The Complexity of Deterministic Automatic Algorithms

Yuhan Ding, Nicholas Clancy, Caleb Hamilton, Fred J. Hickernell, Yizhi Zhang

Room E1-208, Department of Applied Mathematics, Illinois Institute of Technology, 10 W. 32nd St., Chicago, IL 60616

Abstract

Keywords:

1. Introduction

Function recovery and integration are two fundamental examples of numerical problems that arise often in practice. It is desirable to have *automatic* algorithms for solving these problems, i.e., the algorithm should decide adaptively how many and which pieces of function data are needed and then use those data to construct an approximate solution. The error of this approximation should be guaranteed not to exceed the prescribed tolerance, ε, and the number of data required by the algorithm should be reasonable, given the assumptions made about the problem and the error tolerance required.

Most existing theory starts with a Banach space, $\mathcal{G}$, of input functions defined on some set $\mathcal{X}$, and having a semi-norm, $\|\cdot\|_{\mathcal{G}}$. The definition of $(\mathcal{G}, \|\cdot\|_{\mathcal{G}})$ contains assumptions about smoothness, periodicity or other qualities of the input functions. The mathematical problem of interest is defined by a solution operator $S : \mathcal{G} \to \mathcal{H}$, where $\mathcal{H}$ is some other Banach space with its norm $\|\cdot\|_{\mathcal{H}}$. For integration, $\mathcal{H} = \mathbb{R}$, and for function approximation, $\mathcal{H}$ is some superset of $\mathcal{G}$, for example, $\mathcal{L}_\infty(\mathcal{X})$. One then shows that there exists an algorithm, A, that provides an approximate solution differing from the true solution by no more than ε. Furthermore, the number of function data needed, $\mathrm{cost}(A)$, is bounded in terms of ε, typically as follows:

$$\sup_{\substack{f\in\mathcal{G}\\ \|f\|_{\mathcal{G}}\le\sigma}} \|S(f) - A(f)\|_{\mathcal{H}} \le \varepsilon, \qquad \mathrm{cost}(A) \le C_{\mathrm{up}} \left(\frac{\sigma}{\varepsilon}\right)^p. \tag{1}$$

Here it is necessarily assumed that the algorithm is exact if the semi-norm of the input function vanishes, i.e., $S(f) = A(f)$ if $\|f\|_{\mathcal{G}} = 0$. Algorithm A is said to be optimal if any algorithm satisfying the error criterion above must have a cost of at least $\mathcal{O}((\sigma/\varepsilon)^p)$.

A practical drawback of the analysis summarized above is that the definition of the algorithm, at least as far as its cost is concerned, depends a priori on an

upper bound on $\|f\|_{\mathcal{G}}$. Specifically, the theory is derived for functions in a *ball*, $\mathcal{B}_\sigma = \{f \in \mathcal{G} : \|f\|_{\mathcal{G}} \leq \sigma\}$, and the algorithm needs to know the radius, σ. Automatic algorithms try to estimate σ so that the number of function data needed can be determined adaptively. This is the approach taken here. Unfortunately, there is a lack of theory to rigorously justify the estimate of σ and the resulting adaptive algorithm. A general framework for doing so is developed here and some illustrative examples are provided. The key is to look at functions in a *cone* instead of a ball.

A simple example of the kind of practical problem addressed here is computing $\int_0^1 f(x)\,\mathrm{d}x$ for all integrands whose second derivatives are absolutely integrable, i.e., $\|f''\|_1$. The trapezoidal rule computes an approximation with error no greater than ε using $\mathcal{O}(\sqrt{\|f''\|_1/\varepsilon})$ function values. However, except for relatively simple integrands this norm of the second derivative, $\|f''\|_1$ is not available. After evaluating f, at a number of points in $[0,1]$, one may estimate the norm of the first derivative, $\|f'\|_1$, by the corresponding norm of piecewise linear spline, and the error can be bounded rigorously in terms of $\|f''\|_1$. If one assumes that $\|f''\|_1$ is not arbitrarily larger than $\|f'\|_1$ (a cone condition), then the reliable numerical estimate of $\|f'\|_1$ can be used as a surrogate for $\|f''\|_1$, and one can then reliable estimate the integral to the desired precision. This example is discussed in detail in Section 4.

2. General Problem Definition

2.1. Problems and Algorithms

The function approximation, integration, or other problem to be solved is defined by a *solution operator* $S : \mathcal{G} \to \mathcal{H}$, where $\mathcal{G}$ is a Banach space of possible input functions defined on $\mathcal{X}$ with semi-norm $\|\cdot\|_{\mathcal{G}}$, and $\mathcal{H}$ is some other Banach of possible outputs or solutions with norm $\|\cdot\|_{\mathcal{H}}$. It is assumed that the $\mathcal{G}_0 = \{f \in \mathcal{G} : \|f\|_{\mathcal{G}} = 0\}$, the subspace of $\mathcal{G}$ containing functions with vanishing $\mathcal{G}$-semi-norm, has finite dimension. The solution operator is assumed to have a scale property, i.e.,

$$S(cf) = cS(f) \qquad \forall c \geq 0.$$

Examples include the following:

$$\begin{aligned}
\text{Integration:} \quad & S(f) = \int_{\mathcal{X}} f(\boldsymbol{x})\,\rho(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \quad \rho \text{ is fixed} \\
\text{Function Recovery:} \quad & S(f) = f, \\
\text{Poisson's Equation:} \quad & S(f) = u, \quad \text{where} \quad \begin{array}{c} -\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}),\ \boldsymbol{x} \in \mathcal{X}, \\ u(\boldsymbol{x}) = 0\ \forall \boldsymbol{x} \in \partial\mathcal{X} \end{array} \\
\text{Optimization:} \quad & S(f) = \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}).
\end{aligned}$$

The first three examples above are linear problems, which automatically satisfy the scale property for S, but so does the last example, which is a nonlinear problem.

