MS&E 213 / CS 269O : Chapter 9 - Structured Optimization - Linear Programming, Interior Point Methods, and Newton's Method*

By Aaron Sidford (sidford@stanford.edu)

June 9, 2017

1 Goal

The goal in this chapter is to show that we can get even faster algorithms for convex optimization when we are given further structure on the problem. As we have seen when we wish to solve $\min_{x\in\mathbb{R}^n} f(x)$ for convex f if we wish to obtain a *linearly convergent algorithm* that is, an algorithm such that to obtain an ϵ-approximate the number of queries needed scales with $O(\log \epsilon^{-1})$, if all we have is a subgradient oracle then $O(d \cdot \log \epsilon^{-1})$ queries suffice (and are necessary for the feasibility problem). The question we address in this chapter is, when and how can we improve this result?

The main result of this chapter is that when our convex programs have further structure, e.g. we can decompose the convex function into pieces, then in principal we can get faster algorithms. The main class of algorithms we consider here are *interior point methods* and our main motivating examples is *linear programming*.

There is another theme which permeates this chapter. In a fairly broad sense, iterative methods in continuous optimization reduce solving one hard problem, e.g. convex programming, to a sequence of simpler problems, e.g. compute subgradient, project onto simple convex set, add evectors, etc. While the sub-problems we have seen before are fairly simple, here we consider more complicated sub-problems which we assume we can solve.

2 Motivating Example - Linear Programming

One of the key motivating instances of structured convex programs we will consider is the following:

Definition 1 (Linear Programming (Dual Form)). We are given *constraint matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$, *constraint vector* $b \in \mathbb{R}^n$, and *cost vector* $c \in \mathbb{R}^n$ and we wish to solve

$$\min_{\mathbf{A}x\geq b} c^\top x\,.$$

The value of $c^\top x$ is called the *cost of* x and $P = \{x : \mathbf{A}x \geq b\}$ is known as the *feasible region*. We typically use $a_1, ..., a_n$ to denote the rows of a so $P = \cap_{i\in[n]}\{x : a_i^\top x \geq b_i\}$.

[*] These notes are a work in progress. They are not necessarily a subset or superset of the in-class material and there may also be occasional *TODO* comments which demarcate material I am thinking of adding in the future. These notes will converge to a superset of the class material that is TODO-free. Your feedback is welcome and highly encouraged. If anything is unclear, you find a bug or typo, or if you would find it particularly helpful for anything to be expanded upon, please do not hesitate to post a question on the discussion board or contact me directly at sidford@stanford.edu.

This is one of the most fundamental problems in continuous optimization and algorithm design. In the remainder of this section we explore the structure of this problem a little more deeply and give some examples.

2.1 Polytopes and Convex Programming

Note that the feasible region $P = \{x : \mathbf{A}x \geq b\}$ is simply the interesection of a finite number of halfspaces. Such a region is known as a *polytope*. As we know, P, is always convex and if we allow the number of rows of $\mathbf{A}$ to be infinite then P would simply be an arbitrary cloed convex set.

By the above reasoning, linear programming is essentially a finite constraint version of convex programming. Suppose we wished to solve $\min_{x\in\mathbb{R}^n} f(x)$ where f is convex. Equivalently we could solve $\min_{f(x)\leq t} t$. Note that $f(x) \leq t$ is a convex set and thus we could write it as an intersection of an infinite number of halfspaces. Consequently, as we let m grow in an instance of linear programming we are better approximating an arbitrary convex program.

The methods we will build for linear programming will ultimately extend to $\min_{x\in S} c^\top x$ where S is a convex set we have some sort of explicit access to. Consequently, our results are even more general but considering linear programming first will convey the bulk of the difficulty of the problem and the techniques we use to solve it.

2.2 Feasibility v.s. Optimization

Note that we wrote linear programming as a *constrained optimization problem*. We wish to minimize the cost of a vector x subject to the constrainst $\mathbf{A}x \geq b$, i.e. being a member of an explicitly given polytope.

Note that in general, even the problem of checking whether or not a polytope is empty is as hard as computing the value of a linear program. Indeed given a linear programming instance $\min_{\mathbf{A}x\geq b} c^\top x$ we could let $\mathbf{A}'$ be the matrix $\mathbf{A}$ where we added a row for $-c$ and we could let b' be the vector b where we added a entry $-t$ so that

$$\{x : \mathbf{A}'x \geq b'\} = \{x : \mathbf{A}x \geq b\} \cap \{-c^\top x \geq -t\} = \{x : \mathbf{A}x \geq b\} \cap \{c^\top x \leq t\}\,.$$

Consequently, checking emptiness of a polytope lets us compute upper bounds on values for linear programs and find points in these polytopes gives us points of comparable value. Thus, by binary searching on t above we could obtain arbitrarily good estimates on the value of a linear program.

Despite this issue, our algorithms for linear programming will ultimately work by maintaining a feasible point in the polytope. In fact, we will always assume that we can compute a point inside the feasible region easy. At first, this seems like a contradiction, however we do this by performing simple transformations to our linear program. For example, if we wish to solve $\min_{\mathbf{A}x\geq b} c^\top x$ and we do not have an x_0 such that $\mathbf{A}x_0 \geq b$ then we can solve the equivalent problem

$$\min_{\mathbf{A}x+\alpha\vec{1}\geq b,\alpha\geq 0} c^\top x + \beta \cdot \alpha$$

where we choose β to be sufficiently large. Note the choice here of $\vec{1}$ as what αcontributes is fairly arbitrary. For many linear programs there is often a more canonical or natural way to do this reduction as we will see.

2.3 Example: L1 Regression

One natural example of linear programming is ℓ_1 regression. We have $a_1, ..., a_m \in \mathbb{R}^n$ as well as $b_1, ..., b_m \in \mathbb{R}$ and wish to solve

$$\min_{x\in\mathbb{R}^n} \sum_{i\in[m]} |a_i^\top x - b_i|\,.$$

Thus, we wish to be able to predict the b_i from x where we care about the total difference. Letting $\mathbf{A} \in \mathbb{R}^{m \times n}$ be the matrix where row i of $\mathbf{A}$ is a_i we can write this equivalently as

$$\min_{x \in \mathbb{R}^n} \|\mathbf{A}x - b\|_1 = \min_{x \in \mathbb{R}^n, t \in \mathbb{R}^m : -t \cdot \vec{1} \leq \mathbf{A}x - b \leq t \cdot \vec{1}} \sum_{i \in [n]} t_i \,.$$

