a cost vector $c$, our expected loss of policy is $f \cdot c$. Now, we may randomize over $F$ by choosing a vector $p$ representing a distribution over columns of $F$, and the store may likewise choose a distribution $q$ over columns of $C$. We want to find a distribution $p^*$ over policies $F$, which is optimal in the worst case (in terms of the unknown distribution $q$ over cost vectors $C$ of the store):

$$\min_{p \geq 0} \max_{q \geq 0} (Fp) \cdot (Cq) = \max_{q \geq 0} \min_{p \geq 0} (Fp) \cdot (Cq) = (Fp^*) \cdot (Cq^*)$$

where

$$(Fp) = \sum_j f_j p_j$$

$$(Cq) = \sum_j c_j q_j$$

$$(Fp) \cdot (Cq) = (Fp)^T(Cq)$$

and $\{f_j\}$ and $\{c_j\}$ are columns of $F$ and $C$.

1. [8 pts] Write down a linear program which finds the burglar’s optimal strategy $p^*$.

   **Hint:** Try interpreting the problem above as a Lagrangian. Notice that there are equality constraints on both the primal and dual variables; try eliminating one of these constraints using a Lagrange multiplier.

2. [4 pts] Derive the dual of this new LP, and show how we can use the dual’s solution to give the store’s optimal strategy $q^*$.

We now have a method for finding the best way to sneak through the store (as well as a method for finding the best places for the store to put cameras). But wait—the vector $p$ we’re optimizing over in the LP you just derived may be enormous (about $|A|^{|S|}$), and the number of possible camera placements (columns of $C$) may be too. So the LPs may take way too long to solve...

2.3 Constraint Generation

What if we have an efficient way of solving for the optimal burglar strategy $f \in F$ if we know the store’s strategy $q$, as well as an efficient way of solving for the optimal store strategy $c \in C$ if we know the burglar’s strategy $p$? Then we can use constraint generation and still have a good chance at pulling off the robbery! Call the efficient solver for $f$ given $q$ the burglar oracle and the solver for $c$ given $p$ the store oracle.

We’ll maintain a set of constraints $\tilde{F}$ for the burglar strategies; $\tilde{F}$ will be a matrix with columns $f_j \in F$. Likewise, we’ll maintain a set of constraints $\tilde{C}$ for the store where the columns of $\tilde{C}$ are cost vectors $c \in C$. We’ll add more constraints on each iteration, computing closer and closer approximations to the optimal strategies. Here’s the algorithm:

- Initialize $\tilde{F}, \tilde{C}$ with arbitrary columns.
- For $i = 1, 2, \ldots$
Given that we limit the burglar to strategies in $\tilde{F}$ (instead of all possible strategies $F$) and limit the store to $\tilde{C}$, find the optimal strategies $p_i$ and $q_i$. (We can do this using your answer from 2.2 if we replace $F, C$ with $\tilde{F}, \tilde{C}$.)

Assume the store will use the strategy $q_i$, and use the burglar oracle to find the optimal strategy $f_i \in F$. Add $f_i$ to $\tilde{F}$.

Assume the burglar will use strategy $p_i$, and use the store oracle to find the optimal strategy $c_i \in C$. Add $c_i$ to $\tilde{C}$.

Compute lower and upper bounds $v_l$ and $v_u$ on the expected cost incurred by the burglar $v_s$.

If $f_i$ and $c_i$ were already in $\tilde{F}$ and $\tilde{C}$, respectively, or if $v_l$ and $v_u$ are close enough, then return $p_i$ and $q_i$ as optimal strategies.

1. [5 pts] How can we find the lower and upper bounds $v_l$ and $v_u$? Prove your answer.

2. [5 pts] Prove that the algorithm terminates. Prove that the strategies it returns are optimal if it terminates because $f_i, c_i$ were already in $\tilde{F}, \tilde{C}$.

3. (Extra Credit) [3 pts] How could we implement the burglar oracle? Write down an algorithm for this. (If you make use of a common algorithm, there’s no need to write it out as long as you are specific about how you use it.)

## Subgradients 101 [Yi, 20 points]

Since we will have fun with a subgradient programming question later, here we first make sure that everybody loves subgradients! We achieve this by examining some basic properties that show the usefulness of subgradients. In the class we’ve learned the definition of subgradients. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. We say that a vector $g \in \mathbb{R}^n$ is a subgradient of $f$ at a point $x \in \mathbb{R}^n$ iff

$$f(z) \geq f(x) + g^T(z - x), \quad \forall z \in \mathbb{R}^n$$

Also, the set of all subgradients of $f$ at $x \in \mathbb{R}^n$ is called the subdifferential of $f$ at $x$, denoted by $\partial f(x)$.

### 3.1 Existence (7 points)

For subgradients to be useful, we first need to make sure the existence of them. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Prove that the subdifferential $\partial f(x)$ is nonempty for any $x \in \mathbb{R}^n$. (Hint: supporting hyperplane theorem!)