The goal is to find an algorithm $A : \mathcal{G} \to \mathcal{H}$ for which $S(f) \approx A(f)$. Following the definition of algorithms described in [1, Section 3.2], the algorithm takes the form of some function of data derived from the input function:

$$A(f) = \phi(\boldsymbol{L}(f)), \quad \boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f)) \qquad \forall f \in \mathcal{G}. \tag{2}$$

Here the $L_i \in \Lambda$ are real-valued functions defined on $\mathcal{G}$ with the following property:

$$L(cf) = cL(f) \qquad \forall f \in \mathcal{G},\ c \in \mathbb{R},\ L \in \Lambda. \tag{3}$$

One popular choice for Λ is the set of all function values, Λ^{std}, i.e., $L_i(f) = f(\boldsymbol{x}_i)$ for some $\boldsymbol{x}_i \in \mathcal{X}$. Another common choice is the set of all bounded linear functionals, Λ^{lin}. In general, m may depend on f and the choice of L_i may depend on $L_1(f), \ldots, L_{i-1}(f)$. In this article, all algorithms are assumed to be deterministic. There is no random element.

2.2. Non-Adaptive Algorithms

The set $\mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ contains algorithms as just described for which the choice of the L_i and the number of function data used by the algorithm, m, are both assumed to be independent of the input function, i.e., these algorithms are non-adaptive. Furthermore, any $A \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is assumed to satisfy the following scale properties

$$\boldsymbol{L}(cf) = c\boldsymbol{L}(f), \quad \phi(c\boldsymbol{y}) = c\phi(\boldsymbol{y}), \quad A(cf) = cA(f) \qquad \forall c \geq 0,\ \boldsymbol{y} \in \mathbb{R}^m. \tag{4}$$

The cost of a non-adaptive algorithm, $A \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, is fixed and is defined by

$$\text{cost}(A) = \$(\boldsymbol{L}) = \$(L_1) + \cdots + \$(L_m), \tag{5}$$

where $\$: \Lambda \to (0, \infty)$, and $\$(L)$ is the cost of acquiring the data $L(f)$. The cost of L may be the same for all $L \in \Lambda$. Alternatively, it might be a vary with the choice of L. E.g., if f is a function of the infinite sequence real numbers, $(x_1, x_2, \ldots)$, the cost of evaluating the function with arbitrary values of the first d coordinates, $L(f) = f(x_1, \ldots, x_d, 0, \ldots)$, might be d. See [**?**] for examples of this cost model.

The error of an algorithm $A \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is defined as

$$\text{err}(A, \mathcal{G}, \mathcal{H}, S) = \min\{\delta \geq 0 : \|S(f) - A(f)\|_{\mathcal{H}} \leq \delta \|f\|_{\mathcal{G}} \ \ \forall f \in \mathcal{G}\}, \tag{6}$$

When the problem has real-valued solutions, i.e., $\mathcal{H} = \mathbb{R}$, one may also define a one sided error criterion in the worst case:

$$\text{err}_{\pm}(A, \mathcal{G}, \mathbb{R}, S) = \min\{\delta \geq 0 : \pm[S(f) - A(f)] \leq \delta \|f\|_{\mathcal{G}} \ \ \forall f \in \mathcal{G}\} \tag{7}$$

Since $\|\cdot\|_{\mathcal{G}}$ may be a semi-norm, but not a norm, a finite error in any of the above definitions assumes that the algorithm is exact, i.e., $S(f) = A(f)$, for all f with $\|f\|_{\mathcal{G}} = 0$.

The above error criteria are normalized, meaning that the absolute error, $\|S(f) - A(f)\|_{\mathcal{H}}$ is measured with respect to the $\mathcal{G}$-semi-norm of the input function. The complexity of a problem for this set of algorithms, $\mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, is defined as the cost of the cheapest algorithm that satisfies the specified error tolerance, ε:

$$\begin{aligned}&\operatorname{comp}(\varepsilon, \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)) \\ &\qquad = \inf \left\{ \operatorname{cost}(A) : \operatorname{err}(A, \mathcal{G}, \mathcal{H}, S) \le \varepsilon,\ A \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda) \right\}. \end{aligned} \tag{8}$$

Here the infimum of an empty set is defined to be ∞. This means that to guarantee that $\|S(f) - A(f)\|_{\mathcal{H}} \le \varepsilon$, one needs an algorithm with a cost of at least

$$\operatorname{comp}(\varepsilon / \|f\|_{\mathcal{G}}, \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)).$$

This cost increases as either ε decreases or $\|f\|_{\mathcal{G}}$ increases.

Suppose that there is a sequence of nonadaptive algorithms indexed by their cost, and which converge to the true answer:

$$\{A_n\}_{n \in \mathcal{I}}, \qquad A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda), \tag{9a}$$

$$\lim_{\substack{n \to \infty \\ n \in \mathcal{I}}} \operatorname{err}(A_n, \mathcal{G}, \mathcal{H}, S) = 0, \qquad \operatorname{cost}(A_n) = n, \tag{9b}$$

where the countable, non-negative-valued index set,

$$\mathcal{I} = \{n_1, n_2, \ldots\} \quad \text{with } n_i < n_{i+1}, \quad \text{satisfies } \sup_i \frac{n_{i+1}}{n_i} < \infty. \tag{10}$$

This sequence of algorithms is called *nearly optimal* for the problem $(\mathcal{G}, \mathcal{H}, S, \Lambda)$ if it essentially tracks the minimum cost algorithms, namely,

$$\sup_{0 < \varepsilon \le 1} \frac{\min\{n \in \mathcal{I} : \operatorname{err}(A_n, \mathcal{G}, \mathcal{H}, S) \le \varepsilon\} \varepsilon^p}{\operatorname{comp}(\varepsilon, \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda))} < \infty, \qquad \forall p > 0. \tag{11}$$

The sequence is called optimal if above inequality holds fo $p = 0$. A nearly optimal sequence of algorithms may differ in its cost from an optimal algorithm by powers of $\log(\varepsilon^{-1})$, for example.

2.3. Automatic, Adaptive Algorithms

Non-adaptive algorithms, $A \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ need an upper bound on $\|f\|_{\mathcal{G}}$ to guarantee that they meet the prescribed error tolerance for the input function f. Automatic algorithms attempt to estimate $\|f\|_{\mathcal{G}}$ and then determine the number of function data needed to meet the error tolerance. Such automatic, adaptive algorithms are now defined, somewhat differently from the non-adaptive algorithms above. However, in practice automatic algorithms use non-adaptive algorithms as building blocks.

Practical automatic algorithms in $\mathcal{A}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ take the form of ordered pairs of functions

$$(A, W) : \mathcal{G} \times (0, \infty) \times (0, \infty] \to \mathcal{H} \times \{\text{false}, \text{true}\},$$

for which $S(f) \approx A(f;\varepsilon,N_{\max})$. Here $\varepsilon \in (0,\infty)$ is a user-supplied error tolerance, $N_{\max} \in (0,\infty]$ is a user-supplied maximum cost budget, and $W(f;\varepsilon,N_{\max})$ is a Boolean warning flag that is false if the algorithm completed its calculations without attempting to exceed the cost budget, and is true otherwise.

