MS&E 213 / CS 269O : Chapter 1
Introduction to "Introduction to Optimization Theory"*

By Aaron Sidford (sidford@stanford.edu)

October 1, 2020

1 What is this course about?

The central problem we consider throughout this course is as follows. We have a function $f : \mathbb{R}^n \to \mathbb{R}$ and a set $S \subseteq \mathbb{R}^n$ and we wish to minimize f over the points in S. We refer to f as our objective function and we refer to S as the *constraint set* or *feasible region*. We call a point $x \in S$ *feasible* or refer to the condition that $x \in S$ as the *constraints* of the problem and our goal is to minimize the objective function subject to the constraints.

$$\min_{x \in S} f(x) \tag{1.1}$$

This is known as *constrained function minimization* and will be the focus of the class (though frequently we will consider the case where $S = \mathbb{R}^n$, in which case we call the problem *unconstrained function minimization*).

These problems are incredibly well-studied and prevalent. It is hard to think of cases where we have access to a lot of data and then do not want to optimize over that data set, finding the best thing about it in some way. Maybe we have purchase history for some website and we want to know what is the most popular or we want to fit a linear model too it. Maybe we have the map of the world and we want to know the minimum path from one point to another. Maybe we have a bunch of constraints on how we can spend our time, utilities for what we do, and want to find the best schedule. Optimization broadly concerns the problem of finding such optimal solutions and each of these can be easily modeled as such a constrained minimization problem.

The focus of this class is how to solve these optimization problems *efficiently* while making *minimal assumptions* about f (and possibly S). The key question we will ask in the class is under a certain set of mild assumptions on S what algorithm should we design to solve the problem as quickly as possible. Note that there is a lot freedom in these definitions. Beyond the assumptions we made to model our optimization problem as a constrained minimization problem, there are many assumptions we could make on how we get access to f, what we mean by efficiency, and what we assume about f. In the remainder of this section, we will clarify the way in which we will think of optimization in this class and begin to answer these questions.

*These notes are a work in progress. They are not necessarily a subset or superset of the in-class material, there may also be occasional *TODO* comments which demarcate material I am thinking of adding in the future, and citations are often omitted. These notes are intended converge to a superset of the class material that is TODO-free with a more complete set of citations and pointers to the literature. 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 Piazza or contact me directly at sidford@stanford.edu.

1.1 Why constrained function minimization?

Though we will often phrase optimization problems as constrained or unconstrained function minimization problems, there many other ways we could have defined optimization problems. However, for many formulations it can be shown that they are just as expressive and equally difficult to solve.

For example, we could have instead defined our fundamental optimization problem as *constrained function maximization*, where again, given $g : \mathbb{R}^n \Rightarrow \mathbb{R}$ and $S \subseteq \mathbb{R}^n$ the goal is to solve

$$\max_{x \in S} g(x) \,. \tag{1.2}$$

However, given an general purpose algorithm for solving constrained minimization, (1.1), we can easily use it to solve constrained maximization, (1.2), by simply applying the constrained minimization method to the function $f(x) = -g(x)$. Since

$$\min_{x \in S} f(x) = \min_{x \in S} -g(x) = -\max_{x \in S} g(x)$$

we see that a minimizing $x \in S$ for a f is a maximizing $x \in S$ for g and the values of the objectives are relatable by just changing the sign. Similarly, we could apply a constrained maximization method to solve constrained minimization by applying the method to the function $g(x) = -f(x)$.

This idea of relating the difficulty of two different optimization problems by demonstrating that methods for solving them apply to each, is an instance of a general technique of problem *reduction*. This is a technique we will encounter multiple times in this course. A little more formally, we say that problem A can be reduced to problem B if methods for solving B can be efficiently applied to solve instances of problem A. This shows that B is, in some sense, more difficult to solve then A, as we can use solutions for B to solve A. In the preceding example we reduced constrained function minimization and maximization to each other.

These reductions can be somewhat surprising in certain cases. For instance, at first glance constrained minimization, seems more difficult then unconstrained constrained minimization (since it is a special case). However, unconstrained minimization can actually be reduced efficiently to constrained minimization. To see this, for arbitrary constrained minimization problem (1.1) define the objective function $g : \mathbb{R}^n \to \mathbb{R}$ by

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

For such objective function g we see that $\min_{x \in S} f(x) = \min_{x \in \mathbb{R}} g(x)$, as any $x \in \mathbb{R}$ for which $g(x) < \infty$ satisfies $x \in S$.[1] Consequently, any unconstrained minimization method can be applied to g to solve the original constrained problem.

Such reductions will appear a few times in this class. Note that in some cases, as we just saw for reducing constrained to unconstrained function minimization, these reductions may obfuscate some salient properties of the original problem. Another recurring theme in this course will be in identifying the utility of such structure and possibly leverage it to obtain more efficient optimization methods.

1.2 How do we access f?

In order to reduce the assumptions we make about f, throughout this course we will typically assume we only have fairly restrictive access to f. For much of this class, rather than assuming we have f and S specified explicitly, we will instead assume we only have restrictive access to information of f. In particular, typically we will instead only assume that we can access f through invocation of some procedure that gives us only limited information of f at a certain point. Formally, we will call this procedure an *oracle* and we will refer to an invocation of the procedure as a *query* to the oracle. When we only access f through an oracle we will say we only have *oracle access* to f.

[1] Note, technical $\infty \notin \mathbb{R}$, however this reduction still is correct if ∞ is replace with any value strictly larger than $\min_{x \in S} f(x)$.

Three standard oracle assumptions we encounter throughout the class are the following.

Definition 1 (Value Oracle)**.** A *value oracle*, also known as an *evaluation oracle* or a *0-th order oracle* for a function $f : \mathbb{R}^n \to \mathbb{R}$ is a procedure that when queried with a point $x \in \mathbb{R}^n$ outputs the value of f at x, i.e. $f(x)$.

Definition 2 (Gradient Oracle)**.** A *gradient oracle* for a function $f : \mathbb{R}^n \to \mathbb{R}$ is a procedure that when queried with a point $x \in \mathbb{R}^n$ outputs the gradient of f at x, i.e. $\bigtriangledown f(x) \in \mathbb{R}^n$ where $[\bigtriangledown f(x)]_i = \frac{\partial}{\partial x_i} f(x)$.

Definition 3 (First Order Oracle)**.** A *first order oracle* for a function $f : \mathbb{R}^n \to \mathbb{R}$ is a procedure that when queried with a point $x \in \mathbb{R}^n$ outputs both the value of f at x, i.e. $f(x)$, as well as the gradient of f at x, i.e. $\bigtriangledown f(x) \in \mathbb{R}^n$ where $[\bigtriangledown f(x)]_i = \frac{\partial}{\partial x_i} f(x)$.

