Adaptive Algorithms for Problems Involving Black-Box Lipschitz Functions

by

Ilya Baran

B.S., Massachusetts Institute of Technology (2003)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Erik D. Demaine
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Adaptive Algorithms for Problems Involving Black-Box Lipschitz Functions

by

Ilya Baran

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

Suppose we are given a black-box evaluator (an oracle that returns the function value at a given point) for a Lipschitz function with a known Lipschitz constant. We consider queries that can be answered about the function by using a finite number of black-box evaluations. Specifically, we study the problems of approximating a Lipschitz function, approximately integrating a Lipschitz function, approximately minimizing a Lipschitz function, and computing the winding number of a Lipschitz curve in $\mathbb{R}^2$ around a point. The goal is to minimize the number of evaluations used for answering a query. Because the complexity of the problem instances varies widely, depending on the actual function, we wish to design adaptive algorithms whose performance is close to the best possible on every problem instance. We give optimally adaptive algorithms for winding number computation and univariate approximation and integration. We also give a near-optimal adaptive algorithm for univariate approximation when the output of function evaluations is corrupted by random noise. For optimization over higher dimensional domains, we prove that good adaptive algorithms are impossible.

Thesis Supervisor: Erik D. Demaine
Title: Assistant Professor

Acknowledgements

I would like to thank my advisor, Erik Demaine, for guiding me on this project, providing crucial ideas and feedback, and often assisting me in the tricky task of determining the value of various results.

I would also like to thank Dmitriy "Dimdim" Rogozhnikov for many fruitful and interesting discussions, some of which led to the most important ideas in Chapters 4 and 5.

Finally, thanks to my other friends, who provided a wonderful haven from this project and to my parents and sister, who, among other things, gave me free access to their washing machine.

This research was partially supported by the Akamai Presidential Fellowship.

Contents

Chapter 1

Introduction

1.1 Adaptive Algorithms

Algorithms can be analyzed in many different ways. Worst-case analysis measures an algorithm's slowest performance on a problem of size n. Average-case analysis measures the "average" performance on problems of size n (for a suitably defined "average"). Competitive analysis compares an online algorithm's performance on a problem instance to the best possible output of an algorithm that has access to the entire problem instance. In this thesis, I focus on adaptive analysis, which compares the performance of the algorithm to the difficulty of the particular problem instance. The terms "performance" and "difficulty" are intentionally vague because they can be defined in various ways.

As an example, consider the problem of sorting in the comparison model (see [CLRS01] for an overview). Performance for this problem is measured by the number of comparisons an algorithm performs. A well-known lower bound is that $\Omega(n \log n)$ comparisons are necessary in the worst case. Mergesort, for example, performs $O(n \log n)$ comparisons in the worst case, so there is no "better" algorithm than mergesort (neglecting constant factors) from the worst-case point of view. However, if most problem instances are already almost sorted, we may wish for an algorithm that performs better if, for instance, the initial order of the objects has few inversions. Such an algorithm is known, in fact: it requires $O(n + n \log(1 + I/n))$ comparisons to

sort an input that has n objects and I inversions. This is an adaptive algorithm and it performs better for easier instances. Many other adaptive algorithms for sorting are known, for different notions of difficulty. See [ECW92] for a survey of them. It is important to note that an adaptive algorithm does not know the difficulty of the problem (for instance, the number of inversions) in advance—it discovers how difficult the problem is as it runs and adapts to this difficulty.

For some problems, particularly the ones I wish to consider, a natural measure of the difficulty of a problem instance is the performance of the best-possible (correct) algorithm for that problem instance. More formally, if $\mathcal{P}$ is the set of problem instances, $\mathcal{A}$ is the set of algorithms (in some reasonable class) that are correct on all problem instances, and $\mathrm{COST}(A, P)$ is the cost incurred by algorithm $A \in \mathcal{A}$ on problem instance $P \in \mathcal{P}$, then the best-possible performance (denoted $\mathrm{OPT}(P)$) is:

$$\mathrm{OPT}(P) \stackrel{\mathrm{def}}{=} \inf_{A \in \mathcal{A}} \mathrm{COST}(A, P).$$

Our terminology regarding OPT is a bit sloppy: we use OPT to refer to both the performance of the best-possible algorithm and to the best-possible algorithm itself—from the context, it is always clear which meaning is intended.[1]

By definition, for each problem instance P, there is an algorithm whose cost on P is arbitrarily close to $\mathrm{OPT}(P)$. We are interested in adaptive algorithms whose cost on P is not much greater than $\mathrm{OPT}(P)$ for *every* problem instance P. Carrying out an adaptive analysis for an algorithm is generally more difficult than carrying out a worst-case analysis, because the former makes stronger performance guarantees. If difficult problem instances are rare (as sometimes happens in practice), an adaptive algorithm performs much better than an algorithm whose performance on every problem instance is its worst-case performance.

[1]When possible values of COST are not discrete (as can happen with expected costs), a best-possible algorithm may not exist. Because we ignore constant factors, an algorithm whose cost is within an additive constant of the best possible is sufficient for our purposes.

1.2 Problems

In this thesis, we study deterministic adaptive algorithms for answering queries about Lipschitz functions. The Lipschitz condition is a weak condition on a function (it is slightly stronger than continuity), but it allows interesting algorithms. A typical problem is a special case of the following:

> Let S be a subset of $\mathbb{R}^d$, and let M be a metric space. Let f be a function from S to M such that for every $x_1, x_2 \in S$, the Lipschitz condition is satisfied: $d_M(f(x_1), f(x_2)) \leq L\|x_2 - x_1\|_2$. Compute a property of f with error at most ϵ, for $\epsilon > 0$. An algorithm can obtain information about f only by sampling (evaluating) it at a point of S and should use as few samples as possible.

Problems with such formulations are also studied in the field of information-based complexity (see [TWW88] for an overview). However, information-based complexity is primarily concerned with worst-case, average-case, or randomized analysis of more difficult problems, rather than adaptive analysis of algorithms for easier problems. Note that in information-based complexity, the term "adaptive algorithm" refers to an algorithm in which the locations where f is sampled are allowed to depend on the results of previous samples of f. However, we use the term "adaptive algorithm" in the sense of [DLOM00].

This thesis is organized by the type of function for which the problem is posed. In Chapter 2, we study problems on Lipschitz functions from $[0, 1]$ to $\mathbb{R}$. We consider the problem of approximating such a function and the problem of approximating its definite integral. In Chapter 3, we extend the results on the approximation problem to the setting when the range is an arbitrary metric space. We also give an adaptive algorithm for computing the winding number of a Lipschitz curve in $\mathbb{R}^2$. In Chapter 4, we consider the approximation problem when the values of the samples an algorithm makes are corrupted by random noise. Finally, in Chapter 5, we consider the optimization problem when the domain is d-dimensional with $d \geq 2$.

The final goal of investigating a problem in the adaptive framework is to design an optimally adaptive algorithm. Suppose $\mathcal{P}$ is the set of problem instances and each problem instance $P \in \mathcal{P}$ has certain natural parameters, $v_1(P), \ldots, v_k(P)$, with the first parameter $v_1(P) = \mathrm{OPT}(P)$. An algorithm is *optimally adaptive* if its performance on every problem instance $P \in \mathcal{P}$ is within a constant factor of every algorithm's worst-case performance on the family of instances $\{P' \in \mathcal{P} \mid v_i(P') = v_i(P) \text{ for all } i\}$. Note that this definition depends on the choice of parameters. In addition to OPT, we choose reasonable parameters, such as ϵ, the desired output accuracy. For some problems, it is possible to come up with instance-optimal algorithms. An algorithm is *instance optimal* if its performance is within a constant factor of OPT on every problem instance. Obviously, any instance-optimal algorithm is optimally adaptive. In this thesis, we give an instance-optimal algorithm for winding number computation and optimally adaptive algorithms for univariate approximation and deterministic univariate integration. The results we achieve are summarized in Table 6.1 in the conclusion.

When discussing the problems, we assume without loss of generality that the Lipschitz constant is 1. The function and relevant parameters can be scaled to make this the case. Also, when the domain of the function is an interval, we assume that this interval is $[0, 1]$, for the same reason.

1.3 Related Work

There is not very much literature on adaptive algorithms that perform well relative to OPT. One reason is that adaptive analysis makes a much stronger guarantee than, for example, worst-case or average-case analysis, and can be more difficult to perform, as a result. Another reason is that adaptive analysis does not yield itself well to composition. In other words, if algorithm A uses adaptive algorithm B as a subroutine, in order for the analysis of A to take advantage of B's adaptiveness, it must be shown that the problem instances A feeds to B are "easy".

Adaptive algorithms have been considered in the context of various set opera-

tions in [DLOM00], aggregate ranking in [FLN03], and independent-set discovery in [BBD+04]. They have also appeared in the context of distributed algorithms: for example, an adaptive algorithm for the distributed minimum spanning tree problem is given in [Elk04]. There is extensive literature on univariate Lipschitz function optimization, primarily concerned with worst-case or experimental analysis (see, e.g. [HJ95]), but Piyavskii's algorithm [Piy72] has been analyzed in the adaptive framework, first in [Dan71]. The analysis was completed and sharpened in [HJL91] to show that Piyavskii's algorithm is instance-optimal. [BD04] considers the problem of univariate Lipschitz function optimization when the range is $\mathbb{R}^d$ and gives an optimally adaptive algorithm.

Problems on Lipschitz functions similar to ones we consider have been studied outside the adaptive framework. For example, [ZKS03] describe an algorithm for function approximation that is essentially LIPSCHITZ-INTEGRATE-ADAPTIVE, our algorithm for integration (this makes sense because they consider approximation error averaged over $[0, 1]$, whereas we work with worst-case approximation error). However, in that paper, it is only shown that the algorithm makes locally optimal choices (i.e., it is greedy) and no global optimality analysis of the algorithm is performed. The univariate Lipschitz integration problem is used to illustrate information-based complexity concepts in [Wer02], and the trivial algorithm is shown to be optimal in the worst case. In [GW90], problems on Lipschitz curves are considered. A data structure based on Proposition 8 is proposed for solving curve queries, including nearest-point queries and point containment (a special case of winding number) and algorithms based on this data structure are analyzed in the worst case.

1.4 Proof Sets

In order to compare the running time of an algorithm on a problem instance to OPT, we define the concept of a proof set for a problem instance. A set S of points in the domain of f is a *proof set* for problem instance (f, C) (where C represents additional problem instance components) and output x if for every f' that is equal to f on S,

x is a correct output on (f', C). In other words, sampling f at a proof set proves the correctness of the output. For example, suppose that the problem is to compute the definite integral of a Lipschitz function $f\colon [0,1] \to \mathbb{R}$, to an accuracy of ϵ. A proof set for problem instance (f, ϵ) and output x is a set of points $s_1, \ldots, s_n$ in $[0,1]$ such that for any $\hat{f}$ with $\hat{f}(s_i) = f(s_i)$, the output is correct: $\left| x - \int \hat{f} \right| \leq \epsilon$. We say that a set of samples is a proof set for a particular problem instance without specifying the output if some output exists for which it is a proof set.

It is clear from the definition that sampling a proof set is the only way a deterministic algorithm can guarantee correctness: if an algorithm doesn't sample a proof set for some problem instance, we can feed it a problem instance that has the same value on the sampled points, but for which the output of the algorithm is incorrect. Conversely an algorithm can terminate as soon as it has sampled a proof set and always be correct. Thus, deterministic OPT is equal to the size of a smallest proof set.

In this thesis, we consider only deterministic algorithms (in Chapter 4, noise in our model can serve as a random number generator for a deterministic algorithm, so randomization gives no additional power there). However, we remark that several alternative models can be considered. If randomization is allowed, but an algorithm is never allowed to fail, randomized OPT is no more powerful than deterministic OPT on any black-box Lipschitz problem (because if there is a possibility of an algorithm terminating after sampling a set of samples that is not a proof set, there is a possibility of failure). If an algorithm is required to be correct with probability 1, there are some contrived problems on which randomized OPT is more powerful than deterministic OPT. If correctness with probability at least 2/3 is required, randomized OPT becomes asymptotically more powerful than deterministic OPT for integration (we plan to explore this in a forthcoming paper), but no power is gained for univariate approximation and optimization, and winding number.

Chapter 2

Functions from $\mathbb{R}$ to $\mathbb{R}$

2.1 Sampling Univariate Functions

In general, a deterministic adaptive algorithm works by sampling the input function until the samples form a proof set for the problem the algorithm is trying to solve. To analyze such an algorithm, we compare the number of samples it performs to the size of a smallest proof set for that problem instance. To do this, we fix a proof set P, classify the samples the algorithm performs based on their relationship to points in P, and bound the number of points of some of these classes.

Let P be a nonempty finite set of points in $[0, 1]$. Consider the execution of an algorithm which samples a function at points on the interval $[0, 1)$ (if it samples at 1, ignore that sample). Let $s_1, s_2, \ldots, s_n$ be the sequence of samples that the algorithm performs in the order that it performs them. Let I_t be the set of unsampled intervals after sample s_t, i.e., the connected components of $[0, 1) - \{s_1, \ldots, s_t\}$, except make each element of I_t half-open by adding its left endpoint, so that the union of all the elements of I_t is $[0, 1)$. Let $[l_t, r_t)$ be the element of I_{t-1} that contains s_t.

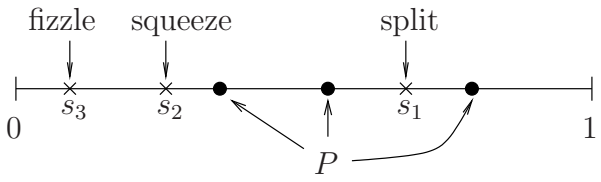

Figure 2-1: Different types of samples.

Then sample s_t is a

$$\begin{aligned} \textit{split} \quad &\text{if} \quad [l_t, s_t) \cap P \neq \emptyset \text{ and } [s_t, r_t) \cap P \neq \emptyset; \\ \textit{squeeze} \quad &\text{if} \quad [l_t, s_t) \cap P \neq \emptyset \text{ or } [s_t, r_t) \cap P \neq \emptyset, \text{ but not both;} \\ \textit{fizzle} \quad &\text{if} \quad [l_t, r_t) \cap P = \emptyset. \end{aligned}$$

These definitions are, of course, relative to P. See Figure 2-1. We can now bound the number of samples of different types. For splits, our bound does not need to know anything about the sampling algorithm.

Proposition 1 *The number of splits is at most* $|P| - 1$.

Proof: Let a_t be the number of intervals of I_t that intersect P. If s_t is a split, then $a_t - a_{t-1} = 1$; otherwise $a_t = a_{t-1}$. Because the intervals of I_t are disjoint, $a_t \leq |P|$. Unless $P = \{1\}$ (in which case, there can be no splits), $a_0 = 1$, so at most $|P| - 1$ splits can occur. □

To bound the number of squeezes, we first want the following easy fact:

Proposition 2 *Let* a_i *for* $1 \leq i \leq |P|$ *and* b *be positive real numbers. If* $\sum_{i=1}^{|P|} a_i \leq \epsilon^{-1}$*, then* $\sum_{i=1}^{|P|} \log_b a_i \leq |P| \log_b(\epsilon^{-1}/|P|)$.

Proof: By the arithmetic-geometric mean inequality, $\sqrt[|P|]{\prod_{i=1}^{|P|} a_i} \leq \frac{\sum_{i=1}^{|P|} a_i}{|P|} \leq \frac{\epsilon^{-1}}{|P|}$. Taking the logarithm with base b of both sides gives us $\frac{\sum_{i=1}^{|P|} \log_b a_i}{|P|} \leq \log_b(\epsilon^{-1}/|P|)$. Multiplying both sides by $|P|$ gives us the desired result. □

For bounding the number of squeezes, we need to assume that the algorithm always samples in the middle of a previously unsampled interval and that samples are never too close together.