### 3.2 Additivity (6 points)

In many cases, the objective function we want to optimize is the sum of several sub-functions (e.g., regularization approaches in machine learning, such as Support Vector Machines in the primal form, require optimizing the sum of empirical loss and regularization penalty). Prove the following theorem, which shows the convenience of defining subdifferential (subgradients) in such cases. Let $f_j : \mathbb{R}^n \to \mathbb{R}, j = 1, 2, \ldots, m$ be $m$ convex functions, and let $f = f_1 + \cdots + f_m$. Then for any $x \in \mathbb{R}^n$,

$$\partial f(x) = \partial f_1(x) + \cdots + \partial f_m(x)$$

Recall that the sum of sets is defined as $A_1 + \cdots + A_m = \{x \mid x = x_1 + \cdots + x_m, \exists x_1 \in A_1, \ldots, x_m \in A_m\}$. 

### 3.3 Optimality (7 points)

We have shown that subgradients exist for any convex function, and their additivity property is convenient. The last thing we want to show is the optimality of using subgradients to minimize a convex function. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function, $S$ be a convex set, and consider the following convex optimization problem:

$$\min_{x \in S} f(x)$$
Prove that a point \( x^* \in \mathbb{R}^n \) is the global minimizer of the problem if and only if \( \exists g \in \partial f(x^*) \) such that \( g^T (x - x^*) \geq 0 \) for any \( x \in S \).

4 Robust Least Squares [Sivaraman, 35 + 11 points]

Let us consider the robust weighted least squares linear regression problem.

\[
\max \alpha \min w \sum_{j=1}^{n} \alpha_j (y_j - X_j w)^2 \tag{5}
\]

subject to \( \alpha \geq 0 \) \hspace{1cm} \tag{6}

\[
\sum_{i=1}^{n} \alpha_i = 1 \tag{7}
\]

Where \( y_j \) is the regressand for the \( j^{th} \) data point and \( X_j \) is a vector of regressors. \( w \) is a vector of weights (one for each regressor).

In this question we will explore two ways to solve this problem.

4.1 Datasets

For most of this question you will work with a synthetic dataset. Download this data from the website and use the following command:

```
load data.mat
```

You will also use the Florida 2K U.S. election dataset. This dataset is already a part of Matlab, and you can load it using this command:

```
load flvote2k
```

4.2 Solution using Constraint Generation

This part of the question requires you to implement robust weighted least squares regression using constraint generation.

1. [12 pts] Implement robust weighted least squares regression using constraint generation. Keep track of the upper and lower bounds on the optimal solution (as discussed in class) through the iterations (your stopping criterion should be to stop when the gap between the upper and lower bounds is smaller than \( 10^{-3} \)). Plot the upper and lower bounds.

Hint 1: The separation/violation oracle, which is a weighted least squares regression problem, has the following closed-form solution.

If we have a weighted least square regression problem with positive weights \( \alpha_1, \ldots, \alpha_n \),

\[
\hat{w} = \arg \min_w \sum_{i=1}^{n} \alpha_i (y_i - X_i w)^2 \tag{8}
\]

the closed form solution is given by:

\[
w = (X^T \text{diag}(\alpha) X)^{-1} X^T \text{diag}(\alpha) y \tag{9}
\]

Where \( \text{diag}(\alpha) \) is a diagonal matrix (all non-diagonal entries are non-zero) with \( \alpha_j \) as the \( j^{th} \) diagonal element. \( X \) is a matrix which has \( X_j \) as its \( j^{th} \) row.

Hint 2: You can use the Matlab command “lscov” to do this.

Hint 3: Your implementation will require the use of an LP solver. I recommend using “linprog”, with the “Simplex” option turned on.
4.3 Projected Supergradients

Now we will explore a different method of solving the robust weighted least squares regression problem, using a projected supergradient algorithm. Supergradients are used in the maximization of concave functions. A supergradient of a function is defined as the negative of the subgradient of the negative of that function, and is a way to extend gradients to non-smooth concave functions.

Consider the case when you have an objective \( c(\alpha) \) that you are maximizing w.r.t \( \alpha \), subject to constraints \( f(\alpha) \leq 0 \). A typical supergradient ascent algorithm proceeds as follows

1. \( t \leftarrow 1 \)
2. \( \alpha \leftarrow \alpha_0 \)
3. while \( t \leq T \) do
   - Compute \( g \in \partial c(\alpha) \)
   - Update \( \alpha \leftarrow \alpha + \eta_t g \)
   - Project \( \alpha \) onto the constraints \( f(\alpha) \leq 0 \)
   - Check convergence
   - \( t \leftarrow t + 1 \)
end while

where \( \alpha_0 \) is a suitable feasible initialization.

1. [3 pts] Consider the original program

\[
\max_{\alpha} \min_w \sum_{j=1}^{n} \alpha_j (y_j - X_j w)^2 \tag{10}
\]
subject to
\[
\alpha \geq 0 \quad \tag{11}
\]
\[
\sum_{i=1}^{n} \alpha_i = 1 \tag{12}
\]

Show that the superdifferential of the objective w.r.t \( \alpha \) has the following simple form:

\[
g(\alpha) = (y - X w_\alpha)^2
\]

where \( w_\alpha = \arg \min_w \sum_{j=1}^{n} \alpha_j (y_j - X_j w)^2 \)

2. [5 pts] Now, we will consider accounting for the constraints in the original program using Euclidean projections. Given a \( \hat{\alpha} \) that does not satisfy the constraints, write a projection routine that solves the following QP.

\[
\min_{\alpha} (\alpha - \hat{\alpha})^T (\alpha - \hat{\alpha})
\]
subject to
\[
\sum_{j=1}^{n} \alpha_j = 1
\]
and \( \alpha \geq 0 \)

Hint: You could use Matlab’s “quadprog”. This shouldn’t be more than a few lines of code.