Note that there are many oracles one can encounter in different situations. For example, a natural oracle in many settings is the following:

Definition 4 (Stochastic Gradient Oracle)**.** A *stochastic gradient oracle* for a function $f \in \mathbb{R}^n$ is a procedure that when given a query point $x \in \mathbb{R}^n$ outputs a random vector $e \in \mathbb{R}^n$ such that $\mathbb{E}e = \bigtriangledown f(x)$ where here the expectation is over the randomness of the oracle.

Note that this oracle is a reasonable abstraction for problems where we have an overwhelming amount of data, want to optimize something about the whole, and can only sub-sample pieces. This occurs in many large scale learning problems. For example, we could imagine that we have n data points, associate each $x \in \mathbb{R}^n$ with a model (or its parameters), and let $f_i(x)$ denote how well the model given by x explains or predicts data point i. The objective, $f(x) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i \in [n]} f_i(x)$, is then average prediction error or loss when using x to predict a data point. Whereas implementing a gradient oracle would involve computing $\frac{1}{n} \sum_{i \in [n]} \bigtriangledown f_i(x)$, i.e. looking at each data point, if instead we just wanted a stochastic gradient oracle, we could just pick a random $i \in [n]$ and use $\bigtriangledown f_i(x)$ as such a random gradient, which would involve just looking at a single data point. Given its prevalence and utility we will work with this oracle model more later in the course.

1.3 Why the oracle model?

In this course, we will often work with the oracle model for a number of reasons. First, it helps reduce the number of assumptions we are making when designing optimization methods. By restricting methods to use only oracle access to the function this may help us clarify what structure is enabling our methods to work and what additional assumptions are needed to obtain improved efficiency. Further, in working with the oracle model in many cases it is possible to prove lower bounds, showing that any method requires a certain number of queries. Consequently, in many cases we can obtain methods that provably near-optimal use an oracle. This can help clarify both the theoretical analysis of different optimization settings and give a framework to leverage towards understanding why certain methods may or may not be effective in practice.

The oracle model may also be beneficial in a number of scenarios that could occur in modeling reality as an optimization problem, when one does not have explicit access to the problem. There are many optimization problems for which exactly specifying the optimization problem can be quite complicated or impossible, though we can obtain access to components of the objective and constraint set. For example, imagine we are trying to set prices, make recommendations, set schedules and f is the total utility obtained when making these decisions. In some setting we may be only able to truly evaluate f by actually making thee decisions and seeing what happens. As another example, the data may be so large or noisy that we cannot interact with the data directly. Maybe we only have stochastic or adversarial noisy access, in this case the oracle model is again natural. In each of these cases, the oracle model can help encapsulate these problems and and provide a mathematical mode for reasoning about how to efficiently optimize given incomplete information.

Even in cases where we have all the information about f the oracle model may be useful for informing the design of new efficient algorithms. For example, given a massive data set accessing the entire data set may be computationally expensive, but sampling a subset of it might be much more efficient. Consequently, by designing methods in oracle models compatible with such sampling access, more efficient methods maybe be obtained.

The oracle model can also be useful for understanding the performance of existing optimization methods for explicit functions. For example, consider a simple, but more-structured optimization problem like *regression*, where we are given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and a vector $b \in \mathbb{R}^n$ and wish to solve, $\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2}\|\mathbf{A}x - b\|_2^2$. When designing algorithms for this problem it can be helpful to understand when having a simple procedure just based on the gradient (e.g. the ability to apply $\mathbf{A}$ and $\mathbf{A}^\top$ to vectors, as $\bigtriangledown f(x) = \mathbf{A}^\top(\mathbf{A}x - b)$) and when it might be important to leverage further algebraic structure of $\mathbf{A}$.Analyzing algorithms through this lens of oracle models can help for this purpose.

In short, oracles provide a nice way to extract the salient information about a problem we wish to use for algorithm design. They also provide a good lens to think about the computational complexity of optimization. Whereas for an explicitly given function, if optimizing it is difficult, i.e. takes a lot of time, proving this is currently typically outside the realm of what we know how to do computational complexity theory. However, if we assume oracle access to the input, in many cases it is possible to obtain provable lower bounds on how many oracle calls are required to obtain a certain accuracy on optimization problems. Thus we may hope for tight bounds on optimization problems under oracle models; ultimately this helps inform us as to what an algorithm can or should be exploiting or not.

1.4 What do we want to do?

Once we have specified our oracle model we wish to design algorithms to minimize our objective function f as efficiently as possible or prove that more efficient minimization is impossible. There are two key questions we need to address here (1) what exactly do we mean by *minimize* and (2) what do we mean by *efficiency*.

There is a serious choice to be made in answering each of these questions. The issue of (1) is important as for almost all the problems we see in this course we will rarely compute an exact minimum value or minimizing point of our objective function (as we will rarely know it exactly). Instead, we will be providing an algorithm that is approximately minimum or optimal in some sense. This is somewhat essential as in the oracle model it is rare to know you are at the exact minimum or optimal value, in some sense the input may just be too noisy. One common notion we will use to parameterize our distance from optimality is the following.

Definition 5 (ϵ-optimal point)**.** For $f : \mathbb{R}^n \to \mathbb{R}$ we call $x \in \mathbb{R}^n$ an *ϵ-optimal point*[2] for some $\epsilon > 0$ if

$$f(x) - f_* \leq \epsilon \text{ where } f_* \stackrel{\text{def}}{=} \inf_{x \in \mathbb{R}^n} f(x)\,.$$

Note that this is a strong notion of optimality as it is global. Although we will achieve this in many cases in other cases we may only obtain local optimality perhaps as measured by the first few moments of the function. For example, we may try to compute an ϵ-critical point.

Definition 6 (ϵ-critical point)**.** For $f : \mathbb{R}^n \to \mathbb{R}$ we call $x \in \mathbb{R}^n$ an *ϵ-critical point* for some $\epsilon > 0$ if $\| \bigtriangledown f(x)\|_2 \leq \epsilon$ where $\|x\|_2 \stackrel{\text{def}}{=} \sqrt{\sum_{i \in [n]} x_i^2}$.[3]

Note that a critical point might not be maximal or minimal apriori but if we have an algorithm that decreases the function value and finds a critical point, this may be a decent proxy for a local minima, that is a point where no direction makes progress if we move in that direction infinitesimally.

[2]This may sometimes be referred to instead as an ϵ-suboptimal point as larger values of ϵ imply that the quality of the point is worse.

[3]Note, other norms could be considered here, but we will focus on the ℓ_2 norm in the majority of the course.

These are just a few notions of optimality we will consider. At various times in the course we may also measure distance to optimal solutions to the problem.

Next, (2), what do we mean by efficiency? The running time of our algorithms are going to have essentially two components (1) how many times did we invoke our oracle, i.e. what is the *oracle complexity* or *query complexity* of the problem[4], and (2) what was the computational complexity of the work we did to manipulate the results of calling the oracle. Note that both of these are important and it is not always clear which is more critical. For example, in the case of setting prices, the oracle calls could be incredibly expensive, however as the size of data gets large, paying running times that scale polynomially can be prohibitively expensive (if the polynomial is large). We will attempt to minimize each and present our running time in terms of both pieces. However, our emphasis will typically be on oracle complexity.