Proposition 3 *Suppose that for all i and j with $i \neq j$, $|s_i - s_j| > \epsilon$ and that for all t, $s_t = (l_t + r_t)/2$. Then if $|P| \leq \epsilon^{-1}/2$, the number of squeezes is at most $|P| \log_2(\epsilon^{-1}/|P|)$.*

Proof: Let $IP_t = \{[a, b) \in I_t \mid [a, b) \cap P \neq \emptyset\}$. If $J \in IP_t$, define $S(J)$ to be the number of squeezes that have occurred to intervals containing J:

$$S(J) = \sum_{i=1}^{t} \begin{cases} 1 \text{ if } s_i \text{ is a squeeze and } J \subset [r_i, l_i), \\ 0 \text{ otherwise.} \end{cases}$$

We claim the invariant that for all t,

$$\sum_{[a,b) \in IP_t} (b - a) \cdot 2^{S([a,b))} = 1. \tag{2.1}$$

We prove (2.1) by induction on t. The base case $t = 0$ is clear because unless P is trivial, $IP_0 = \{[0, 1)\}$ and $S([0, 1)) = 0$ because there have been no squeezes. For the inductive step, assume (2.1) holds for $t - 1$. If s_t is a fizzle, no intervals containing points of P are affected and the sum remains the same. If s_t is a split, interval $[l_t, r_t)$ is replaced by $[l_t, s_t)$ and $[s_t, r_t)$ in IP_t and $S([l_t, s_t)) = S([s_t, r_t)) = S([l_t, r_t))$ because no new squeezes have occurred and the sum remains unchanged. Finally, if s_t is a squeeze, the interval $[l_t, r_t)$ in IP_t is replaced by an interval J of half the length, but $S(J) = S([l_t, r_t)) + 1$, so the sum remains the same.

Now, by assumption, each interval of I_t is longer than ϵ, so, writing (2.1) for $t = n$ gives:

$$\sum_{[a,b) \in IP_n} \epsilon \cdot 2^{S([a,b))} < \sum_{[a,b) \in IP_n} (b-a) \cdot 2^{S([a,b))} = 1, \quad \text{which means} \quad \sum_{[a,b) \in IP_n} 2^{S([a,b))} < \epsilon^{-1}.$$

Using Proposition 2, we obtain $\sum_{[a,b) \in IP_n} S([a, b)) < |P| \log_2(\epsilon^{-1}/|P|)$, which implies that the total number of squeezes is less than $|P| \log_2(\epsilon^{-1}/|P|)$. □

Propositions 1 and 3 tell us that adaptive analyses of sampling algorithms for problems involving univariate Lipschitz functions can focus on the number of fizzles.

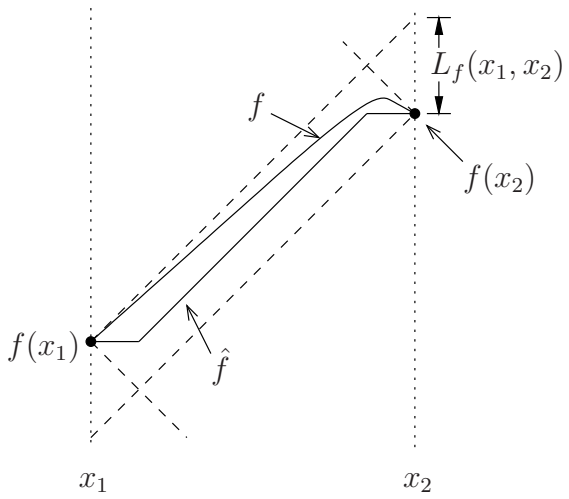

Figure 2-2: Illustration of looseness. Lipschitz bounds are dashed.

2.2 Approximation

We first demonstrate the application of Propositions 1 and 3 to the problem of univariate function approximation. The problem is formally defined as follows:

Problem LIPSCHITZ-APPROX:

Given:	(f, ϵ) where $f\colon [0,1] \to \mathbb{R}$, $0 < \epsilon < 1/2$
Such that:	For $x_1, x_2 \in [0,1]$, $\lvert f(x_2) - f(x_1)\rvert \leq \lvert x_2 - x_1\rvert$
Compute:	$\hat{f}\colon [0,1] \to \mathbb{R}$ such that for all $x \in [0,1]$, $\lvert f(x) - \hat{f}(x)\rvert \leq \epsilon$

The issue of how $\hat{f}$ is to be represented can be ignored, because the algorithms we discuss always output a piecewise-linear function with a small number of pieces, while OPT and lower bounds do not depend on the representation of $\hat{f}$.

To solve this problem, we notice that once f is sampled at x_1 and x_2, we can use the Lipschitz condition to bound f on $[x_1, x_2]$, as shown in Figure 2-2. The *looseness* of this bound is defined to be

$$L_f(x_1, x_2) = (x_2 - x_1) - |f(x_2) - f(x_1)|.$$

When it is clear which f we are talking about, we just write $L(x_1, x_2)$.

Proposition 4 *Given a Lipschitz function f, looseness has the following properties:*

(1) $0 \leq L(x_1, x_2) \leq x_2 - x_1$.

(2) If $x_1' \leq x_1 < x_2 \leq x_2'$ *then* $L(x_1, x_2) \leq L(x_1', x_2')$.

Proof: This follows immediately from the definition of looseness and the Lipschitz condition on f. □

As the following proposition shows, the accuracy with which we can approximate f on $[x_1, x_2]$ given only the value of f at x_1 and x_2 is precisely $L(x_1, x_2)/2$.

Proposition 5 *Let* $f\colon [0,1] \to \mathbb{R}$ *be Lipschitz and let* $[x_1, x_2]$ *be a subinterval of* $[0,1]$. *Then:*

(1) *If* $HI(x) = \min(f(x_1) + |x - x_1|, f(x_2) + |x_2 - x|)$ *and* $LO(x) = \max(f(x_1) - |x - x_1|, f(x_2) - |x_2 - x|)$, *the function defined as* $\hat{f}(x) = (HI(x) + LO(x))/2$ *is within* $L(x_1, x_2)/2$ *of* f *on* $[x_1, x_2]$.

(2) *For any function* $\hat{f}$ *on* $[x_1, x_2]$, *there is a Lipschitz function* g *and a point* $x \in (x_1, x_2)$ *such that* $g(x_1) = f(x_1)$, $g(x_2) = f(x_2)$, *and* $|\hat{f}(x) - g(x)| \geq L(x_1, x_2)/2$.

Proof: First we show part (1). Notice that the Lipschitz condition on f implies that $LO(x) \leq f(x) \leq HI(x)$. This implies that at any x, $|f(x) - \hat{f}(x)| \leq (HI(x) - LO(x))/2$. Consider first the case $f(x_1) \leq f(x_2)$. Note that $HI(x) \leq f(x_1) + x - x_1$ and $LO(x) \geq f(x_2) + x - x_2$ which implies that

$$HI(x) - LO(x) \leq x_2 - x_1 - (f(x_2) - f(x_1)) = L(x_1, x_2).$$

Similarly, for the case $f(x_1) > f(x_2)$, we note that $HI(x) \leq f(x_2) + x_2 - x$ and $LO(x) \geq f(x_1) - x + x_1$ and so

$$HI(x) - LO(x) \leq x_2 - x_1 - (f(x_1) - f(x_2)) = L(x_1, x_2).$$

Thus, $|f(x) - \hat{f}(x)| \leq (HI(x) - LO(x))/2 \leq L(x_1, x_2)/2$, which proves part (1).

For part (2), note that both $HI(x)$ and $LO(x)$ are Lipschitz and $HI(x_1) = LO(x_1) = f(x_1)$ and $HI(x_2) = LO(x_2) = f(x_2)$. Let $x = (x_1 + x_2)/2$ and note

that $HI(x) = \min(f(x_1), f(x_2)) + (x_2 - x_1)/2$ and $LO(x) = \max(f(x_1), f(x_2)) - (x_2 - x_1)/2$ so $HI(x) - LO(x) = L(x_1, x_2)$. By the triangle inequality, $|HI(x) - \hat{f}(x)| + |LO(x) - \hat{f}(x)| \geq L(x_1, x_2)$, which implies that either $|HI(x) - \hat{f}(x)| \geq L(x_1, x_2)/2$ or $|LO(x) - \hat{f}(x)| \geq L(x_1, x_2)/2$. □

Proposition 5 immediately gives the criterion for a proof set:

Corollary 1 *Let $P = \{x_1, x_2, \ldots, x_n\}$ such that $0 \leq x_1 < x_2 < \cdots < x_n \leq 1$. Then P is a proof set for problem instance (f, ϵ) if and only if $x_1 \leq \epsilon$, $x_n \geq 1 - \epsilon$, and for all i with $1 \leq i \leq n - 1$, $L_f(x_i, x_{i+1}) \leq 2\epsilon$.*

Proof: If $x_1 \leq \epsilon$, $x_n \geq 1 - \epsilon$, and $L_f(x_i, x_{i+1}) \leq 2\epsilon$ then f can be approximated to within ϵ with $f(x_1)$ on $[0, x_1]$, $f(x_n)$ on $[x_n, 1]$, and a pasting of functions of the form $(HI(x) + LO(x))/2$ as in Proposition 5(1).

In the other direction, if $x_1 > \epsilon$, $f(0)$ could be anywhere in the range $[f(x_1) - x_1, f(x_1) + x_1]$ and so no value of $\hat{f}(0)$ is guaranteed to be within ϵ of $f(0)$. Similarly, P cannot be a proof set if $x_n < 1 - \epsilon$. If $L_f(x_i, x_{i+1}) > 2\epsilon$ for some i, Proposition 5(2) shows that f cannot be approximated to within ϵ on (x_i, x_{i+1}). □

The simplest way to obtain a proof set is to sample f at $\epsilon, 3\epsilon, 5\epsilon, \ldots, 1 - \epsilon$. Proposition 4(1) guarantees that this is a proof set regardless of f. Thus, a trivial algorithm can solve LIPSCHITZ-APPROX using $O(\epsilon^{-1})$ samples. On the other hand, if $f(x) = 0$ for all x, then $L_f(x_1, x_2) = x_2 - x_1$ for all $x_1 < x_2$, which implies that $\Omega(\epsilon^{-1})$ samples are necessary for a proof set. So, as with most other problems we consider, the worst-case scenario is simple and not very interesting for LIPSCHITZ-APPROX and we now focus on adaptive algorithms.

Figure 2-3 gives an adaptive algorithm for this problem. We will show, using the tools in Section 2.1, that this algorithm is optimally adaptive. The idea is to start by sampling f at 0 and 1, and then sample inside previously unsampled intervals where looseness is higher than 2ϵ until none are left. For bookkeeping, we store all of the sampled points in a singly linked list, ordered by the parameter value. We also maintain a "To-Do" stack which contains pointers to endpoints of intervals we have

Algorithm LIPSCHITZ-APPROX-ADAPTIVE

L is a linked list of (PARAMETER,VALUE) pairs and S is a stack of pointers into objects in L.

1. Add $(0, f(0))$ and $(1, f(1))$ to L and push a pointer to $(0, f(0))$ onto S
2. Do until S is empty:
  3. $P_1 \leftarrow \text{TOP}[S]$
  4. $P_2 \leftarrow \text{NEXT}[L, P_1]$
  5. LOOSENESS $\leftarrow$ PARAMETER$[P_2]$ − PARAMETER$[P_1]$ − |VALUE$[P_1]$ − VALUE$[P_2]$|
  6. If LOOSENESS $\leq 2\epsilon$ then POP(S) and continue to step 2
  7. $x \leftarrow$ (PARAMETER$[P_1]$ + PARAMETER$[P_2]$)/2
  8. Insert $(x, f(x))$ into L after P_1 and push a pointer to it onto S
9. Output the interpolation of the values stored in L as described in Corollary 1.

Figure 2-3: LIPSCHITZ-APPROX-ADAPTIVE

not subdivided.

The correctness of this algorithm follows from Corollary 1, because a pointer to an element of L is removed from S only when the looseness of the interval defined by that element and the next element is no more than 2ϵ. Thus, at the end of the execution the looseness of any interval defined by adjacent elements of L is no more than 2ϵ.

We can now bound the adaptive performance of this algorithm:

Theorem 1 LIPSCHITZ-APPROX-ADAPTIVE *makes* $O\left(\text{OPT}(f,\epsilon)\log\frac{\epsilon^{-1}}{\text{OPT}(f,\epsilon)}\right)$ *samples on problem instance* (f, ϵ).

Proof: Let P be a proof set for (f, ϵ) of size $\text{OPT}(f, \epsilon)$. We show that among the samples LIPSCHITZ-APPROX-ADAPTIVE makes in Step 8, there are no fizzles. Suppose the contrary, that at some point a sample performed in Step 8 is a fizzle. Let $x_i = \text{PARAMETER}[P_i]$, for $i = 1, 2$, at that time. Because Step 8 is a fizzle, the interval $[x_1, x_2)$ contains no points of P, but (as the check in Step 6 failed) $L(x_1, x_2) > 2\epsilon$. But Proposition 4(2) implies that one of the conditions for P being a proof set is violated, which is a contradiction. So every sample is either a squeeze or a split. The number of splits is bounded by $O(\text{OPT}(f, \epsilon))$ using Proposition 1. Notice that no interval of length less than 2ϵ is ever subdivided and intervals are always sampled in the middle.

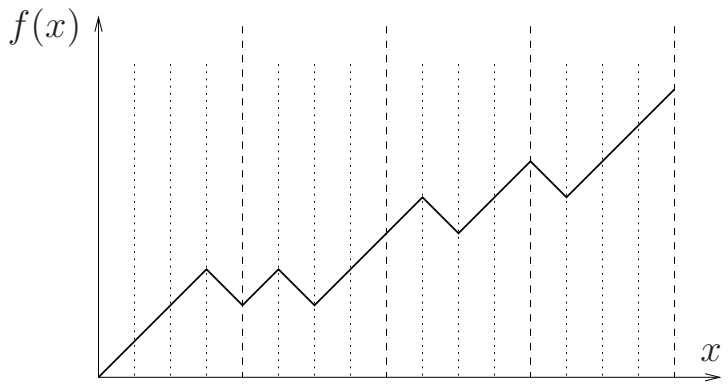

Figure 2-4: Adaptive lower bound construction for LIPSCHITZ-APPROX with $n = 16$ and $k = 4$.

So Proposition 3 bounds the number of squeezes by $O\left(\mathrm{OPT}(f,\epsilon)\log\frac{\epsilon^{-1}}{\mathrm{OPT}(f,\epsilon)}\right)$, as necessary. □

It remains to prove that LIPSCHITZ-APPROX-ADAPTIVE is optimally adaptive, i.e., that any algorithm incurs a logarithmic penalty over OPT.

Theorem 2 *For any algorithm and for any $\epsilon > 0$ and any integer k such that $0 < k < \epsilon^{-1}/2$, there exists a problem instance (f,ϵ) of* LIPSCHITZ-APPROX *with $\mathrm{OPT}(f,\epsilon) = O(k)$ on which that algorithm performs $\Omega(k\log(\epsilon^{-1}/k))$ samples.*

Proof: Let $n = k\lfloor\epsilon^{-1}/2k\rfloor$. Divide the parameter space $[0,1]$ into n equal regions and group them into k groups of n/k regions each. In each group, let $n/k-1$ regions have slope 1 and let one region have slope -1, as shown in Figure 2-4. Clearly, sampling at 0, 1, and every point where the slope of f changes is a proof set because the looseness of all unsampled intervals is 0. This implies that for any such function, $\mathrm{OPT} \le 2k+2 = O(k)$.

