COMS 4995-8: Advanced Algorithms (Spring'21) March 25, 2021

Lecture 20: Ellipsoid Algorithm and Gradiant Descent

Instructor: *Alex Andoni* Scribes: *Greg Marzo, Alex Yao*

1 Introduction

In this lecture, we 1) conclude our discussion of the Ellipsoid Algorithm for tackling linear programming and 2) begin our exploration of gradient descent.

2 Ellipsoid Algorithm

To recap from the previous class, we left off discussing the feasibility problem which is defined as follows: Given $F \triangleq \{x \in \mathbb{R}^n : Ax \leq b\}$, return some $x \in F$ if the set is non-empty or report that $F = \varnothing$.

To address this problem using the Ellipsoid Algorithm, we first address the approximation version of the feasibility problem: Given F from the feasibility problem as well as $\epsilon > 0$, return some $x \in F$ if the set is non-empty or report that $\text{vol}(F) < \epsilon$. If $\epsilon = \exp(-poly(n, m)B)$, essentially ϵ is exponentially small, this is enough to solve the non-approximate version.

We first define key terms used in the algorithm.

Definition 1 (Ball)**.** *Fix $r > 0$, a ball*

$$B_r(y) = \{x \in \mathbb{R}^n : ||x - y|| \leq r\}$$

Definition 2 (Axis-Aligned Ellipsoid)**.** *Fix $\lambda_1, ..., \lambda_n, r > 0$, an axis-aligned ellipsoid*

$$E_{r,\lambda}(y) = \{x : \sum_{i=1}^{n} (\frac{x_i - y_i}{\lambda_i})^2 \leq r^2\}$$

Definition 3 (General Ellipsoid)**.** *Fix $r > 0$, rank n matrix $A \in \mathbb{R}^{n \times n}$, a general ellipsoid*

$$\begin{aligned} E_{r,A}(y) &= \{x + y : x^T A^T A x \leq r^2\} \\ &= \{x + y : ||Ax||_2^2 \leq r^2\} \end{aligned}$$

Alternatively,

$$E_{r,A}(y) = \{x + y : || \underbrace{\Lambda}_{\text{diagonal matrix}} \overbrace{R}^{\text{unitary rotation matrix}} x||_2^2 \leq r^2\}$$

2.1 Algorithm Idea

1. Start with a starting ellipsoid E^0 such that $F \subseteq E^0$.

2. Iterate and compute E^t for timestep t such that

   (a) $F \subseteq E^t$

   (b) $\text{vol}(E^t) < \text{vol}(E^{t-1})$

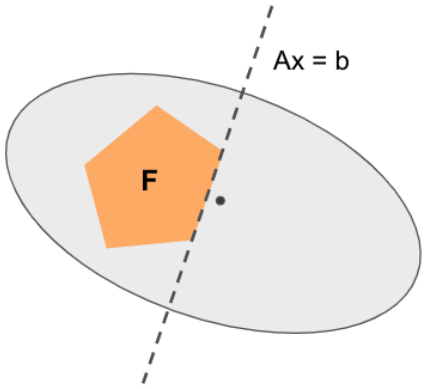
3. Terminate when the center y is in the target region F or when $\text{vol}(E^t) < \epsilon$.

2.2 Exact Algorithm

Algorithm 1: Ellipsoid Algorithm

Result: Report $x \in F$, or that $\text{vol}(F) < \epsilon$
Start with $E^0 \supseteq F$, where $E^0 = $ (ball) $B_r(y_0)$ of sufficient radius r and center y_0.
for $t = 0, ..., T$ **do**
  if $y^t \in F$ **then**
    we are done: report $x = y^t$
  else
    $\exists$ some violated constraint i: $A_i y^t > b_i$
    Define $E^{t+1}, y^{t+1} =$ smallest ellipsoid containing $E^t \cap \{A_i x \leq b_i\}$
    if $vol(E^{t+1}) < \epsilon$ **then**
      we are done: report $\text{vol}(F) < \epsilon$
    end
  end
end

Figure 1: $F \subseteq E^t \cap \{A_i x \leq b_i\} \subset E^t$



2.3 Analysis

We note that correctness of this algorithm follows by definition. If the algorithm terminates, it is correct. Now, we turn to the runtime analysis. Specifically, how many iterations (T) will this algorithm take?