1.5 What do the algorithms look like?

Since we just have oracle access, just have local information, most algorithms we will see are essentially greedy local methods. They run some simple iterative procedure: essentially, query the oracle, make an improvement and repeat. Such algorithms are known as *iterative methods* and despite the simplicity of the procedure efficient iterative methods and their analysis can be quite complicated. This comes from the fact that sometimes to get the best performance somewhat complicated performance measures need to be used. Both finding the best thing to do with all the information available can be complex and finding the best proxies for progress can be difficult. Figuring out how to design algorithms and these potential functions for analysis is an integral part of this course. While we will start with fairly simple potential functions, they will get more complicated as the course progress.

This class can in fact be viewed, in part, as a introduction to the theory of iterative methods. These are powerful tools in designing fast algorithms for all sorts of problems (both continuous and discrete). They have been the workhorse in practice behind many machine learning algorithms and they have recently used to get numerous breakthrough results in the theory of computation. This is also an incredibly active area of research that this course should equip you to go into.

1.6 What do the problems look like?

So far we have summarized what the course will be about. We will introduce some broad class of objective functions f, specify an oracle model for accessing f, design an iterative method for optimizing f, and characterize the performance of the method in terms of the number of oracle calls (typically the number of iterations of the algorithm) and the total computational cost (typically the work per iteration). By doing this we will have a good sense of the computational boundaries for continuous optimization.

What we haven't specified is what exactly these classes of optimization problems will look like. In the remainder of this chapter we will consider some basic classes of f and apply this lens. This will give us a feel for what kind of assumptions are natural. We will conclude these notes with a summary of the main problem classes we will consider in the future.

2 General Unconstrained Optimization and Lower Bounds

Let us start with a very general class of unconstrained function minimization problems which we call *bounded unconstrained function minimization in 1 dimension*. Suppose we have an objective function $f : \mathbb{R}^n \to \mathbb{R}$ and we wish to minimize it. Further, suppose we just have an value oracle for f, i.e. for any $x \in \mathbb{R}^n$ we can compute $f(x)$ with a single query. Lastly, suppose we are even promised that the minimizer of f lies in

[4]Note that this is different than the computational complexity of the oracle, which would be how expensive the oracle is to implement.

$[0,1]$, i.e. there is $x_* \in [0,1]$ with $f(x_*) = f_* \stackrel{\text{def}}{=} \inf_{x\in\mathbb{R}^n} f(x)$, and that $f(x) \in [0,1]$ for all $x \in \mathbb{R}$. What is the oracle complexity for this problem?

Unfortunately this problem seems quite difficult. It doesn't seem like information about any point gives you information about any other point in the domain. We can actually make this formal and show that there is no finite number T and deterministic method, i.e. a method where the points queried and what is output is a deterministic function of the input and results of the oracle, such that the method always outputs an ϵ-optimal point for any $\epsilon < 1$ in at most T steps. In other words, we will show that this problem has an infinite (deterministic) oracle complexity.[5]

While the impossibility of this optimization problem seems intuitive, proving this formally at first glance may seem a bit more challenging. Note proving this impossibility is a somewhat strong claim in that we want to give a lower bound for *all (deterministic)* methods. Formally, we wish to show that for any method and any value $T \geq 0$, there is function that meets the desired properties such that the the method will output the incorrect answer after T steps. What perhaps may be tricky in this analysis is that we need to show this for any method. While the focus of much of this class will be on designing optimization methods, the methods we introduce are informed by known lower bounds for many of the problems we consider. Consequently, it is beneficial to see how such lower bounds can be proven and we give some examples in this chapter.

Our approach to prove this lower bound is based on the idea of a *resisting oracle*, a natural and powerful strategy for proving lower bounds for deterministic methods.[6] We will show that we can provide a procedure for implementing the oracle such that the output is always consistent with some function f that meets the desired criteria to be a valid input function, but for which the output of the method is always incorrect. This approach may also be called an *adversarial oracle* as we will be implementing the oracle, i.e. choosing the function, adversarially in response to what the algorithm does. Ultimately, if we can design such an oracle then it will prove (and typically find) an input on which the algorithm fails.

To more formally prove the lower bound, we first consider the structure of an arbitrary (deterministic) method. Let $\mathcal{F}_{\text{input}}$ be the set of all functions that are "valid input" for the problem, i.e.

$$\mathcal{F}_{\text{input}} \stackrel{\text{def}}{=} \left\{ f : \mathbb{R} \to [0,1] \,|\, \exists x_* \in [0,1] \text{ such that } x_* = \inf_{x\in\mathbb{R}} f(x) \right\}$$

and for each valid input function, $f \in \mathcal{F}_{\text{input}}$, let $S_{\text{out}}(f)$ be the set of "valid output" for that f, i.e. the set of 1/2-optimal points $S_{\text{out}}(f) \stackrel{\text{def}}{=} \{x \in \mathbb{R} \,|\, f(x) \leq \inf_{x\in\mathbb{R}} f(x) + \frac{1}{2}\}$. An arbitrary method is simply given an oracle for $f \in \mathcal{F}_{\text{input}}$ and then queries points $x_1, x_2, ..., x_T$ and then outputs a point x_{out}. Each x_i is a function just of the oracles output on $x_1, ..., x_{i-1}$ and x_{out} is just a function $x_1, ..., x_T$ and the results of the oracle. The algorithm succeeds, i.e. solves the optimization problem, if x_{out} is a valid output point for f, i.e. x_{out} is 1/2-optimal, or equivalently $x_{\text{out}} \in S_{\text{out}}$, and fails otherwise.

Given the discussion of the preceding paragraph, if we wish to show that T queries are insufficient to solve the optimization problem, it suffices to show that for any algorithm there is some $f \in \mathcal{F}_{\text{input}}$ for which the output of the oracle at $x_1, ..., x_T$ is consistent with f being the input and yet, $x_{\text{out}} \notin S_{\text{out}}(f)$. Note that to show this it suffices to show that there are two $f_1, f_2 \in \mathcal{F}_{\text{input}}$ such that the output of the oracle is consistent with the input being either f_1 and f_2 and the set of valid points to output for f_1 and f_2 are disjoint, i.e. $S_{\text{out}}(f_1) \cap S_{\text{out}}(f_2) = \emptyset$. This would also shows that more than T queries are needed since no matter what x_{out} is, it will be incorrect for one of f_1 or f_2, and therefore the algorithm will output an incorrect answer on one of the two functions as input.

With this in mind, the idea of the resisting oracle is to specify what the output of the oracle is, as a function of the points queried and then show that so long as T is not too large, there are two valid functions consistent with the oracle output, such that set of valid output points for the two functions are disjoint. In other words,

[5] The same is true for randomized algorithms, but proving lower bounds against such algorithms is outside the scope of the course (but I am happy to discuss in office hours or over Piazza).