Now we show that any algorithm needs additional samples on some function of this type. Consider a group, and consider an interval I consisting of the region on which f has negative slope in that group and an adjacent region in that group. Because the value of f is the same on the endpoints of I, $L(I) = 2/n \ge 4\epsilon$. So any set of points which does not include a point in I cannot be a proof set for that problem instance. In other words, any algorithm must find the region on which f has negative

slope in every group, or an adjacent one. The endpoints of the groups are the same for all functions of this type, so sampling in one group provides no information about the location of negative-slope regions in other groups. Therefore each search for the negative-slope region must be done independently. Moreover, sampling inside a group only gives information as to whether the negative-slope region is to the left or to the right of the sampled point. This means that the problem in every group is as hard as a continuous binary search on an interval of length $1/k$ for a region of length $2/n$, which requires $\Omega(\log(n/k))$ time in the worst case. The algorithm needs to perform k independent such searches, giving the desired lower bound. □

2.3 Integration

We now turn to the problem of univariate integration. The problem is formulated as follows:

Problem LIPSCHITZ-INTEGRATE:

$$\begin{aligned} \textbf{Given:} \quad & (f, \epsilon), \quad \text{where} \quad f\colon [0,1] \to \mathbb{R}, \quad 0 < \epsilon < 1/2 \\ \textbf{Such that:} \quad & \text{For } x_1, x_2 \in [0,1],\ |f(x_2) - f(x_1)| \le |x_2 - x_1| \\ \textbf{Compute:} \quad & M \in \mathbb{R} \text{ such that } \left| M - \int_0^1 f(x)\,dx \right| \le \epsilon \end{aligned}$$

At first glance, it seems that integration can be reduced to approximation. Indeed, given a black box for f on $[0,1]$, if we know explicitly a function $\hat{f}$ that is guaranteed to be within ϵ of f everywhere, then the integral of $\hat{f}$ must be within ϵ of the true integral of f. Thus, every proof set for approximation is also a proof set for integration, and LIPSCHITZ-APPROX-ADAPTIVE can be trivially transformed to solve LIPSCHITZ-INTEGRATE. The resulting algorithm is not optimally adaptive, however. This is because OPT for integration can be asymptotically smaller than

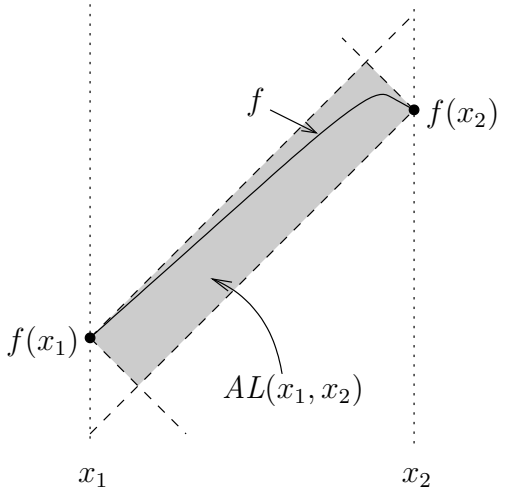

Figure 2-5: Illustration of area-looseness.

OPT for approximation. For example, consider the function

$$f(x) = \begin{cases} 0 & \text{if } x \leq \sqrt{\epsilon}, \\ x - \sqrt{\epsilon} & \text{if } x > \sqrt{\epsilon}. \end{cases}$$

In this case, because f is constant on $[0, \sqrt{\epsilon}]$, samples at intervals of 2ϵ are necessary there for an approximation, resulting in $\text{OPT}_{\text{APPROX}} = \Theta(\epsilon^{-1/2})$. However, $\text{OPT}_{\text{INTEGRATE}} = 3$ because samples at 0, $\sqrt{\epsilon}$ and 1 form a proof set.

In order to obtain an optimally adaptive algorithm for LIPSCHITZ-INTEGRATE, we use the same idea as LIPSCHITZ-APPROX-ADAPTIVE, but we use "area-looseness" instead of looseness for determining where to sample, and stop when the total area-looseness is smaller than 2ϵ. Formally, if we have a function f, the *area-looseness* of the interval (x_1, x_2) is defined to be

$$AL_f(x_1, x_2) = \frac{1}{2}\left((x_2 - x_1)^2 - (f(x_2) - f(x_1))^2\right).$$

See Figure 2-5. If the expression for AL_f is factored using the difference of squares formula, then the terms in the resulting expression correspond to the lengths of the sides of the gray rectangle in the figure.

We note some properties of area-looseness, the first two of which are analogous to

looseness properties in Proposition 4:

Proposition 6 *Area-Looseness has the following properties:*

(1) $0 \le AL(x_1, x_2) \le (x_2 - x_1)^2/2$.

(2) If $x_1' \le x_1 < x_2 \le x_2'$ *then* $AL(x_1, x_2) \le AL(x_1', x_2')$.

(3) If $x \in [x_1, x_2]$, *then* $AL(x_1, x) + AL(x, x_2) \le AL(x_1, x_2)$.

(4) $AL\big(x_1, \frac{x_1+x_2}{2}\big) + AL\big(\frac{x_1+x_2}{2}, x_2\big) \le AL(x_1, x_2)/2$.

Proof: Part (1) is obvious, and (2) follows from (3), so we prove (3) and (4):

(3) The Lipschitz condition implies that $|x_1 - x| \ge |f(x_1) - f(x)|$ and $|x_2 - x| \ge |f(x_2) - f(x)|$, and therefore $|(x_1 - x)(x_2 - x)| \ge |(f(x_1) - f(x))(f(x_2) - f(x))|$. In addition, $(x_1 - x)(x_2 - x) \le 0$, so $(f(x_1) - f(x))(f(x_2) - f(x)) \ge (x_1 - x)(x_2 - x)$. Multiplying through, we get

$$f(x_1)f(x_2) + f(x)^2 - f(x_1)f(x) - f(x_2)f(x) \ge x_1x_2 + x^2 - xx_1 - xx_2.$$

Rearranging, we get

$$-x_1x_2 + f(x_1)f(x_2) \ge x^2 - xx_1 - xx_2 - f(x)^2 + f(x)f(x_1) + f(x)f(x_2).$$

Adding $\frac{x_2^2+x_1^2+f(x_1)^2+f(x_2)^2}{2}$ to both sides to complete the squares, we get

$$\frac{(x_2 - x_1)^2 - (f(x_1) - f(x_2))^2}{2} \ge$$
$$\ge \frac{(x - x_1)^2 - (f(x_1) - f(x))^2}{2} + \frac{(x_2 - x)^2 - (f(x_2) - f(x))^2}{2}.$$

By the definition of AL, this inequality is equivalent to (3).

(4) Let $x_m = (x_1 + x_2)/2$. The proposition claims that if f is Lipschitz,

$$\frac{(x_2 - x_1)^2 - (f(x_2) - f(x_1))^2}{2} \ge$$
$$\ge (x_m - x_1)^2 - (f(x_m) - f(x_1))^2 + (x_2 - x_m)^2 - (f(x_2) - f(x_m))^2.$$

Because $(x_m - x_1) = (x_2 - x_m) = (x_2 - x_1)/2$, the claim can be written as

$$\frac{(x_2 - x_1)^2 - (f(x_2) - f(x_1))^2}{2} \geq \frac{(x_2 - x_1)^2}{2} - (f(x_m) - f(x_1))^2 - (f(x_2) - f(x_m))^2.$$

Thus, we need to show that

$$(f(x_2) - f(x_1))^2 \leq 2(f(x_m) - f(x_1))^2 + 2(f(x_2) - f(x_m))^2.$$

Notice that the right-hand side can be written as $af(x_m)^2 + bf(x_m) + c$, where $a = 4$ and $b = -4f(x_1) - 4f(x_2)$. This expression has a single global minimum at $f(x_m) = -b/2a$, which is $f_m = (f(x_1) + f(x_2))/2$. Therefore, we have:

$$2(f_m - f(x_1))^2 + 2(f(x_2) - f_m)^2 \leq 2(f(x_m) - f(x_1))^2 + 2(f(x_2) - f(x_m))^2.$$

But we can rewrite the left-hand side as:

$$2\left(\frac{f(x_2) - f(x_1)}{2}\right)^2 + 2\left(\frac{f(x_2) - f(x_1)}{2}\right)^2 = (f(x_2) - f(x_1))^2,$$

which gives us the claim. □

We now need to show that area-looseness is the right property for bounding the accuracy of an integral approximation on an interval. To do this, we prove an analogue of Proposition 5:

Proposition 7 *Let $f\colon [0,1] \to \mathbb{R}$ be Lipschitz and let $[x_1, x_2]$ be a subinterval of $[0,1]$. Then:*
(1) $\left|(x_2 - x_1) \cdot \frac{f(x_1)+f(x_2)}{2} - \int_{x_1}^{x_2} f(x)\,dx\right| \leq AL(x_1, x_2)/2$.
(2) *For any $\hat{M} \in \mathbb{R}$, there is a Lipschitz function g such that $g(x_1) = f(x_1)$, $g(x_2) = f(x_2)$, and $\left|\hat{M} - \int_{x_1}^{x_2} g(x)\,dx\right| \geq AL(x_1, x_2)/2$.*

Proof: For part (1), define the functions $HI(x) = \min(f(x_1)+|x-x_1|, f(x_2)+|x_2-x|)$ and $LO(x) = \max(f(x_1) - |x - x_1|, f(x_2) - |x_2 - x|)$. By the Lipschitz condition,

$LO(x) \le f(x) \le HI(x)$ on $[x_1, x_2]$. Therefore,

$$\int_{x_1}^{x_2} LO(x)\,dx \le \int_{x_1}^{x_2} f(x)\,dx \le \int_{x_1}^{x_2} HI(x)\,dx$$

so

$$\left| \int_{x_1}^{x_2} \frac{LO(x)+HI(x)}{2}\,dx - \int_{x_1}^{x_2} f(x)\,dx \right| \le \frac{1}{2}\left(\int_{x_1}^{x_2} HI(x)\,dx - \int_{x_1}^{x_2} LO(x)\,dx \right).$$

Note that $f(x_1)+x-x_1 \le f(x_2)+x_2-x$ precisely when $x \le (f(x_2)-f(x_1)+x_1+x_2)/2$. Therefore, we have

$$\begin{aligned}
\int_{x_1}^{x_2} HI(x)\,dx &= \int_{x_1}^{(f(x_2)-f(x_1)+x_1+x_2)/2} (f(x_1)+x-x_1)\,dx + \\
&\quad + \int_{(f(x_2)-f(x_1)+x_1+x_2)/2}^{x_2} (f(x_2)+x_2-x)\,dx = \\
&= \frac{3f(x_1)+f(x_2)+x_2-x_1}{4} \cdot \frac{f(x_2)-f(x_1)+(x_2-x_1)}{2} + \\
&\quad + \frac{f(x_1)+3f(x_2)+x_2-x_1}{4} \cdot \frac{f(x_1)-f(x_2)+(x_2-x_1)}{2} = \\
&= f(x_1) \cdot \frac{f(x_2)-f(x_1)+(x_2-x_1)}{4} + f(x_2) \cdot \frac{f(x_1)-f(x_2)+(x_2-x_1)}{4} + \\
&\quad + \frac{f(x_1)+f(x_2)+x_2-x_1}{4} \cdot (x_2-x_1) = \\
&= (x_2-x_1)\frac{f(x_1)+f(x_2)}{2} + \frac{(f(x_2)-f(x_1))(f(x_1)-f(x_2))}{4} + \frac{(x_2-x_1)^2}{4} = \\
&= (x_2-x_1)\frac{f(x_1)+f(x_2)}{2} + AL(x_1,x_2)/2.
\end{aligned}$$

Similarly, $\int_{x_1}^{x_2} LO(x)\,dx = (x_2-x_1)\frac{f(x_1)+f(x_2)}{2} - AL(x_1,x_2)/2$. Substituting the integrals for HI and LO gives the inequality of (1).

To show part (2), notice that both HI and LO are Lipschitz and $HI(x_1) = LO(x_1) = f(x_1)$ and $HI(x_2) = LO(x_2) = f(x_2)$. If $\hat{M} \ge \int_{x_1}^{x_2} \frac{LO(x)+HI(x)}{2}\,dx$, let

$g(x) = LO(x)$; otherwise let $g(x) = HI(x)$. Then

$$\left|\hat{M} - \int_{x_1}^{x_2} g(x)\,dx\right| \geq \frac{1}{2}\left(\int_{x_1}^{x_2} HI(x)\,dx - \int_{x_1}^{x_2} LO(x)\,dx\right) = \frac{1}{2}\cdot AL(x_1, x_2).$$

and (2) follows. □

We can now formulate the criterion for a proof set for LIPSCHITZ-INTEGRATE:

Corollary 2 *Let $P = \{x_1, x_2, \ldots, x_n\}$ such that $0 \leq x_1 < x_2 < \cdots < x_n \leq 1$. Then P is a proof set for problem instance (f, ϵ) if and only if $x_1^2 + (1 - x_n)^2 + \sum_{i=1}^{n-1} AL(x_i, x_{i+1}) \leq 2\epsilon$.*

Proof: The value $M = x_1 \cdot f(x_1) + (1 - x_n) \cdot f(x_n) + \sum_{i=1}^{n-1}\left((x_{i+1} - x_i) \cdot \frac{f(x_i)+f(x_{i+1})}{2}\right)$ is within ϵ of $\int_0^1 f(x)\,dx$ because

$$\left|x_1 \cdot f(x_1) - \int_0^{x_1} f(x)\,dx\right| \leq \frac{x_1^2}{2}, \qquad \left|(1 - x_n) \cdot f(x_n) - \int_{x_n}^1 f(x)\,dx\right| \leq \frac{(1 - x_n)^2}{2},$$

and for each i, $\left|(x_{i+1} - x_i) \cdot \frac{f(x_i)+f(x_{i+1})}{2} - \int_{x_i}^{x_{i+1}} f(x)\,dx\right| \leq \frac{1}{2} \cdot AL(x_i, x_{i+1})$ by Proposition 7.

In the other direction, if $x_1^2 + (1 - x_n)^2 + \sum_{i=1}^{n-1} AL(x_i, x_{i+1}) > 2\epsilon$, then two functions, f_{HI} and f_{LO}, can be constructed such that $f_{HI}(x_i) = f(x_i) = f_{LO}(x_i)$ but $\int_0^1 f_{HI}(x)\,dx - \int_0^1 f_{LO}(x)\,dx > 2\epsilon$, which means that no matter what value an algorithm outputs after sampling P, that value will be incorrect for either f_{HI} or f_{LO}. □

This corollary, together with Proposition 6, immediately shows the correctness of a trivial algorithm. Let $n = \lceil \epsilon^{-1}/4 \rceil$ and let the algorithm make n samples, at $\frac{1}{2n}, \frac{3}{2n}, \ldots, \frac{2n-1}{2n}$ and output the integral M as in the proof of Corollary 2. It is correct because the area-looseness of every interval is at most $(1/n)^2/2$. Because there are $n - 1$ intervals, the total area-looseness of all of them is at most $(n - 1)/(2n^2)$. Also, $x_1^2 = (1 - x_n)^2 = 1/(2n)^2$, so $x_1^2 + (1 - x_n)^2 + \sum_{i=1}^{n-1} AL(x_i, x_{i+1}) = n/(2n^2) \leq 2\epsilon$. Therefore, as in LIPSCHITZ-APPROX, $\Theta(\epsilon^{-1})$ samples are always sufficient (and if, for instance, f is constant, necessary).

Algorithm LIPSCHITZ-INTEGRATE-ADAPTIVE

L is a linked list of (PARAMETER, VALUE) pairs and Q is a priority queue of (AL, ELEM) pairs where the first element is a real number (and defines the order of Q) and the second element is a pointer into an element of L.

1. Add $(0, f(0))$ and $(1, f(1))$ to L and insert $(AL(0,1), \&(0, f(0)))$ into Q
2. A-LOOSENESS $\leftarrow AL(0,1)$.
3. Do while A-LOOSENESS $> 2\epsilon$:
  4. $(\text{AL}, P_1) \leftarrow$ EXTRACT-MAX$[Q]$
  5. $P_2 \leftarrow$ NEXT$[L, P_1]$
  6. $x \leftarrow (\text{PARAMETER}[P_1] + \text{PARAMETER}[P_2])/2$
  7. $\text{AL}_1 \leftarrow AL(\text{PARAMETER}[P_1], x)$, $\text{AL}_2 \leftarrow AL(x, \text{PARAMETER}[P_2])$
  8. Insert $(x, f(x))$ into L after P_1
  9. Insert $(\text{AL}_1, \&P_1)$ and $(\text{AL}_2, \&(x, f(x)))$ into Q
  10. A-LOOSENESS $\leftarrow$ A-LOOSENESS $-$ AL $+ \text{AL}_1 + \text{AL}_2$
11. Compute and output M using the values stored in L as described in Corollary 2.

Figure 2-6: LIPSCHITZ-INTEGRATE-ADAPTIVE

Figure 2-6 gives an adaptive algorithm for this problem, analogous to LIPSCHITZ-APPROX-ADAPTIVE. The algorithm maintains the total area-looseness of the current unsampled intervals, the unsampled intervals themselves in a linked list, and uses a priority queue to choose the unsampled interval with the largest area-looseness at every step and sample in the middle of it.