Claim 4. $vol(E^{t+1}) \leq vol(E^t) \cdot (1 - \frac{1}{2n})$

Proof. The proof is omitted from lecture since we did not show how to compute the exact update for E^{t+1}, y^{t+1}. However, there are update methods that obtain the bound in this claim. □

As a result of Claim 4, we know

$$\begin{aligned}\text{vol}(E^T) &\leq \text{vol}(E^0) \cdot (1 - \frac{1}{2n})^T \\ &\leq \underbrace{(2R)^n}_{\text{ball volume bound}} \cdot (1 - \frac{1}{2n})^T\end{aligned}$$

In the worst case where we run until the volume is less than ϵ,

$$\begin{aligned}(2R)^n \cdot (1 - \frac{1}{2n})^T &\leq \epsilon \\ -n \log 2R + T \cdot \frac{\theta(1)}{2n} &\geq \log \frac{1}{\epsilon}\end{aligned}$$

$$\begin{aligned}T &\geq \theta(n)[\log \frac{1}{\epsilon} + n \cdot poly(n, m) \cdot B] \\ \#\text{ of iterations} &\approx \theta(n \log \frac{1}{\epsilon} + poly(n, m) \cdot B)\end{aligned}$$

Given our approximate algorithm, it is enough to take $\epsilon = \exp(-poly(n, m)B)$ to solve our original non-approximate feasibility problem. The resulting overall runtime is $poly(n, m, B)$.

Remark 5. *The only "access" needed to the input constraints $Ax \leq b$ is through a "separation oracle".*

Definition 6 (Separation Oracle)**.** *SO_F : Given $y \in \mathbb{R}^n$, report $y \in F$ or return some violated constraint hyperplane $Ax = b$ such that F and y are on opposite sides.*

The maximum number of calls we may need to make to $SO_F = \theta(n) \cdot [\log \frac{1}{\epsilon} + n \log R]$. Note there is no dependency on the number of constraint equations. In some cases, we can write separation oracles that work on linear problems with exponential number of constraints in only polynomial time, resulting in an overall polynomial runtime.

3 Gradient Descent

Switching gears to address the optimization of finding the minimum of a function f(x), we define 2 variants of the optimization problem:

- Unconstrained optimization:

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

- Constrained optimization:

$$min_{\substack{x \in \mathbb{R}^n \\ x \in F}} f(x)$$

A constrained optimization can be reduced to unconstrained by defining a new function g as:

$$g(x) = \begin{cases} f(x) & x \in F \\ +\infty & x \notin F \end{cases}$$

where it is enough to solve for the minimum of $g(x)$ as an unconstrained optimization problem.

3.1 Setting up Gradient Descent

In our Gradient Descent process, the basic idea is that we start with a $x^0 \in \mathbb{R}^n$ and on each iteration from $t = 0...T$, compute some new x^{t+1} based on the previous x^t via some function f

Naturally, 2 key questions arise:

1. How will we find a new x^{t+1} from x^t?
2. How many iterations T will it take to find the global minimization of $min_{x \in \mathbb{R}^n} f(x)$?