2.4 Example: Minimum Cost Flow

Here we see a more combinatorial example I believe is particularly interesting. We have some large graph $G = (V, E)$. Now there is some stuff we wish to send between the vertices. Each vertex $v \in V$ requires b_v units of stuff on net where b_i negative means it is sending the stuff. Nowever, every edge $(a, b) \in E$ can afford to carry between 0 and $u_{(a,b)}$ units of stuff. Our goal is to figure out how to send the stuff so that every vertex v receives b_v units of stuff we don't send more stuff then the edges can carry and we minimize some cost on the stuff (we assume that every edge $(a, b) \in E$ has some cost $c_{(a,b)}$). If we let $f \in \mathbb{R}^E$ denote how much stuff we put on the edges then we see we wish to minimize

$$\sum_{(a,b) \in E} c_{(a,b)} \cdot f_{(a,b)} = c^\top f \,.$$

We wish to have $0 \leq f \leq u$ to ensure the *capacity constraints* are met and we wish to have that for all $v \in V$

$$\sum_{(a,b) \in E} f(a, v) - \sum_{(v,a) \in E} f(v, a) = b_v \,.$$

We can write the constraints more compactly as $\mathbf{M}f = b$ for some matrix $\mathbf{M}$ and thus we have that this problem is equivalent to

$$\min_{\mathbf{M}f = b, 0 \leq f \leq u} c^\top f$$

which we can write as a linear program (note that $\mathbf{M}f = b$ is equivalent to $\mathbf{M}f \geq b$ and $\mathbf{M}f \leq b$).

This problem encapsulates multiple graph optimization problems including maximum flow and minimum cut as we may see in the homework or later in the notes.

3 Approach - Newton's Method

So how do we want to build a faster algorithm for linear programming? Note, that we can compute separation oracles for $\mathbf{A}x \geq b$ and thus we can use algorithms for the feasibility problem to solve this in $O(n \log \epsilon^{-1})$ iterations. The question we address here is, can we do better. We start by taking a closer look at the only other linearly convergent algorithms we have seen so far, gradient descent for smooth convex functions. We will use this as a stepping stone for motivating Newton's method and the interior point methods we will build for linear programming (and more broadly, convex programming).

3.0.1 Gradient Descent Again

Let's take a closer look at gradient descent to both motivate our algorithms and introduce some notation and analysis tricks that will be useful throughout. Suppose we have a L-smooth μ-strongly convex $f : \mathbb{R}^n \to \mathbb{R}$ that is twice differentiable. We saw that the method

$$x_{k+1} = x_k - \frac{1}{L} \bigtriangledown f(x_k)$$

yields an ϵ-suboptimal point with $O(\frac{L}{\mu} \log(\frac{f(x_0) - f_*}{\epsilon}))$ iterations.

What does the geometry of this problem look like? Recall that since f is twice differentiable we have that it is L-smooth and μ-strongly convex if and only if for all $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$ we have

$$\mu \cdot \|z\|_2^2 \preceq z^\top \bigtriangledown^2 f(x) z \leq L \cdot \|z\|_2^2 \,.$$

Another way to write this is that for all $x \in \mathbb{R}^n$ ad $z \in \mathbb{R}^n$ we have

$$\mu \cdot z^\top \mathbf{I} z \preceq z^\top \bigtriangledown^2 f(x) z \leq L \cdot z^\top \mathbf{I} z \,.$$

How should we interpret this? Note that $\bigtriangledown^2 f(x)$ is always symmetric and that for any symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ there is an orthonormal basis of eigenvectors $v_1(\mathbf{A}), ..., v_n(\mathbf{A}) \in \mathbb{R}^n$ with corresponding eigenvalue $\lambda_1(\mathbf{A}), ..., \lambda_n(\mathbf{A})$, i.e. $v_i(\mathbf{A}) \in \mathbb{R}^n$ with the following holding for all $i, j \in [n]$

$$v_i(\mathbf{A})^\top v_j(\mathbf{A}) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \text{ and } \mathbf{A} v_i(\mathbf{A}) = \lambda \cdot v_i(\mathbf{A}) \,.$$

Now, letting $\mathbf{V}(\mathbf{A})$ be the matrix such that column i of $\mathbf{V}(\mathbf{A})$ is $v_i(\mathbf{A})$ we see that $\mathbf{V}(\mathbf{A})^\top \mathbf{V}(\mathbf{A}) = \mathbf{I}$ and consequently $\mathbf{V}(\mathbf{A})\mathbf{V}(\mathbf{A})^\top = \mathbf{I}$. From this we have that we can write any $x \in \mathbb{R}^n$ by

$$x = \mathbf{V}(\mathbf{A})\mathbf{V}(\mathbf{A})^\top x = \sum_{i \in [n]} v_i(\mathbf{A}) v_i(\mathbf{A})^\top x = \sum_{i \in [n]} v_i(\mathbf{A}) \cdot \left(v_i(\mathbf{A})^\top x\right)$$

and we have that

$$\mathbf{A} x = \mathbf{A} \left(\sum_{i \in [n]} v_i(\mathbf{A}) \cdot \left(v_i(\mathbf{A})^\top x\right) \right) = \sum_{i \in [n]} \lambda_i(\mathbf{A}) \cdot \left[v_i(\mathbf{A})^\top x\right] \cdot v_i(\mathbf{A})$$

and therefore for $\mathbf{\Lambda}(\mathbf{A}) = \mathbf{diag}(\lambda(\mathbf{A}))$ we have

$$\mathbf{A} = \mathbf{V}(\mathbf{A}) \mathbf{\Lambda}(\mathbf{A}) \mathbf{V}(\mathbf{A})^\top = \sum_{i \in [n]} \lambda_i(\mathbf{A}) \cdot v_i(\mathbf{A}) v_i(\mathbf{A})^\top \,.$$

Consequently we have that

$$z^\top \mathbf{A} z = \lambda_i \cdot (v_i(\mathbf{A})^\top z)^2$$

and

$$R_{\mathbf{A}}(z) = \frac{z^\top \mathbf{A} z}{z^\top z} = \frac{\sum_{i \in [n]} \lambda_i \cdot (v_i(\mathbf{A})^\top z)^2}{\sum_{i \in [n]} (v_i(\mathbf{A})^\top z)^2} \,.$$

Thus $z \cdot R_{\mathbf{A}}(z)$ for all z with $\|z\|_2 \leq 1$, i.e. $E_{\mathbf{A}} = \{x \in \mathbb{R}^n \,:\, x = z \cdot R_{\mathbf{A}}(z), \|z\|_2 \leq 1\}$, is an ellipse where the axis are the $v_i(\mathbf{A})$ and the radii are the λ_i. Furthermore, we see that the condition

$$\mu \cdot z^\top \mathbf{I} z \preceq z^\top \bigtriangledown^2 f(x) z \leq L \cdot z^\top \mathbf{I} z$$

is equivalent to $E_{\mathbf{A}} \subseteq L \cdot E_{\mathbf{I}} = E_{L \cdot \mathbf{I}}$ and $\mu \cdot E_{\mathbf{I}} = E_{\mu \cdot \mathbf{I}} \subseteq E_{\mathbf{A}}$.

