First-order Methods on Large-scale Convex Optimization Problems

Qingxuan Jiang

Abstract

In this report, we summarize some recent advances in first-order methods for large-scale convex optimization algorithms, including the computation tree method, the restart method, and the perceptron method. As a case study, we implement the restart method on the the linear programming feasibility problem and observe a reduction of number of iterations to around one-half of the usual subgradient methods. We also perform experiments on influence of matrix shape, scale and sparsity on the performance of the restart method.

1 Background

1.1 Subgradient Method on Linear Programming

We first consider the simple linear programming problem:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b \end{aligned}$$

where we can represent $A = \begin{bmatrix} — & a_1^\top & — \\ & \vdots & \\ — & a_m^\top & — \end{bmatrix} \in \mathbb{R}^{m\times n}$, $b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m$, $c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \in \mathbb{R}^n$.

Assume that the constraint set has a non-empty interior, and we are given one interior point $e \in \mathbb{R}^n$ s.t. $Ae \geq b$.

The paper [5] gives a nice framework of applying the subgradient method to this linear programming problem. This can be summarized in the following steps:

(1) **Defining a "descent goal"**: We fix some scalar z with $z < c^\top e$. (This represents a better objective value that we would like to descend to.)

(2) **Projection**: For any point $x \in \mathbb{R}^n$ with $c^\top x = z$ having the descent goal as objective value, we project e in the direction $x - e$ onto the boundary $\pi(x)$. We then define $\gamma(x)$ s.t. $\pi(x) = e + \frac{1}{\gamma(x)}(x - e)$.
From properties of projection, we know that γ is convex and Lipschitz continuous, with Lipschitz constant δ equal to the distance from e to $\{x \in \text{boundary} : c^\top x = c^\top e\}$. Also, if the projection never intersects the boundary, then we define $\gamma(x) = -\infty$.

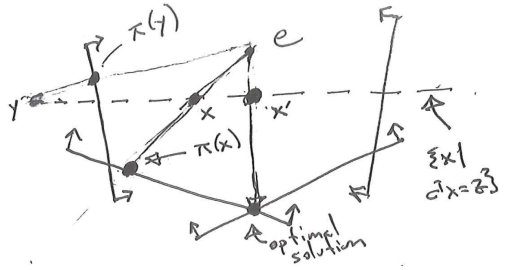

Figure 1: Illustration of projection. The dashed line represents the constraint $c^\top x = z$, and for each x we project e in the direction of x toward the boundary of the constraint set, and we denote $\pi(x)$ to be the projected point.

(3) **Minimization condition**: We know that if we pick some point $r = e + \lambda(x - e)$, then $c^\top r = c^\top e + \lambda(z - c^\top e)$ with both $c^\top e$ and z fixed values and $z - c^\top e > 0$. Thus, minimizing the objective function would be equivalent to picking λ as large as possible. For fixed x, the largest feasible λ is achieved when we move to $\pi(x)$, with $\lambda = \frac{1}{\gamma(x)}$. Thus, we see that minimizing the objective function is equivalent to minimizing $\gamma(x)$ over all choices of $x \in \mathbb{R}^n$ with $c^\top x = z$:

$$\begin{array}{ll} \min\limits_x & c^\top x \\ \text{s.t.} & Ax \geq b \end{array} \iff \begin{array}{ll} \min\limits_x & \gamma(x) \\ \text{s.t.} & c^\top x = z \end{array}$$

(4) **Subgradient computation**: To compute the subgradient, we first identify the actual boundary constraint we're faced with, i.e. we find one index $j \in \{1, 2, \ldots, m\}$ s.t. $a_j^\top \pi(x) = b_j$. Then we can pick the subgradient $g(x) = \frac{1}{a_j^\top(\pi(x) - e)} a_j$. Note that this index might not be unique, in which case we just need to pick an arbitrary one of them.

(5) **Subgradient algorithm**: Once we have the subgradient, we can perform either of the following methods for optimization:
(5-1) **Subgradient method**: $x_{k+1} = x_k - \frac{\epsilon}{\|g(x_k)\|^2} g(x_k)$.
(5-2) **Projected subgradient method**: $x_{k+1} = x_k - \frac{\epsilon}{\|g(x_k)\|^2} P_C(g(x_k))$, where $P_C(g(x_k)) = g(x_k) - \frac{c^\top g(x_k)}{\|c\|^2} c$ is the orthogonal projection of $g(x_k)$ onto $\{x : c^\top x = 0\}$.

1.2 Computation Trees on Subgradient Methods

One of the recent advances for accelerating subgradient methods on large-scale optimization problems is the computational tree methods introduced in [4]. The method incorporates sparsity assumptions for large-scale optimization problems to split the variables into several groups, and then arrange the groups in a tree-like structure to speed up and parallelize computation.

Problem: Consider the convex optimization problem $\min\limits_{x \in \mathbb{R}^n} f(x)$, where $f(x) = \max\{f_1(x), \cdots, f_M(x)\}$ is the maximum of several convex components $f_i(x)$ for $i = 1, \ldots M$.

Approach: The method can be summarized in the following steps:
(1) **Splitting of variables**: We will split the variables $x = \begin{bmatrix} x^{[1]} \\ \vdots \\ x^{[I]} \end{bmatrix}$ into I "groups", where each $x^{[i]} = \begin{bmatrix} x_1^{[i]} \\ \vdots \\ x_{j_i}^{[i]} \end{bmatrix}$ represent variables in one certain group for $1 \leq i \leq I$. We then impose the following sparsity conditions:

Assumption 1: For each $1 \leq i \leq I$, the variables in the group $x^{[i]}$ appear in only a small number of functions f_j.

Assumption 2: For each $1 \leq m \leq M$, the function f_m involves variables from only a small number of groups of variables $x^{[i]}$.

(2) **Arranging the computation tree**: WLOG we consider the simple case where the splitting of variables above produce a binary tree. Let $\overline{x}$ be the current iterate. The following figure illustrates the bottom layer of the tree: for each root, we choose the index j such that $f_j(\overline{x})$ gives the maximum value among all branches f_m below this root. We then propagate this process up the tree until we get to the top node.
(3) **Finding a subgradient on the computation tree**: To find a subgradient $\overline{g}$ for f at $\overline{x}$, the above step shows that we can first find the index j such that f_j gives the maximum value. Then the subgradient of f is just any subgradient of f_j at $\overline{x}$. This is easy to compute as f_j involves "few" variables from Assumption 2 above.