The correctness of the algorithm is clear from Corollary 2: the algorithm stops precisely when the total area-looseness of the unsampled intervals is no more than 2ϵ. Next, we analyze the algorithm's performance.

Theorem 3 LIPSCHITZ-INTEGRATE-ADAPTIVE *makes* $O(\text{OPT} \cdot \log(\epsilon^{-1}/\text{OPT}))$ *samples on problem instance* (f, ϵ).

Proof: We actually compare the number of samples to $\text{OPT}(f, \epsilon/2)$ rather than to $\text{OPT}(f, \epsilon)$. We can do this because if we take a proof set for $\text{OPT}(f, \epsilon)$ and sample in the middle of every unsampled interval, then by Proposition 6(4), we obtain a proof set for $\text{OPT}(f, \epsilon/2)$. Thus, $\text{OPT}(f, \epsilon/2) \leq 2 \cdot \text{OPT}(f, \epsilon) + 1$. So let P be a proof set for $(f, \epsilon/2)$ of size $\text{OPT}(f, \epsilon/2)$.

First, we argue that no interval of length smaller than 4ϵ is ever subdivided.

Suppose for contradiction that among n intervals $I_1, \ldots, I_n$ of lengths $a_1, \ldots, a_n$, interval I_k with $a_k < 4\epsilon$ is chosen for subdivision. By Proposition 6(1), $AL(I_i) \leq a_i^2/2$, so $\sqrt{AL(I_k)/2} \leq 2\epsilon$. On the other hand, $\sum a_i = 1$, so $\sum \sqrt{2AL(I_i)} \leq 1$. Multiplying the inequalities, we get $\sum AL(I_i) \leq \sum \sqrt{AL(I_i)AL(I_k)} \leq 2\epsilon$. But this implies that the algorithm should have terminated, which is a contradiction.

Now, we count the number of samples relative to P. The number of splits is $O(|P|)$ by Proposition 1. The above paragraph shows that we can use Proposition 3 to conclude that there are $O(|P| \log(\epsilon^{-1}/|P|))$ squeezes. This does not conclude the analysis because, unlike for the approximation problem, fizzles are now possible. However, we now show that there are $O(|P|)$ fizzles, proving the theorem.

A fizzle occurs when an interval not containing a point of P is chosen for subdivision. Consider the situation after n points have been sampled. Let the sampled points be $0 = x_1 \leq x_2 \leq \cdots \leq x_n = 1$. Because the total area-looseness of intervals between points of P is at most ϵ, by repeated application of Proposition 6(2,3), we have

$$\sum_{[x_i, x_{i+1}) \cap P = \emptyset} AL(x_i, x_{i+1}) \leq \epsilon.$$

The algorithm has not terminated, so the total area-looseness must be more than 2ϵ, which implies that

$$\sum_{[x_i, x_{i+1}) \cap P \neq \emptyset} AL(x_i, x_{i+1}) > \epsilon.$$

Because there are at most $|P|$ elements in the sum on the left-hand side, the largest element must be greater than $\epsilon/|P|$. Therefore, there exists a k such that $[x_k, x_{k+1})$ contains a point of P and $AL(x_k, x_{k+1}) > \epsilon/|P|$. The algorithm always chooses the interval with the largest area looseness, so if a fizzle occurs, the area-looseness of the chosen interval must be at least $\epsilon/|P|$.

Now let S_t be the set of samples made by the algorithm after time t. Define A_t as follows: let $\{y_1, y_2, \ldots, y_n\} = S_t \cup P$ with $0 = y_1 \leq y_2 \leq \cdots \leq y_n$ and let $A_t = \sum_{i=1}^{n-1} AL(y_i, y_{i+1})$. Clearly, $A_t \geq 0$, $A_t \geq A_{t+1}$ (by Proposition 6(3)), and therefore, $A_t \leq A_0 \leq 2\epsilon$. Every fizzle splits an interval between adjacent y's into

two. Because the area-looseness of the interval before the split was at least $\epsilon/|P|$, by Proposition 6(4), A_t decreases by at least $\epsilon/(2|P|)$ as a result of every fizzle. Therefore, there can be at most $4|P|$ fizzles during an execution. □

We now prove a matching lower bound that shows that the logarithmic factor is necessary and that LIPSCHITZ-INTEGRATE-ADAPTIVE is optimally adaptive:

Theorem 4 *For any deterministic algorithm and for any $\epsilon > 0$ and any integer k such that $0 < k < \epsilon^{-1}/2$, there exists a problem instance (f, ϵ) of* LIPSCHITZ-INTEGRATE *with $\mathrm{OPT}(f, \epsilon) = O(k)$ on which that algorithm performs $\Omega(k \log(\epsilon^{-1}/k))$ samples.*

Proof: The proof is very similar to the proof of Theorem 2. The construction is essentially the same, but with different constants. Let $n = k\lfloor 1/\sqrt{2\epsilon k}\rfloor$. Divide the parameter space $[0, 1]$ into n equal regions and group them into k groups of n/k regions each. In each group, let $n/k - 1$ regions have slope 1 and let 1 region have slope -1 (see Figure 2-4 for an illustration). As with approximation, sampling at 0, 1, and every point where the slope of f changes is a proof set because the area looseness of all unsampled intervals is 0. This implies that for any such function, $\mathrm{OPT} \leq 2k + 2 = O(k)$.

Now we show that any algorithm needs additional samples on some function of this type. Consider a group, and consider an interval I consisting of the region on which f has negative slope in that group and an adjacent region in that group. Because the value of f is the same on the endpoints of I, $AL(I) = 2/n^2 \geq 4\epsilon/k$. Therefore, an algorithm that does not find the negative-slope region or an adjacent one in at least $k/2$ groups will not be correct on some inputs. Finding the negative-slope region in each group is a binary search that needs $\Omega(\log(n/k)) = \Omega(\log(\epsilon^{-1}/k))$ samples and this needs to be done independently for $\Omega(k)$ groups, giving us the bound. □

The above theorem shows that LIPSCHITZ-INTEGRATE-ADAPTIVE cannot be improved by more than a constant factor with respect to the number of samples. However, notice that in addition to sampling f, the algorithm maintains numbers in a

priority queue Q. If Q is implemented as a heap, then an execution that makes n samples requires an additional $O(n \log n)$ time for the extract-max operations on Q.

In fact, this logarithmic overhead can be avoided. The key to eliminating this overhead is to observe that a strict priority queue is not required for the algorithm. As long as the extract-max operation guarantees that the value of the element extracted is at least half (or any constant factor) of the maximum value stored in Q, the proof of Theorem 3 holds (although the constants increase). Therefore, an approximate priority queue is sufficient. Furthermore, notice that the queue stores area-loosenesses of unsampled intervals, which can vary from 0 to 1. However, if n is the number of samples made by the algorithm, then no interval whose area-looseness is smaller than $2\epsilon/n$ is ever subdivided, and because n is no more than ϵ^{-1}, the queue only needs to store numbers between $2\epsilon^2$ and 1. Finally, notice that the maximum value in Q is always decreasing.

The above observations allow us to implement Q so that all operations can be done in $O(n + \log(1/\epsilon))$ time. The implementation consists of k buckets numbered 1 through k, where $k = \lfloor \log_2(\epsilon^{-2}) \rfloor$ and a variable that initially points to bucket 1. Bucket i stores values in the range $[2^{-i}, 2^{1-i})$. An insert operation simply inserts the value into the proper bucket. An extract-max operation first increments the variable until it points to a non-empty bucket and then extracts any value from that bucket. Insertion takes constant time per operation (assuming that computing the floor of the log of a number takes constant time) and extraction takes constant time plus the time to increment the variable. Since there are only $\Theta(\log(1/\epsilon))$ buckets, the total time for all of the operations is $O(n + \log(1/\epsilon))$.

Chapter 3

Functions from $\mathbb{R}$ to $\mathbb{R}^d$

3.1 Lipschitz Curves

When the range of a univariate Lipschitz function is $\mathbb{R}^d$ with $d > 1$, such a function is a parametric curve. The study of adaptive algorithms on Lipschitz curves was initiated in [BD04]. In this thesis, we apply similar techniques to different problems.

The observation that leads to the algorithms is that once two points on a curve are sampled, the curve between these two points must lie inside an ellipse. The following three propositions (proved in [BD04]) formalize this idea.

Given a Lipschitz curve C and an interval $[x_1, x_2] \subseteq [0, 1]$, define the ellipse

$$E_C(x_1, x_2) = \Big\{ p \in \mathbb{R}^d \;\Big|\; \|C(x_1) - p\| + \|C(x_2) - p\| \leq x_2 - x_1 \Big\}.$$

See Figure 3-1.

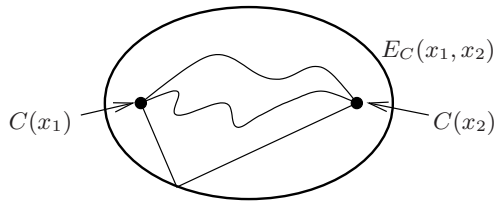

Figure 3-1: Some possible curves C inside an ellipse.

Proposition 8 *For an interval $J = [x_1, x_2] \subseteq [0, 1]$, $C(J) \subseteq E_C(x_1, x_2)$.*

Proposition 9 *Let $J = (x_1, x_2) \subseteq [0, 1]$ and let C be a Lipschitz curve. Then for every point p in $E_C(x_1, x_2)$, there is a Lipschitz curve C' such that $C(x) = C'(x)$ for $x \notin J$ and for some $x \in J$, $C'(x) = p$.*

Proposition 10 *If $J' = [x'_1, x'_2]$ and $J = [x_1, x_2]$ and $J' \subseteq J$, then $E_C(x'_1, x'_2) \subseteq E_C(x_1, x_2)$.*

3.2 Approximation

We now consider the approximation problem when the range of the function is $\mathbb{R}^d$. There is no fundamental difference from the one-dimensional case. We can even generalize LIPSCHITZ-APPROX-ADAPTIVE and Theorem 1 to the setting where the range is an arbitrary metric space M (with metric d_M). We define the generalized looseness as follows: Let $f\colon [0, 1] \to M$ be a function, let $G_{(x_1,x_2)}$ be the set of Lipschitz functions equal to f on $[0, x_1]$ and on $[x_2, 1]$, and let H be the set of all functions from $[0, 1]$ to M. Define

$$GL_f(x_1, x_2) = \min_{h \in H} \max_{g \in G_{(x_1,x_2)}} \max_{x \in [x_1,x_2]} d_M(g(x), h(x)).$$

In other words, GL is the maximum possible error on $[x_1, x_2]$ of the best possible approximator, h. Note that for $M = \mathbb{R}$, generalized looseness is precisely half of regular looseness. Note also that like looseness, GL does not depend on f other than at x_1 and x_2, so computing GL does not require additional samples.

In order to get an algorithm for solving LIPSCHITZ-APPROX when the range of f is M, we modify LIPSCHITZ-APPROX-ADAPTIVE to compute GL instead of L in Step 5, to compare GL to ϵ in step 6, and to output a pasting of the best-possible approximators (the h's) in Step 9. Correctness follows from the definition of GL: the algorithm stops only when there is a function that approximates f with error ϵ on every interval.

We now analyze the performance of the modified algorithm. We start by proving an analogue of the first part of Proposition 4 for GL:

Proposition 11 *Given f, and an interval $[x_1, x_2]$, $0 \leq GL(x_1, x_2) \leq (x_2 - x_1)/2$.*

Proof: Since $d_M \geq 0$, $GL(x_1, x_2) \geq 0$. For the second inequality, let $x_m = (x_1 + x_2)/2$ and let $h(x) = f(x_1)$ for $x \leq x_m$ and $h(x) = f(x_2)$ for $x > x_m$. Now consider any Lipschitz function g with $g(x_1) = f(x_1)$ and $g(x_2) = f(x_2)$ and consider any point $x \in [x_1, x_2]$. If $x \leq x_m$, then

$$d_M(g(x), h(x)) = d_M(g(x), f(x_1)) \leq d_M(g(x_1), g(x)) + d_M(g(x_1), f(x_1)) \leq$$
$$\leq |x - x_1| + 0 \leq x_m - x_1 = (x_2 - x_1)/2.$$

Similarly, if $x > x_m$, we also get $d_M(g(x), h(x)) \leq (x_2 - x_1)/2$. Therefore, there exists an h such that for any g and x, the error is smaller than $(x_2 - x_1)/2$. □

We are now ready to prove that the algorithm is within a logarithmic factor of OPT. The proof is analogous to that of Theorem 1.

Theorem 5 *The modified algorithm for approximation uses $O(\mathrm{OPT} \cdot \log(\epsilon^{-1}/\mathrm{OPT}))$ samples on problem instance (f, ϵ).*

Proof: Let P be a set of points that OPT samples on (f, ϵ). By Proposition 11, no interval of length smaller than 2ϵ is ever subdivided. Therefore, by Propositions 1 and 3, the number of splits and squeezes is $O(|P| \log(\epsilon^{-1}/|P|))$. We now show that there are no fizzles.

Suppose, for contradiction, that the algorithm samples in the interval (x_1, x_2), but there are no points of P in that interval. It must be the case that $GL(x_1, x_2) > \epsilon$, because otherwise the modified algorithm would not sample there. Now let h be the function that OPT outputs. By definition of GL, there exists a Lipschitz g that is equal to f everywhere except possibly on (x_1, x_2), such that $d_M(g(x), h(x)) > \epsilon$ for some x. But then if OPT is given the problem instance (g, ϵ), it would also output h because f and g are the same everywhere OPT samples, but h would not be a correct output. □

We now show using messy algebra that GL is easy to compute when M is Euclidean d-space.

Proposition 12 *If M is $\mathbb{R}^d$ with the Euclidean metric and $d > 1$, then $GL(x_1, x_2) = \sqrt{(x_2 - x_1)^2 - \|f(x_2) - f(x_1)\|^2}/2$.*

Proof: Let e_i be the unit vectors. Rotate and translate the coordinate system so that $f(x_1)$ is the origin and e_1 points towards $f(x_2)$. Let $a = \|f(x_2) - f(x_1)\|$. Then, $f(x_2)$ is at ae_1.

Notice that the points $e_1 a/2 \pm e_2\sqrt{(x_2 - x_1)^2 - a^2}/2$ are both at distance $(x_2 - x_1)/2$ from both $f(x_1)$ and $f(x_2)$. Therefore, there are Lipschitz functions g_1 and g_2 equal to f outside (x_1, x_2) such that $g_1((x_2 - x_1)/2) = e_1 a/2 + e_2\sqrt{(x_2 - x_1)^2 - a^2}/2$ and $g_2(x_2 - x_1/2) = e_1 a/2 - e_2\sqrt{(x_2 - x_1)^2 - a^2}/2$. At that point, the distance between g_1 and g_2 is $\sqrt{(x_2 - x_1)^2 - a^2}$ and therefore any function h is at distance at least $\sqrt{(x_2 - x_1)^2 - a^2}/2$ from one of them. Therefore, $GL(x_1, x_2) \geq \sqrt{(x_2 - x_1)^2 + a^2}/2$.