As in (2), the algorithm takes the form of some function of function data: $A(f;\varepsilon,N_{\max}) = \phi\left(\boldsymbol{L}(f);\varepsilon,N_{\max}\right)$. Now, however, the algorithm is allowed to be adaptive. The choice of L_2 may depend on the value of $L_1(f)$, the choice of L_3 may depend on $L_1(f)$ and $L_2(f)$, etc. The number of function data used by the algorithm, m, may also be determined adaptively. The choice of how many and which function data to use depends on ε and $N_{\max}$. The goal of the algorithm is to make $\|S(f) - A(f;\varepsilon,N_{\max})\|_{\mathcal{H}} \le \varepsilon$, but this is not a requirement of the definition.

The cost of the algorithm for a specified input function is defined analogously to (5):

$$\operatorname{cost}(A,f;\varepsilon,N_{\max}) = \$(\boldsymbol{L}) = \$(L_1) + \cdots + \$(L_m).$$

Because of the potentially adaptive nature of the algorithm, namely, that m may depend on f, it follows that the cost may depend on f as well as A. The input parameter $N_{\max}$ tells the algorithm to ensure that $\operatorname{cost}(A,f;\varepsilon,N_{\max}) \le N_{\max}$ for all f and ε. This is a practical consideration since the user does not want to wait indefinitely for an answer.

The cost of the algorithm is expected to scale with the $\mathcal{G}$-semi-norm of the integrand. This means that the cost of an algorithm generally increases as $\|f\|_{\mathcal{G}}$ increases. The cost of the algorithm may also depend on $\mathcal{N}$, the subset of $\mathcal{G}$ where the functions of interest lie. To represent this idea one defines

$$\operatorname{cost}(A,\mathcal{N},\varepsilon,N_{\max},\sigma) = \sup\{\operatorname{cost}(A,f;\varepsilon,N_{\max}) : f \in \mathcal{N},\ \|f\|_{\mathcal{G}} \le \sigma\}.$$

Here the set $\mathcal{N}$ is allowed to depend on the algorithm inputs, ε and $N_{\max}$, but not on the unknown σ.

For automatic algorithms, returning an approximation with the desired error is not enough. One also wants the algorithm to be confident that the answer is correct. A successful algorithm for $\mathcal{N} \subseteq \mathcal{G}$, denoted $(A,W) \in \mathcal{A}(\mathcal{N},\mathcal{H},S,\Lambda)$, is one that meets the prescribed error tolerance and does not raise the warning flag. Specifically, success is defined as

$$\operatorname{succ}(A,W,\mathcal{N},\varepsilon,N_{\max}) = \begin{cases} \text{true} & \text{if } \|S(f) - A(f;\varepsilon,N_{\max})\|_{\mathcal{H}} \le \varepsilon \ \&\ W(f;\varepsilon,N_{\max}) = \text{false} \quad \forall f \in \mathcal{N}, \\ \text{false} & \text{otherwise.} \end{cases}$$

The above are absolute error criteria for success. One might also define relative error criteria instead, but finding successful algorithms for relative error is a non-trivial exercise and will be considered in future work.

The complexity of a problem is defined as the cost of the cheapest successful

algorithm with $\mathcal{G}$-semi-norm no greater than σ:

$$\begin{aligned}\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S), \Lambda, N_{\max}, \sigma) \\ = \inf \{\operatorname{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) : \operatorname{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = \text{true}, \\ (A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)\}. \quad (12)\end{aligned}$$

Here the infimum of an empty set is defined to be ∞.

The set of non-adaptive algorithms, $\mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, defined in the previous subsection is a subset of the automatic algorithms $\mathcal{A}(\mathcal{G}, \mathcal{H}, S, \Lambda)$. Algorithms in $\mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ are not affected by the error tolerance ε and do not recognize a cost budget $N_{\max}$. Moreover, the warning flag for an algorithm in $\mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is always returned as false. Whereas the non-adaptive algorithms are inherently impractical by themselves, they are vital components of automatic, adaptive algorithms.

2.4. Cones of Functions

All algorithms can be fooled by some input functions, even if these functions are sufficiently smooth. Error analysis such as that outlined in (1) rules out fooling functions with large error by restricting the size of $\|f\|_{\mathcal{G}}$.

As mentioned above, it is often difficult to know how large $\|f\|_{\mathcal{G}}$ is a priori and so practical automatic algorithms try to estimate it. The framework described here rules out fooling functions whose $\mathcal{G}$-semi-norms cannot be estimated reliably. This is done by considering $\mathcal{F}$, a subspace of $\mathcal{G}$, with its own semi-norm $\|\cdot\|_{\mathcal{F}}$. The semi-norm $\|\cdot\|_{\mathcal{F}}$ is considered to be stronger than $\|\cdot\|_{\mathcal{G}}$ in the following sense:

$$\min_{f_0 \in \mathcal{F}_0} \|f - f_0\|_{\mathcal{G}} \leq C_{\mathcal{F}} \|f\|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}, \tag{13a}$$

where $\mathcal{F}_0 = \{f \in \mathcal{F} : \|f\|_{\mathcal{F}} = 0\}$ is a finite dimensional subspace of $\mathcal{F}$. Moreover, it is assumed that any $f \in \mathcal{G}$ with zero $\mathcal{G}$-semi-norm must also lie in $\mathcal{F}$ and have zero $\mathcal{F}$-semi-norm:

$$\mathcal{G}_0 \subseteq \mathcal{F}_0. \tag{13b}$$

Given $\tau > 0$, let $\mathcal{C}_\tau \subset \mathcal{F}$ denote a *cone* of functions whose $\mathcal{F}$-semi-norms are no greater than τ times their $\mathcal{G}$-semi-norms:

$$\mathcal{C}_\tau = \{f \in \mathcal{F} : \|f\|_{\mathcal{F}} \leq \tau \|f\|_{\mathcal{G}}\}. \tag{14}$$