(4) **Subgradient update on the computation tree**: Finally, we consider the subgradient method update $\overline{x}_+ = \overline{x} - \frac{\epsilon}{\|\overline{g}\|} \overline{g}$. Note that $\overline{x}_+$ differs from $\overline{x}$ in only a few variables (the ones that f_j rely on in Step 3), and each of these variables appear in only a few functions. If we assume that the updated variables only appear in a constant number of functions f_i's, then tree can be updated in $O(\log M)$ time (by making this update on a constant number of branches for the tree).

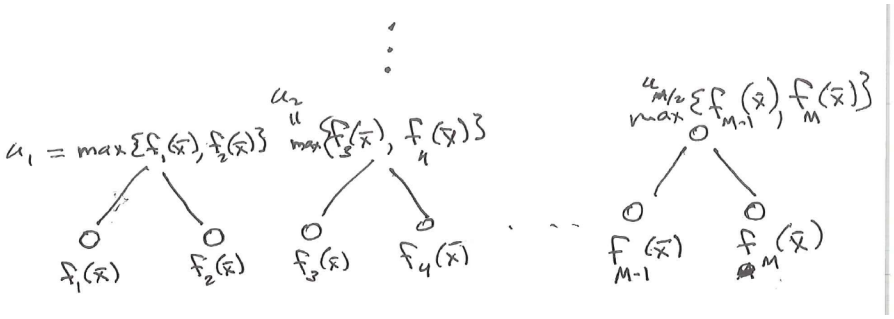

Figure 2: Illustration of Computation Tree

2 Subgradient Method on Convex Feasibility Problem

In this section, we state the convex feasibility problem that we are mainly concerned with in this paper.

Inputs: We're given m closed convex sets $Q_1, \cdots, Q_M \subseteq \mathbb{R}^n$, and for each Q_i we're given one of its interior points $e_i \in \mathbb{R}^n$. (Here M is assumed to be large, so that this is a large-scale problem.)

Goal: We want to find a point in the intersection $\bigcap_{i=1}^{M} Q_i$, assuming that it is non-empty.

Approach: We consider combining the Projection approach in Section 1.1 and the Computation Tree approach in Section 1.2, using the following steps:

(1) **Projection**: For some iterate x and any $1 \leq i \leq M$, we project e_i in the direction of $x - e_i$ onto the boundary point $\pi_i(x)$ of the convex set Q_i. We then define $\gamma_i(x)$ to be the coefficient satisfying $\pi_i(x) = e_i + \frac{1}{\gamma_i(x)}(x - e_i)$. If the projection never intersects the boundary, we define $\gamma_i(x) = -\infty$.

Recall from previous discussion in Section 1.1 that $\gamma_i(x)$ is Lipschitz continuous with Lipschitz constant $\delta_i = \frac{1}{r_i}$ where r_i is the radius of the largest ball centered at e_i and contained in Q_i.

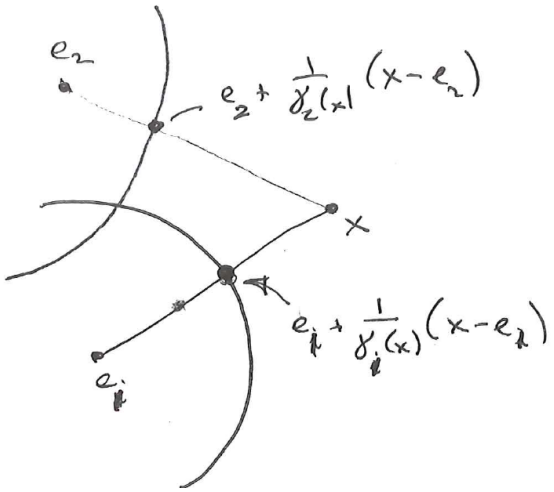

Figure 3: Illustration of Projection on Convex Sets

(2) **Goal condition**: Since we have $\gamma_i(x) \leq 1$ if and only if $x \in Q_i$, our goal is to find x s.t. $\gamma_i(x) \leq 1$ for all $1 \leq i \leq M$. Thus, if we let $\gamma(x) = \max\{\gamma_1(x), \cdots, \gamma_M(x)\}$, then the goal would be to find x s.t. $\gamma(x) \leq 1$.

(3) **Evaluation of subgradient method**: We consider the above problem as $\min \gamma(x)$ where $\gamma(x) = \max\{\gamma_1(x), \cdots, \gamma_M(x)\}$. For some iterate $\overline{x}$, we can choose a j such that $\gamma_j(\overline{x})$ gives the maximum value among all γ_m with $1 \leq m \leq M$.

Then we can find a subgradient $\overline{g}$ for γ at $\overline{x}$ by finding any subgradient of γ_j at $\overline{x}$. The subgradient method would

give the update $\overline{x}_+ = \overline{x} - \frac{\epsilon}{\|\overline{g}\|}\overline{g}$.

Also note that if we have sparsity assumptions on our problem, then we can apply the Computation Tree Method in section 1.2 to this step to speed up and parallelize the computation.

3 Example: Linear Programming Feasibility Problem

In this section, we apply the subgradient method to the linear programming feasibility problem, where we take each convex set in the above framework as a half-space.

3.1 Problem Statement and Approach

As an simple illustration of our algorithm, we first discuss how we may apply this algorithm to the linear programming feasibility problem, i.e. finding a feasible solution to $Ax \geq b$, given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. We summarize the experiment settings in the following steps:

(1) **Generation of half-spaces** Q_i: Let us denote $A = \begin{bmatrix} — & a_1^\top & — \\ & \vdots & \\ — & a_m^\top & — \end{bmatrix}$ and $b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$. Then we can rewrite the linear programming feasibility problem in terms of our framework, by letting each $Q_i = \{a_i^\top x \geq b_i\}$ represent a half-space for each $1 \leq i \leq m$.