This gives us a nice ordering on symmetric matrices. We say $\mathbf{A} \preceq \mathbf{B}$ if and only if $x^\top \mathbf{A} x \preceq x^\top \mathbf{B} x$ for all x and define $\prec$,$\succ$, and $\succeq$ analogously. As we have seen these correspond to the ellipsoid containment inequalities we have given above. This is known the Lowener order on symmetric matrices. It behaves a lot like standard inequalities on scalars, e.g. if $\mathbf{A} \preceq \mathbf{B}$ and $\mathbf{B} \preceq \mathbf{C}$ then $\mathbf{A} \preceq \mathbf{C}$. Furthermore, if $\mathbf{A} \preceq \mathbf{B}$ and $\mathbf{B} \preceq \mathbf{A}$ then $\mathbf{A} = \mathbf{B}$. Also if $\mathbf{A} \preceq \mathbf{B}$ then $\mathbf{A} + \mathbf{C} \preceq \mathbf{B} + \mathbf{C}$. Furthermore we have that $\mu \mathbf{I} \preceq \mathbf{A} \preceq L \mathbf{I}$ if and only if $\lambda_i(\mathbf{A}) \in [\mu, L]$ for all $i \in [n]$. Note that our interpretation of $E_{\mathbf{A}}$ as an ellipse gets a little muddle if the eigenvalues of $\mathbf{A}$ are negative and thus when this does not hold, i.e. the eigenvalues of $\mathbf{A}$ are non-negative we give such matrices a special name and call that positive semidefininite (PSD), i.e. $\mathbf{A} \succeq \mathbf{0}$. Furthermore, we call $\mathbf{A}$ positive definite (PD) if all its eigenvalues are positive, i.e $\mathbf{A} \succ \mathbf{0}$.

With this interpretation we see that twice differentiable f is convex if and only if its Hessian is everywhere PSD and we see that is is μ-strongly convex and L-smooth if and only if the ellipse corresponding to the Hessian is contained in a ball of radius L and contains a ball of radius μ, i.e. the countours of f are approximated by balls up to a factor of L/μ.

3.1 Towards Newton's Method

With this analysis of gradient descent in mind, what is the problem with applying it to linear programming? In short, we do not hope that in general linear programs should look like balls. We can easily make them squeeze into an arbitrary subsace. Furthermore, simply solving $\mathbf{A}x = b$ is a special case of linear programming. Consequently, any reasonable approximation to the structure of a linear programming could be arbitrarily badly approximated by a ball. We have no hope of bounding L/μ for linear programming while obtaining a linearly convergent algorithm.

Instead, perhaps we can approximate a linear program by an ellipse in various contexts. In some sense this is what the ellipsoid method does. This is a natural thing to do. Suppose that we don't have $\mu \cdot \mathbf{I} \preceq \bigtriangledown^2 f(x) \preceq L \cdot \mathbf{I}$ and instead have that $\mu \cdot \mathbf{H} \preceq \bigtriangledown^2 f(x) \preceq L \cdot \mathbf{H}$ for some symmetric PD matrix $\mathbf{H}$. This would suggest that the contours of f looking like the $\mathbf{H}$ ellipsoid. How would we get a fast algorithm in this case?

A natural idea would simply be to perform a change of basis. We could simply pick some invertible $\mathbf{M} \in \mathbb{R}^{n \times n}$ and minimize $g(y) \overset{\text{def}}{=} f(\mathbf{M}y)$. It is not too hard to see by chain rule that $\bigtriangledown g(y) = \mathbf{M}^\top \bigtriangledown f(\mathbf{M}y)$ and $\bigtriangledown^2 g(y) = \mathbf{M}^\top \bigtriangledown^2 f(\mathbf{M}y)\mathbf{M}$.

What $\mathbf{M}$ should we pick? Note that for every symmetric PSD matrix $\mathbf{H}$ the matrix

$$\mathbf{H}^{1/2} = \sum_{i \in [n]} \sqrt{\lambda_i(\mathbf{H})} \cdot v_i(\mathbf{H}) v_i(\mathbf{H})^\top$$

is the unique symmetric PSD matrix such that $\mathbf{H}^{1/2}\mathbf{H}^{1/2} = \mathbf{H}$. We call this the square root of $\mathbf{H}$. Furthermore we see that

$$\mathbf{H}^{-1/2} = \sum_{i \in [n]} \frac{1}{\sqrt{\lambda_i(\mathbf{H})}} \cdot v_i(\mathbf{H}) v_i(\mathbf{H})^\top = \left[\mathbf{H}^{1/2}\right]^{-1}$$

is both the square root of the inverse and the inverse of the square root of $\mathbf{H}$. Thus a natural choice would be to pick $\mathbf{M} = \mathbf{H}^{-1/2}$. Note that this would yield that

$$\mu \cdot \mathbf{I} \preceq \bigtriangledown^2 g(y) \preceq L \cdot \mathbf{I}$$

and thus gradient descent

$$y_{k+1} = y_k - \frac{1}{L} \bigtriangledown g(y_k)$$

achieves $g(y_{k+1}) - g_* \leq \epsilon$ in $O(\frac{L}{\mu} \log((g(y_0) - g_*)/\epsilon)$ iterations. However, what does this give for f? It gives

$$f(\mathbf{H}^{-1/2} y_k) - f_* \leq \epsilon$$

however

$$\mathbf{H}^{-1/2} y_{k+1} = \mathbf{H}^{-1/2} y_k - \frac{1}{L} \mathbf{H}^{-1} \bigtriangledown f(\mathbf{H}^{-1/2} y_k)$$

and consequently we have just proven that letting

$$x_{k+1} = x_k - \frac{1}{L} \mathbf{H}^{-1} \bigtriangledown f(x_k)$$

yields an ϵ-suboptimal x_k in the same amount of time. Cosnequently, we get fast convergence rate and the cost of the iteration is the cost of one gradient computation and solving one lienar system in $\mathbf{H}$. In general this is $O(n^3)$ by Guassian elimination or $O(n^\omega)$ by fast matrix multiplication, where currently $\omega < 2.373$.

However, if we believe it is the case that the Hessian doesn't change too much. Rather than using such a $\mathbf{H}$ we could also just use the Hessian at x_k, i.e.

$$x_{k+1} = x_k - \eta \bigtriangledown^2 f(x_k)^{-1} \bigtriangledown f(x_k)\,.$$

This is known as damped Newton iteration. In the case when $\eta = 1$ this is known as Newton's method and corresponds to picking the next point to be the minimizer of the second order Taylor approximation of f, i.e.

$$x_{k+1} = \text{argmin}_{x \in \mathbb{R}^n} f(x_k) + \bigtriangledown f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top \bigtriangledown^2 f(x_k)(x - x_k)\,.$$

Now, achieving fast convergnece of iterative methods given a second order oracle, i.e. access to $f(x)$, $\bigtriangledown f(x)$, and $\bigtriangledown^2 f(x)$ is an interesting active area of research. However, we will only need to use a fairly basic set of results (all of which could have been recovered from facts about first order minimization under a change of basis.