For any $f \in \mathcal{C}_\tau$ both $\|f\|_{\mathcal{F}}$ and $\|f\|_{\mathcal{G}}$ must be finite, but they can be arbitrarily large. There is no need to assume an upper bound on their sizes, but it is possible to obtain reliable upper bounds for both $\|f\|_{\mathcal{F}}$ and $\|f\|_{\mathcal{G}}$ by sampling f. An upper bound on $\|f\|_{\mathcal{G}}$ for $f \in \mathcal{C}_\tau$ can be computed in terms of the data $\boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f))$ because $\|\cdot\|_{\mathcal{F}}$ is a stronger semi-norm than $\|\cdot\|_{\mathcal{G}}$ in the sense of (13a), and because $\|f\|_{\mathcal{F}}$ is no larger than a multiple of $\|f\|_{\mathcal{G}}$ (see Lemma 1 below). This upper bound on $\|f\|_{\mathcal{G}}$ then automatically implies an upper bound on $\|f\|_{\mathcal{F}}$ from the definition of the cone. These reliable bounds on both $\|f\|_{\mathcal{F}}$ and $\|f\|_{\mathcal{G}}$ may be used to obtain a bound on the error of the algorithm for estimating $S(f)$ (see Theorem 2 below).

2.5. Results that One Wishes to Prove

The previous subsections define the problem to be approximated and the notation describing the difficulty of the problem and the efficiency of the algorithms. This subsection summarizes the results that are proved in general in the next section and illustrated for specific cases in the following sections.

i. *Upper bound on the complexity.* One wishes to bound the complexity of solving the problem successfully, $\mathrm{comp}(\mathcal{N}, \varepsilon, N_{\max}, \sigma, \Lambda)$, in terms of some power of ε/σ for $\mathcal{N}$ suitably defined as a subset of the cone $\mathcal{C}_\tau$. This is done in Theorem 2.

ii. *An algorithm that achieves the upper bound.* Upper bounds on the complexity are sometimes found in a non-constructive way. However, it is desirable to identify an explicit successful algorithm, $(A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)$, that achieves these upper bounds. This is also done in Theorem 2.

iii. *Penalty for not knowing the $\mathcal{F}$- and $\mathcal{G}$-semi-norms of f.* The optimal successful algorithm must find an upper bound on $\|f\|_{\mathcal{F}}$ or $\|f\|_{\mathcal{G}}$ rather than assuming such an upper bound. One hopes that the extra cost relative to the situation of knowing a priori bounds on these semi-norms is not too great. A positive result is shown in Theorem 2.

iv. *Lower bound on the complexity.* The difficulty of the problem is provided by lower bounds on the complexity. These are given in Theorem 3.

3. General Theorems

This section provides rather general theorems about the complexity of automatic algorithms. In some sense, these theorems are roadmap or an outline because their assumptions are non-trivial and take effort to be be verified for specific problems of interest. On the other hand, the assumptions are reasonable as is demonstrated in the later sections where concrete cases are are discussed.

3.1. Bounding the $\mathcal{G}$-Semi-Norm

As mentioned in Section 2.4 automatic algorithms require a reliable upper bound on $\|f\|_{\mathcal{G}}$ for all f in the cone $\mathcal{C}_\tau$. This can be obtained using a non-adaptive algorithm $G \in \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$, provided that one has error bounds, $\mathrm{err}_{\pm}(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})$, as defined in (7). These upper and lower error bounds imply

$$-\mathrm{err}_-(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})\,\|f\|_{\mathcal{F}} \le \|f\|_{\mathcal{G}} - G(f) \le \mathrm{err}_+(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})\,\|f\|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}.$$

Noting that $\|f\|_{\mathcal{F}} \le \tau \|f\|_{\mathcal{G}}$ for all f in the cone $\mathcal{C}_\tau$ implies the lemma below.

Lemma 1. *Any nonadaptive algorithm $G \in \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$ yields an approximation to the $\mathcal{G}$-semi-norm of functions in the cone $\mathcal{C}_\tau$ with the following upper and lower error bounds:*

$$\frac{G(f)}{1 + \tau\,\mathrm{err}_-(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})} \le \|f\|_{\mathcal{G}} \le \frac{G(f)}{1 - \tau\,\mathrm{err}_+(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})} \qquad \forall f \in \mathcal{C}_\tau.$$

The upper bound assumes that

$$\mathrm{err}_+(G, \mathcal{F}, \mathbb{R}, \|\cdot\|_{\mathcal{G}}) < 1/\tau \tag{15}$$

3.2. Two-Stage, Automatic Algorithms

Computing an approximate solution to the problem $S : \mathcal{C}_\tau \to \mathcal{H}$, e.g., integration or function approximation, depends on non-adaptive algorithms. Suppose that there is a sequence of such of algorithms, $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, indexed by their cost as defined in (9), and for which upper error bounds are known for both the spaces $\mathcal{G}$ and $\mathcal{F}$:

$$\mathrm{err}(A_n, \mathcal{G}, \mathcal{H}, S) \le h(n), \qquad \mathrm{err}(A_n, \mathcal{F}, \mathcal{H}, S) \le \tilde{h}(n), \tag{16}$$

for some functions h and $\tilde{h}$. The definitions of these errors in (6) then implies upper bounds on the error of $A_n(f)$ in terms of the $\mathcal{G}$-semi-norm of f:

$$\begin{aligned} \|S(f) - A_n(f)\|_{\mathcal{H}} &\le \min(\mathrm{err}(A_n, \mathcal{G}, \mathcal{H}, S)\,\|f\|_{\mathcal{G}}\,, \mathrm{err}(A_n, \mathcal{F}, \mathcal{H}, S)\,\|f\|_{\mathcal{F}}) \\ &\le \min(h(n), \tau\tilde{h}(n))\,\|f\|_{\mathcal{G}} \qquad \forall f \in \mathcal{C}_\tau. \end{aligned} \tag{17}$$

Algorithm 1. (Automatic, Adaptive, Two-Stage). Let $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ be Banach spaces as described above, let S be the solution operator, let ε be a positive error tolerance, and let $N_{\max}$ be the maximum cost allowed. Let τ be a fixed positive number, and let $G \in \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}\,, \Lambda)$ be an algorithm as described in Lemma 1 and satisfying (15). Moreover, let $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, be a sequence of algorithms as described in (9) and (16). Given an input function f, do the following:

Stage 1. Estimate $\|f\|_{\mathcal{G}}$. First compute $G(f)$ at a cost of $N_G = \mathrm{cost}(G)$. Define the inflation factor

$$\mathfrak{C} = \frac{1}{1 - \tau\,\mathrm{err}_+(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})} \ge 1. \tag{18}$$

Then $\mathfrak{C}G(f)$ provides a reliable upper bound on $\|f\|_{\mathcal{G}}$.