(2) **Generation of initial interior point** e_i: For our specific setup of convex sets as half-spaces, we let $e_i = \lambda_i a_i$. In this case, any $\lambda_i \geq \frac{b_i}{\|a_i\|_2^2}$ would suffice, though we need to choose a suitable λ_i. There is a balance involved in the size of λ_i: having λ_i too small would result in very large Lipschitz constant, while a very large λ_i would make our interior far from the intersection set, and both factors may result in slower convergence.
For simplicity, we choose the λ_i such that the Lipschitz constant is 1. In this case we should have $\frac{a_i^\top(\lambda_i a_i)-b_i}{\|a_i\|^2} = 1$, and this implies $\lambda_i = \frac{b_i+\|a_i\|}{\|a_i\|^2}$. Thus, we'll pick $e_i = \frac{b_i+\|a_i\|}{\|a_i\|^2}a_i$ as the initial points. (Alternatively, we can also think of this as choosing the point e_i such that the distance from e_i to the constraint $a_i^\top x - b_i \geq 0$ is 1.)

(3) **Initialization of** x_0: $x_0 \in \mathbb{R}^n$ is initialized randomly. In the algorithm we would first check if x_0 already satisfies the condition. If not, we would proceed with the following iterations.

(4) **Iteration Step 1: Evaluate** γ_i **and find one index** i **with** $\gamma(x_k) = \gamma_i(x_k)$:
To evaluate γ, we recall that $\pi_i(x_k) = e_i + \frac{1}{\gamma_i(x_k)}(x - e_i)$. In our case, this implies $a_i^\top(e_i + \frac{1}{\gamma_i(x)}(x - e_i)) = b_i$, so we have $\frac{1}{\gamma_i(x_k)} = \frac{b_i - a_i^\top e_i}{a_i^\top(x_k - e_i)}$, and $\gamma_i(x_k) = \frac{a_i^\top(x_k - e_i)}{b_i - a_i^\top e_i} = \frac{a_i^\top x_k - b_i}{b_i - a_i^\top e_i} + 1$.

To combine the evaluation of all $\gamma_i(x_k)$ into matrix operations, we denote $\Gamma(x_k) = \begin{bmatrix} \gamma_1(x_k) \\ \vdots \\ \gamma_M(x_k) \end{bmatrix}$. From the previous expression, we can write $\Gamma(x_k) = (Ax_k - b)./(b - diag(Ae)) + 1$, where ./ represent pointwise division. Also, note that we can pre-compute $b - diag(Ae)$ since it is a constant throughout all iterations.

Then we find the maximal of such $\gamma_i(x_k)$ and let i be one such index.

(5) **Iteration Step 2: Find the subgradient of** γ_i **at** x_k: The subgradient is given by $g_i(x_k) = \frac{1}{a_i^\top(\pi_i(x_k) - e_i)}a_i$, where $\pi_i(x_k) = e_i + \frac{1}{\gamma_i(x_k)}(x_k - e_i) = e_i + \frac{b_i - a_i^\top e_i}{a_i^\top(x_k - e_i)}(x_k - e_i)$.

(6) **Iteration Step 3: Update step of the subgradient**: We perform the update $x_{k+1} = x_k - \alpha_k g_k$. As known from properties of subgradient algorithms, we know that any choice of α_k with $\sum_{k=1}^{\infty} \alpha_k = \infty$ would give convergence. This includes the following choices:
(6-1) **Fixed step size**: $\alpha_k = \frac{\epsilon}{\|g(x_k)\|_2^2}$.

(6-2) **Harmonic series**: $\alpha_k = \frac{1}{k}$.
(6-3) **Polyak's step size**: $\alpha_k = \frac{f(x_k)-f^*}{\|g_k\|^2}$, in cases where we know the optimal value f^*.

3.2 Simplified version of entire algorithm

The above steps are summarized in Algorithm 3.2.

Algorithm 1 Algorithm for finding interior points in the union of half-planes

Input: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ refer to the m half planes in $\mathbb{R}^n$, α_k is the step size factor for the k-th iteration
Initialize: $e_i = \frac{b_i+\|a_i\|}{\|a_i\|^2}a_i$ for $i = 1, \cdots, m$; $x \in \mathbb{R}^n$ is picked randomly; $C = b - diag(Ae)$
Algorithm:
for $k = 1, 2, \cdots$
    $\Gamma = (A * x - b)./C + 1$
    $[\gamma, j] = \max\{\Gamma(1), \cdots, \Gamma(n)\}$
    $\pi = e_j + \frac{b_j - a_j^\top e_j}{a_j^\top (x-e_j)}(x - e_j)$
    $g = \frac{1}{a_j^\top(\pi - e_j)}a_j$
    $x = x - \alpha_k * g$
    check convergence
end

3.3 Practical Considerations and Implementation

Implementation of the above method can be found in the Github Repository.[1] Here we note several implementation details not described above:

- Without loss of generality, we will assume that each $b_i < 0$, which guarantees that 0 is one solution inside the intersection. (Of course, this information is not given to the algorithm itself, and it's only to ensure that we have a nonempty intersection.)
- To apply the step size for Polyak's rule, we need to know the optimal value for $\min_x \gamma(x)$, which is not known for this problem. Instead, we will use $\gamma(0)$ as an approximation to the optimal value. This guarantees that the algorithm would converge to some $\gamma(x^*) < \gamma(0) \leq 1$, which is good enough for the purpose of finding a point in the intersection. (Note that this also requires us to give the information $\gamma(0)$ to the algorithm, which is typically not available. In reality, we would just pick 1 as the approximation, perform Polyak's step for some iterations, and then switch to the fixed-step algorithm for small ϵ.)

We also recall results from convergence theorems for subgradient methods:

- For the fixed step size $\alpha_k = \frac{\epsilon}{\|g(x_k)\|_2^2}$, the algorithm is guaranteed to reach within ϵ of optimal value in $\left(\frac{M \cdot dist(x_0, X^*)}{\epsilon}\right)^2$ steps, given that f is M-Lipschitz.
- For Polyak's step size, the algorithm is guaranteed to reach within ϵ of optimal value in $2 \cdot \left(\frac{M}{\mu}\right)^2 \log_2\left(\frac{M \cdot dist(x_0, X^*)}{\epsilon}\right)$ steps, given that f is M-Lipschitz, and f possesses the linear growth condition $f(x) - f^* \geq \mu \cdot dist(x, X^*)$ for some $\mu > 0$.
- The choice of $\alpha_k = \frac{1}{k}$ and other rules with $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$ only guarantees convergence, while there is no known convergence bounds for such cases.