4 Interior Point Methods

Our main idea for solving linear programming is to attempt to repeatedly use Newton's method, or (since it will suffice) first order methods in a sufficiently chosen change of basis. The idea is that we will try to reduce linear programming to a sequence of unconstrained optimization problems, for each of which the Hessian is fairly constant. We will then try to show that such a method converges quickly to optimum.

The framework we use is *interior point methods*. Now there are numerous variants of this method, and we won't discuss all. However, to reduce the difficult non-smooth problem of linear programming to nicer, smooth minimization problems these methods all use either implicitly or explicictly a *barrier function*. This is simply a function

$$p : \mathbb{R}^n \to \mathbb{R}$$

that is twice differentiable, such that $\lim_{x \to \partial(\mathbf{A}x \geq b)} p(x) \to \infty$. We only need p defined on $P_0 \overset{\text{def}}{=} \text{int}(P) = \{x \in \mathbb{R}^n : \mathbf{A}x > b\}$.

In these notes we will analyze a particular variant of interior point methods known as *path following methods which* trade off how much we care about minimizing cost versus staying away from the constraints of the linear program. In other words they solve the following

$$\min_{x \in \mathbb{R}^n} f_t(x) \overset{\text{def}}{=} t \cdot c^\top x + p(x)\,.$$

For well behaved p and large enough t this converges to x_*. We let $x^{(t)} \overset{\text{def}}{=} \text{argmin}_{x \in \mathbb{R}^n} f_t(x)$ and note that $\lim_{t \to 0} x^{(t)} = x_*$, in other words the minimizers of this form a path to the solution of the linear program (known as the central path). Consequently, while this optimization problem is as hard as linear programming for large values of t, we will ultimately show that we can transform the program so that it is easy to compute x_t for some value of t and then show that we can use this point to compute $x_{t'}$ for t' within a range of t and then repeat.

The most common choice of p for this problem is known as the logarithmic barrier function,

$$p_l(x) = -\sum_{i \in [n]} \ln(a_i^\top x_i - b_i)$$

We call $a_i^\top x - b_i = s(x)_i$ the *slack* of constraint i as it is how far away a point is from this constraint being tight and call $s(x)$ the *slack vector.*

Now our approach for linear programming is more clear, we will simply start with some $x_0 \approx x^{(t_0)}$ and then let $t_{k+1} = t_k \cdot (1 + \alpha)$ and $x_{k+1} = x_k - \bigtriangledown^2 f_{t'}(x_k)^{-1} \bigtriangledown f_{t'}(x_k)$. So long as we can show that the Hessian doesn't change too quickly for a sufficiently large value of α and we can bound how large a value of t we need, we will achieve an effecient algorithm. However, to analyze this we first need to parameterize how exactly we measure the Hessian changing and anlyze a single step of Newton's method.

5 Newton's Method Analysis

To analyze this method, we will find it easier to provide some new proof of gradient descent and thereby, Newton's method.

To start, recall our first analysis of gradient descent where we show that for a L-smooth, μ-strongly convex function a step of gradient descent decreased the function error by a multiplicative factor depending only on smoothness and strong convexity.

Theorem 2 (Gradient Descent (Function Progress))**.** *If f is a L-smooth and μ-strongly convex function and we let $y = x - \frac{1}{L} \bigtriangledown f(x)$ then $f(y) - f_* \leq \left(1 - \frac{\mu}{L}\right)[f(x) - f_*]$.*

Using the analysis of mirror descent we can alsos show that the squared distance to the optimum also decreases at the same rate.

Theorem 3 (Gradient Descent (Function Progress))**.** *If f is a L-smooth and μ-strongly convex function and we let $y = x - \frac{1}{L} \bigtriangledown f(x)$ then $\|y - x_*\|_2^2 \leq \left(1 - \frac{\mu}{L}\right)\|x - x_*\|_2^2$.*

Proof. Expanding the definition of $\|\cdot\|_2^2$ yields that

$$\|y - x_*\|_2^2 = \|x - x_* - \frac{1}{L} \bigtriangledown f(x)\|_2^2 = \|x - x_*\|_2^2 - \frac{2}{L} \bigtriangledown f(x)^\top (x - x_*) + \frac{1}{L^2}\| \bigtriangledown f(x)\|_2^2$$

However,

$$f(x_*) \geq f(x) + \bigtriangledown f(x)^\top (x_* - x) + \frac{\mu}{2}\|x - x_*\|_2^2$$

and by smoothness $\| \bigtriangledown f(x)\|_2^2 \leq 2L \cdot [f(x) - f_*]$and therefore

$$\|y - x_*\|_2^2 \leq \|x - x_*\|_2^2 - \frac{2}{L}[f(x) - f_*] - \frac{\mu}{L}\|x - x_*\|_2^2 + \frac{2}{L}[f(x) - f_*] .$$

□

Now, we will find it convenient to perform our analysis not in terms of either of these potential functions, but rather in terms of the the norm of the gradient, i.e. how much a gradient descent step moves in $\|\cdot\|_2$. As we know, function error, norm of gradient, and squared distance to optimum are all relatable to each other up to constant and thus we can do our analysis with any of these up to logarithmic factors. However, we will find the norm of the gradient the cleanest to reason about.

To perform this analysis we will find it convenient to work with the operator norm of a matrix.

Definition 4 (Matrix Operator Norm)**.** For any matrix $\mathbf{A}$ and any norm $\|\cdot\|$ we let $\|\mathbf{A}\| = \max_{x \neq 0} \frac{\|\mathbf{A}x\|}{\|x\|}$.

This definition is convenient as trivially we have that for all $x \in \mathbb{R}^n$, $\|\mathbf{A}x\| \leq \|\mathbf{A}\| \cdot \|x\|$ (just consider the $x = 0$ case separately). We will also use the following lemma which lets us characterize the operator norm of a symmetric matrix.