Stage 2. Estimate $S(f)$. Choose the sample size need to approximate $S(f)$, namely, $N_A = N_{\min}(\varepsilon/(\mathfrak{C}G(f)))$, where

$$N_{\min}(a) = \min\left\{n \in \mathcal{I} : \min(h(n), \tau\tilde{h}(n)) \le a\right\}, \quad a \in (0, \infty). \tag{19}$$

If $N_A \le N_{\max} - N_G$, then $S(f)$ may be approximated within the desired error tolerance and within the cost budget. Set the warning flag to false, $W = 0$. Otherwise, recompute N_A to be within budget, $N_A = \widetilde{N}_{\max} := \max\{n \in \mathcal{I} : n \le N_{\max} - N_G\}$, and set the warning flag to true, $W = 1$. Compute $A_{N_A}(f)$ as the approximation to $S(f)$.

Return the result $(A_{N_A}(f), W)$, at a total cost of $N_G + N_A$.

Theorem 2. *Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, ε, $N_{\max}$, $\widetilde{N}_{\max}$, $\mathfrak{C}$, and τ be given as described in Algorithm 1, and assume that $\mathcal{F}$ satisfies* (13)*. Let*

$$\mathfrak{c} = 1 + \tau \operatorname{err}_-(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}) \geq 1. \tag{20}$$

Let $\mathcal{C}_\tau$ be the cone of functions defined in (14) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\mathcal{G}$-semi-norms. Let*

$$\begin{aligned}\mathcal{N} &= \left\{ f \in \mathcal{C}_\tau : N_{\min}\left(\frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\,\|f\|_{\mathcal{G}}}\right) \leq \widetilde{N}_{\max} \right\} \\ &= \left\{ f \in \mathcal{C}_\tau : \|f\|_{\mathcal{G}} \leq \frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\min(h(\widetilde{N}_{\max}), \tau\tilde{h}(\widetilde{N}_{\max}))} \right\}\end{aligned} \tag{21}$$

be a subset of the cone $\mathcal{C}_\tau$ that lies inside a $\mathcal{G}$-semi-norm ball of rather large radius (since $\min(h(\widetilde{N}_{\max}), \tau\tilde{h}(\widetilde{N}_{\max}))$ is assumed to be tiny). Then it follows that Algorithm 1 is successful for all functions in this set of nice *functions $\mathcal{N}$, i.e.,* $\operatorname{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = 1$*. Moreover, the cost of this algorithm is bounded above in terms of the $\mathcal{G}$-semi-norm of the input function as follows:*

$$\operatorname{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) \leq N_G + N_{\min}\left(\frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\sigma}\right). \tag{22}$$

The upper bound on the cost of this specific algorithm provides an upper bound on the complexity of the problem, $\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda), N_{\max}, \sigma)$*. If the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is nearly optimal for the problems $(\mathcal{G}, \mathcal{H}, S, \Lambda)$ and $(\mathcal{F}, \mathcal{H}, S, \Lambda)$ as defined in* (11)*, then Algorithm 1 does not incur a significant penalty for not knowing $\|f\|_{\mathcal{G}}$ a priori, i.e., for all $p > 0$,*

$$\sup_{0<\varepsilon/\sigma\leq 1} \frac{\operatorname{cost}(A, \mathcal{N}, \varepsilon, \infty, \sigma)}{\operatorname{comp}(\varepsilon/\sigma, \mathcal{A}_{\text{non}}(\mathcal{J}, \mathcal{H}, S, \Lambda))} \left(\frac{\varepsilon}{\sigma}\right)^p < \infty, \qquad \mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}. \tag{23}$$

Proof. The definition of $\mathfrak{C}$ in (18) implies that the true $\mathcal{G}$-semi-norm of f is bounded above by $\mathfrak{C}G(f)$ according to Lemma 1. The upper bound on the error of the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$ in (17) then implies that

$$\|S(f) - A_n(f)\|_{\mathcal{H}} \leq \min(h(n), \tau\tilde{h}(n))\mathfrak{C}G(f) \qquad \forall f \in \mathcal{C}_\tau.$$

This error upper bound may be made no greater than the error tolerance, ε, by choosing the algorithm cost, n, to satisfy the condition in Stage 2. of Algorithm 1, provided that this can be done within the maximum cost budget. In this case, the algorithm is successful, as claimed in the theorem.

To ensure that the algorithm does not attempt to overrun the cost budget, one must limit the $\mathcal{G}$-semi-norm of the input function. The definition of $\mathfrak{c}$ in (20) implies that $G(f) \leq \mathfrak{c}\,\|f\|_{\mathcal{G}}$ according to Lemma 1. This means that for any function, f, with actual $\mathcal{G}$-semi-norm $\sigma = \|f\|_{\mathcal{G}}$, the upper bound on its $\mathcal{G}$-semi-norm computed via Lemma 1 is no greater than $\mathfrak{C}\mathfrak{c}\sigma$. Thus, after using N_G samples to estimate $\|f\|_{\mathcal{G}}$, functions in $\mathcal{N}$ as defined in (21) never need

more than $N_{\max} - N_G$ additional samples to estimate $S(f)$ with the desired accuracy. This establishes that Algorithm 1 must be successful for all $f \in \mathcal{N}$. It furthermore establishes an upper bound on the cost of the algorithm as given in (22).

Now consider the penalty for not knowing $\|f\|_{\mathcal{G}}$ in advance. If the sequence of nonadaptive algorithms, $\{A_n\}_{n\in\mathcal{I}}$, used to construct Algorithm 1 are nearly optimal for solving the problem on both $\mathcal{F}$ and $\mathcal{G}$, as defined in (11), then it follows that for $\mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}$,

$$\begin{aligned}
&\sup_{0<\varepsilon/\sigma\leq 1} \frac{\operatorname{cost}(A, \mathcal{N}, \varepsilon, \infty, \sigma)}{\operatorname{comp}(\varepsilon/\sigma, \mathcal{A}_{\text{non}}(\mathcal{J}, \mathcal{H}, S, \Lambda))} \left(\frac{\varepsilon}{\sigma}\right)^p \\
&\qquad = \sup_{0<\varepsilon/\sigma\leq 1} \frac{\operatorname{cost}(A, \mathcal{N}, \varepsilon, \infty, \sigma)}{\min\{n \in \mathcal{I} : \operatorname{err}(A_n, \mathcal{J}, \mathcal{H}, S) \leq \varepsilon/\sigma\}} \left(\frac{\varepsilon}{\sigma}\right)^{p/2} \\
&\qquad\qquad \times \sup_{0<\varepsilon/\sigma\leq 1} \frac{\min\{n \in \mathcal{I} : \operatorname{err}(A_n, \mathcal{J}, \mathcal{H}, S) \leq \varepsilon/\sigma\}}{\operatorname{comp}(\varepsilon/\sigma, \mathcal{A}_{\text{non}}(\mathcal{J}, \mathcal{H}, S, \Lambda))} \left(\frac{\varepsilon}{\sigma}\right)^{p/2}
\end{aligned}$$