Our experiments show that Polyak's step size converges in the least steps, followed by the harmonic step size and the fixed step size. This is expected as Polyak's step size guarantees an additional log factor that significantly reduces the steps needed.

[1] Github Repository: https://github.com/mathreader/first-order-method-feasibility

3.4 Extension to Equality Constraints

Now that we've solved the linear programming feasibility problem $Ax \geq b$ with inequalities, it is natural to think about how to incorporate equality constraints also in the problem.

The mathematical formulation is that we want to find a point in the region $\begin{cases} Ax \geq b \\ Mx = d \end{cases}$ for $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, $M \in \mathbb{R}^{l \times n}$, $d \in \mathbb{R}^l$ $(l \leq n)$. This corresponds to the original half-space intersection problem plus additional conditions that require the points to be in a specific subspace. For simplicity, we assume that M has full row rank, i.e. $rank(M) = l$.

(1) **Does changing equality to inequality work?**: Maybe the most natural idea we can think about is to directly change equality constraints into inequality constraints, thus solving $\begin{bmatrix} A \\ M \\ -M \end{bmatrix} x \geq \begin{bmatrix} b \\ d \\ -d \end{bmatrix}$.
However, if we think more deeply into how the subgradient method works, we will find that the method may be problematic, since the updates from the subgradient method may cause our iterations to fall off the subspace $Mx = d$. The takeaway is that we need to have some kind of projection involved so that we can keep our iterates on the subspace $Mx = d$.

(2) **Approach (Projected subgradient method)**: From our motivations above, we will consider applying the projected subgradient method.

Let P be the orthogonal projection onto the null space of M, and suppose that the initial point satisfy $Mx_0 = d$. Then the prjected subgradient method has the update $x_{k+1} = x_k - \frac{\epsilon}{\|P(g_k)\|^2} P(g_k)$.

We also note that this procedure is equivalent to the usual subgradient method, if we perform a change of basis from the standard basis to $v_1, \cdots, v_n$ where $v_1, \cdots, v_l$ represent a basis of the subspace, and apply subgradient method in this alternate coordinates.

To actually compute the projection, we note that for matrix $M \in \mathbb{R}^{l \times n}$ with full column rank, we have $range(M^\top)$ orthogonal to $null(M)$. The projection matrix of $x \in \mathbb{R}^n$ onto $range(M^\top)$ is given by $P = M^\top (MM^\top)^{-1} M$, and thus the projection onto its orthogonal complement $null(M)$ is $P' = I - P = I - M^\top (MM^\top)^{-1} M$.

The time complexity of computing the projection matrix is $O(2ln^2 + l^2 n + l^3)$, which is cubic, though it is only done as a pre-processing step at the beginning, so it won't affect the complexity of the iterations.

3.5 Convergence of Subgradient Method on Convex Feasibility Problem

In this section, we provide an argument that shows the convergence of such subgradient methods on general convex feasibility problems. We denote $Q = \bigcap_{i=1}^{M} Q_i$ be the feasible set.
(1) **Minimum $\overline{x}$ of maximal distance** $D(x)$: We define $D(x) = \max_{i=1,2,\cdots,M} \|x - e_i\|$. This represents the distance from x to the furthest e_i. We let $\overline{x} = \arg\min_{x \in Q} D(x)$ be the optimal point that minimizes the maximal distance $D(x)$ in Q.

(2) **Maximal radius $r(x)$ of fitted ball**: For any $x \in Q$, we define $r(x) = \sup\{r | B(x, r) \subseteq Q\}$. This represents the maximal radius of the ball with center x that is contained entirely in the intersection. [$r(x)$ can approach infinity if Q is unbounded.]

(3) **Proportion $\alpha(x)$ of radius and distance**: For $x \in Q$, we define $\alpha(x) = \frac{r(x)}{D(x)}$. This represents the proportion between maximal radius within Q and maximal distance to e_i. Let $\alpha^* = \sup_{x \in Q} \alpha(x)$ be the maximal proportion that can be attained.
(3-1) If we assume $e_i \notin Q$ for all i, then for all interior points of bounded Q, we must have $0 < \alpha(x) < 1$. For boundary points we have $\alpha(x) = 0$.

In fact, even if Q is unbounded, we still have $0 < \alpha(x) < 1$. The limit can only approach 1 if the constraints are parallel (and feasible region in the same direction) for LP problems.
(3-2) If we allow the e_i to be on the boundary, then we have $0 < \alpha(x) \leq 1$. All cases are similar to the above, with an additional case of $\alpha(x) = 1$ being when the maximal circle at x is tangent to at least one boundary point of each Q_i with all e_i lying on the tangent points.

(4) **Relation between $\alpha(x)$ and $\gamma(x)$**: We have $\|x - e_i\| \leq \max_i \|x - e_i\| = D(x)$ and $\|\pi_i(x) - x\| \geq r(x)$. Thus,

$$\gamma_i(x) = \frac{\|x - e_i\|}{\|\pi_i(x) - e_i\|} = \frac{\|x - e_i\|}{\|x - e_i\| + \|\pi_i(x) - x\|} = \frac{1}{1 + \frac{\|\pi_i(x) - x\|}{\|x - e_i\|}} \leq \frac{1}{1 + \frac{r(x)}{D(x)}} = \frac{1}{1 + \alpha(x)}$$

Thus, we must have $\gamma(x) = \min_i \gamma_i(x) \leq \frac{1}{1+\alpha(x)}$.

(5) **Key proposition regarding "best approximation point"**: We can show that $\exists x \in Q$ s.t. $D(x) \leq 2D(\overline{x})$ and $\alpha(x) \geq \frac{1}{2}\alpha^*$. Combining this with (4) gives us $\exists x \in Q$ s.t. $D(x) \leq 2D(\overline{x})$ and $\gamma(x) \leq \frac{1}{1+\alpha(x)} \leq \frac{1}{1+\frac{1}{2}\alpha^*} < 1$ (since $0 < \alpha^* < 1$).

(6) **Preliminaries to the convergence theorem**:
Let $\gamma' = \frac{1}{1+\frac{1}{2}\alpha^*}$ be an upper bound on the optimal value of $\gamma(x)$.
We define $X^*(\gamma') = \{x | \gamma(x) \leq \gamma'\}$ to be the set of points having optimal value at most γ'.
Let $dist(x_0, X^*(\gamma')) = \min_{x^* \in X^*(\gamma')} \|x_0 - x^*\|$ be the minimal distance from initial point x_0 to the set $X^*(\gamma')$.