Lemma 5. *If $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric then $\|\mathbf{A}\|_2 = \max_{i \in [n]} |\lambda_i(\mathbf{A})|$.*

Proof. We have that

$$\|\mathbf{A}\|_2^2 = \left(\max_{x \neq 0} \frac{\|\mathbf{A}x\|_2}{\|x\|_2}\right)^2 = \max_{x \neq 0} \frac{x^\top \mathbf{A}^\top \mathbf{A} x}{x^\top x}$$

However, since $\mathbf{A}$ is symmetric the eigenvalues of $\mathbf{A}^\top \mathbf{A}$ are the eigenvalues of $\mathbf{A}$ squared, i.e. $\lambda_i(\mathbf{A})^2$. Thus $\|\mathbf{A}\|_2 = \max_{i \in [n]} \sqrt{\lambda_i(\mathbf{A})^2} = \max_{i \in [n]} |\lambda_i(\mathbf{A})|$. □

Using this we analyze how a step of gradient descent affects the norm of the gradient.

Theorem 6. *If f is a L-smooth and μ-strongly twice differentiable convex function and we let $y = x - \eta \bigtriangledown f(x)$ then*

$$\| \bigtriangledown f(y) \|_2 \leq \max\left\{1 - \eta L, 1 - \eta \mu\right\} \| \bigtriangledown f(x) \|_2 \,.$$

and thus when $\eta = \frac{1}{L}$ we have that $\| \bigtriangledown f(y) \|_2 \leq (1 - \frac{\mu}{L}) \| \bigtriangledown f(x) \|_2$

Proof. For all $t \in [0,1]$ let $x_t = x + t(y - x)$. Integrating yields that

$$\bigtriangledown f(x_1) - \bigtriangledown f(x_0) = \int_0^1 \bigtriangledown^2 f(x_t) \cdot (y - x) dt = \int_0^1 -\eta \bigtriangledown^2 f(x_t) \bigtriangledown f(x) dt \,.$$

Consequently,

$$\bigtriangledown f(x_1) = \int_0^1 \left[\mathbf{I} - \eta \bigtriangledown^2 f(x_t)\right] \bigtriangledown f(x) dt$$

and therefore

$$\| \bigtriangledown f(y) \|_2 = \| \int_0^1 \left[\mathbf{I} - \eta \bigtriangledown^2 f(x_t)\right] \bigtriangledown f(x) dt \|_2 \leq \int_0^1 \| \left[\mathbf{I} - \eta \bigtriangledown^2 f(x_t)\right] \bigtriangledown f(x) \|_2 dt$$

However, we have that since $\lambda_i(\bigtriangledown^2 f(x_t)) \in [\mu, L]$ we have that $\lambda_i(\mathbf{I} - \eta \bigtriangledown^2 f(x_t)) \in [1 - \eta L, 1 - \eta \mu]$ and the result follows. □

So how do we extend this to Newton's method? If we perform the step $y = x - \bigtriangledown^2 f(x)^{-1} \bigtriangledown f(x)$ then we likely want to look at the norm of the gradient in the norm induced by the Hessian, i.e. $\|y - x\|_{\bigtriangledown^2 f(x)} = \| \bigtriangledown f(x) \|_{\bigtriangledown^2 f(x)^{-1}}$ where for symmetric PD $\mathbf{A}$ we let $\|x\|_{\mathbf{A}} \stackrel{\text{def}}{=} \sqrt{x^\top \mathbf{A} x}$ (note that this is always a norm when $\mathbf{A}$ is PD). We call this Newton decrement, it is simply the norm of the gradient in the change of basis induced by the hessian.

Definition 7 (Newton Decrement)**.** For twice differentiable $f : \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$ we let $\delta_f(x) \stackrel{\text{def}}{=} \| \bigtriangledown f(x) \|_{\bigtriangledown^2 f(x)^{-1}}$ denote the *Newton decrement*.

Note that in our analysis of gradient descent, we really only required that the Hessian be bounded by the identity along the line between x and y. Similarly, we only require Hessian be bounded by the Hessian at x along the line between y and x. Below we provide our main Theorem for Newton's method. Note that it is essentiall the same as our last proof regarding gradient descent and we could prove it by simply applying it under a change of basis, however for completeness and concreteness we prove it directly.

Theorem 8. *Let f be a twice differentiable function and let $y = x - \eta \bigtriangledown^2 f(x)^{-1} \bigtriangledown f(x)$. Now suppose that $\bigtriangledown^2 f(y) \succeq \alpha \bigtriangledown^2 f(x)$ and that for $t \in [0,1]$ and $x_t = x + t \cdot (y - x)$ we have*

$$\mu \cdot \bigtriangledown^2 f(x) \preceq \int_0^1 \bigtriangledown^2 f(x_t) dt \preceq L \cdot \bigtriangledown^2 f(x)$$

then $\delta_f(y) \leq \sqrt{(1/\alpha)} \cdot \max\left\{1 - \eta L, 1 - \eta \mu\right\} \delta_f(x)$.

Proof. For notational convenience let $\mathbf{H} \stackrel{\text{def}}{=} \bigtriangledown^2 f(x)$. Integrating yields that

$$\bigtriangledown f(x_1) - \bigtriangledown f(x_0) = \int_0^1 \bigtriangledown^2 f(x_t) \cdot (y - x) dt = \int_0^1 -\eta \bigtriangledown^2 f(x_t) \mathbf{H}^{-1} \bigtriangledown f(x) dt \,.$$

and therefore re-arranging yields that

$$\mathbf{H}^{-1/2} \bigtriangledown f(x_1) = \int_0^1 \left[\mathbf{I} - \eta \mathbf{H}^{-1/2} \bigtriangledown^2 f(x_t)\mathbf{H}^{-1/2}\right] \mathbf{H}^{-1/2} \bigtriangledown f(x) dt$$

and therefore

$$\begin{aligned}\| \bigtriangledown f(y)\|_{\mathbf{H}^{-1}} &= \| \left[\mathbf{I} - \eta \mathbf{H}^{-1/2} \int_0^1 \bigtriangledown^2 f(x_t) d_t \mathbf{H}^{-1/2}\right] \mathbf{H}^{-1/2} \bigtriangledown f(x)\|_2 \\ &\leq \|\mathbf{I} - \eta \mathbf{H}^{-1/2} \int_0^1 \bigtriangledown^2 f(x_t) d_t \mathbf{H}^{-1/2}\|_2 \cdot \delta_f(x) \,.\end{aligned}$$

However, since $\mu \cdot \mathbf{H} \preceq \int_0^1 \bigtriangledown^2 f(x_t) dt \preceq L \cdot \mathbf{H}$ we have that $\mu \cdot \mathbf{I} \preceq \mathbf{H}^{-1/2} \int_0^1 \bigtriangledown^2 f(x_t) dt \mathbf{H}^{-1/2} \preceq L \cdot \mathbf{I}$ we have that $\lambda_i(\mathbf{H}^{-1/2} \int_0^1 \bigtriangledown^2 f(x_t) dt \mathbf{H}^{-1/2}) \in [\mu, L]$ and therefore $\|\mathbf{I} - \eta \mathbf{H}^{-1/2} \int_0^1 \bigtriangledown^2 f(x_t) dt \mathbf{H}^{-1/2}\|_2 \in [1 - \eta L, 1 - \eta \mu]$. The result then follows from the fact that $(\alpha \mathbf{H})^{-1} \succeq \bigtriangledown^2 f(x)^{-1}$. □

In the next section we show how to use that to get closer to the central path for our barrier function

6 Centering to the Central Path

Let's see how to use the analysis we just did of Newton's method to compute central path points (given crude approximations to them).