If $a = 0$, then the Lipschitz condition immediately implies that $h(x) = f(x_1)$ is within $(x_2 - x_1)/2$ of any g and the proposition follows. If $a = x_2 - x_1$, let $h(x) = (x - x_1)e_1$. Then $h = g$ and the proposition follows. So assume $0 < a < x_2 - x_1$. Let $v(x) = \left(\frac{x_2 - x_1}{a}\right) x + \frac{a^2 + x_1^2 - x_2^2}{2a}$ and let $h(x) = v(x)e_1$. Consider a Lipschitz function g with $g = f$ everywhere but on (x_1, x_2) and fix an x. We need to show that $\|g(x) - h(x)\| \leq \sqrt{(x_2 - x_1)^2 - a^2}/2$. Rotate the coordinate system (keeping 0 and e_1 fixed) so that $g(x) = be_1 + ce_2$ for some b and c. Then $\|g(x) - h(x)\| = \sqrt{c^2 + (b - v(x))^2}$. Assume without loss of generality that $b \geq v(x)$ (otherwise, exchange $f(x_1)$ and $f(x_2)$). By the Lipschitz condition, $b^2 + c^2 \leq (x - x_1)^2$. Therefore, $c^2 + (b - v(x))^2 \leq (x - x_1)^2 - 2bv(x) + v(x)^2 \leq (x - x_1)^2 - v(x)^2$. So we need to prove that $4(x - x_1)^2 - 4v(x)^2 \leq (x_2 - x_1)^2 - a^2$. Note that the right-hand side does not depend on x and the left-hand side is quadratic in x. The left-hand side can be written as:

$$\left(4 - 4\left(\frac{x_2 - x_1}{a}\right)^2\right) x^2 + \left(-8x_1 - \frac{4(x_2 - x_1)(a^2 + x_1^2 - x_2^2)}{a^2}\right) x + q$$

where q does not depend on x. The coefficient of x^2 is negative, so the left-hand side

has a global maximum at

$$\frac{\left(8x_1 + \frac{4(x_2-x_1)(a^2+x_1^2-x_2^2)}{a^2}\right)}{\left(8 - 8\left(\frac{x_2-x_1}{a}\right)^2\right)} = \frac{x_1a^2 + (x_2-x_1)(a^2+x_1^2-x_2^2)/2}{a^2-(x_2-x_1)^2} =$$

$$= \frac{x_1a^2 + x_2a^2 + (x_2-x_1)(x_1^2-x_2^2)}{2(a^2-(x_2-x_1)^2)} = \frac{x_1+x_2}{2}.$$

But at $x = (x_1+x_2)/2$, $v(x) = a/2$, so $4(x-x_1)^2 - 4v(x)^2 = (x_2-x_1)^2 - a^2$ and the proposition holds. □

Finally, we note that Theorem 2, the lower bound on adaptive approximation algorithms, carries over without modification to $d > 1$. Therefore, the modified approximation algorithm is optimally adaptive.

3.3 Winding Number

We now consider the problem of computing the winding number of a curve in two dimensions around a given point (which is assumed to be the origin).

Problem WINDING-NUMBER:

Given:	$C\colon [0,1] \to \mathbb{R}^2$
Such that:	$C(0) = C(1)$, for all $x \in [0,1]$, $C(x) \neq \mathbf{0}$,
	and for $x_1, x_2 \in [0,1]$, $\|C(x_2) - C(x_1)\| \leq \|x_2 - x_1\|$
Compute:	The winding number of C around the origin

We first characterize proof sets for WINDING-NUMBER.

Proposition 13 *Let $P = \{x_1, x_2, \ldots, x_n\}$ with $0 = x_1 < x_2 < \cdots < x_n = 1$. P is a proof set for problem instance C if and only if for all i the origin is not in the interior of $E_C(x_i, x_{i+1})$.*

Proof: First we show that if no ellipse interior contains the origin, P is a proof set. Given a curve C' with $C'(x_i) = C(x_i)$, let $H : [0,1] \times [0,1] \to \mathbb{R}^2$ be defined

by $H(x,t) = t \cdot C'(x) + (1-t) \cdot C(x)$. Clearly H is a homotopy between C and C'. Suppose that $H(x,t) = \mathbf{0}$ for some x and t. Let x_k and x_{k+1} be consecutive points in P such that $x_k \leq x \leq x_{k+1}$. If $x_{k+1} - x_k = \|C(x_{k+1}) - C(x_k)\|$, then $C'(x) = C(x)$ and by assumption, the image of C does not contain the origin. Otherwise $C'(x) \neq \mathbf{0}$, $C(x) \neq \mathbf{0}$ and both $C'(x)$ and $C(x)$ are points in $E_C(x_k, x_{k+1})$. Because E_C is convex and its boundary does not contain any line segments, no convex combination of $C'(x)$ and $C(x)$ contains the origin, contradicting the assumption that $H(x,t) = \mathbf{0}$. Therefore, C' and C have a homotopy in $\mathbb{R}^2 - \{\mathbf{0}\}$ and hence the same winding number. In particular, C' can be a parameterization of the polygon with vertices at $C(x_i)$, so that polygon has the same winding number as C.

Conversely, if the origin is in the interior of $E_C(x_k, x_{k+1})$, we show that there are two Lipschitz functions, C_1 and C_2 such that $C_1(x_i) = C_2(x_i) = C(x_i)$ but C_1 and C_2 have different winding numbers. Let $C_1 = C_2 = C$ everywhere except on (x_k, x_{k+1}). Now let $a = x_{k+1} - x_k - \|C(x_{k+1})\| - \|C(x_k)\|$. Because the interior of $E_C(x_k, x_{k+1})$ contains the origin, $a > 0$. Let p be a point such that $\|p\| \leq a/14$ and such that p is not on the line segment between $C(x_k)$ and the origin nor on the line segment between $C(x_{k+1})$ and the origin. Then by the triangle inequality, $\|C(x_k) - p\| + \|C(x_{k+1}) - p\| \leq x_{k+1} - x_k - a/7$. Now let C_1 on (x_k, x_{k+1}) consist of the line segment from $C(x_k)$ to p followed by the line segment from p to $C(x_{k+1})$. Let C_2 on (x_k, x_{k+1}) consist of the line segment from $C(x_k)$ to p, followed by the counterclockwise circle of radius $a/14$ (arclength smaller than $a/2$) around the origin followed by the line segment from p to $C(x_{k+1})$. See Figure 3-2. Then the winding number of C_2 is one greater than that of C_1. □

A set of points may be a proof set even if it doesn't contain 0 or 1, but the smallest proof set that does contain 0 and 1 has at most two points more than the overall smallest proof set.

Notice that no bound on the number of samples necessary for this problem can be given without knowing C. If C is contained in a disk of radius ϵ around the origin, $\Omega(1/\epsilon)$ samples are necessary to determine the winding number. However, we can

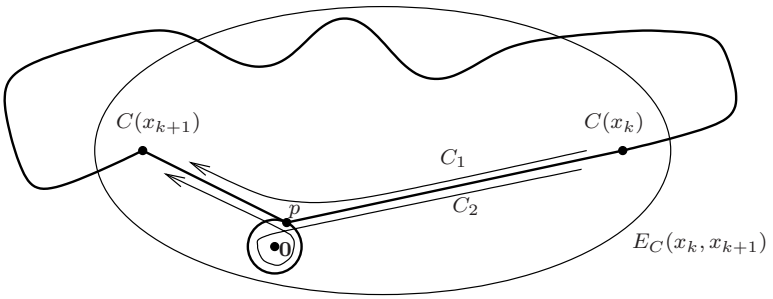

Figure 3-2: Illustration for the second part of the proof of Proposition 13.

Algorithm WINDING-NUMBER-ADAPTIVE

L is a linked list of (PARAMETER,POINT) pairs and S is a stack of pointers into objects in L.

1. Add $(0, C(0))$ and $(1, C(1))$ to L and push a pointer to $(0, C(0))$ onto S
2. Do until S is empty:
  3. $P_1 \leftarrow \text{TOP}[S]$
  4. $P_2 \leftarrow \text{NEXT}[L, P_1]$
  5. $\text{SURPLUS} \leftarrow \text{PARAMETER}[P_2] - \text{PARAMETER}[P_1] - \|\text{POINT}[P_1]\| - \|\text{POINT}[P_2]\|$
  6. If $\text{SURPLUS} \leq 0$ then $\text{POP}(S)$ and continue to Step 2
  7. $x \leftarrow \text{PARAMETER}[P_1] + \|\text{POINT}[P_1]\| + \text{SURPLUS}/2$
  8. Insert $(x, C(x))$ into L after P_1 and push a pointer to it onto S
9. Output the winding number of the polygon defined by the points in L around $\mathbf{0}$

Figure 3-3: WINDING-NUMBER-ADAPTIVE

give an adaptive algorithm that makes at most twice as many samples as OPT on any problem instance (see Figure 3-3).

The algorithm proceeds analogously to LIPSCHITZ-APPROX-ADAPTIVE. As it samples the curve, it maintains a set of ellipses around the unsampled intervals. At each step, it samples in some interval (it doesn't matter which) whose ellipse contains the origin, thus replacing it with two smaller intervals (with smaller ellipses). When no ellipse contains the origin, the algorithm computes the winding number of the polygon formed by connecting all the sampled points (this is a simple linear-time operation) and outputs it. A key point is that we don't sample in the middle of an unsampled interval—the location where we sample in the parameter space depends on the geometry of the already sampled points.

The algorithm is correct because it only stops subdividing an interval once the ellipse around it does not contain the origin (i.e. SURPLUS ≤ 0) and so always samples a proof set.

We now show that WINDING-NUMBER-ADAPTIVE is instance optimal.

Theorem 6 *On problem instance C, algorithm* WINDING-NUMBER-ADAPTIVE *performs at most $2 \cdot \mathrm{OPT}(C)$ samples.*

Proof: Let P be a proof set of size $\mathrm{OPT}(C)$.

Let us call an interval *alive* if its ellipse interior contains the origin. Clearly, the algorithm only subdivides alive intervals. We now show that when WINDING-NUMBER-ADAPTIVE makes a sample, if one of the resulting intervals is alive, then so is the other.

Suppose that WINDING-NUMBER-ADAPTIVE samples at x inside the previously unsampled interval (x_1, x_2). This can only happen when SURPLUS > 0, that is, $\|C(x_1)\| + \|C(x_2)\| < x_2 - x_1$. From Step 7,

$$x = x_1 + \|C(x_1)\| + \text{SURPLUS}/2 = \frac{x_2 + x_1 + \|C(x_1)\| - \|C(x_2)\|}{2}.$$

Now

$$x - x_1 - \|C(x)\| - \|C(x_1)\| = \frac{x_2 - x_1 - \|C(x_1)\| - \|C(x_2)\|}{2} - \|C(x)\|$$

and

$$x_2 - x - \|C(x)\| - \|C(x_2)\| = \frac{x_2 - x_1 - \|C(x_1)\| - \|C(x_2)\|}{2} - \|C(x)\|$$

so

$$x - x_1 - \|C(x)\| - \|C(x_1)\| = x_2 - x - \|C(x)\| - \|C(x_2)\| = \text{SURPLUS}/2 - \|C(x)\|.$$

So if SURPLUS$/2 \geq \|C(x)\|$, then neither resulting interval is alive, otherwise, both are alive. Call a sample a *birth sample* if both resulting intervals are alive and call it

a *death sample* otherwise.

Notice that if an interval (x_1, x_2) is alive, then it must contain a point of P; otherwise P is not a proof set, by Propositions 10 and 13. Therefore, every time a birth sample is made, this sample is a split. So by Proposition 1, there are at most $|P| - 1$ birth samples.

Let A_t be the number of alive intervals at time t in the execution of WINDING-NUMBER-ADAPTIVE. Because we start with one interval, $A_0 \leq 1$. Every birth sample increases A_t by 1 and every death sample decreases it by 1. But A_t is always positive, so the number of death samples is at most one greater than the number of birth samples. Therefore, the total number of birth and death samples is at most $2|P| - 1$. Adding the extra sample in Step 1 that establishes $C(0)$ (and $C(1)$), we find that WINDING-NUMBER-ADAPTIVE makes at most $2|P|$ samples. □

We can show that the factor of 2 over OPT is necessary for deterministic algorithms:

Theorem 7 *For any deterministic algorithm and any $k > 0$, there exists a problem instance C of* WINDING-NUMBER *for which $k \leq \mathrm{OPT}(C) \leq 2k$, but the algorithm performs at least $2 \cdot \mathrm{OPT}(C) - 1$ samples on C.*

Proof: Fix the algorithm and k. Consider the execution of the algorithm on problem instance $C(x) = (0, 1/4k)$, the constant curve at distance $1/4k$ from the origin. Let $x_1, x_2, \ldots, x_n$ be the points where the algorithm samples on this problem instance. The OPT for this problem instance is clearly $2k$, so if $n \geq 4k$, we are done. Otherwise, we construct a problem instance C', such that $C'(x_i) = C(x_i)$ for all i (so the algorithm makes the same n samples on C' as on C), but $\mathrm{OPT}(C') \leq \lceil n/2 \rceil$.

The curve C' also lives on the y axis and is constructed to have the maximum possible y coordinate at every point, while still obeying the Lipschitz condition and being equal to C on the x_i's. See Figure 3-4. C' has a local maximum at parameter value $(x_i + x_{i+1})/2$ for every i and has value $(0, 1/4k + (x_{i+1} - x_i)/2)$ there. We show that taking every other local maximum is sufficient for a proof set. In the following paragraph, we work with indices mod n and parameter values mod 1 (this is justifiable

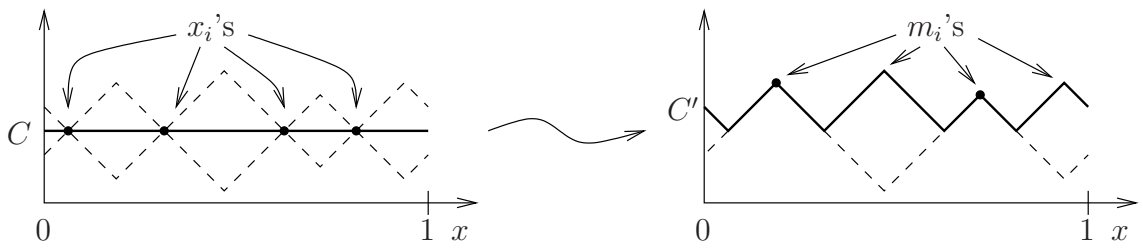

Figure 3-4: Construction of C'. Lipschitz bounds are dashed.

since we must have $C(0) = C(1)$). Also, since C' is entirely on the y axis, we work with it as if it were a real-valued function.

Let $m_i = (x_i + x_{i+1})/2$. To show that $m_1, m_3, \ldots, m_n$ form a proof set, it suffices to show that $C'(m_i) + C'(m_{i+2}) \geq m_{i+2} - m_i$. We have

$$C'(m_i) + C'(m_{i+2}) = 1/2k + (x_{i+1} + x_{i+3} - x_i - x_{i+2})/2.$$

Because the x_i's form a proof set, $C(x_{i+1}) + C(x_{i+2}) \geq x_{i+2} - x_{i+1}$, so $0 \geq x_{i+2} - x_{i+1} - 1/2k$. Adding this to the above equation gives

$$C'(m_i) + C'(m_{i+2}) \geq (-x_{i+1} + x_{i+3} - x_i + x_{i+2})/2 = (m_{i+2} - m_i),$$

as required. Therefore, $\mathrm{OPT}(C')$ is at most half of $n + 1$. On the other hand, $\|C'\| < 1/2k$, so any adjacent points in a proof set cannot be farther apart than $1/k$ in parametric space. Hence, $\mathrm{OPT}(C') \geq k$. □

It seems likely that a randomized algorithm (such as selecting the initial point randomly and running the deterministic algorithm) could improve the constant, but so far, we have been unable to prove it.

Chapter 4

Noisy Samples

4.1 Noisy Approximation Problem

So far we have assumed that sampling f always returns a precise value. Now we investigate what happens if the samples an algorithm makes are corrupted by noise. For simplicity, we assume that the noise is normally distributed, has a known variance σ^2, and that all samples are independent. The performance of algorithms, including OPT, is now measured as the expected number of samples performed on a problem instance.

Due to the noise, it is unavoidable that some executions of the algorithm will be completely wrong. A natural desired property is that the algorithm should be (ϵ, δ)-accurate. This means that the probability of the output error being greater than ϵ should be less than δ. In other words, the user specifies the desired accuracy and how often the algorithm is allowed to fail to achieve this accuracy. Note that the probability of failure is taken over the executions of the algorithm on a particular problem instance and no prior distribution is assumed over problem instances.

Formally, the problem is defined as follows:

Problem NOISY-LIPSCHITZ-APPROX:

Given:	$(f, \sigma, \epsilon, \delta)$ where $f: [0,1] \to \mathbb{R},\ 0 < \sigma,\ 0 < \epsilon < 1/2,\ 0 < \delta < 1/12$
Such that:	For $x_1, x_2 \in [0,1]$, $\lvert f(x_2) - f(x_1) \rvert \leq \lvert x_2 - x_1 \rvert$
	and sampling at x returns a sample from $N(f(x), \sigma^2)$
Compute:	$\hat{f} : [0,1] \to \mathbb{R}$ such that for all $x \in [0,1]$, $\lvert f(x) - \hat{f}(x) \rvert \leq \epsilon$
	The output should be correct with probability at least $1 - \delta$.