(7) **Convergence theorem**: We consider the subgradient method to minimize $\gamma(x)$, with update $\alpha_k = \frac{\epsilon}{\|g_k\|^2}$. The convergence theorem for this specific algorithm states that for any $\epsilon > 0$, if $k \geq \left(\frac{dist(x_0, X^*(\gamma'))L}{\epsilon}\right)^2$, then $\exists l = 0, 1, \cdots, k$ s.t. $\gamma(x_l) \leq \gamma' + \epsilon$.
From (5) we know that if we start with $x_0 = e_i$ for some index i, then $dist(x_0, X^*(\gamma')) \leq 2D(\overline{x})$.
For the specific problem regarding half-spaces, we have chosen the initial interior points in a certain way such that the Lipschitz constant $L = 1$.
Thus, assuming that we can pick $\epsilon < 1 - \gamma' = 1 - \frac{1}{1+\frac{1}{2}\alpha^*}$, the subgradient method will converge in $(\frac{2D(\overline{x})}{\epsilon})^2$ steps.

4 Restart Method

In this section, we consider an acceleration routine called the restart method, which allows us to parallelize the above subgradient method on multiple machines. We will apply this method to our above algorithm and report the speedup under different settings.

4.1 Background

We first briefly summarize the ideas of the restart method, as introduced in the paper [6].

(1) **Problem setting**: In the most general setting, consider the convex optimization problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & x \in Q \end{aligned}$$

with convex function f, closed convex set Q, and nonempty optimal solution set X^* with optimal value f^*.

Assume that we are also given a first-order method, which takes in an initial feasible point x_0 and accuracy $\epsilon > 0$, and returns an ϵ-optimal solution.

(2) **Goal**: We would like to improve the convergent rate of first-order methods via an adaptive restart scheme. The scheme involves several instances of the first-order method with different objective accuracy that communicate results via restarting under certain conditions.

(3) **Restart procedure**:

(3-1) **Setting**: Consider running $N+2$ copies of the first-order method fom_n for $n = -1, 0, \cdots, N$, each with objective accuracy $2^n\epsilon$. (We can run those copies either sequential or in parallel.)

(3-2) **Task for each** fom_n: Each machine fom_n has the goal of reducing the objective value by $2^n\epsilon$. Thus, we can think in terms of fom_N having the largest step size and is the coarsest update, while fom_{-1} has the smallest step size and performs the finest updates.

Once a machine reaches its optimization goal, it will send the best iterate to the inbox of the machine that is one step finer than it. At the same time, the machine would constantly check its own inbox, in case the previous machine has sent it a better iterate. Whenever the inbox contains a better iterate that what the machine currently has, the machine will restart with the initial iterate being the one from the inbox.

4.2 Applying restart method to the linear programming feasibility problem

(1) **Initial iteration using Polyak's step size**: Recall that in our problem, we're trying to minimize $\gamma(x)$ to be smaller than 1. We would like to apply the restart method above with several subgradient methods with different ϵ chosen.

Since we have no bound on the initial value of $\gamma(x_0)$, it makes sense to first reduce γ to a small value before applying the restart method (which, after all, has all of its machines being fixed step subgradient methods).

Thus, we can apply Polyak's method first to find an iterate $\gamma(x_k) \leq \frac{3}{2}$. We now rename the above iterate to be our new initial point x_0.

(2) **Restart method**: We consider L different subgradient methods, with the l-th method having step-size $\epsilon_l = (\frac{1}{2})^l$. We then apply the same procedure as above to create a restart method for our problem.

4.3 Experiments on the restart method

In this section, we perform experiments on the restart method. In general, we observe that the restart method, even without parallelization, uses around 2-3 times fewer iterations than applying Polyak alone.

We also observe that the exact rate of improvement depends on the shape, scaling, and sparsity of the matrix. Thus, we will consider varying parameters in these aspects and try to explain how different choice of these parameters may affect the performance of the restart method.

4.3.1 Experiment on size and shape of the matrix

(1) **Setup of the experiment**: We first run experiments on different sizes of matrix $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. The entries of the matrix A is drawn from a uniform distribution between $[-1, 1]$, and the entries of the vector b is drawn from a uniform distribution between $[-1, 0]$.

We consider picking $m \in \{1000, 2000, 4000, 8000\}$ and $n \in \{100, 200, 400, 800\}$. We run 5 experiments with each setting of (m, n) and take the average of the number of iterations.

(2) **Result of the experiment**: In the following matrix, rows represent number of variables $\{100, 200, 400, 800\}$, and columns represent number of inequalities $\{1000, 2000, 4000, 8000\}$. The entries represent the "proportion" of iterations used by Polyak's method divided by iterations used by restart method.

	1000	2000	4000	8000
100	1.9818	2.2070	2.3694	2.2188
200	1.8750	2.5791	2.2392	2.5052
400	2.0489	2.3248	2.4061	2.6026
800	1.7803	2.0465	2.6680	2.4458

Table 1: Experiment on size of matrix

We can see that the restart method have better performance compared to Polyak when the vector become higher-dimensional and the number of inequalities become larger.

4.3.2 Experiment on scaling of the matrix

(1) **Setup of the experiment**: We now run experiments on different scaling of the coordinates. In this case we fix the dimension $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ with $m = 2000$, $n = 400$.

The entries of A are drawn from a uniform distribution between $[-1, 1]$. The entries of b are first drawn from a uniform distribution between $[-1, 0]$, but then we pick the first k coordinates and divide by 10. (This corresponds to a shrinking of the region by a factor of 10 in that coordinate.) The number k is drawn from $\{0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000\}$. We perform the experiment on 5 different trials of A and b and take the average.

(2) **Result of the experiment**: Each column in the following table represents the number of coordinates that is shrunk by a factor of 10.

	0	250	500	750	1000	1250	1500	1750	2000
proportion	2.5328	2.8013	2.8328	2.6028	3.4560	2.7580	2.7002	2.8243	3.2362

Table 2: Experiment on scaling of matrix

Currently the table doesn't show any clear patterns. This is probably because there are two factors that complicate the issue: One is that shrinking all coordinates by 10 may change the performance of the two methods, and the other is that the uneven distribution of different coordinates may change the performance of the two methods. Further experiments are needed to separate these two factors so that they are not confounded by each other.