The first of these suprema is finite by comparing the convergence rates of the sequence algorithms, $\{A_n\}_{n\in\mathcal{I}}$, in (16) with the cost of the automatic algorithm given by (22). The second of these suprema is finite for all $p > 0$ by the near optimality of $\{A_n\}_{n\in\mathcal{I}}$. □

There are a several remarks that may facilitate understanding of this result.

Remark 1. There are three main conditions to be checked for this theorem to hold.

i. An algorithm, G, to approximate the semi-norm in the larger space, $\|\cdot\|_{\mathcal{G}}$, must be identified and its error must be bounded.
ii. Both the error functions h and $\tilde{h}$, for the sequence of nonadaptive algorithms, $\{A_n\}_{n\in\mathcal{I}}$, must be computed explicitly for the the automatic algorithm to be defined.
iii. The near optimality of this sequence of nonadaptive algorithms must be verified to ensure that there is no significant penalty for not having an a priori upper bound on $\|f\|_{\mathcal{G}}$.

Sections 4 and 5 provide concrete examples where these conditions are checked.

Remark 2. If $\tilde{h}$ is unknown, then one may take $\tilde{h} = \infty$, and the algorithm still satisfies the error tolerance with a cost upper bound given in (22). The optimality result in (23) then only holds for $\mathcal{G}$, and not $\mathcal{F}$. The analogy holds if h is unknown. However, at least one of these two functions h or $\tilde{h}$, must be known for this theorem to have a meaningful result.

Remark 3. The cost of Algorithm 1, as given by (22), depends on σ, which is essentially the $\mathcal{G}$-semi-norm of the input function, f. Thus, if σ is smaller, the cost will correspondingly be smaller. Moreover, σ is not an input parameter for the algorithm. Rather the algorithm reliably estimates $\|f\|_{\mathcal{G}}$, and then adjusts the number of samples used (the cost) accordingly.

Remark 4. The definition of the set of algorithms for which the Algorithm 1 is guaranteed to work, $\mathcal{N}$, depends somewhat on $\|f\|_{\mathcal{G}}$, but only because of the practical constraint of a cost budget of $N_{\max}$. This dependence disappears if one lifts this constraint by taking $N_{\max} \to \infty$. The primary constraint determining the success of the algorithm is that $f \in \mathcal{C}_\tau$.

Remark 5. Instead of choosing τ as an input parameter for Algorithm 1, one may alternatively choose the inflation factor $\mathfrak{C} > 1$. This then implies that

$$\tau = \left(1 - \frac{1}{\mathfrak{C}}\right) \frac{1}{\mathrm{err}_+(G, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})}, \tag{24}$$

which is equivalent to (18).

Remark 6. It is observed in the examples of Sections 4 and 5 that for the sequence of algorithms $\{A_n\}_{n \in \mathcal{I}}$

$$\tau \tilde{h}(n) \le h(n) \quad \forall n \in \mathcal{I}. \tag{25}$$

or equivalently, $\min(h(n), \tau \tilde{h}(n)) = \tau \tilde{h}(n)$. This then simplifies Algorithm 1 in the computation of the sample size for A in (19) and also in Theorem 2 in the definition of $\mathcal{N}$ in (21) and the upper bound on the cost in (22).

3.3. Embedded Nonadaptive Algorithms $\{A_n\}_{n \in \mathcal{I}}$

Suppose that the sequence of nonadaptive algorithms,

$$\{A_n\}_{n \in \mathcal{I}} = \{A_{n_1}, A_{n_2}, \ldots\},$$

are *embedded*, i.e., $A_{n_{i+1}}$ uses all of the data used by A_{n_i} for $i = 1, 2, \ldots$. An example would be a sequence of trapezoidal rules for integration that uses a power of two trapezoids for increasing powers. Furthermore, it is supposed that the data used by G, the algorithm used to estimate the $\mathcal{G}$-semi-norm of f, is the same data used by A_{n_1}, and so $n_1 = N_G$. Then the total cost of Algorithm 1 can be reduced; it is simply N_A, as given in Stage 2, instead of $N_G + N_A$. Moreover, $\widetilde{N}_{\max}$ may then be taken to be $N_{\max}$, and the cost bound of the automatic algorithm in (22) does not need the term N_G.

It is observed in the examples of Sections 4 and 5 that for $n_1 = N_G$, and G and τ satisfying condition (15), it follows that

$$\tau \tilde{h}(n) \le h(n) \quad \forall n \ge n_1. \tag{26}$$

or equivalently, $\min(h(n), \tau \tilde{h}(n)) = \tau \tilde{h}(n)$. This then simplifies Algorithm 1 in the computation of the sample size for A in (19) and also in Theorem 2 in the definition of $\mathcal{N}$ in (21) and the upper bound on the cost in (22).

Remark 7. Again suppose that $\{A_n\}_{n \in \mathcal{I}}$ is an embedded sequence of algorithms and that the sequence of nonadaptive algorithms and that the data used by the A_{n_i} can also b G, the algorithm used to estimate the $\mathcal{G}$-semi-norm of f, is the same data used by A_{n_1}

$$\{A_n\}_{n \in \mathcal{I}} = \{A_{n_1}, A_{n_2}, \ldots\},$$

are *embedded*, i.e., $A_{n_{i+1}}$ uses all of the data used by A_{n_i} for $i = 1, 2, \ldots$. Furthermore, suppose that the data used by G, the algorithm used to estimate the $\mathcal{G}$-semi-norm of f, is the same data used by A_{n_1}, and so $n_1 = N_G$. Then total cost of Algorithm 1 can be reduced; it is simply N, as given in Stage 2, instead of $N + N_G$. Moreover, $\widetilde{N}_{\max}$ may then be taken to be $N_{\max}$, and the cost bound of the automatic algorithm in (22) does not need the term N_G.