[6] There is a variety of literature on proving lower bounds for randomized algorithms as well, however that will not be discussed at length in this course.

by giving such a method for implementing the oracle, i.e. a *resisting oracle*, we can hope to find a function on which the method fails.

Let's apply this strategy to show that no algorithm can always output the correct answer for bounded unconstrained function minimization in 1 dimension in T queries for any finite T. As discussed, to show this we want a family of valid input function, i.e. a subset of $\mathcal{F}_{\text{input}}$, where many functions in the class disjoint valid output. For our problem of bounded unconstrained function minimization in 1 dimension, for all $z \in [0,1]$ we let $f_z : \mathbb{R} \to \mathbb{R}$ be defined for all $x \in \mathbb{R}$ as follows

$$f_z(x) = \begin{cases} 1 & \text{if } x \notin z \\ 0 & \text{if } x = z \end{cases}. \tag{2.1}$$

Note, that clearly each $f_z \in \mathcal{F}_{\text{input}}$. Further the minimizer of f_z is 0 and the only ϵ-approximate minimizer of f_z for any $\epsilon < 1$ is the point z. Consequently, $S_{\text{out}}(f_z) = \{z\}$ and therefore for any $z_1 \neq z_2$ there is no output that is valid for both f_{z_1} and f_{z_2}, i.e. $S_{\text{out}}(f_{z_1}) \cap S_{\text{out}}(f_{z_1}) = \emptyset$.

Leveraging these properties of f_z, consider the resisting oracle always outputting 1, i.e. consider an execution of the algorithm where the oracle returns 1 as the value of the function at every point queried. If the points queried are $x_1, ..., x_T$ then the output of the oracle is consistent with f_z for all $z \notin \{x_1, ..., x_T\}$, i.e. $f_z(x_i) = 1$ for all such z and $i \in [T]$. Consequently, there are an uncountably infinite number of $z \in [0,1]$ where f_z is consistent with the points queried with disjoint 1/2-optimal points (we only needed two) and therefore there is some input on which the algorithm outputs the incorrect answer f_z for $z \notin \{x_1, ..., x_T\} \cup \{x_{\text{out}}\}$.

The idea of implementing the oracle to prove a lower bound may be somewhat counter-intuitive at first glance, but it is a useful lower bounding technique. In the following lemma, we collect the ideas above to provide a short formal proof that it is impossible for any algorithm to always output a ϵ-optimal point for any $\epsilon < 1$ in T queries in any finite T.

Lemma 7. *There is no finite value $T > 0$ such that there is a deterministic algorithm which when given a value oracle for $f : \mathbb{R} \to \mathbb{R}$ that has the property that the minimizer of f lies in $[0,1]$ and has $f(x) \in [0,1]$ for all $x \in \mathbb{R}$ outputs an ϵ-approximate minimizer for $\epsilon < 1$ in at most T queries.*

Proof. Consider running an arbitrary method for T steps to solve this problem in the setting where in response to each query the oracle returns 1. Let $x_1, ..., x_T$ be the points queried and x_{out} be the point returned. Note that there is exists some $z \in [0,1]$ with $z \notin \{x_1, ...x_t\} \cup \{x_{\text{out}}\}$ and f_z, defined as in (2.1), is a valid input function consistent with the oracle returning 1 at each point. However, x_{out} is not ϵ-optimal for f_z and therefore the output of the algorithm is incorrect when f_z is the input. □

In other words, fully general optimization with a finite number of queries, even in one dimension, is impossible. This example highlights that in this general setting optimization is essentially the same as finding the smallest element in an uncountably infinite set of elements using only a finite number of queries to their values and therefore, impossible to do with a finite number of queries. Note we could try to changer our optimality criteria, i.e. seek a critical point, but again the function could easily be non-differentiable. In the next section we will discuss a different way to change the assumptions of the problem so that approximate minimization in a finite number of queries is possible.

3 Continuous Function Minimization and (Near)-Optimal Oracle Complexity

The difficulty of the problem in the previous section stemmed from the fact that the value of the function at a point told us very little information about the function. It didn't even give us any local information about the function. To get around this issue we could try assuming just a little more structure on f, we could

assume that f does not change too quickly, or more precisely that f is continuous. This is a class primarily about continuous optimization, the efficient optimization over infinite sets with some continuity structure (as opposed to discrete optimization where finite sets are considered).

Unfortunately, simply adding the assumption that f is continuous in the previous section would be insufficient to make the query complexity finite (as we will show later in this section). Instead, to obtain methods which compute approximately optimal points with a finite number of queries, we instead add the assumption that f is continuous in a quantifiable sense. In particular, in this section we will assume that we can bound the change in function value by any two points x and y by their distance in some norm $\|\cdot\|$, i.e. $\|x-y\|$. This assumption is known as Lipschitz continuity and we define it and recap the definition of norm briefly below.

Definition 8 (Lipschitz Continuous)**.** We say a function $f : \mathbb{R}^n \to \mathbb{R}$ is *L-Lipschitz (Continuous) with respect to a norm* $\|\cdot\|$ if for all $x, y \in \mathbb{R}^n$ we have

$$|f(x) - f(y)| \leq L \cdot \|x - y\| .$$

Definition 9 (Norm)**.** Recall that $\|x\|$ for $x \in \mathbb{R}^n$ is called a norm on $\mathbb{R}^n$ if the following hold, (1) (absolute homogeneity) $\forall \alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$ it holds $\|\alpha x\| = |\alpha| \cdot \|x\|$ (2) (triangle inequality) $\forall x, y \in \mathbb{R}^n$ it holds $\|x + y\| \leq \|x\| + \|y\|$, (3) $\|x\| = 0$ if and only $x = \vec{0}$. If conditions (1) and (2) hold (but not necessarily (3)) we call $\|\cdot\|$ a *semi-norm.*

Typically in this class we will take $\|\cdot\|$ to be the Euclidean norm, that is $\|x\| = \|x\|_2 \stackrel{\text{def}}{=} \sqrt{\sum_{i\in[n]} x_i^2}$. Whenever the norm is not specified or explicitly stated to be general, we will assume that it is the Euclidean norm. Occasionally we will work with other Schatten p-norms for $p \geq 1$, i.e. $\|x\|_p \stackrel{\text{def}}{=} (\sum_{i\in[n]} |x_i|^p)^{1/p}$ with $\|x\|_\infty \stackrel{\text{def}}{=} \max_{i\in[n]} |x_i|$. Note that all norms are equivalent up to scaling in finite dimensions and thus if we use a norm ever to define a limit it doesn't matter, but if we use it in the definition of Lipschitz, it my affect the actual value of L.

Now the question we ask in this section is how many oracle calls are need to compute an ϵ-optimal point for a L-Lipschitz function, where we have a bound on a minimizer of the function. Before addressing this question we first study the structure of L-Lipschitz functions (Section 3.1), then we will use this to obtain upper and lower bounds for 1-dimensional Lipschitz function minimization (Section 3.2), and then we discuss generalizations to higher dimensions (Section 3.3)

3.1 Structure of L-Lipschitz Functions

Before designing methods and lower bounds for Lipschitz function minimization, here we first study the structure of Lipschitz functions a little more. This is a common approach we will apply when considering new class of functions to optimize.