4.3.3 Experiment on sparsity of the matrix

(1) **Setup of the experiment**: We now run experiments that explore cases where we introduce sparsity into the matrix. In this case, we fix the dimension $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ with $m = 2000$, $n = 400$.

The entries of A are first drawn from a uniform distribution of $[-1, 1]$, then for each entry, with probability $p \in \{0.2, 0.4, 0.6, 0.8, 1\}$ we keep the value and otherwise we replace it by 0. The entries of b are chosen from a uniform distribution of $[-1, 0]$. Similarly, we perform the experiment on 5 different trials and take the average.

(2) **Result of the experiment**: Each column in the following table represent a trial with some probability of non-zero entries.

	0.2	0.4	0.6	0.8	1
proportion	2.1439	2.3723	2.1670	2.3893	2.9343

Table 3: Experiment on sparse matrix input

While more experiment is definitely needed, the current trend we see is that the restart method perform better with full matrices instead of sparse matrices.

5 The Perceptron method

Here we introduce the Perceptron method, a class of first-order methods that has similar inspirations as the Perceptron learning algorithm in machine learning.

5.1 Perceptron method on Linear Programming Feasibility Problem

(1) **Homogenization of the Problem**: Recall that our original problem of finding an interior point $x \in \mathbb{R}^n$ satisfying $Ax \geq b$ for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

We can perform homogenization by transforming the problem to the equivalent problem $\begin{cases} Ax - x_{n+1}b \geq 0 \\ x_{n+1} > 0 \end{cases}$.

To remove the constraint $x_{n+1} > 0$, we can rewrite $\overline{A} = \begin{bmatrix} A & -b \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(n+1)}$ and $\overline{x} = \begin{bmatrix} x \\ x_{n+1} \end{bmatrix} \in \mathbb{R}^{n+1}$. We then modify the interior points are then modified to $\overline{e}_i = \begin{bmatrix} e_i \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1}$, with the additional interior point $\overline{e}_{n+1} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1}$. Then the original problem is equivalent to the homogeneous problem $\overline{A}\overline{x} > 0$.

(2) **Cone properties of the new problem**: Let $K = \{x \in \mathbb{R}^{n+1} : Ax > 0\}$. Then for each e_i, we'll have every point in $e_i + K$ satisfy $\gamma_i(x) = 0$.
This is because for any $x \in K$, we consider $\gamma_i(x) = \frac{\|x - e_i\|}{\|\pi_i(x) - e_i\|}$. In this expression we have $\|x - e_i\|$ a fixed value, although $\|\pi_i(x) - e_i\|$ will approach infinity as the projection will never intersect with the boundary, and in this process we have $\gamma_i(x) \to 0$.

(3) **Properties of region with lowest γ value**: We now consider $S = \bigcap_{i=1}^{n+1} (e_i + K)$.
(3-1) We know that each $e_i + K \subseteq \{x \in \mathbb{R}^{n+1} : a_i^\top x \geq 0\}$, so $S \subseteq \{x \in \mathbb{R}^{n+1} : Ax \geq 0\} = K$.

(3-2) We also know that each $e_i + K = \{x \in \mathbb{R}^{n+1} : Ax \geq Ae_i\}$. Thus, $S = \bigcap_{i=1}^{n+1} (e_i + K) = \{x \in \mathbb{R}^{n+1} : Ax \geq \max_i(Ae_i)\}$. If we assume K has nonempty interior, we can just scale that vector by some factor to find a point in the interior of S, so S is also nonempty.

(3-3) Each $x \in S$ has the property that $\gamma_i(x) = 0$ for any i, from (3). Thus, we have $\gamma(x) = \max_i \gamma_i(x) = 0$.

The above statements combined shows that we have $\min \gamma(x) = 0$, which is achieved in points in the set S. Thus, for this particular optimization problem, we know the exact minimum of γ to be 0. (This is different from the previous subgradient method, where we do not know the minimum of this γ except that it is less than 1, so for the Perceptron method, Polyak's step size is always well-defined.)

(4) **Subgradient method**: We start at $x_0 = \mathbf{0}$ (where we have $\gamma(x_0) = 1$). We want some x such that $\gamma(x) < 1$.

We can apply the subgradient method $x_{k+1} = x_k - \frac{\epsilon}{\|g_k\|^2} g_k$ with $\epsilon = \frac{1}{2}$. This would guarantee to produce iterate x_l satisfying $\gamma(x_l) - 0 \leq \epsilon = \frac{1}{2}$.

Alternatively, if we want to apply Polyak's rule, we can also plug in $\gamma^* = 0$, i.e. $x_{k+1} = x_k - \frac{\gamma(x_k)}{\|g_k\|^2} g_k$ would be the Polyak update.

(5) **The usual perceptron learning algorithm in machine learning** : Recall that in the usual perceptron method, given data points $a_1, \cdots, a_m$, we would like to find a weight vector x such that $x^\top a_i > 0$ for all $i = 1, \cdots, m$. (Note that the above notations are used for aligning with our previous discussion. In machine learning literature, weights are usually denoted by w and data points $x_1, \cdots, x_m$, and target conditions $w^\top x_i > 0$.)

We start with $x_0 = 0$. In each update of the perceptron method, we would choose a random i such that $x^\top a_i \leq 0$. We then perform the update $x_{k+1} = x_k + a_i$.

(6) **Relation of our method to the perceptron learning algorithm**: The key observation is that our subgradient method is essentially doing the update $x_{k+1} = x_k + \epsilon \cdot proj(e_i, a_i)$, where $proj(e_i, a_i)$ represent the projection of e_i onto the direction of a_i, and the index of a_i is picked specifically (instead of random) such that $a_i^\top x_k$ is the largest among all i.