Remark 8. Assuming embedded algorithms, $\{A_n\}_{n\in\mathcal{I}}$, satisfying the conditions of the previous remark, and assuming that for every $n \in \mathcal{I}$, there exists an algorithm G_n that estimates $\|\cdot\|_{\mathcal{G}}$, one may modify the two-stage Algorithm 1 to become a multi-stage algorithm. Starting with $i = 1$,

Stage 1. Compute $G_{n_i}(f)$ as in Stage 1, and then $\mathfrak{C}_i$ as in (18).

Stage 2. Check whether n_i is large enough to satisfy the error tolerance, i.e.,

$$\frac{\min(C_1, C_2\tau n_i^{p_1-p_2})\mathfrak{C}_i G_{n_i}(f)}{n_i^{p_1}} \le \varepsilon.$$

If this is true, then set $W = 0$. Otherwise, if this inequality fails to hold and $n_{i+1} > N_{\max}$, then set the $W = 1$. In both cases, return $(A_{n_i}(f), W)$ and terminate the algorithm. If the error tolerance is not yet satisfied, and $n_{i+1} \le N_{\max}$, then increment i by one and return to Stage 1.

This multi-stage algorithm, where Stages 1 and 2 are repeated until the error tolerance is reached, has the advantage that the $G_{n_i}(f)$, the estimates of $\|f\|_{\mathcal{G}}$ should generally improve as i increases. This means that the final sample size may not need to be so large as in the two-stage algorithm.

3.4. Lower Complexity Bounds for the Algorithms

Lower complexity bounds are typically proved by constructing fooling functions. First, a lower bound is derived for the complexity of problems defined on $\mathcal{F}$- and $\mathcal{G}$-semi-norm balls of input functions. This technique is generally known [**?**]. Then it is shown how to extend this idea for the cone $\mathcal{C}_\tau$.

Consider the Banach spaces $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, and the *linear* solution operator $S : \mathcal{G} \to \mathcal{H}$. Let Λ be the set of bounded linear functionals that can be used as data. Suppose that for any $n > 0$, and for all $\boldsymbol{L} \in \Lambda^m$, satisfying $\$(\boldsymbol{L}) \le n$, there exists an $f_1 \in \mathcal{F}$, depending on n and the L_i, with solution norm one, zero data, and bounded $\mathcal{F}$ and $\mathcal{G}$ semi-norms, i.e.,

$$\|S(f_1)\|_{\mathcal{H}} = 1, \quad \boldsymbol{L}(f_1) = \boldsymbol{0}, \quad \|f_1\|_{\mathcal{G}} \le g(n), \quad \|f_1\|_{\mathcal{F}} \le \tilde{g}(n)\,\|f_1\|_{\mathcal{G}}\,, \tag{27}$$

for some positive, non-decreasing functions g and $\tilde{g}$ defined in $(0, \infty)$. For example, one might have $g(n) = an^p$ and $\tilde{g}(n) = bn^q$ with positive a, b, p, and q. Since the data for the function f_1 are all zero, it follows that $A(f_1) = A(-f_1)$ for any algorithm, A, adaptive or not, that is based on the information $\boldsymbol{L}(f_1)$. Then,

by the triangle inequality and (27) the error for one of the fooling functions $\pm f_1$ must be at least one:

$$\begin{aligned}
&\max(\|S(f_1) - A(f_1)\|_{\mathcal{H}}, \|S(-f_1) - A(-f_1)\|_{\mathcal{H}}) \\
&\quad = \max(\|S(f_1) - A(f_1)\|_{\mathcal{H}}, \|-S(f_1) - A(f_1)\|_{\mathcal{H}}) \\
&\quad \geq \frac{1}{2}\left[\|S(f_1) - A(f_1)\|_{\mathcal{H}} + \|S(f_1) + A(f_1)\|_{\mathcal{H}}\right] \\
&\quad \geq \frac{1}{2}\|[S(f_1) - A(f_1)] + [S(f_1) + A(f_1)]\|_{\mathcal{H}} = \|S(f_1)\|_{\mathcal{H}} = 1.
\end{aligned}$$

Furthermore, applying (27), for $\mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}$, it follows that any nonadaptive algorithm satisfying the error tolerance ε must have a cost n satisfying the following inequality:

$$\begin{aligned}
\varepsilon &\geq \sup_{f \neq 0} \frac{\|S(f) - A(f)\|_{\mathcal{H}}}{\|f\|_{\mathcal{J}}} \\
&\geq \frac{\max(\|S(f_1) - A(f_1)\|_{\mathcal{H}}, \|S(-f_1) - A(-f_1)\|_{\mathcal{H}})}{\|f_1\|_{\mathcal{J}}} \\
&\geq \frac{1}{\|f_1\|_{\mathcal{J}}} \geq \begin{cases} \dfrac{1}{g(n)}, & \mathcal{J} = \mathcal{G}, \\ \dfrac{1}{g(n)\tilde{g}(n)}, & \mathcal{J} = \mathcal{F}. \end{cases}
\end{aligned}$$

This implies lower bounds on the complexity of nonadaptive algorithms, as defined in (8):

$$\mathrm{comp}(\varepsilon, \mathcal{A}_{\mathrm{non}}(\mathcal{J}, \mathcal{H}, S, \Lambda)) \geq \begin{cases} g^{-1}(\varepsilon^{-1}), & \mathcal{J} = \mathcal{G}, \\ (g\tilde{g})^{-1}(\varepsilon^{-1}), & \mathcal{J} = \mathcal{F}, \end{cases}$$

where g^{-1} and $(g\tilde{g})^{-1}$ denote the inverse functions of g and $g\tilde{g}$, respectively. Thus, the cost of solving the problem within error tolerance ε for input functions in a $\mathcal{G}$-semi-norm ball of radius σ is at least $g^{-1}(\sigma\varepsilon^{-1})$ and for input functions in a $\mathcal{F}$-semi-norm ball of radius σ is at least $(g\tilde{g})^{-1}(\sigma\varepsilon^{-1})$

Turning to the problem of solving functions in the cone $\mathcal{C}_\tau$, the lower bound on the complexity becomes a bit more difficult to derive. Note that condition (27) allows the fooling function f_1 to lie *outside* this cone for $bn^q > \tau$. Thus, when considering the cone of input functions, the fooling function must be modified as described below.