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is L-Lipschitz with respect to some norm. What does this imply about the function? The assumption implies that for all $x, y \in \mathbb{R}^n$ we can bound the change in f between x and y, i.e. $|f(x)-f(y)|$, by the distance between x and y in the norm, i.e. $\|x - y\|$. Formally, this states that

$$|f(x) - f(y)| \leq L\|x - y\| \text{ for all } x, y \in \mathbb{R}^n .$$

Note that by definition of absolute value, this is equivalent to stating that

$$-L\|x - y\| \leq f(x) - f(y) \leq L\|x - y\| \text{ for all } x, y \in \mathbb{R}^n .$$

By re-arranging this in turn is equivalent to stating that s

$$f(x) - L\|x - y\| \leq f(y) \leq f(x) + L\|x - y\| \text{ for all } x, y \in \mathbb{R}^n .$$

Consequently, this assumption is equivalent to say that at any point x by querying a value oracle, i.e. computing $f(x)$, we can obtain global upper and lower bounds on the function, i.e. $f(x) + L\|x - y\|$ and $f(x) - L\|x - y\|$. In other words, one oracle query provides these global constraints on what the function can look like.

In 1-dimension when the norm is $\|\cdot\|_p$ for any p, this norm is the same as the absolute value, $|\cdot|$ and by letting $y = x + d$ for arbitrary d we see that for $f : \mathbb{R} \to \mathbb{R}$ the assumption of being L-Lipschitz is equivalent to the assumption that

$$f(x) - L|d| \leq f(x + d) \leq f(x) + L|d| .$$

In other words, the assumption states that the function changes at rate at most L. This is suggestive that Lipschitz continuity is in some sense a bound on the slope, or first derivative of a function. Indeed, that is the case and later in the course we will prove the following characterization of Lipschitz differentiable functions.

Lemma 10. *Differentiable $f : \mathbb{R}^n \to \mathbb{R}$ is L-Lipschitz with respect to a norm $\|\cdot\|$ if and only if $\|\bigtriangledown f(x)\|_* \leq L$ for all $x \in \mathbb{R}^n$ where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$. (See the appendix or later chapters for definition.)*

3.2 1-Dimensional Lipschitz Function Minimization

With the structure of Lipschitz functions more firmly established, let's return to our example. Consider the following problem which we will call *1-dimensional Lipschitz function minimization* where we are given a value oracle for L-Lipschitz $f : \mathbb{R} \to \mathbb{R}$ that is minimized at a point in $[0, 1]$ we wish to compute an ϵ-optimal point for $\epsilon > 0$. This is the same as the *bounded unconstrained function minimization in 1 dimension* problem we considered in the previous section, except that we added the assumption that f is L-Lipschitz (with respect to ℓ_2, which in 1 dimension is equivalently $|\cdot|$) and we dropped the assumption that $f(x) \in [0, 1]$ for all x, as it does not affect the complexity of the methods we will develop.

What is the query complexity of this problem and how should we obtain an efficient method? As we discussed in the previous section, we know that if we move a point by δ that the value of the function can change by at most $L|\delta|$. Consequently, so long as we query a point within distance ϵ/L of a minimizer then so long as we return the point we query which has minimum value, we will obtain an ϵ-optimal point. Consequently, we can obtain a method that provably computes an ϵ-optimal point, just by querying enough points that every point in $[0, 1]$ is close enough to one of the points we query, and then returning the point of minimum value. We formally show that this works in the following lemma.

Lemma 11. *There is an algorithm which when given a value oracle for L-Lipschitz $f : \mathbb{R} \to \mathbb{R}$ with $f(x_*) = f_* = \inf_{x \in \mathbb{R}} f(x)$ for some $x_* \in [0, 1]$ and $\epsilon > 0$ computes an ϵ-optimal point with $\lceil \frac{L}{\epsilon} \rceil$ queries.*

Proof. Consider the method which queries the value oracle at the point i/k for all $i \in [k]$ and then returns the queried point with minimum value. Note that $|x_* - \frac{i_*}{k}| \leq \frac{1}{k}$ for some $i_* \in [k]$ and therefore $|f(i_*/k) - f_*| \leq \frac{L}{k}$. Consequently

$$f(i_*/k) = f_* + f(i/k) - f_* \leq f_* + \frac{L}{k} .$$

Consequently, for any $k \geq L/k$ the point i_*/k is ϵ-optimal. Further, since the point output by the method has value at most $f(i_*/k)$ we see that the output point is also ϵ-optimal. Therefore, picking $k = \lceil L/\epsilon \rceil$ yields the result. □

Now a natural question to ask is, is this optimal? Can we improve? To provide such a lower bound, we apply the resisting oracle framework described in the previous section. To do so, as discussed we want to find a large family of L-Lipchitz functions minimized in $[0, 1]$ with disjoint sets of ϵ-optimal points. To aid this construction, we first provide the following on how various operations affect whether a function is L-Lipschitz.

Lemma 12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that is L_f-Lipschitz with respect to a norm $\|\cdot\|$ and let $g : \mathbb{R}^n \to \mathbb{R}$ be function that is L_g-Lipschitz with respect to $\|\cdot\|$. Further, let $c \in \mathbb{R}$ and $\alpha \in \mathbb{R}^n$ be arbitrary. The following hold*

- *(1) $h : \mathbb{R}^n \to \mathbb{R}$ defined as $h(x) = f(x) + g(x)$ for all $x \in \mathbb{R}$ is $L_f + L_g$-Lipschitz with respect to $\|\cdot\|$.*
- *(2) $h : \mathbb{R}^n \to \mathbb{R}$ defined as $h(x) = \min\{f(x), g(x)\}$ for all $x \in \mathbb{R}$ is $\max\{L_f, L_g\}$-Lipschitz with respect to $\|\cdot\|$.*
- *(3) $h : \mathbb{R}^n \to \mathbb{R}$ defined as $h(x) = \max\{f(x), g(x)\}$ for all $x \in \mathbb{R}$ is $\max\{L_f, L_g\}$-Lipschitz with respect to $\|\cdot\|$.*
- *(4) $h : \mathbb{R}^n \to \mathbb{R}$ defined as $h(x) = c \cdot f(x - a)$ for all $x \in \mathbb{R}$ is $|c| \cdot L_f$-Lipschitz with respect to $\|\cdot\|$.*

Proof. (1) For all $x, y \in \mathbb{R}^n$ we have

$$h(y) \leq f(x) + L_f \cdot \|x - y\| + g(x) + L_g \cdot \|x - y\| = h(x) + (L_f + L_g)\|x - y\| .$$