Notice that we don't require the output function to be Lipschitz or even continuous. Moreover, our algorithm potentially outputs a discontinuous function. As shown at the end of the proof of Proposition 17, this is not a serious limitation.

For this problem, there is no distinction between deterministic and randomized algorithms because a deterministic algorithm may obtain random numbers simply by sampling the function, while δ allows a probability of failure.

In order to carry out an adaptive analysis of an algorithm for NOISY-LIPSCHITZ-APPROX, we need to compare its execution to that of the best possible algorithm for the problem. However, because now an algorithm is allowed to fail with a certain probability, it is more difficult to characterize OPT. In order to get around this difficulty, we compare the algorithm's performance to NOPT, which we define to be the number of samples performed by the best possible algorithm for the corresponding noiseless instance. We then show that $\mathrm{NOPT}(f, \epsilon)$ is asymptotically related to $\mathrm{OPT}(f, \sigma, \epsilon, \delta)$.

We start by proving an additional property of NOPT, which we didn't need in Section 2.2, but which we need here. We show that decreasing ϵ does not increase NOPT too much:

Proposition 14 $\mathrm{NOPT}(f, \epsilon/2) \leq 2 \cdot \mathrm{NOPT}(f, \epsilon) + 1$

Proof: We show that given a set of n samples x_i that satisfies the conditions of Corollary 1 for f and ϵ, there is a set of $2n+1$ samples that satisfies these conditions for f and $\epsilon/2$. First note that sampling at $\epsilon/2$ and $1-\epsilon/2$ satisfies the "end" conditions at

a cost of 2 additional samples. Now for all i with $1 \leq i \leq n-1$, we show we can insert an additional sample x between x_i and x_{i+1} so that $L(x_i, x) \leq \epsilon$ and $L(x, x_{i+1}) \leq \epsilon$. This will result in $n-1$ additional samples, for a total of $n+2+n-1 = 2n+1$ samples.

Fix a particular i. Let $a = L(x_i, x_{i+1})$. We know that $a \leq 2\epsilon$. Suppose that $f(x_{i+1}) \geq f(x_i)$ (the other case is symmetric) and consider the function

$$g(x) = f(x_i) + (x - x_i) - a/2 = f(x_{i+1}) - (x_{i+1} - x) + a/2$$

Clearly, $f(x_i) \geq g(x_i)$ and $f(x_{i+1}) \leq g(x_{i+1})$. Because g is continuous and Lipschitz functions are continuous, by the intermediate value theorem, there is an $x \in [x_i, x_{i+1}]$ for which $f(x) = g(x)$. This will be our new sample—we show $L(x_i, x) \leq \epsilon$:

$$\begin{aligned} L(x_i, x) &= (x - x_i) - |f(x) - f(x_i)| = \\ &= (x - x_i) - |g(x) - f(x_i)| = \\ &= (x - x_i) - |f(x_i) + (x - x_i) - a/2 - f(x_i)| = \\ &= (x - x_i) - |(x - x_i) - a/2| \leq a/2 \leq \epsilon. \end{aligned}$$

Similarly, we can show that $L(x, x_{i+1}) \leq \epsilon$. □

4.2 Sampling the Normal Distribution

To solve the noisy approximation problem, we use an (ϵ, δ)-accurate algorithm for estimating the mean of a normal distribution as a building block. We start by considering the mean of a fixed number of samples as the estimator.

Proposition 15 *Consider the normal distribution $N(\mu, \sigma^2)$ and let $G(\sigma, \epsilon, \delta)$ be the minimum number of samples from $N(\mu, \sigma^2)$ necessary for the sample mean to be an (ϵ, δ)-accurate estimate of μ (with $\delta < 1/3$). Then $G(\sigma, \epsilon, \delta) = \Theta(\frac{\sigma^2 \log(1/\delta)}{\epsilon^2})$.*

Proof: Assume without loss of generality that $\mu = 0$. Because the normal distribution is symmetric, the probability that the sample mean is off by more than

ϵ is exactly twice the probability that it is greater than ϵ. The sample mean of n samples is distributed as $N(0, \sigma^2/n)$. Thus, we want to determine the minimum n such that if $\hat{\mu}$ is a sample from $N(0, \sigma^2/n)$, then $\Pr[\hat{\mu} > \epsilon] < \delta/2$. Equivalently, we need to have $\Pr[c\epsilon^{-1}\hat{\mu} > c] < \delta/2$. Let $c = \sqrt{n}\epsilon/\sigma$. Note that $c\epsilon^{-1}\hat{\mu}$ is distributed as $N(0, c^2\sigma^2\epsilon^{-2}/n) = N(0, 1)$. Therefore, we need to show that if c is the smallest number such that the probability that a sample from $N(0, 1)$ is greater than c is smaller than $\delta/2$, then $c = \Theta(\sqrt{\log(1/\delta)})$.

The probability density function of $N(0, 1)$ is $e^{-x^2/2}/\sqrt{2\pi}$. We need $\int_c^\infty \frac{e^{-x^2/2}}{\sqrt{2\pi}}\, dx < \delta/2$. Equivalently, we need $\int_{c/\sqrt{2}}^\infty e^{-y^2}\, dy < \delta\sqrt{\pi/4}$. It is well known (see [AS72]) that

$$\frac{e^{-c^2/2}}{c/\sqrt{2} + \sqrt{c^2/2 + 2}} \leq \int_{c/\sqrt{2}}^\infty e^{-y^2}\, dy \leq \frac{e^{-c^2/2}}{c/\sqrt{2} + \sqrt{c^2/2 + 4/\pi}}.$$

From the right inequality, if $c > \sqrt{2\log(1/\delta)}$, then $\int_{c/\sqrt{2}}^\infty e^{-y^2}\, dy < \delta\sqrt{\pi/4}$. On the other hand, from the left inequality, if $c < \sqrt{\log(1/\delta)/5}$, we have

$$\int_{c/\sqrt{2}}^\infty e^{-y^2}\, dy \geq \frac{e^{-c^2/2}}{c/\sqrt{2} + \sqrt{c^2/2 + 2}} \geq \frac{\delta^{1/10}}{2\sqrt{\log(1/\delta)/10 + 2}} \geq \frac{\delta^{1/10}}{\sqrt{2/5\delta + 38/5}} =$$

$$= \delta\sqrt{\pi/4}\sqrt{\frac{\delta^{1/5}/(\delta^2\pi/4)}{2/5\delta + 38/5}} = \delta\sqrt{\pi/4}\sqrt{\frac{10}{\pi(\delta^{4/5} + 19\delta^{9/5})}}.$$

So $\int_{c/\sqrt{2}}^\infty e^{-y^2}\, dy > \delta\sqrt{\pi/4}$ if $\delta \leq 1/3$. □

Thus, an algorithm to find an (ϵ, δ)-accurate value of f at x is to sample $G(\sigma, \epsilon, \delta)$ times and return the average. The following proposition shows that it is impossible to do better and will be useful in proving lower bounds on OPT. Unfortunately, the proof is quite messy and we have not been able to find a cleaner one.

Proposition 16 *Any algorithm that distinguishes between $N(0, \sigma^2)$ and $N(\epsilon, \sigma^2)$, with error probability at most $\delta < 1/12$, uses $\Omega(G(\sigma, \epsilon, \delta))$ samples in expectation when the true distribution is $N(0, \sigma^2)$.*

Proof: In general, at every step, the algorithm may choose to either sample the distribution, or output an answer. Call an algorithm *symmetric* if it outputs 0 after samples $x_1, \ldots, x_k$ whenever it outputs ϵ after $\epsilon - x_1, \ldots, \epsilon - x_k$. We first show that for every algorithm, there is a symmetric algorithm whose performance is within a constant factor.

We start by taking the algorithm that distinguishes between $N(0, \sigma^2)$ and $N(\epsilon, \sigma^2)$ with error probability at most δ and transforming it by amplification (this is where we lose the constant) to have error at most $\delta/2$. We now transform it to be symmetric by simultaneously simulating two "copies" of the algorithm. Each new sample x is fed unchanged to the first copy, and as $\epsilon - x$ to the second copy. If the first copy outputs a (and the second copy either outputs $\epsilon - a$ or needs more samples), the algorithm stops and outputs a. If the second copy outputs a, the algorithm stops and outputs $\epsilon - a$. If both copies output a, the algorithm outputs 0 or ϵ with equal probability. This algorithm errs with probability at most δ because by the union bound, with probability δ, neither copy makes a mistake. The expected number of samples this algorithm performs is no larger than that of either copy, since it terminates whenever either copy terminates.

Now let k be the expected number of samples the symmetric algorithm makes. By Markov's inequality, the probability that the algorithm requires more than $4k$ samples is less than $1/4$. So we may relax the conditions of the algorithm to turn it into one that does the following: it first performs $4k$ samples, then with probability $3/4$ it must output an answer; with probability $1/4$ it is allowed to ask an oracle that always gives the correct answer. If the algorithm decides to output the answer, it can make errors with probability at most $4\delta/3$. Let ERR be the event that the algorithm outputs incorrectly and $STEPS4k$ be the event that it outputs without asking the oracle. Then:

$$\Pr[ERR] = \Pr[ERR \wedge STEPS4k] = \Pr[STEPS4k] \cdot \Pr[ERR \mid STEPS4k] \leq \delta.$$

Let the $4k$ samples the algorithm makes be $x_1, \ldots, x_{4k}$ and let $\mathbf{x} = (x_1, \ldots, x_{4k})$ be the corresponding point in $\mathbb{R}^{4k}$. Let $f_0(\mathbf{x})\colon \mathbb{R}^{4k} \to [0,1]$ be the probability that the algorithm outputs 0 on that input. Because the algorithm is symmetric, it is completely characterized by f_0. However, for notational convenience, let $f_\epsilon(\mathbf{x})$ be the probability that the algorithm outputs ϵ, so $f_\epsilon(\mathbf{x}) = f_0(\epsilon - \mathbf{x})$, where $\epsilon - \mathbf{x}$ is shorthand for $(\epsilon - x_1, \ldots, \epsilon - x_n)$. Let

$$PDF_{a,\sigma} = \frac{e^{-(x-a)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

be the probability density function of a single sample. Then the PDF of $\mathbf{x}$ is

$$P_a(\mathbf{x}) = \prod_{i=1}^{4k} PDF_{a,\sigma}(x_i) = \frac{e^{\frac{-\sum_{i=1}^{4k}(x_i-a)^2}{2\sigma^2}}}{(\sigma\sqrt{2\pi})^{4k}} = PDF_{a\sqrt{4k},\sigma}\left(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k}\right) \cdot \frac{e^{\frac{-\left\|\mathbf{x}-\frac{\mathbf{x}(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k})}{\|\mathbf{x}\|}\right\|^2}{2\sigma^2}}}{(\sigma\sqrt{2\pi})^{4k-1}}$$

where a is the mean of the true distribution and $\mathbf{1}$ is the vector $(1, \ldots, 1)$ so $\mathbf{1}/\sqrt{4k}$ is a unit vector. In order for the algorithm to be correct, f_0 and f_ϵ must satisfy these conditions:

$$\int_{\mathbb{R}^{4k}} f_\epsilon(\mathbf{x}) P_0(\mathbf{x}) d\mathbf{x} \leq \delta \quad \text{and} \quad \int_{\mathbb{R}^{4k}} (f_\epsilon(\mathbf{x}) + f_0(\mathbf{x})) P_0(\mathbf{x}) d\mathbf{x} \geq 3/4.$$

Now let $A = \{\mathbf{x} \in \mathbb{R}^{4k} \mid \sum_{i=1}^{4k} x_i > 2k\epsilon\}$. Note that for all $\mathbf{x} \in A$, $P_0(\mathbf{x}) < P_0(\epsilon - \mathbf{x})$. Also, clearly if $\mathbf{x} \in A$, then $\epsilon - \mathbf{x} \notin A$. Now let $S = \{\mathbf{x} \in A \mid f_0(\mathbf{x}) > 0\}$. If S is non-empty, we can lower the probability of failure (i.e., reduce the first integral without changing the second) by letting

$$f_0'(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in S, \\ f_0(\mathbf{x}) + f_\epsilon(\mathbf{x}), & \text{if } \epsilon - \mathbf{x} \in S, \\ f_0(\mathbf{x}), & \text{otherwise.} \end{cases}$$

Therefore, we may assume that f_0 is 0 on A (in other words, the algorithm never outputs 0 when the mean of the samples is closer to ϵ).

Now we split A: let $B = \{\mathbf{x} \in \mathbb{R}^{4k} \mid \sum_{i=1}^{4k} x_i > 4k\epsilon\}$ and $C = \{\mathbf{x} \in \mathbb{R}^{4k} \mid 2k\epsilon < \sum_{i=1}^{4k} x_i \leq 4k\epsilon\}$. We want to show that we can transform the algorithm to one with $f_\epsilon = 1$ on B that is correct with probability $1-\delta$ and asks the oracle with probability at most $1/4+\delta$. Suppose that $\int_B (1-f_\epsilon(\mathbf{x}))P_\epsilon(\mathbf{x})d\mathbf{x} = 1/2 - \int_B f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x})d\mathbf{x} = c > 0$. We know that $\int_A f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x})d\mathbf{x} \geq 3/4 - \delta \geq 1/2$ if $\delta \leq 1/4$. Therefore, $\int_C f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x}) \geq c$. Let $D \subset C$ be such that $\int_D f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x}) = c$. Now we can transform the algorithm by letting $f'_\epsilon = 1$ on B and $f'_\epsilon = 0$ on D and $f'_\epsilon = f_\epsilon$ everywhere else. The integral $\int_A f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x})d\mathbf{x}$ is unchanged and the integral $\int_{\mathbb{R}^{4k}-A} f_0(\mathbf{x})P_\epsilon(\mathbf{x})d\mathbf{x}$ was at most δ to begin with, so the transformed algorithm outputs after $4k$ samples with probability at least $3/4 - \delta$. We now show that the transformed algorithm errs with probability no more than δ. The error probability of the new algorithm is $\int_A f'_\epsilon(\mathbf{x})P_0(\mathbf{x})d\mathbf{x}$. This is equal to

$$\int_A f_\epsilon(\mathbf{x})P_0(\mathbf{x})d\mathbf{x} + \int_B (1 - f_\epsilon(\mathbf{x}))P_0(\mathbf{x})d\mathbf{x} - \int_D f_\epsilon(\mathbf{x})P_0(\mathbf{x})d\mathbf{x}.$$

Thus, we need to show that $\int_B (1 - f_\epsilon(\mathbf{x}))P_0(\mathbf{x})d\mathbf{x} \leq \int_D f_\epsilon(\mathbf{x})P_0(\mathbf{x})d\mathbf{x}$. Now let $g_1(\mathbf{x}) = (1 - f_\epsilon(\mathbf{x}))P_\epsilon(\mathbf{x})$ and let $g_2(\mathbf{x}) = f_\epsilon(\mathbf{x})P_\epsilon(\mathbf{x})$. By construction, $\int_B g_1(\mathbf{x})d\mathbf{x} = \int_D g_2(\mathbf{x})d\mathbf{x}$. Notice that

$$P_0(\mathbf{x})/P_\epsilon(\mathbf{x}) = \frac{PDF_{0,\sigma}\left(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k}\right)}{PDF_{\epsilon\sqrt{4k},\sigma}\left(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k}\right)} = \frac{e^{-(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k})^2/(2\sigma^2)}}{e^{-(\mathbf{x}\cdot\mathbf{1}/\sqrt{4k}-\epsilon\sqrt{4k})^2/(2\sigma^2)}} = e^{\left(\frac{-\epsilon^2 4k - 2\epsilon(\mathbf{x}\cdot\mathbf{1})}{2\sigma^2}\right)}$$

and the rightmost expression is a decreasing function of $\mathbf{x}\cdot\mathbf{1}$. Because $\mathbf{x}\cdot\mathbf{1}$ is greater everywhere on B than on D, we have $\int_B g_1(\mathbf{x})P_0(\mathbf{x})/P_\epsilon(\mathbf{x})d\mathbf{x} \leq \int_D g_2(\mathbf{x})P_0(\mathbf{x})/P_\epsilon(\mathbf{x})d\mathbf{x}$, as necessary. It is easy to modify the transformed algorithm into one that uses the oracle with probability at most $1/4$, but has error probability 2δ. This algorithm has $f_\epsilon = 1$ on B. In other words, if the mean of the $4k$ samples is at least ϵ, it outputs ϵ. Now consider the probability that the mean of $4k$ samples from $N(0,\sigma)$ is at least ϵ (because in this case the algorithm makes a mistake). If $4k \leq G(\sigma,\epsilon,4\delta)$, this probability is at least 2δ, so the algorithm cannot work unless $k = \Omega(G(\sigma,\epsilon,\delta))$. $\square$

4.3 Noisy Algorithm

Because we can only obtain approximate function values, we need to show that we can reconstruct a function approximation. The following proposition shows that if the error in the function values is less than $\epsilon/2$ and the sample looseness is no more than ϵ, we can achieve an accuracy of ϵ:

Proposition 17 *Suppose that* $0 \leq x_1 \leq \cdots \leq x_n \leq 1$ *are points and we are given* n *approximate function values* $v_1, \ldots, v_n$ *at these points, such that* $(x_{i+1} - x_i) - |v_{i+1} - v_i| \leq \epsilon$. *If* f *is a Lipschitz function such that* $v_i \in [f(x_i) - \epsilon/2, f(x_i) + \epsilon/2]$, *then using* v_i*'s,* f *can be reconstructed to within* ϵ.