3. [5 pts Extra Credit] In practice gradient projection algorithms are a good idea only if we can solve the projection problem quickly. Consider the following algorithm:
for \( k = \{1,2,\ldots,n\} \)
Set the \( k-1 \) smallest values of \( \alpha \) to 0.
Set the *other* entries to, \( \alpha_j = \hat{\alpha}_j + \lambda \) where \( \lambda \) is chosen so that the sum of entries of \( \alpha \) is 1.
If all entries of \( \alpha \geq 0 \) break.
end for

Show, that the algorithm above computes the Euclidean projection onto the convex set defined by the constraints \( \alpha \geq 0, \sum_{i=1}^{n} \alpha_i = 1 \) efficiently and exactly.

Hint 1: Introduce a Lagrange multiplier for the equality constraint. Argue that if you knew the value of this Lagrange multiplier at optimum, then the update equation \( \alpha_j = \max(0, \hat{\alpha}_j + \lambda) \) is correct.

Hint 2: Show that the search procedure described above finds the value of the Lagrange multiplier at the optimum.

4. [5 pts] Now, we will put this all together. Implement the supergradient ascent algorithm described above, using the following set of parameters.

\[
\eta_t = \frac{k}{\sqrt{t}}
\]
where \( k \) is suitably chosen to ensure convergence in a small number of iterations.

Note: \( k = 0.0001 \), works in my implementation, but since this depends on how you initialize \( \alpha \) you might have to play around with it.

Check convergence, by comparing the values of \( \alpha \) in successive iterations. Stop when the \( L_2 \)-norm of the difference vector (between successive \( \alpha \)s) is smaller than \( 10^{-3} \). Use \( T = 200 \) just in case (you shouldn’t need this). When the algorithm converges, you can use the optimal \( \alpha \)s to find the optimal \( w \)s.

5. [3 pts Extra Credit] Try different learning rate schedules and comment on the differences in performance. You could try,
   (a) \( \eta_t = \frac{k}{\sqrt{t}} \), for different values of \( k \).
   (b) \( \eta_t = \frac{k}{t} \), for different values of \( k \).
   (c) Also, you could try the geometric series from an earlier version of this homework, \( \eta_t = \frac{\eta_1}{1-t} \), with \( \eta_1 = 1 \).

4.4 Comparisons

1. [2 pts] Compare the two algorithms in terms of the solutions they return. Are the \( \alpha \)s, and \( w \)s exactly the same? Are they similar? If they aren’t exactly the same, which algorithm achieves a better objective value at OPT? Comment on your observations.

2. [3 pts] Compare the two algorithms with ordinary least squares regression. Particularly, define an unweighted score and a worst case score as follows. The worst case score is defined as

\[
S_W = \max_{\alpha} \sum_{i=1}^{n} \alpha_i (y_i - X_i w)^2 \quad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i = 1 \quad \alpha_i \geq 0, \, i = 1, \ldots, n
\]

while the unweighted least square score is defined as

\[
S_U = \frac{1}{n} \sum_{i=1}^{n} (y_i - X_i w)^2
\]

Report the unweighted and worst case scores for the three algorithms. What do you observe?
3. [3 pts Extra Credit] You could also define an average case score, which is

\[ S_A = \int \sum_{j=1}^{n} \alpha_j \geq 0, \sum_{i=1}^{n} \alpha_i (Y_i - X_i w)^2 \, d\alpha_1 \ldots d\alpha_n \]  

(15)

How would you compute this? Comment about the average case scores of the least squares and robust least squares methods.

4.5 The butterfly ballot and outliers

1. [5 pts] In the 2000 election for US president, the counting of votes in Florida was controversial. In Palm Beach county in South Florida, for example, voters used a so-called butterfly ballot. Some believe that the layout of the ballot caused some voters to cast votes for Buchanan when their intended choice was Gore.

Look at http://www.mathworks.com/access/helpdesk/help/toolbox/curvefit/excludedata.html, to see an example of the kind of analysis you can do with this data.

Load the Florida election dataset. You will see 4 variables, County, the county name, and Gore, Bush, and Buchanan, which are the number of votes for each of these three candidates.

Now, define the Gore and Bush variables to be your regressors, and the Buchanan variable to be the regressand. Run the two robust least squares algorithms on the data.

Plot a graph of the worst case weights \( \alpha \). Typically these are maximized at “outliers” in your regression. Do you see any outliers in the data? List the three counties which have the highest weights \( \alpha \) for both robust least squares algorithms.

Confirm that these are actually outliers by plotting the residuals at each data point from an ordinary least squares fit. What do you observe?