If we assume that $f(x)$ is a "nice" function, meaning that if we move a slight amount $\delta \in \mathbb{R}^n$ from $x \in \mathbb{R}^n$, the function's gradient won't change too drastically, we can use the Taylor Expansion to expand function f(x) like so:

$$f(x + \delta) \approx f(x) + \nabla f(x)^\intercal \delta + \delta^\intercal \nabla^2 f(y) \delta$$

where $y \in [x, x + \delta]$

Here, was also define the gradient and the Hessian as follows:

Definition 7. $\nabla f(x) = (\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), ...) \in \mathbb{R}^n$

Definition 8. $\nabla^2 f(x)$ *is some matrix size* $n \times n$ *where*

$$entry_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

3.2 Finding δ

To recap what we have established, we update x^{t+1} to be $x^t + \delta$. Here, suppose we establish $\|\delta\| \leq \epsilon$. We can find the best δ using the following form:

$$\delta = argmin_{\|\delta\| \leq \epsilon} f(x) + \nabla f(x)^\intercal \delta + \delta^\intercal \nabla^2 f(y) \delta$$

A question arises now on how do we estimate $\delta^\intercal \nabla^2 f(y) \delta$? There are two options we can do.

3.2.1 Option 1

$$f(x + \delta) \approx f(x) + \nabla f(x)^\intercal \delta + O(\epsilon^2)$$

We can take advantage of the fact that $O(\epsilon^2) << \nabla f(x)^\intercal \delta$ and effectively ignore the $O(\epsilon^2)$ term. Thus:

$$\begin{aligned} \delta &= argmin_{\|\delta\| \leq \epsilon} f(x) + \nabla f(x)^\intercal \delta \\ &= argmin_{\|\delta\| \leq \epsilon} \nabla f(x)^\intercal \delta \\ &= \eta(-\nabla f(x)) \end{aligned}$$

where here, $\|\eta - \nabla f(x)\| = \epsilon$ and $\eta = \frac{\epsilon}{\|\nabla f(x)\|}$
In context of the Gradient Descent Algorithm, $x^{t+1} = x^t - \eta \nabla f(x)$

3.2.2 Option 2

Definition 9. *β-**smooth:** Set $\beta > 0$. We call f β-smooth if:*

$$\begin{aligned}
&\forall x, y \in \mathbb{R}^n : \|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\| \\
&\Leftrightarrow \forall \delta \in \mathbb{R}^n : \delta^\intercal \nabla^2 f(x) \delta \leq \beta \|\delta\|^2 \\
&\Leftrightarrow \textit{max eigenvalue of } \nabla^2 f(x) \leq \beta
\end{aligned}$$

We can derive the optimization of δ for Gradient Descent in the following way:

$$\begin{aligned}
f(x + \delta) &\approx f(x) + \nabla f(x)^\intercal \delta + \delta^\intercal \nabla^2 f(y) \delta \\
&\leq f(x) + \nabla f(x)^\intercal \delta + \beta \frac{\|\delta\|^2}{2}
\end{aligned}$$

and δ now gets set as:

$$\begin{aligned}
\delta &= argmin_\delta f(x) + \nabla f(x)^\intercal \delta + \beta \frac{\|\delta\|^2}{2} \\
&= argmin_\delta \nabla f(x)^\intercal \delta + \frac{\beta}{2} \|\delta\|^2 \\
&= argmin_{\eta : \delta = \eta \nabla f(x)} - \eta \|\nabla f(x)\|^2 + \frac{\beta}{2} \eta^2 \|\nabla f(x)\|^2 \\
&= argmin_{\eta : \delta = \eta \nabla f(x)} \eta (\frac{\beta}{2} \eta - 1)
\end{aligned}$$

Thus, the optimal solution becomes $\eta = \frac{1}{\beta}$ and the optimal δ becomes $\delta = -\frac{1}{\beta} \nabla f(x)$
In context of the Gradient Descent algorithm,

$$\begin{aligned}
f(x + \delta) &\leq f(x) - \frac{1}{\beta} \|\nabla f(x)\| + \frac{\beta}{2} \frac{1}{\beta^2} \|\nabla f(x)\|^2 \\
&= f(x) - \frac{1}{2\beta} \|\nabla f(x)^2\|
\end{aligned}$$