Again we have a barrier function $p(x)$ where in our particular case

$$p_l(x) \stackrel{\text{def}}{=} - \sum_{i \in [n]} s_i(x) \text{ where } s(x) = \mathbf{A}x - b \,.$$

We also have $\bigtriangledown p_l(x) = -\mathbf{A}^\top \mathbf{S}_x^{-1} \vec{1}$ and $\bigtriangledown^2 p_l(x) = \mathbf{A}^\top \mathbf{S}_x^{-2} \mathbf{A}$ where $\mathbf{S}_x = \mathbf{diag}(s(x))$ and our goal is to comptue

$$x^{(t)} = \operatorname{argmin}_{x \in \mathbb{R}^n} f_t(x) \text{ where } f_t(x) \stackrel{\text{def}}{=} t \cdot c^\top x - p(x) \,.$$

Now, clearly $\bigtriangledown^2 f_t(x) = \bigtriangledown^2 p(x)$ and thus if we want to understand the performance of Newton's method we need to understand how $\bigtriangledown^2 p(x)$ changes as we change x. We analyze this in the following.

Lemma 9 (Change in Barrier Hessian)**.** *For $x \in P_0$ if $\|x - y\|_{\bigtriangledown^2 p(x)} \leq \alpha < 1$ then $y \in P_0$ and*

$$(1 - \alpha)^2 \bigtriangledown^2 p(y) \preceq \bigtriangledown^2 p(x) \preceq (1 + \alpha)^2 \bigtriangledown^2 p(y) \,.$$

Proof. Note that $\mathbf{A}(x - y) = \mathbf{A}x - b - [\mathbf{A}y - b] = s(x) - s(y)$. Consequently

$$\begin{aligned}\|x - y\|^2_{\bigtriangledown^2 f(x)} &= (x - y)^\top \mathbf{A}^\top \mathbf{S}_x^{-2} \mathbf{A}(x - y) = (s(x) - s(y)^\top \mathbf{S}_x^{-2}(s(x) - s(y) \\ &= \sum_{i \in [n]} \frac{(s(x) - s(y))^2}{s(x)^2} = \sum_{\in [n]} \left(1 - \frac{s(y)}{s(x)}\right)^2 \,.\end{aligned}$$

Therefore, if $\|x - y\|_{\bigtriangledown^2 f(x)} \leq \alpha$ we have that for all $i \in [n]$ it is the case that

$$\left|1 - \frac{s(y)_i}{s(x)_i}\right| \leq \alpha$$

and therefore $-\alpha s(x)_i \leq s(y)_i - s(x)_i \leq \alpha s(x)_i$ yielding that $(1 - \alpha)\mathbf{S}_x \preceq \mathbf{S}_y \preceq (1 + \alpha)\mathbf{S}_x$ and since $\alpha < 1$. Since $\alpha < 1$ and $x \in P_0$ if and only if $S_x \succ \mathbf{0}$ we have that $y \in P_0$. Furthermore, we see that since $\mathbf{S}_y$ is a diagonal matrix we have that $(1 - \alpha)^2 \mathbf{S}_x^2 \preceq \mathbf{S}_y^2 \preceq (1 + \alpha)\mathbf{S}_x^2$[1] Consequently, $(1 - \alpha)^{-2} \mathbf{S}_x^{-2} \succeq \mathbf{S}_y^{-2} \succeq (1 + \alpha)^{-2} \mathbf{S}_x^{-2}$ yielding the result. □

[1] Note that for symmetric PSD matrices $\mathbf{A} \preceq \mathbf{B}$ does not necessarily imply $\mathbf{A}^2 \preceq \mathbf{B}^2$.

Using this lemma we can analyze the effect of a newton step on $f_t(x)$. For notational convenience we let $\delta_t(x) \stackrel{\text{def}}{=} \delta_{f_t}(x)$.

Lemma 10. *For $x \in P_0$ and let $y = x - \bigtriangledown^2 f_t(x)^{-1} \bigtriangledown f_t(x)$ for $\delta_t(x) < 1$. Then $y \in P_0$ and*

$$\delta_t(y) \leq \left(\frac{1+\delta_t(x)}{1-\delta_t(x)}\right) \cdot \delta_t(x)^2$$

and if $\delta_t(x) \leq \frac{1}{3}$ we have that $\delta_t(y) \leq 2\delta_t(x)^2$.

Proof. For all $v \in [0,1]$ and $x_v = x + v(y-x)$. Note that $\|x_v - x\|_{\bigtriangledown^2 f_t(x)} = v \cdot \|y - x\|_{\bigtriangledown^2 f_t(x)} = v \cdot \delta_{f_t}(x) < 1$ by choice of η. Consequently by Lemma 9 we have that $y \in P_0$ and

$$(1 + v \cdot \delta_t(x))^{-2} \bigtriangledown^2 f_t(x) \preceq \bigtriangledown^2 f_t(x_v) \preceq (1 - v \cdot \delta_t(x))^{-2} \bigtriangledown^2 f_t(x)$$

Since

$$\int_0^1 (1 - v \cdot \delta_t(x))^{-2} dv = \frac{1}{\delta_t(x)} \cdot \left[\frac{1}{1-\delta_t(x)} - \frac{1}{1}\right] = \frac{1}{1-\delta_t(x)}$$

and

$$\int_0^1 (1 + v \cdot \delta_{f_t}(x))^{-2} dv = \frac{1}{\delta_{f_t}(x)} \cdot \left[\frac{-1}{1+\delta_{f_t}(x)} - \frac{-1}{1}\right] = \frac{1}{1+\delta_{f_t}(x)}$$

we have that

$$\left(\frac{1}{1+\delta_{f_t}(x)}\right) \cdot \bigtriangledown^2 f_t(x) \preceq \int_0^1 \bigtriangledown^2 f_t(x_v) dv \preceq \left(\frac{1}{1-\delta_{f_t}(x)}\right) \cdot \bigtriangledown^2 f_t(x)\,.$$