Proof: For every interval (x_i, x_{i+1}) we construct $\hat{f}$ such that $|\hat{f} - f| \leq \epsilon$ on that interval. Fix an i and suppose that $v_i \leq v_{i+1}$ (the other case is symmetric). We know that for $x \in [x_i, x_{i+1}]$,

$$f(x) \leq f(x_i) + (x - x_i) \leq v_i + \epsilon/2 + (x - x_i), \quad \text{and}$$
$$f(x) \geq f(x_{i+1}) - (x_{i+1} - x) \geq v_{i+1} - \epsilon/2 - (x_{i+1} - x).$$

We average the upper and lower bounds on f to get $\hat{f}$:

$$\hat{f}(x) = \frac{(v_i + \epsilon/2 + (x - x_i)) + (v_{i+1} - \epsilon/2 - (x_{i+1} - x))}{2}.$$

To show that $\hat{f}$ is indeed within ϵ of f, we show that the difference between the upper and lower bound is at most 2ϵ:

$$(v_i + \epsilon/2 + (x - x_i)) - (v_{i+1} - \epsilon/2 - (x_{i+1} - x)) = (x_{i+1} - x_i) - (v_{i+1} - v_i) + \epsilon \leq 2\epsilon.$$

This proves the proposition. We note that we have just shown that every Lipschitz function that is within $\epsilon/2$ of the v_i's is within ϵ of $\hat{f}$. This immediately implies that any two such Lipschitz functions are within 2ϵ of each other. So if an algorithm outputs any Lipschitz function within $\epsilon/2$ of the v_i's, it is guaranteed to be accurate to within 2ϵ. Hence, at a small cost to accuracy, the output can be made Lipschitz. □

Algorithm NLAA(NOISY-LIPSCHITZ-APPROX-ADAPTIVE)

L is a linked list of (PARAMETER,VALUE) pairs and S is a stack of pointers into objects in L.

MULTI-SAMPLE(x): Make $G(\sigma, \epsilon/4, \delta\epsilon^2)$ samples of f at x and return their mean.

1. Add $(0, \text{MULTI-SAMPLE}(0))$ and $(1, \text{MULTI-SAMPLE}(1))$ to L
2. Push a pointer to the first element of L onto S
3. Do until S is empty:
  4. $P_1 \leftarrow \text{TOP}[S]$
  5. $P_2 \leftarrow \text{NEXT}[L, P_1]$
  6. $\text{LOOSENESS} \leftarrow \text{PARAMETER}[P_2] - \text{PARAMETER}[P_1] - |\text{VALUE}[P_1] - \text{VALUE}[P_2]|$
  7. If $\text{LOOSENESS} \leq \epsilon$ then $\text{POP}(S)$ and continue to Step 3
  8. $x \leftarrow (\text{PARAMETER}[P_1] + \text{PARAMETER}[P_2])/2$
  9. Insert $(x, \text{MULTI-SAMPLE}(x))$ into L after P_1 and push a pointer to it onto S
10. Output the interpolation of the values stored in L as described in Proposition 17.

Figure 4-1: NLAA

Proposition 17 immediately gives us a noisy version of the trivial algorithm: Let the x_i's be spaced at intervals of ϵ throughout $[0, 1]$ and sample the function $G(\sigma, \epsilon/2, \delta\epsilon)$ times at each x_i. Then set v_i to be the average of the samples at x_i and use Proposition 17 to construct $\hat{f}$. By Proposition 15, the probability of v_i being off by more than $\epsilon/2$ is at most $\delta\epsilon$. Because there are ϵ^{-1} v_i's, by the union bound, the probability of one of them being off by more than $\epsilon/2$ is at most δ. Thus, by Proposition 17, the probability that the algorithm is off by more than ϵ is at most δ. Clearly the trivial algorithm uses $\Theta(\frac{\sigma^2 \log(1/\delta\epsilon)}{\epsilon^3})$ samples.

Figure 4-1 gives an algorithm that does better in cases with lower OPT. The (rather obvious) idea is to use an analogue of LIPSCHITZ-APPROX-ADAPTIVE. That is, sample $G(\sigma, \Theta(\epsilon), \delta\epsilon)$ times in the middle of an unsampled interval, whenever the condition of Proposition 17 is not satisfied.

In the execution, Step 8 is never reached if $x_{i+1} - x_i \leq \epsilon$, so MULTI-SAMPLE is called at most ϵ^{-1} times. This means that the probability of none of the calls "failing" is at least $1 - \delta\epsilon > 1 - \delta$. Conditioned on this, correctness of NLAA follows from Proposition 17, as the algorithm only stops when the conditions of that proposition are satisfied.

We now turn to analyzing NLAA's performance. It makes $\Theta(G(\sigma, \epsilon, \delta\epsilon^2)\epsilon^{-1})$ samples in the worst case (for instance, in correct executions when f is constant). We would like to analyze its performance relative to OPT, but that seems extremely difficult. Instead, we analyze its performance relative to NOPT and then in the next sections show that $\mathrm{OPT}(f, \sigma, \epsilon, \delta) = \Theta(G(\sigma, \epsilon, \delta)\mathrm{NOPT}(f, \epsilon))$.

Theorem 8 NLAA *performs* $O(G(\sigma, \epsilon, \delta\epsilon)\cdot\mathrm{NOPT}(f, \epsilon)\log(\epsilon^{-1}/\mathrm{NOPT}(f, \epsilon)))$ *samples on the problem instance* $(f, \sigma, \epsilon, \delta)$.

Proof: To analyze the algorithm's performance relative to NOPT, we compare its execution to that of LIPSCHITZ-APPROX-ADAPTIVE on $(f, \epsilon/4)$, which, by Theorem 1, performs $O(\mathrm{NOPT}(f, \epsilon/4)\log(4\epsilon^{-1}/\mathrm{NOPT}(f, \epsilon/4)))$ samples.

First, suppose that no call to MULTI-SAMPLE fails—that is, each call returns a value within $\epsilon/4$ of $f(x)$. We show that NLAA makes at most as many calls to MULTI-SAMPLE as the noiseless algorithm, with its error tolerance set to $\epsilon/4$, makes samples. Because both NLAA and LIPSCHITZ-APPROX-ADAPTIVE always sample in the middle of an unsampled interval whose length is greater than $\Theta(\epsilon)$, an execution of each algorithm corresponds to a subtree of a complete binary tree of $\Theta(\epsilon^{-1})$ nodes. The root of this tree is the entire interval $[0, 1]$, the next level consists of the intervals $[0, 1/2]$ and $[1/2, 1]$, and so on. A node is in the subtree if the corresponding interval appeared between points in L during the execution of the algorithm. Therefore, the number of calls to MULTI-SAMPLE or samples (in the noisy and noiseless setting, respectively) is equal to the size of the subtree.

We show that the subtree corresponding to the noisy algorithm (in the case of no failures) is no bigger than the subtree corresponding to the noiseless algorithm. To obtain the noiseless algorithm, Step 7 is replaced by "If $L(x_i, x_{i+1}) \leq \epsilon/2$, continue" (see Step 6 in LIPSCHITZ-APPROX-ADAPTIVE). But because $|\hat{f}(x) - f(x)| \leq \epsilon/4$, we have $|\hat{f}(x_i) - \hat{f}(x_{i+1})| \geq |f(x_i) - f(x_{i+1})| - \epsilon/2$, so

$$(x_{i+1} - x_i) - |\hat{f}(x_{i+1}) - \hat{f}(x_i)| \leq (x_{i+1} - x_i) - |f(x_{i+1}) - f(x_i)| + \epsilon/2 = L(x_i, x_{i+1}) + \epsilon/2.$$

Therefore, if the children of a node are not included in the noiseless subtree, they are also not included in the noisy subtree. So NLAA performs $O(\text{NOPT}(f, \epsilon/4) \cdot \log(4\epsilon^{-1}/\text{NOPT}(f, \epsilon/4)))$ calls to MULTI-SAMPLE. By two applications of Proposition 14, this is the claimed bound.

The probability of MULTI-SAMPLE failing at least once during an execution is at most $\delta\epsilon$, and the NLAA makes fewer than ϵ^{-1} calls to MULTI-SAMPLE, even in case of failure, so the expected number of extra calls to MULTI-SAMPLE due to failure is at most δ, which is smaller than 1. □

4.4 Bounds on OPT in Terms of NOPT

For the purposes of comparing algorithms to OPT once their performance is expressed in terms of NOPT, it is important to have a lower bound on OPT in terms of NOPT. We show the following such bound:

Theorem 9 $\text{OPT}(f, \sigma, \epsilon, \delta) = \Omega(G(\sigma, \epsilon, \delta) \cdot \text{NOPT}(f, \epsilon))$

Proof: We show that every algorithm for the noisy problem is forced to solve at least $\text{NOPT}(f, 2\epsilon)/2$ independent instances of the distinguishing-between-two-Gaussians problem, which gives the desired bound.

First of all, let $n = \text{NOPT}(f, 2\epsilon)$ and let x_i for $1 \leq i \leq n$ be a smallest proof set for the noiseless problem instance. So $L(x_i, x_{i+1}) \leq 4\epsilon < L(x_i, x_{i+2})$. As we are proving a lower bound on the noisy algorithm, we are allowed to give the noisy algorithm free extra information to simplify the argument. So suppose we give the noisy algorithm the exact function values at all x_i's where i is odd. With this information, if i is odd, sampling in an interval $[x_i, x_{i+2}]$ gives the algorithm no information about the behavior of f in any other interval (because both $f(x_i)$ and $f(x_{i+2})$ are already known exactly).

Now consider a particular odd i and the interval $[x_i, x_{i+2}]$. Because $L(x_i, x_{i+2}) > 4\epsilon$, by Proposition 5(2), there is a point $x \in [x_i, x_{i+2}]$ and a Lipschitz function f' equal to f at x_i and x_{i+2} such that $|f'(x) - f(x)| > 2\epsilon$. By choosing f' correctly,

we can also guarantee that x is the point where $|f' - f|$ is maximized and that $|f'(x) - f(x)| < 3\epsilon$. Extend f' to $[0,1]$ by setting $f' = f$ everywhere except on $[x_i, x_{i+2}]$. Now any function $\hat{f}$ that approximates f within ϵ cannot approximate f' within ϵ, at least at x. So the best possible algorithm has to at least distinguish f from f' (which only differ on $[x_i, x_{i+2}]$). Because x is the point where f and f' differ most, by a simple reduction, distinguishing f from f' is at least as hard as distinguishing $f(x)$ from $f'(x)$, which differ by no more than 3ϵ. Thus, on every interval, the best possible algorithm must use $\Omega(G(\sigma, 3\epsilon, \delta))$ samples in expectation to distinguish $f(x)$ from $f'(x)$, by Proposition 16. Because there are $n/2$ intervals, the total number of samples necessary is $\Omega(n \cdot G(\sigma, \epsilon, \delta))$. □

Together, Theorems 8 and 9 show that the number of samples NLAA performs is at most

$$O\left(\mathrm{OPT}(f, \sigma, \epsilon, \delta) \cdot \frac{(\log(1/\delta) + \log(1/\epsilon))\log(1/\epsilon)}{\log(1/\delta)}\right) = O(\mathrm{OPT}(f, \sigma, \epsilon, \delta)\log^2(1/\epsilon)).$$

We now turn to proving an upper bound on OPT in terms of NOPT in order to show the tightness of Theorem 9. It is easy to prove the upper bound

$$\mathrm{OPT}(f, \sigma, \epsilon, \delta) = O\left(G\left(\sigma, \epsilon, \frac{\delta}{\mathrm{NOPT}(f, \epsilon)}\right)\mathrm{NOPT}(f, \epsilon)\right).$$

Tightening this to $\mathrm{OPT}(f, \sigma, \epsilon, \delta) = O(G(\sigma, \epsilon, \delta) \cdot \mathrm{NOPT}(f, \epsilon))$ is a bit tricky. The idea is to take advantage of the fact that the best possible algorithm is optimized for a particular instance and only sample with higher accuracy in regions where the input function doesn't look like the one for which the algorithm is optimized.

First, we prove a proposition related to Proposition 17:

Proposition 18 *Let $0 \le x_1 \le \cdots \le x_n \le 1$ be a proof set for $(f, \epsilon/4)$. Suppose f' is a Lipschitz function such that $|f'(x_i) - f(x_i)| \le \epsilon/2$ for all i. Then for all x, $|f'(x) - f(x)| \le \epsilon$.*

Proof: If $x < x_1$, then because $x_1 \le \epsilon/4$, we have $f(x) \in [f(x_1) - \epsilon/4, f(x_1) + \epsilon/4]$, and $f'(x) \in [f'(x_1) - \epsilon/4, f'(x_1) + \epsilon/4] \subset [f(x_1) - 3\epsilon/4, f(x_1) + 3\epsilon/4]$. The maximum

possible difference between points in these two intervals is ϵ. Similarly for $x > x_n$.

Otherwise, for some i, $x \in [x_i, x_{i+1}]$. Suppose that $f(x_i) \leq f(x_{i+1})$ (the other case is symmetric). Then by the Lipschitz condition, $f(x) \in [f(x_{i+1}) - (x_{i+1} - x), f(x_i) + (x - x_i)]$ and $f'(x) \in [f'(x_{i+1}) - (x_{i+1} - x), f'(x_i) + (x - x_i)] \subset [f(x_{i+1}) - (x_{i+1} - x) - \epsilon/2, f(x_i) + (x - x_i) + \epsilon/2]$. But $L_f(x_i, x_{i+1}) \leq \epsilon/2$, so $f(x)$ is contained in an interval of the form $[a - \epsilon/4, a + \epsilon/4]$ and $f'(x)$ is contained in an interval of the form $[a - 3\epsilon/4, a + 3\epsilon/4]$ (the a's are the same), which proves that the maximum possible difference between $f(x)$ and $f'(x)$ is ϵ. □

Theorem 10 $\mathrm{OPT}(f, \sigma, \epsilon, \delta) = O(G(\sigma, \epsilon, \delta) \cdot \mathrm{NOPT}(f, \epsilon))$

Proof: Let x_i's be a smallest noiseless proof set of size n for $(f, \epsilon/4)$. Then $L_f(x_i, x_{i+1}) \leq \epsilon/2$ and $n = O(\mathrm{NOPT}(f, \epsilon))$ by Proposition 14. Let f be the function the algorithm is optimized for and let f' be the function fed to it. The algorithm is as follows:

1. Make $G(\sigma, \epsilon/4, \delta/4)$ samples of f' at each x_i and let v_i's be the sample means.
2. Starting with $k = 3$, do while $2^{-k} > \epsilon^3/4$ (and increment k on each loop):
  3. Mark all i's for which $|v_i - f(x_i)| > \epsilon/4$. If none marked, output f.
  4. Make $G(\sigma, \epsilon/4, \delta/2^k)$ samples of f' at each marked x_i and let v_i's be the means.
5. Output the result of running the "trivial" algorithm passing it $\delta/2$ instead of δ.