Consequently, by symmetry $h(x) \leq h(y) + (L_f + L_g)\|x - y\|$ and the result follows.
(2) For all $x, y \in \mathbb{R}^n$ and monotonicity of min in its arguments we have

$$\begin{aligned} h(y) &\leq \min\{f(x) + L_f \cdot \|x - y\|, g(x) + L_g \cdot \|x - y\|\} \\ &\leq \min\{f(x) + \max\{L_f, L_g\} \cdot \|x - y\|, g(x) + \max\{L_f, L_g\} \cdot \|x - y\|\} \\ &= h(x) + \max\{L_f, L_g\} \cdot \|x - y\| \end{aligned}$$

and again the result follows by symmetry in x and y.
(3) For all $x, y \in \mathbb{R}^n$ and monotonicity of max in its arguments we have

$$\begin{aligned} h(y) &\leq \max\{f(x) + L_f \cdot \|x - y\|, g(x) + L_g \cdot \|x - y\|\} \\ &\leq \max\{f(x) + \max\{L_f, L_g\} \cdot \|x - y\|, g(x) + \max\{L_f, L_g\} \cdot \|x - y\|\} \\ &= h(x) + \max\{L_f, L_g\} \cdot \|x - y\| \end{aligned}$$

and again the result follows by symmetry in x and y.
(4) For all $x, y \in \mathbb{R}^n$ we have

$$\begin{aligned} |h(y) - h(x)| &= |c \cdot f(x - a) - c \cdot f(y - a)| = |c| \cdot |f(x - a) - f(y - a)| \\ &\leq |c| \cdot L_f \cdot \|(x - a) - (y - a)\| = |c| \cdot L_f \cdot \|x - a\| . \end{aligned}$$

□

This lemma provides a variety of tools for combining Lipschitz functions to create new ones, however it doesn't immediately yield non-trivial Lipschitz functions we can use to design our family of difficult functions to optimize. Constant functions, i.e. $f(x) = a$ for all x are trivially 0-Lipschitz (since $f(x) - f(y) = 0$) however to obtain our lower bound we will want a possibly more complex function. In the following lemma we show that norms are in fact always one Lipschitz in that norm. This lemma is also known as "reverse triangle inequality" and is a useful technical tool in general

Lemma 13 (Reverse Triangle Inequality)**.** *Any norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is 1-Lipschitz with respect to $\|\cdot\|$, i.e.*

$$|\|x\| - \|y\|| \leq \|x - y\| \text{ for all } x, y \in \mathbb{R}^n .$$

Proof. By triangle inequality

$$\|x\| - \|y\| \leq \|x - y + y\| - \|y\| \leq \|x - y\| + \|y\| - \|y\| = \|x - y\|.$$

Consequently, by symmetry $-(\|x\| - \|y\|) = \|y\| - \|x\| \leq \|x - y\|$ and combining yields the result. □

Leveraging Lemma 12 and Lemma 13 we can provide a simple family of functions to lower bound the oracle complexity for solving 1-dimensional Lipschitz function minimization. The approach we apply is a natural extension of the approach in the previous section. Whereas in the previous section the difficult functions we considered were constant everywhere except for a single point, here we consider functions that are constant everywhere except for a small region, within which we have the function change as quickly as possible while the function still being L-Lipschitz. This allows us to obtain functions that are constant over large regions, L-Lipschitz, but only have a small region that is ϵ-optimal. Further, the construction we obtain actually applies for any norm.

Lemma 14. *For any $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, $z \in \mathbb{R}^n$, and $\alpha \geq 0$ the function $f_{z,\alpha}(x)$ defined for all $x \in \mathbb{R}^n$ as*

$$f_{z,\alpha} \stackrel{\text{def}}{=} \min\{1, 1 - \alpha + L\|x - z\|\} \tag{3.1}$$

is L-Lipschitz with respect to $\|\cdot\|$. Further $f_{z,\alpha}(x) = 1$ for all x with $\|x - z\| \geq \frac{\alpha}{L}$ and if $\epsilon < \alpha$ a point $x \in \mathbb{R}^n$ is ϵ-optimal if and only if $\|x - z\| \leq \frac{\epsilon}{L}$.

Proof. By Lemma 13, $\|x\|$ is 1-Lipschitz of x with respect to $\|\cdot\|$. Further, by Lemma 12 this implies that $L\|x - z\|$ is L-Lipschitz with respect to $\|\cdot\|$. Further, since the constant function, $1 - \alpha$, is 0-Lipschitz applying Lemma 12 twice shows that $1 - \alpha + L\|x - z\|$ and $f_{z,\alpha}$ are both 1-Lipschitz with respect to $\|\cdot\|$.

Now, note that $f_{z,\alpha}(z) = 1 - \alpha$ and $f_{z,\alpha}(x) \geq 1 - \alpha$ for all x. Consequently, z is the minimizer and x is ϵ-optimal if and only if $f_{z,\alpha}(x) \leq 1 - \alpha + \epsilon$. However, since $\alpha > \epsilon$ this happens if and only if $1-\alpha+L\|x-z\| \leq 1-\alpha+\epsilon$ or equivalently $\|x-z\| \leq \frac{\epsilon}{L}$. Further, once $\|x-z\| \geq \frac{\alpha}{L}$ we have $1-\alpha+L\|x-z\| \geq 1$ and $f_{z,\alpha}(x) = 1$. □

Note that in the particular case of $n = 1$ and the norm being $|\cdot|$ this lemma implies that for any $\alpha \geq \epsilon$ there is a function that is L-Lipschitz and constant everywhere except for a region of length $2\alpha/L$ (i.e. where $\|x - z\| \geq \alpha/L$). Further, this implies that for any interval of length larger than $\frac{4\epsilon}{L}$ we can construct two L-Lipschitz functions which both have value 1 everywhere outside the interval and disjoint sets of ϵ-optimal points.

Lemma 15. *For $n = 1$, $\|\cdot\| = |\cdot|$ and any $c \in \mathbb{R}$ and $\alpha, L \geq 0$ the functions $f_{c+\frac{\alpha}{L},\alpha}$ and $f_{c-\frac{\alpha}{L},\alpha}$ defined as in 3.1 have disjoint sets of ϵ-optimal points for all $\epsilon < \alpha$ and have value 1 outside the interval $[c - \frac{2\alpha}{L}, c + \frac{2\alpha}{L}]$.*

Proof. By Lemma 14 the two functions are L-Lipschitz and have value that is not 1 in the intervals $[c, c+\frac{2\alpha}{L}]$ and $[c - \frac{2\alpha}{L}, c]$ respectively. Further, the ϵ-optimal points for the functions lie in the intervals $[c, c + \frac{2\epsilon}{L}]$ and $[c - \frac{2\epsilon}{L}, c]$ respectively. □

Using this we see that if we consider the resisting oracle that always returns value 1 (which is always consistent with the 1-Lipschitz function that is the constant function, i.e. $f(x) = 1$ for all $x \in \mathbb{R}$) then if the algorithm does not query a point in an interval of length $\frac{4\epsilon}{L}$ contained in $[0, 1]$ then the algorithm will fail on some input. Since there are two functions with disjoint ϵ-optimal points that are consistent with the oracle. Using this we can prove the following lower bound.