Furthermore, since $\bigtriangledown^2 f_t(y) \succeq (1 + \delta_{f_t}(x))^2 \bigtriangledown^2 f_t(x)$ by Theorem 8 we have

$$\begin{aligned} \delta_{f_t}(y) &\leq (1 + \delta_{f_t}(x)) \cdot \max\left\{\left|1 - \frac{1}{1-\delta_{f_t}(x)}\right|, \left|1 - \frac{1}{1+\delta_{f_t}(x)}\right|\right\} \cdot \delta_{f_t}(x) \\ &\leq (1 + \delta_{f_t}(x)) \cdot \max\left\{\frac{\delta_{f_t}(x)}{1-\delta_{f_t}(x)}, \frac{\delta_{f_t}(x)}{1+\delta_{f_t}(x)}\right\} \cdot \delta_{f_t}(x)^2 \end{aligned}$$

yielding the result. □

Note that this is a very good rate of convergence provided $\delta_{f_t}(x)$ is a small constant. For example, whenever, $\delta_{f_t}(x) \leq \frac{1}{4}$ one step of Newton's method will halve the size of the Newton decrement. Moreover, we have that if we repeat this algorithm, i.e. $x_{k+1} = x_k - \bigtriangledown^2 f_t(x)^{-1} \bigtriangledown f_t(x)$ then $\delta_{f_t}(x_k) \leq (2\delta f_t(x_0)^{2^k}$ and therefore we can achieve $\delta_{f_t}(x_k) \leq \epsilon$ with $k = O(\log\log(\frac{1}{\epsilon}))$. This is known as quadratic convergence as the number of bits of precision in the answer grows quadratically, i.e. squares, with every iteration.

We conclude by showing how to relate the norm of the gradient to the distance to the optimum point in the Hessian norm.

Lemma 11. *Suppose that for $x \in P_0$ we have that $\delta_{f_t}(x) < \frac{1}{4}$ then $\|x - x^{(t)}\|_{\bigtriangledown^2 f(x)} \leq 4 \cdot \delta_t$*

Proof. Let $x(y) = \bigtriangledown^2 f_t(x)^{-1/2} y$ and let $g(y) = f_t(x(y))$. Note that $\| \bigtriangledown g(y)\|_2 = \| \bigtriangledown f_t(x(y))\|_{\bigtriangledown^2 f(x)^{-1}}$ by chain rule. Furthermore, note that for $\|y - z\|_2 \leq \alpha$ we have that $\|x(y) - x(z)\|_{\bigtriangledown^2 f_t(x)} \leq \alpha$ and therefore by Lemma 9 we have that in the ball of radius α around x the function $g(y)$ is $(1+\alpha)^{-2}$-strongly convex and $(1-\alpha)^2$ smooth. Furthermore, this implies that restricted to this ball we know that $\| \bigtriangledown^2 f(x)^{1/2}(x - x_*)\|_2 \leq \left[\frac{1}{(1+\alpha)^{-2}}\right]^{-1} \| \bigtriangledown g(\bigtriangledown^2 f(x)^{1/2} x)\|_2 = (1+\alpha)^2 \delta_t$ and thus when $(1+\alpha)^2 \delta_t \leq \alpha$ we have that x_* lies inside the ball. Consequently $\|x - x_*\|_{\bigtriangledown^2 f(x)} \leq \alpha$ for the smallest α such that $\delta_t \leq \frac{\alpha}{(1+\alpha)^2}$. However, since $4 \geq (1 + \frac{4}{4})^2$ we have the desired result. □

7 Moving Along the Central Path

In the last section we showed that if we were sufficiently close to the central path, i.e. have $\delta_{f_t}(x) \leq \frac{1}{4}$, then we can quickly get arbitrarily close to $x^{(t)}$. What remains to show is how to use a point near the central path for one value of t to get close to the central path for another value.

We analyze this in a few steps. First we provide the following lemma showing that the contribution of the barrier to the gradient in the Hessian inverse norm is small.

Lemma 12. *For all $x \in P_0$ we have $\| \bigtriangledown p(x)\|_{(\bigtriangledown^2 p(x))^{-1}} \leq \sqrt{m}$.*

Proof. Recall that $\bigtriangledown^2 p(x) = \mathbf{A}^\top \mathbf{S}_x^{-2} \mathbf{A}$ and $\bigtriangledown p(x) = \mathbf{A}^\top \mathbf{S}_x^{-1} \vec{1}$ consequently we have that

$$\| \bigtriangledown p(x)\|_{\bigtriangledown^2 p(x)^{-1}} = \sqrt{\vec{1}^\top \mathbf{P} \vec{1}} \text{ where } \mathbf{P} = \mathbf{S}_x^{-1} \mathbf{A} \left(\mathbf{A}^\top \mathbf{S}_x^{-2} \mathbf{A}\right)^{-1} \mathbf{A}^\top \mathbf{S}_x^{-1} .$$

Now $\mathbf{P}$ is a symmetric matrix PSD matrix with $\mathbf{P}^2 = \mathbf{P}$, i.e. it is a projection matrix, and therefore $\mathbf{P} \preceq \mathbf{I}$. Since $\sqrt{\vec{1}^\top \mathbf{I} \vec{1}} = \sqrt{m}$ we have the desired result. □

Using this we bound how much the Newton decrement changes as we change t.

Lemma 13. *For all $x \in P_0$ and $t > 0$ and $t' = (1 + \alpha)t$ we have that*

$$\delta_{t'}(x) \leq (1 + \alpha) \cdot \delta_t(x) + \alpha\sqrt{m} .$$

Proof. Recall that $f_t(x) = t \cdot c^\top x + p(x)$ and $\bigtriangledown f_t(x) = t \cdot c + \bigtriangledown p(x)$. Consequently

$$\bigtriangledown f_{t'}(x) = (1 + \alpha)t \cdot c + \bigtriangledown p(x) = (1 + \alpha) \bigtriangledown f_t(x) - \alpha \bigtriangledown p(x) .$$

Therefore, we have that

$$\begin{aligned} \delta_{f_{t'}}(x) &= \| \bigtriangledown f_{t'}(x)\|_{\bigtriangledown^2 f_{t'}(x)^{-1}} = \|(1 + \alpha) \bigtriangledown f_t(x) - \alpha \bigtriangledown p(x)\|_{\bigtriangledown^2 f_{t'}(x)^{-1}} \\ &\leq (1 + \alpha) \cdot \delta_{f'}(x) + \alpha \| \bigtriangledown p(x)\|_{\bigtriangledown^2 p(x)^{-1}} . \end{aligned}$$

The result follows from Lemma 12

$$\delta_{f_{t'}}(x) = \| \bigtriangledown f_{t'}(x)\|_{\bigtriangledown^2 f_{t'}(x)^{-1}}$$

□

From this we see that we can change t' by a constant factor with $O(\sqrt{m})$ iterations of Newton.