This can be deduced by plugging in $b = 0$ in Section 3.1, part (5):

$$g_i(x_k) = \frac{1}{a_i^\top (\pi_i(x_k) - e_i)} a_i = \frac{1}{a_i^\top \left(\frac{-a_i^\top e_i}{a_i^\top (x_k - e_i)} (x_k - e_i) \right)} a_i = -\frac{1}{a_i^\top e_i} a_i$$

and thus

$$x_{k+1} = x_k - \frac{\epsilon}{\|g_k\|^2} g_k = x_k - \epsilon \frac{-\frac{1}{a_i^\top e_i} a_i}{(\frac{1}{a_i^\top e_i})^2 \|a_i\|^2} = x_k + \epsilon \frac{a_i^\top e_i}{\|a_i\|^2} a_i = x_k + \epsilon (e_i^\top \frac{a_i}{\|a_i\|}) \frac{a_i}{\|a_i\|}$$

Let $\hat{a}_i = \frac{a_i}{\|a_i\|}$, then we see that $(e_i^\top \frac{a_i}{\|a_i\|}) \frac{a_i}{\|a_i\|} = (e_i^\top \hat{a}_i)\hat{a}_i = proj(e_i, \hat{a}_i) = proj(e_i, a_i)$ is simply the projection of e_i onto the direction of a_i. Thus, out update is essentially $x_{k+1} = x_k + \epsilon \cdot proj(e_i, a_i)$.

If we consider our construction of e_i, the distance of e_i to $a_i^\top x - b = 0$ is always 1, so we have $proj(e_i, a_i) = \frac{a_i}{\|a_i\|}$. Thus, we know that we're essentially doing $x_{k+1} = x_k + \frac{\epsilon}{\|a_i\|} a_i$, so the update is just a weighted version of perceptron update (though the randomness in choosing i in the original algorithm is fundamentally different from our choice of index with largest $\gamma_i(x) = \frac{\|x - e_i\|}{\|\pi_i(x) - e_i\|}$.)

5.2 Theoretical Deductions - Ellipsoid Problems

(1) **Representation of ellipsoid**: An ellipsoid can be represented as the set $\{y \in \mathbb{R}^n | y^\top M y + c^\top y + d \leq r\}$, where $M \in \mathbb{R}^{n \times n}$ is a symmetric positive definite matrix, $c \in \mathbb{R}^n$ is any vector, and $d, r \in \mathbb{R}$ are scalars.

For an ellipsoid to contain the vector $\mathbf{0}$ in its interior, we need to have $d < r$.

For simplicity, in the case $d < r$ we can scale both M and c by $\frac{1}{r-d}$, and thus we can always assume the ellipsoid is represented as $\{y \in \mathbb{R}^n | y^\top M y + c^\top y \leq 1\}$.

(2) **Picking an interior point of the ellipsoid**:
We consider rewriting $y^\top M y + c^\top y \leq 1$ in the alternate form $(y - e)^\top M (y - e) \leq r'$ for some vector e.
For this equivalence to hold, we can expand the second inequality into $y^\top M y - 2(Me)^\top y \leq r' - e^\top M e$.
Thus, we need to have $\begin{cases} -2Me = c \\ r' - e^\top M e = 1 \end{cases}$, so $e = -\frac{1}{2} M^{-1} c$. [This is unique since M is symmetric positive definite, so it also has a symmetric positive definite inverse.]
(We note that the above automatically ensures that $r' > 0$, as $r' = 1 + e^\top M e \geq r - d > 0$, as M is symmetric positive definite.)

(2) **The normal vector to a boundary point of the ellipsoid**: Let $f(y) = y^\top M y + c^\top y$, then the boundary of the ellipsoid would be $\{y \in \mathbb{R}^n | f(y) = 1\}$. The normal vector to a boundary point of the ellipsoid would be the direction that reduces f the most, i.e. the gradient of f. We would have $\nabla f(y) = 2My + c$.

(3) **Subgradient step**: The subgradient for the ellipsoid is given by $g_i(x_k) = \frac{1}{\langle d(x_k), \pi_i(x) - e_i \rangle} d(x_k)$, where $d(x_k) = 2M\pi_i(x_k) + c$.

5.3 Convergence of the Perception method

In this section, we present the outline for a proof of the convergence of the perceptron method above, with ideas from the paper [2].

In the following, let us denote $X^* = \bigcap_{i=1}^{n+1} (e_i + K) = \{x : \gamma(x) = 0\}$, and $x = \arg\min\{x \in X^* : \|x\| = 1\}$.

We will first analyze the property of an arbitrary iteration point x_k with $\gamma(x_k) < \frac{1}{2}$.

(1) **Distance to origin**: Recall that from usual analysis of the subgradient method, even though $\gamma(x_k)$ is not strictly decreasing in subgradient method, the distance to optimal set is strictly decreasing: $dist(x_{k+1}, X^*) \leq dist(x_k, X^*) - (2(f(x_k) - f^*) - \epsilon) \frac{\epsilon}{M^2}$.

In fact, following the same line of proof, we have $dist(x_k, x') \leq dist(x_0, x')$ for any $x' \in X^*$. Thus, this also holds for $x^* = \arg\min\{\|x\|, x \in X^*\}$. Thus, we have $|x_k - x^*| \leq |x_0 - x^*| = |x^*|$, and this implies $\|x_k\| \leq 2\|x^*\|$.

(2) **Radius of largest ball contained in** K: Let r^* be the largest scalar such that $B(x^*, r^*) \subseteq K$.

(3-1) We first show that we must have $r^* \geq 1$ in our case. This is because for any $x' \in X^*$, if we extend the ray from e_i to x' we'll never intersect with the plane $V_i = \{x : a_i^\top x = b\}$. This means $dist(x', V_i) \geq dist(e_i, V_i) = 1$ for any $x' \in X^*$, $i = 1, \cdots, n$, so we have $r^* \geq 1$.
Thus, we know that $B(x^*, 1) \subseteq K$, and if normalized $B\left(\frac{x^*}{\|x^*\|}, \frac{1}{\|x^*\|}\right) \subseteq K$.

(3-2) Now assume that $B(x^*, 1)$ is the largest ball contained in K. Then from $\gamma(x_k) < \frac{1}{2}$, we have $\frac{\|x_k - e_i\|}{\|\pi_i(x_k) - e_i\|} < \frac{1}{2}$, and thus $\frac{\|x_k - \pi_i(x_k)\|}{\|e_i - \pi_i(x_k)\|} > \frac{1}{2}$. From similar triangle property, we have $\frac{proj(x_k - \pi_i(x_k), a_i)}{proj(e_i - \pi_i(x_k), a_i)} > \frac{1}{2}$, so $dist(x_k, V_i) > \frac{1}{2}$.
Thus, we know that $B(x_k, \frac{1}{2}) \subseteq K$, so if we project x_k to the unit ball, we have $B\left(\frac{x_k}{\|x_k\|}, \frac{1}{2\|x_k\|}\right) \subseteq K$. Since $x_k \leq 2\|x^*\|$, we have $B\left(\frac{x_k}{\|x_k\|}, \frac{1}{4\|x^*\|}\right) \subseteq K$.