Lemma 16. *If an algorithm always computes an ϵ-optimal point of a function $f : \mathbb{R} \to \mathbb{R}$ that is L-Lipschitz, has $f(x_*) = f_* \stackrel{\text{def}}{=} \inf_{x \in \mathbb{R}^n} f(x)$ for $x_* \in [0, 1]$, and has $f(x) \in [0, 1]$ for all $x \in \mathbb{R}$ with T queries to a value oracle, $T \geq \frac{L}{4\epsilon} - 1$.*

Proof. Let $0 \leq x_1 \leq x_2 \leq ... \leq x_k \leq 1$ be the $k \leq T$ points queried by the algorithm (not necessarily in order) that lied between 0 and 1 when the algorithm was given value 1 as the result of each of its value queries. Now consider the intervals $[0, x_1], [x_2, x_3], ..., [x_{k-1}, x_{k-1}], [x_{k-1}, 1]$. Since these $k+1$ intervals have lengths summing to 1 at least one of these intervals has length at least $1/(k+1) \geq 1/(T+1)$. If $T < \frac{L}{4\epsilon} - 1$ then one of these intervals has length $> \frac{L}{4\epsilon}$ and by Lemma 15 there are two functions consistent with the input and disjoint ϵ-optimal points and the algorithm will output an incorrect answer when one of these functions is input. Consequently $T \geq \frac{L}{4\epsilon} - 1$ as desired. □

We have just shown an upper bound of $\lceil \frac{L}{\epsilon} \rceil$ on the number of queries to get an ϵ-optimal point and a lower bound of $\frac{L}{4\epsilon} - 1$ queries. These are the same up to additive and multiplicative constants and in much of the course we will ignore such constants when studying the complexity of problems. However, it is interesting and illustrative here to leverage our upper and lower bounds to see if either can be improved. Indeed, the arguments sharpened slightly to show that $L/(2\epsilon)$ is the optimal number of queries up to an additive constant (with a small change to the upper bound algorithm).

First we improve the upper bound. Note that the analysis of the upper bound algorithm we provided earlier cannot be improved in the worst case. This analysis stemmed from the fact that we argued that every point in $[0, 1]$ was at distance at most $1/k$ to a point we queried of the form i/k for $i \in [k]$. Further, if $x_* = 0$, this is indeed the case. However, every point in an interval of length $1/k$ is in fact distance at most $1/(2k)$ from one of the endpoints of the interval. Consequently, simply by querying 0 as well as i/k for all $i \in [k]$ we obtain the following.

Lemma 17. *There is an algorithm which when given a value oracle for L-Lipschitz $f : \mathbb{R} \to \mathbb{R}$ with $f(x_*) = f_* = \inf_{x \in \mathbb{R}} f(x)$ for some $x_* \in [0, 1]$ and $\epsilon > 0$ computes an ϵ-optimal point with $\lceil \frac{L}{2\epsilon} \rceil + 1$ queries.*

Proof. Consider the algorithm which queries the value oracle at the points i/k for all $i \in [k]$ and also queries the point 0 and then returns the point, q_*, for which the oracle output the smallest value. Note that the intervals $[0, 1/k], [1/k, 2/k], ..., [k - 1/k, 1]$ partition $[0, 1]$ and that x_* lies in one of these intervals. Since the length of each interval is $1/k$ we see that x_* is distance $1/(2k)$ from one of the endpoints. Since we queried the oracle at each endpoint we have that $|x_* - q| \leq 1/(2k)$ for one of the queried points $q \in [0, 1]$ and therefore

$$f(q_*) \leq f(q) = f_* + f(q) - f_* \leq f_* + L|q - x_*| \leq f_* + \frac{L}{2k}.$$

Picking $k = \lceil L/(2\epsilon) \rceil$ and noting that this algorithm queried $k + 1$ points yields the result. □

To improve the lower bound we leverage that when the resisting oracle outputs one there is another case in which there may be two $f_{z,\alpha}$ with disjoint ϵ-optimal points. In particular, if two intervals between the queried points that have length $> \frac{\epsilon}{L}$ then the are also two different $f_{z,\alpha}$ consistent with the oracle output and yet have disjoint ϵ-optimal points, i.e. where z is the midpoint of each interval. Applying this idea yields the following lemma.

Lemma 18. *If an algorithm always computes an ϵ-optimal point of a function $f : \mathbb{R} \to \mathbb{R}$ that is L-Lipschitz, has $f(x_*) = f_* \stackrel{\text{def}}{=} \inf_{x \in \mathbb{R}^n} f(x)$ for $x_* \in [0, 1]$, and has $f(x) \in [0, 1]$ for all $x \in \mathbb{R}$ with T queries to a value oracle, $T \geq \frac{L}{2\epsilon} - 2$.*

Proof. Let $0 \leq x_1 \leq x_2 \leq ... \leq x_k \leq 1$ be the $k < T$ points queried by the algorithm (not necessarily in order) that lie between 0 and 1 when the algorithm is given value 1 as the result of each of its value queries. Now consider the intervals $[0, x_1], [x_2, x_3], ..., [x_{k-1}, x_{k-1}], [x_{k-1}, 1]$. If one interval has length $> 4\epsilon/L$ then by Lemma 15 the algorithm will be incorrect on some input. Further, if two intervals have length $> 2\epsilon/L$ then by Lemma 14 the algorithm will be incorrect on some input (since the function could be the $f_{z,\alpha}$, for z set as the midpoint of the an intervals and α set as half the interval length for either interval). Consequently, we have that the largest interval has length $\leq 4\epsilon/L$ and the others all have length $\leq 2\epsilon/L$. Since the lengths

of these intervals sum to 1 and there are $k+1$ intervals we have $2k\epsilon/L + 4\epsilon k/L \geq 1$ and $2\epsilon(k+2) \geq L$. As $T \geq k$ the result follows. □

With the preceding two lemma we have proven the optimal query complexity for this problem up to additive constants. While we could improve further, e.g. by shifting the query points slightly in the upper bound, throughout most of the class we will not be too focused on these exact constants (both additive and multiplicative) and will be more focused on the rates. We will informally use the notation $O(\cdot)$ to hide additive and multiplicative constants in our upper bounds and $\Omega(\cdot)$ to hide the same in our lower bounds. In this notation the above results can be restated as saying that $O(L/\epsilon)$ queries suffice to compute an ϵ-optimal point and $\Omega(L/\epsilon)$ queries are needed. We say that the oracle or query complexity of a problem is $\Theta(\alpha)$ if $O(\alpha)$ queries suffice and $\Omega(\alpha)$ queries are needed. Consequently, we say that the query complexity of this one-dimensional Lipschitz optimization problem is $\Theta(L/\epsilon)$.