Lemma 14. *Suppose we have $x_0 \in P_0$ and $t_0 > 0$ such that $\delta_{t_0}(x_0) \leq \frac{1}{32}$ and the for all $k \geq 0$ let $t_{k+1} = (1 + \frac{1}{16\sqrt{m}})t_k$ and $x_{k+1} = x_k - \bigtriangledown^2 f_{t_{k+1}}(x)^{-1} \bigtriangledown f_{t_{k+1}}(x_k)$ then for all k we have $x_k \in P_0$ and $\delta_{f_{t_k}}(x_k) < \frac{1}{32}$ and $t_k = (1 + \frac{1}{16\sqrt{m}})^k t_0$.*

Proof. Note that by Lemma 13 we have that for all $k \geq 0$

$$\delta_{t_{k+1}}(x_k) \leq \left(1 + \frac{1}{16\sqrt{m}}\right) \cdot \delta_{t_k}(x_k) + \frac{1}{16\sqrt{m}} \cdot \sqrt{m} \leq 2 \cdot \frac{1}{32} + \frac{1}{16} = \frac{1}{8} .$$

Furthermore, by Lemma 10 we have that

$$\delta_{t_{k+1}}(x_{k+1}) \leq 2 \cdot \delta_{t_{k+1}}(x_k)^2 \leq \frac{2}{64} \leq \frac{1}{32} .$$

□

Consequently we see that if t_0 is a constant and we have x_0 with $\delta_{t_0}(x_0) \leq \frac{1}{32}$ then we can compute x_k with $\delta_{t_k}(x_k) \leq \frac{1}{32}$ for $t_k \geq 2^{\frac{1}{\epsilon}}$ with $O(\sqrt{m} \log \epsilon^{-1})$ iterations of Newton's method. Thus we can move along the central path quickly.

To use the results of the previous section to solving a linear programming, all we are missing is analysis of how far down the central path we need to go to obtain a good cost, how close we need to be to the path, and how to get an initial point.

We start by bounding the cost of central path points.

Lemma 15. *For all $t > 0$ we have that $c^\top(x^{(t)} - x_*) \leq \frac{m}{t}$.*

Proof. We know that

$$\vec{0} = \triangledown f_t(x^{(t)}) = t \cdot c + \triangledown p(x^{(t)}) = t \cdot c - \mathbf{A}\mathbf{S}_{x^{(t)}}^{-1}\vec{1}$$

consequently

$$t \cdot c^\top(x^{(t)} - x_*) = \vec{1}^\top \mathbf{S}_{x^{(t)}}^{-1} \mathbf{A}(x^{(t)} - x_*) = \vec{1}^\top \mathbf{S}_{x^{(t)}}^{-1}(s(x^{(t)}) - s(x_*)) \leq m\,.$$

□

Next, we bound the quality of a point near the central path.

Lemma 16. *For all $x \in P_0$ and $t > 0$ such that $\delta_{f_t}(x) < \frac{1}{4}$ we have*

$$c^\top(x - x^{(t)}) \leq \frac{1}{t}\left[m + 4\delta_t^2(x)\right]$$

Proof. Since $\triangledown f_t(x) = t \cdot c - \mathbf{A}\mathbf{S}_x^{-1}\vec{1}$ we have that

$$c^\top(x - x^{(t)}) = \frac{1}{t}\left(\triangledown f_t(x) + \mathbf{A}\mathbf{S}_x^{-1}\right)^\top(x - x_t) = \frac{1}{t} \triangledown f_t(x)^\top(x - x_t) + \mathbf{S}_x^{-1}(s_x - s_{x_t})\,.$$

However, by Lemma 11 then $\|x - x_t\|_{\triangledown^2 f_t(x)} \leq 4 \cdot \delta_t(x)$ and

$$\triangledown f_t(x)^\top(x - x_t) = \triangledown f_t(x) \triangledown^2 f_t(x)^{-1/2} \triangledown^2 f_t(x)^{1/2}(x - x_t) \leq \| \triangledown f_t(x)\|_{\triangledown^2 f_t(x)^{-1}} \cdot \|x - x_t\|_{\triangledown^2 f_t(x)} \leq 4\delta_t^2(x)$$

and the result follows. □

Putting these lemmas together we see that to get an ϵ-approximate solution we just need to get $t = \Omega(\frac{m}{\epsilon})$ which we have shown we can do with $O(\sqrt{m}\log(\epsilon^{-1}))$ iterations of Newton's method.

All that remains is to show how to get an initial point on the central path. Let's suppose we have some $x_0 \in \text{int}(P)$ so that $\triangledown p(x_0)$ is finite. Note that if we let $c' = \triangledown p(x_0)$ and let $f_t'(x) = t \cdot (c')^\top x + p(x)$ then we have that $\triangledown f_t'(x) = t \cdot (c') + \triangledown p(x)$ and therefore $x_0 = \text{argmin}_{x \in P} f_1'(x)$. In other words, x_0 is on the central path for a different cost function. Now, careful inspection of our previous analysis shows that we can decrease t at more or less the same rate as we can increase it. Consequently, we can compute $\text{argmin}_x f_t'(x)$ for very small t and then switch the cost function to the true cost function. In other words, we can follow this new central path to the center of the polytope and then follow the true central path down to the minimizer. Overall this gives us that we just need to solve $O(\sqrt{m}\log(\alpha/\epsilon))$ linear systems where α depends on how close to the center we need to get and how close to the boundary we start. Note that for many linear programs this α value can be bounded more directly.

8 Generalizing

How can we generalize the results in the previous sections? Suppose more broadly we want to solve

$$\min_{x \in S} c^\top x$$

for some convex set S and we are given some $x_0 \in S_0 \stackrel{\mathrm{def}}{=} \mathrm{int}(S)$. Our approach is essentially the same, we just need to find some barrier function p for S and then we can just minimize

$$\min_{x \in S} t \cdot c^\top x + p(x)\,.$$

However, with some thought and care, we can see that we only really used two properties of p in our analysis. The first is that if $x \in P_0$ and $\|x - y\|_{\bigtriangledown^2 p(x)} \leq \alpha < 1$ then $y \in P_0$ and $(1-\alpha)^2 \bigtriangledown^2 p(y) \preceq \bigtriangledown^2 p(x) \preceq (1+\alpha)^2 \bigtriangledown^2 p(y)$. The second is that $\| \bigtriangledown p(x)\|_{\bigtriangledown^2 p(x)} \leq \sqrt{\nu}$ for some value of ν. There is a slightly more general condition that implies and it is known as ν-self-concordance. There is a broad theory for construction self-concordant barrier function and our analysis shows that given a ν-self-concordant barrier we can compute an ϵ-suboptimal point with roughly $O(\sqrt{v} \log(1/\epsilon))$ linear system solve. Furthermore, it can be shown that every n-dimensional convex set admits a $O(n)$-self-concordant barrier and thus cutting plane methods are always improvable in theory.