(3) **Implications of properties (1)(2)**: The above properties tell us that after we run the algorithm to the point where $\gamma(x_k) < \frac{1}{2}$, we can get a point x_k that is not too far from the origin (i.e. $\|x_k\| \leq 2\|x^*\|$), and not too close from the boundary (i.e. $B\left(\|x_k\|, \frac{1}{2}\right) \subseteq K$, or $B\left(\frac{x_k}{\|x_k\|}, \frac{1}{4\|x^*\|}\right) \subseteq K$).

(4) **Analyzing Perceptron method via condition number**:
The condition number, as defined in [2], is $\rho(A) = \max\limits_{\|x\|=1, Ax \geq 0} \min\limits_i (\frac{a_i^\top x}{\|a_i\|})$.

We would like to reinterpret the above algebraic expression from a geometric standpoint. We know that $\frac{a_i^\top x}{\|a_i\|}$ represent the projection of x on the direction a_i perpendicular to the constraint $a_i^\top x > b_i$. Thus, for $x \in K$, $\min\limits_i (\frac{a_i}{\|a_i\|} x)$ represent the largest radius $r(x)$ of a ball with center x that is contained in K.
Since when we scale x, the same scaling happens for the radius $r(x)$, we can write $\rho(A) = \max\limits_{\|x\|=1, Ax \geq 0} r(x) = \max\limits_{x \in K} \frac{r(x)}{\|x\|}$.
In later analysis, we'll refer to $\rho(A)$ as the optimal condition number for the problem, and $\rho(x) = \frac{r(x)}{\|x\|}$ as the condition number at some point $x \in K$.

(5) **Interpretation of our results with condition number**: We know from above discussion that when our algorithm terminates at some $\gamma(x_k) < \frac{1}{2}$, we have $B(\frac{x_k}{\|x_k\|}, \frac{1}{4\|x^*\|}) \subseteq K$. Then we have $\frac{r(x_k)}{\|x_k\|} = \frac{1}{2\|x_k\|} \geq \frac{1}{4\|x^*\|} = \frac{1}{4}\rho$. So the condition number at x_k is at least one-fourth of the condition number at x^* (a.k.a. optimal condition number of the problem).

(6) **Convergence rate of perceptron learning algorithm**: Suppose the linear program has a solution, i.e. K is nonempty. Let $z = \arg\max\limits_{\|x\|=1, x \in K} \min\limits_i (\frac{a_i^\top x}{\|a_i\|})$ with $\rho = \min\limits_i (\frac{a_i^\top z}{\|a_i\|})$
We know that for any x, $\frac{x^\top z}{\|x\|} \leq \|z\| = 1$ since the projection of z onto the direction of x can only have smaller length.
We start with $x_0 = 0$ and do the updates $x_{k+1} = x_k + \epsilon \overline{a}_k$, where $\overline{a}_k = \frac{a_k}{\|a_k\|}$.
Then we have $x_{k+1}^\top x_{k+1} = (x_k + \epsilon \overline{a}_k)(x_k + \epsilon \overline{a}_k) = x_k^\top x_k + 2\epsilon \overline{a}_k^\top x_k + \epsilon^2 \overline{a}_k^\top \overline{a}_k < x_k^\top x_k + \epsilon^2$.
Also, $x_{k+1} z = (x_k + \epsilon \overline{a}_k) z = x_k^\top z + \epsilon \overline{a}_k^\top z \geq x_k^\top z + \epsilon\rho$.
Then we have $\frac{x_k^\top z}{\|x_k\|} > \frac{k\epsilon\rho}{\sqrt{k\epsilon^2}} = \sqrt{k}\rho$. If $k \geq \frac{1}{\rho^2}$, then this quantity would be larger than 1, a contradiction to property of z.
Thus, the algorithm must have stopped (converged) before $\frac{1}{\rho^2}$ steps.

(7) **Convergence rate of the perceptron method for optimization**: Note that even though the perceptron method has convergence $\frac{1}{\rho^2}$, it is only for finding solution with $\gamma \leq 1$, and it doesn't have further guarantees. Thus, our algorithm can be seen as an "ordered" perceptron learning algorithm, by careful choice of the indices i and continuation of running the algorithm even after we're in K to get inside the region.

That said, to analyze the convergence of the Perceptron method, we still need to apply a similar argument as the subgradient method above, but not the analysis for the perceptron learning algorithm. (Note that the inequality $x_{k+1}^\top x_{k+1} < x_k^\top x_k + \epsilon^2$ within the perceptron learning algorithm proof depends on $\overline{a}_k^\top x_k < 0$, which is not true in our further runs inside K.)

6 Future Research Directions

In this section, we outline some of the connections to other methods and future research directions that can be pursued within the above framework:

- In practice, how do such first-order methods on linear programming compare to the classical simplex algorithms?
- How should we Nesterov's computation tree method concretely in our framework?
- How can we potentially prove the convergence of perceptron method, if we try to introduce randomness as in the perceptron learning algorithm?
- Another common way of dealing with such feasibility problems is first defining smoothing of the original function [3], and then applying the FISTA method [1]. Is it possible to contrast the subgradient method with the smoothing method, in terms of both analysis and performance?

7 Acknowledgement

I would like to thank Professor James Renegar for offering me the opportunity of having this first exposure to optimization research. His careful guidance of the proof techniques and implementation ideas has enabled me to learn and implement various first-order methods in the short duration of one month as a sophomore. His passion toward optimization and his clear interpretations of the beauty of the field have eventually sparked my interest in pursuing a PhD with optimization as one of the main focuses.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2:183–202, 2009.

[2] J. Dunagan and S. S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114:101–114, 2008.

[3] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[4] Y. Nesterov. Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, 146:275–297, 2014.

[5] J. Renegar. Efficient subgradient methods for general convex optimization. *SIAM Journal on Optimization*, 26:2649–2676, 2016.

[6] J. Renegar and B. Grimmer. A simple nearly-optimal restart scheme for speeding-up first order methods. *arXiv preprint arXiv:1803.00151*, 2018.