3.3 Lipschitz Function Minimization in Higher Dimensions

A natural question to ask then is, what happens if we are in higher dimensions. Interestingly, the strategy provided in the previous section generalizes to higher dimensions and arbitrary norms. More broadly, we could ask, if we are given a L-Lipschitz function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to some norm and promised that its minimizer lies in some set $S \subseteq \mathbb{R}^n$, how many queries to a value oracle are needed to find an ϵ-optimal point?

The approach of the previous section generalizes to this setting. To prove an upper bound on the query complexity of this problem, it suffices to prove that there is a set X of points such that every point in S is distance at most L/ϵ from one of these points. Provided this is found, then simply querying all the points in T and outputting the point of minimum value yields an ϵ-optimal point in $|X|$-queries. Such sets T are known as *ϵ-nets* and there are known bounds on their size for different norms and dimensions.

Similarly, to prove a lower bound, since Lemma 14 applied to every norm, it suffices to show that no matter what points are queried there are two points that are distance greater than ϵ/L from every point and each other. This gives a lower bound by considering the algorithm executed on the the $f_{z,\alpha}$ for z set to be one of these points and α set to be just larger than ϵ/L. Alternatively, one can simply find a set of k-points that are all sufficiently far from from each such that any method needs to query a point close to at least $k-2$ of them (See Lemma 20).

In the remainder of this section we briefly give such an argument for a generalization of the setting in the previous section where the algebra is particularly simple. This is provided to yield a sample of such arguments and (as is typical for the rest of the course) we will not optimize the multiplicative and additive constants in the bounds.

Lemma 19. *If $f : \mathbb{R}^n \to \mathbb{R}$ is L-Lipschitz with respect to $\|\cdot\|_\infty$ such that $f_* = f(x_*)$ for some $x_* \in [0,1]^n$ and $f(x) \in [0,1]$ for all $x \in [0,1]^n$ then for all $\epsilon \in (0,1)$ there is an algorithm that finds an ϵ-optimal point with $\lceil \frac{L}{\epsilon} \rceil^n$ queries to a value oracle.*

Proof. Consider the method that queries the value oracle at points $x \in \mathbb{R}^n$ for all vectors where $x_i = j_i/k$ for $j_i \in [k]$ and return the point with the minimum value. Note that there are k^n such points. Note that for all $i \in [n]$ it is the case that $|x_*(i) - \frac{j}{k}| \leq \frac{1}{k}$ for some $j \in [k]$ and therefore for some point q that was queried we have that $\|q - x_*\|_\infty \leq \frac{1}{k}$ and therefore

$$f(q) - f_* = |f(q) - f(x_*)| \leq L\|q - x_*\|_\infty \leq \frac{L}{k}.$$

Consequently, $f(q) \leq f_* + \frac{L}{k}$ and setting $k = \lceil L/\epsilon \rceil$ then yields the result. □

Lemma 20. *If an algorithm that always computes an ϵ-optimal point of a function $f : \mathbb{R}^n \to \mathbb{R}$ that is L-Lipschitz with respect to $\|\cdot\|_\infty$, has $f(x_*) = f_* \stackrel{\text{def}}{=} \inf_{x\in\mathbb{R}^n} f(x)$ for $x_* \in [0,1]^n$, and has $f(x) \in [0,1]$ for all $x \in \mathbb{R}$ with T queries to a value oracle, $T \geq (\frac{L}{2\epsilon} - 1)^n - 1$.*

Proof. Let $C \subseteq [0,1]^n$ be the set of points x with $x_i = j_i/k$ for $j_i \in [k]$. Note that $|C| = k^n$ and for all $x, y \in C$ e have $\|x - y\|_\infty \geq \frac{1}{k}$. Now consider running the method on a resisting oracle that always outputs 1. Note that if a for a point queried q and $x \in C$ it is the case that $\|x - q\|_\infty \leq \frac{1}{2k}$ then by reverse triangle inequality for every $y \in C$ with $y \neq x$ it is the case that

$$\|q - y\|_\infty \geq \|x - y\|_\infty - \|x - q\|_\infty \geq \frac{1}{k} - \frac{1}{2k} = \frac{1}{2k}.$$

Consequently, by Lemma 14 we see that unless the algorithm queries at least $k^n - 1$ points there will be be two functions $f_{z,\frac{L}{2k}}$ for $z \in C$ consistent with the oracle returning 1. Further, since $C \subseteq [0,1]^n$ we see that these functions are valid input and by Lemma 14 if $\frac{L}{2k} > \epsilon$ then these functions have disjoint ϵ-optimal points and therefore in this case at least $k^n - 1$ queries are needed for one of these $f_{z,\frac{L}{2k}}$ to not cause the algorithm to yield the incorrect output. Therefore, $T \geq k^n - 1$ for all integer k satisfying $\frac{L}{2k} > \epsilon$ and the result follows. □

Together these lemmas show that global minimization of Lipschitz functions is possible. However, they also show that performing such minimization requires a number of queries that scales exponentially with dimension. It shows that to halve the error, an extra exponential in n number of queries are needed. As the course proceeds we will see how to obtain improved error dependencies in different cases.

4 What Else?

What will we do in the rest of the course? Now that we understand the goals of the course and the framework by which we study problems, we can now betterexplain the structure of the course. In the next few lectures we will restrict our attention to the minimization of *smooth functions* that is functions where the gradient doesn't change to quickly, i.e. the gradient is Lipschitz. We will show that under this assumption we may not be able to compute global optimum efficiently, but we can compute critical points fairly sufficiently.

With the structure of smooth functions in hand we will add the assumption that our function is convex, i.e. that $f(\alpha x + (1-\alpha)y) \leq \alpha \cdot f(x) + (1-\alpha) \cdot f(y)$ for all x and y in the domain and $\alpha \in [0,1]$, that is the function lies underneath the line between any two points. These assumptions will be enough that we can provably find global optimum of our function. We will consider different algorithms that better exploit the exact convexity and smoothness assumptions we make. We will also discuss extensions of this to other norms and other oracle models which may be useful in various settings.

Next, we study the structure of convex functions that are continuous, i.e. Lipschitz, but not necessarily smooth. Here the structure is a little more complicated and we will consider two classes of algorithms. We will consider algorithms with a great dependence on the desired ϵ-accuracy but a poor (i.e. polynomial) dependence on dimension, these are known as cutting plane method and they underly the theory of optimization and are critical for many of the best guarantees we have for solving both combinatorial and continuous problems. We will also consider algorithms with a worse dependence on ϵ but a better dependence on dimension, the algorithms will be called mirror descent or subgradient descent; these are used in various online, streaming, stochastic, and practical settings to get good performance. Along the way, with a firm understanding of smooth and non-smooth convex optimization we will also cover some more advanced topics in iterative methods like acceleration and variance reduction.

Finally, we will conclude the course by studying higher-order methods, that is iterative methods which compute not just the gradient but the Hessian of a function, i.e. Newton's method. Moreover, we will

show how to use such methods to outperform cutting plane methods for broad classes of more structured optimization. In particular, we will introduce interior point methods which provide the fastest known algorithms for problems like linear programming and semidefinite programming.