To prove correctness, we first consider the case $f' = f$. Note that the loop completes only if there is a v_i marked in each round. The probability of a particular v_i being marked in all rounds (once a v_i is unmarked, it stays unmarked) is at most the probability of it being marked in the last round, which is at most $\delta\epsilon^3$. Thus, the probability of any v_i staying marked is at most $\delta\epsilon^2$ so with probability at least $1 - \delta\epsilon^2 > 1 - \delta$, f is correctly output.

If $f' \neq f$, we only care about the case when f is not within ϵ of f' (otherwise, it is fine if f is output). In this case, by the contrapositive of Proposition 18, for some x_i, $|f'(x_i) - f(x_i)| \geq \epsilon/2$. This means that the probability of that x_i getting unmarked in round k is at most $\delta/2^k$. By the union bound, the probability of it getting unmarked at all is at most $\delta/2$. Thus, with probability at least $1 - \delta/2$ the algorithm reaches

Step 5 (and conditioned on that, the output is correct with probability $1-\delta/2$). This proves that the algorithm is correct with probability at least $1-\delta$, as necessary.

We now analyze expected performance for $f' = f$. The expected number of samples made at x_i in Steps 1–4 is at most (roughly)

$$G(\sigma, \epsilon/4, \delta/4) + \frac{\delta}{4} G(\sigma, \epsilon/4, \delta/8) + \frac{\delta}{8} G(\sigma, \epsilon/4, \delta/16) + \cdots < 2G(\sigma, \epsilon/4, \delta/4).$$

Thus the total expected number of samples in Steps 1–4 is $O(nG(\sigma, \epsilon, \delta))$, as claimed. Step 5 is reached with probability less than $\delta\epsilon^2$ and makes $\epsilon^{-1}G(\sigma, \epsilon, \delta\epsilon)$ samples. Thus, the expected number of samples in Step 5 is $O(\frac{\delta(\log \delta^{-1} + \log \epsilon^{-1})}{\epsilon})$, which is small compared to $G(\sigma, \epsilon, \delta)$. □

Chapter 5

Functions Whose Domain is in $\mathbb{R}^d$

5.1 Higher Dimensional Domains

When $d > 1$, devising and analyzing adaptive algorithms becomes much more difficult. There are several reasons for this difficulty:

(1) If the domain is $[0, 1]$ and an algorithm has sampled $0 = x_1 < x_2 < \cdots < x_n = 1$, and $x \in [x_i, x_{i+1}]$, the Lipschitz condition for x_i and x is a stronger constraint on $f(x)$ than the condition for x_j and x if $i \geq j$. No obvious analogue of this statement is true when the domain is higher-dimensional. The implication is that OPT may use a single well-placed sample to tighten bounds on the function very far away, but finding this well-placed sample will be difficult for the algorithm.

(2) When $d = 1$, every connected domain is an interval. This is not true for $d > 1$. Assuming that the algorithm knows the shape of the domain a priori is reasonable for lower bounds, but not for algorithms. Making the shape of the domain part of the input to the algorithm presents its own difficulties, such as representation, as well analyzing the amount of time the algorithm spends doing computations with the shape.

(3) When $d = 1$, after a finite number of samples, Lipschitz bounds are piecewise-linear with few pieces (see, e.g., Figure 2-2). However, when $d > 1$, the region determined by the Lipschitz condition is an intersection of cones and has a combinatorial structure analogous to that of a Voronoi diagram. Computations that algorithms

may need to perform on this structure may have very high cost, even compared to the sampling cost.

The second and third reasons point to potential inadequacies of computation models, but the first is fundamental to the problem.

5.2 Lower Bounds for Function Minimization

We present a lower bound that shows that when the domain is a closed unit ball $\mathbb{B}^d$, good adaptive algorithms do not exist for the function minimization problem.

Problem LIPSCHITZ-BALL-MINIMIZE:

Given:	(f, ϵ) where $f: \mathbb{B}^d \to \mathbb{R}$, $0 < \epsilon < 1/2$
Such that:	For $x_1, x_2 \in \mathbb{B}^d$, $\lvert f(x_2) - f(x_1)\rvert \leq \lVert x_2 - x_1\rVert$
Compute:	x such that for all $y \in \mathbb{B}^d$, $f(x) \leq f(y) + \epsilon$

Theorem 11 *For any ϵ with $1/2 > \epsilon > 0$ and for any deterministic algorithm, there exists function g such that $\mathrm{OPT}(g, \epsilon) = 2$, but the algorithm makes $\Omega(\epsilon^{1-d})$ samples.*

Proof: Consider the "hat" function $f(x) = \max(1 - \lVert x\rVert, 2\epsilon)$. Let $B_1, B_2, \ldots, B_n$ be a tight packing of balls of radius $3\epsilon/2$ with centers on $\mathbb{S}^d$, the boundary of $\mathbb{B}^d$. The number of these balls is $\Theta(\epsilon^{1-d})$, where the constant in Θ may depend on d. Let $c_1, \ldots, c_n$ be the centers of the balls. Let $g_i: \mathbb{B}^d \to \mathbb{R}$ be defined as

$$g_i(x) = \begin{cases} \epsilon/2 + \lVert x - c_i\rVert, & \text{for } x \in B_i, \\ f(x), & \text{otherwise.} \end{cases}$$

See Figure 5-1. Because B_i is contained in the region where $f = 2\epsilon$, all g_i's are Lipschitz. The only solutions to the minimization problem for (g_i, ϵ) are on B_i, because $g_i(c_i) = \epsilon/2$, but everywhere not on B_i, g_i is at least 2ϵ. For each g_i, sampling at 0 and c_i is sufficient to prove that c_i is a solution because by the Lipschitz condition, $\lvert g_i(x) - 1\rvert < \lVert x\rVert$, which implies that $g_i(x) > 0$. Therefore, $\mathrm{OPT}(g_i, \epsilon) = 2$.

Now sampling g_i anywhere except at B_i gives no information about which of the balls contains the minimum (beyond that it is not the ball, if any, that contains the

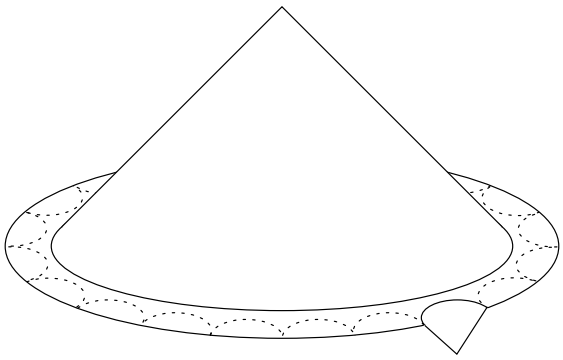

Figure 5-1: Illustration of g_i for Theorem 11 with $d = 2$. Other B_j's are shown dotted.

sample). Therefore, an algorithm is forced to do a linear search over the balls, which gives the desired result. □

This lower bound heavily relies on the fact that the domain is a ball. The generalization of this bound to other domain shapes is the following:

Proposition 19 *Let f have domain S. Let $(b_1, r_1), \ldots, (b_m, r_m)$ be balls centered at $b_i \in S$ with radii r_i such that for any point $x \in S$, there is an i for which $\|x - b_i\| \leq r_i + 2\epsilon$. Suppose n non-intersecting balls of radius $R > \epsilon$ with centers in S can be packed between the m balls. Then there is a family of n problem instances, such that* $\mathrm{OPT} \leq m + 1$ *for each member of the family, but any algorithm uses $\Omega(n)$ samples for some problem instance.*

Proof: Assume that none of the m balls is contained within another one (otherwise, it may be removed without violating the premise). Let $f(x) = \max(\max_i(2\epsilon + r_i - \|x - b_i\|), 2\epsilon)$. Clearly, f is Lipschitz because it is the maximum of Lipschitz functions. Note that $f(b_i) = 2\epsilon + r_i$ because none of the balls contain each other. We now show that sampling at the m b_i's proves that $f \geq 0$ everywhere on S. Given $x \in S$, pick an i such that $\|x - b_i\| \leq r_i + 2\epsilon$. By the Lipschitz condition, $f(x) \geq f(b_i) - r_i - 2\epsilon = 0$.

Let the n packed balls be $B_1, \ldots, B_n$ with centers at $c_1, \ldots, c_n$. Now, as in the proof of Theorem 11, let each member of the problem instance family be defined as

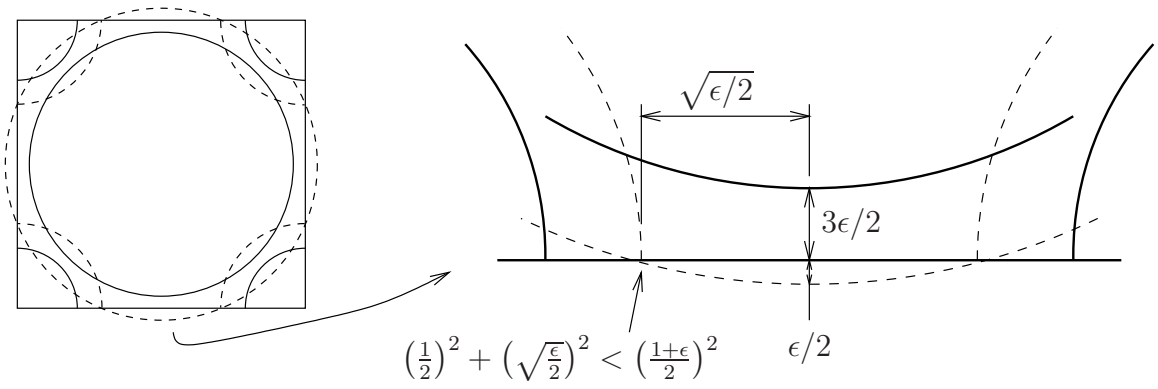

Figure 5-2: Construction for proving a lower bound on optimization over the square. Balls (b_i, r_i) are shown solid and $(b_i, r_i + 2\epsilon)$ are dashed.

$$g_i(x) = \begin{cases} 2\epsilon - R + \|x - c_i\|, & \text{for } x \in B_i, \\ f(x), & \text{otherwise.} \end{cases}$$

For g_j, OPT samples all of the b_i's and c_j. Any algorithm must find the ball that contains the "dip" out of the n possible balls. □

We can now apply Proposition 19 to prove a lower bound when S is a square. It is likely that similar methods can be used to show an $\Omega(\epsilon^{(1-d)/2})$ lower bound when S is a d-dimensional hypercube.

Theorem 12 *If $S = [0,1]^2$, for any ϵ with $1/4 > \epsilon > 0$, there is a family of problem instances with* $\mathrm{OPT} = O(1)$ *for each problem instance, such that any algorithm requires $\Omega(\sqrt{1/\epsilon})$ samples on some problem instance.*

Proof: Let $R = 5\epsilon/4$, and let $m = 5$. Consider 5 balls centered at the four corners of S and at the center of S. Let the ball in the center have radius $1/2 - 3\epsilon/2$ and let each ball in the corner have radius $1/2 - \sqrt{\epsilon/2} - 2\epsilon$. As shown in Figure 5-2, every point in the square is within 2ϵ of one of these balls. Along each edge, there is an interval of length $2\sqrt{\epsilon}$ around the middle such that every point on that interval is at distance at least $3\epsilon/2$ from the nearest ball. It is possible to pack $2\sqrt{\epsilon}/(5\epsilon/2) = \Omega(\sqrt{1/\epsilon})$ balls of radius R centered along that interval that do not overlap either any of the five balls or each other. The theorem follows by Proposition 19. □

Chapter 6

Conclusion

6.1 Results Summary

The main contribution of this thesis is deterministic optimally adaptive algorithms for some basic problems and progress in this direction for other problems. Table 6.1 summarizes the most important results.

Univariate black-box Lipschitz functions are proving to be very fertile ground for good adaptive algorithms. From a practical point of view, however, the applicability of the algorithms described in this thesis is limited due to the assumption that the Lipschitz constant must be known a priori, which is often not the case in practice. Nevertheless, we hope that the concepts and analysis techniques introduced in this

Table 6.1: Results summary—bounds proved on factors over OPT

	$\mathbb{R} \to \mathbb{R}$	$\mathbb{R} \to \mathbb{R}^d$	$\mathbb{R}^d \to \mathbb{R}$
Approximation	$\Omega(\log), O(\log)$	$\Omega(\log), O(\log)$	?
Noisy Approximation	$O(\log^2)$	?	?
Integration	$\Omega(\log), O(\log)$	–	?
Winding Number	–	$O(1)$	–
Optimization	$O(1)$ [HJL91]	$\Omega(\log), O(\log)$ [BD04]	$\mathbb{B}^d$: $\Omega(\epsilon^{1-d})$ $[0,1]^2$: $\Omega(\sqrt{1/\epsilon})$

thesis will prove useful in developing adaptive algorithms for more practical problems.

6.2 Future Research

There are many related problems left unexplored. We have shown that generalizing adaptive optimization algorithms to higher-dimensional domains is impossible, but perhaps integration or approximation do have good adaptive algorithms in higher dimensions. In particular, if Monte Carlo algorithms are allowed, integration does not suffer from the "curse of dimensionality," so the existence of good adaptive Monte Carlo algorithms would be especially interesting. Even for univariate functions, adaptive Monte Carlo integration algorithms are significantly more powerful than adaptive deterministic algorithms, as I plan to show in a forthcoming paper.

For the noisy approximation problem, there is a $\log^2$ gap between upper and lower bounds, so the "adaptive complexity" of noisy approximation is open. We have analyzed the extent to which the approximation problem becomes more difficult with random noise, but have not considered integration or optimization with noise. Also, we make very strong assumptions about the noise; are algorithms possible if we do not assume that the noise components in samples are i.i.d.?

We do not know what assumptions other than the Lipschitz condition lead to interesting adaptive algorithms. For example, we could consider functions whose n^{th} derivative is Lipschitz, functions that are sums or products of Lipschitz functions (e.g., we are given several Lipschitz functions and need to compute the minimum of their sum), or functions chosen at random according to some distribution.

Bibliography

[AS72] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, chapter 7. Dover, New York, 1972.

[BBD+04] Therese Biedl, Broňa Brejova, Erik D. Demaine, Angèle M. Hamel, Alejandro López-Ortiz, and Tomáš Vinař. Finding hidden independent sets in interval graphs. *Theoretical Computer Science*, 310(1–3):287–307, January 2004.

[BD04] Ilya Baran and Erik D. Demaine. Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, Brooklyn, NY, June 2004. To appear.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, McGraw-Hill Book Company, second edition, 2001.

[Dan71] Y.M. Danilin. Estimation of the efficiency of an absolute-minimum-finding algorithm. *USSR Computational Mathematics and Mathematical Physics*, 11:261–267, 1971.

[DLOM00] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752, San Francisco, California, January 2000.

[ECW92] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, December 1992.

[Elk04] Michael L. Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 352–361, New Orleans, LA, January 2004.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[GW90] Oliver Günther and Eugene Wong. The arc tree: an approximation scheme to represent arbitrary curved shapes. *Computer Vision, Graphics, and Image Processing*, 51:313–337, 1990.

[HJ95] Pierre Hansen and Brigitte Jaumard. Lipschitz optimization. In Reiner Horst and Panos M. Pardalos, editors, *Handbook of Global Optimization*, pages 407–494. Kluwer, 1995.

[HJL91] Pierre Hansen, Brigitte Jaumard, and Shi-Hui Lu. On the number of iterations of piyavskii's global optimization algorithm. *Mathematics of Operations Research*, 16(2):334–350, May 1991.

[Piy72] S.A. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12:57–67, 1972.

[TWW88] J.F. Traub, G.W. Wasilkowski, and H. Woźniakowski. *Information-Based Complexity*. Academic Press, New York, 1988.

[Wer02] Arthur G. Werschulz. An overview of information-based complexity. Technical Report CUCS-022-02, Computer Science Department, Columbia University, October 2002.

[ZKS03] Z. Zabinsky, B.P. Kristinsdottir, and R.L. Smith. Optimal estimation of univariate black-box lipschitz functions with upper and lower error bounds. *Computers and Operations Research*, 30(10):1539–1553, 2003.