It is assumed that there exists a function f_0 with non-zero $\mathcal{G}$-semi-norm lying in the interior of the cone $\mathcal{C}_\tau$, i.e.,

$$\|f_0\|_{\mathcal{G}} > 0, \qquad \|f_0\|_{\mathcal{F}} \leq \tau_0 \|f_0\|_{\mathcal{G}}, \qquad \tau_0 < \tau. \tag{28}$$

Furthermore, suppose that for each $n > 0$, and for all $\boldsymbol{L} \in \Lambda^m$, satisfying $\$(\boldsymbol{L}) \leq n$, there exists f_1 as described above in (27). Under these assumptions, one may show the following lower bound on the complexity of solving the problem S for functions in the cone $\mathcal{C}_\tau$.

Theorem 3. *Suppose that functions f_0 and f_1 can be found that satisfy conditions* (27) *and* (28)*. It then follows that the complexity of the problem, defined by* (12)*, assuming infinite cost budget, over the cone of functions $\mathcal{C}_\tau$ is*

$$\begin{aligned}&\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{H}, S, \Lambda), \infty, \sigma)\\ &\qquad\geq \min\left(g^{-1}\left(\frac{\sigma(\tau-\tau_0)}{2(2\tau-\tau_0)\varepsilon}\right), (g\tilde{g})^{-1}\left(\frac{\sigma\tau(\tau-\tau_0)}{2(2\tau-\tau_0)\varepsilon}\right)\right).\end{aligned}$$

Proof. Let A be a successful, possibly adaptive, algorithm for all functions lying in the cone $\mathcal{C}_\tau$. Given an error tolerance, ε, and a positive σ, let f_0 be a function satisfying (28) and choose

$$c_0 = \frac{\sigma\tau}{\|f_0\|_{\mathcal{G}}\,(2\tau-\tau_0)}. \tag{29a}$$

Provide the algorithm A with the input function $c_0 f_0$, and let $\boldsymbol{L}(c_0 f_0) = c_0 \boldsymbol{L}(f_0)$ be the data vector extracted by A to obtain the estimate $A(c_0 f_0)$. Let $n = \$(\boldsymbol{L})$ denote the cost of this algorithm for the function $c_0 f_0$, and define two fooling functions, $f_\pm = c_0 f_0 \pm c_1 f_1$, in terms of the f_0 and f_1 satisfying conditions (27) with c_1 satisfying

$$c_1 = \frac{(\tau-\tau_0)c_0\,\|f_0\|_{\mathcal{G}}}{\|f_1\|_{\mathcal{G}}\,[\tilde{g}(n)+\tau]} = \frac{\sigma\tau(\tau-\tau_0)}{\|f_1\|_{\mathcal{G}}\,(2\tau-\tau_0)[\tilde{g}(n)+\tau]}. \tag{29b}$$

These fooling functions must lie inside the cone $\mathcal{C}_\tau$ because

$$\begin{aligned}\|f_\pm\|_{\mathcal{F}} - \tau\,\|f_\pm\|_{\mathcal{G}} &\leq c_0\,\|f_0\|_{\mathcal{F}} + c_1\,\|f_1\|_{\mathcal{F}} - \tau(c_0\,\|f_0\|_{\mathcal{G}} - c_1\,\|f_1\|_{\mathcal{G}})\\ &\qquad\qquad\text{by the triangle inequality}\\ &\leq c_1[\tilde{g}(n)+\tau]\,\|f_1\|_{\mathcal{G}} - (\tau-\tau_0)c_0\,\|f_0\|_{\mathcal{G}} \qquad \text{by (27), (28)}\\ &= 0 \qquad\qquad \text{by (29).}\end{aligned}$$

Moreover, both fooling functions have $\mathcal{G}$-semi-norms no greater than σ, since

$$\begin{aligned}\|f_\pm\|_{\mathcal{G}} &\leq c_0\,\|f_0\|_{\mathcal{G}} + c_1\,\|f_1\|_{\mathcal{G}}\\ &= \frac{\sigma\tau}{2\tau-\tau_0}\left[1+\frac{\tau-\tau_0}{\tilde{g}(n)+\tau}\right] \qquad \text{by (29)}\\ &\leq \frac{\sigma\tau}{2\tau-\tau_0}\left[1+\frac{\tau-\tau_0}{\tau}\right] = \sigma.\end{aligned}$$

Following the argument earlier in this section, it is noted that the data used by algorithm A for both fooling functions is the same, i.e., $\boldsymbol{L}(f_\pm) = \boldsymbol{L}(c_0 f_0)$, and so $A(f_\pm) = A(c_0 f_0)$. Consequently, by the same argument used above,

$$\varepsilon \geq \max(\|S(f_+) - A(f_+)\|_{\mathcal{H}}, \|S(f_-) - A(f_-)\|_{\mathcal{H}}) \geq c_1\,\|S(f_1)\|_{\mathcal{H}} = c_1.$$

Since A is successful for these two fooling functions, c_1, as defined in (29), must be no larger than the error tolerance, which implies by (27) that

$$\begin{aligned}\frac{\sigma\tau(\tau-\tau_0)}{\varepsilon} \leq \frac{\sigma\tau(\tau-\tau_0)}{c_1} &= \|f_1\|_{\mathcal{G}}\,(2\tau-\tau_0)[\tilde{g}(n)+\tau] \\ &\leq (2\tau-\tau_0)g(n)[\tilde{g}(n)+\tau)] \\ &\leq 2(2\tau-\tau_0)g(n)\max(\tilde{g}(n),\tau).\end{aligned}$$

Since A is an arbitrary successful algorithm, this inequality provides a lower bound on the cost, n, that any such algorithm requires. This then implies the lower bound on the complexity of the problem. □

Remark 9. Now suppose that the sequence of nonadaptive algorithms used to construct the adaptive, automatic Algorithm 1, and the fooling functions in Theorem 3 have comparable powers, namely $p_1 = p$ and $p_2 = p + q$. It then follows by comparing the upper bound on the cost in Theorem 2 to the lower bound in Theorem 3 that Algorithm 1 is optimal.

4. Approximation of One-Dimensional Integrals

Yizhi needs to write this section.

5. $\mathcal{L}_\infty$ Approximation of Univariate Functions

Yuhan Ding needs to fix this section as soon as she can.

Let $\mathcal{H}$ be the space of continuous functions on $[0,1]$ with $\mathcal{L}_\infty$ first derivatives. Suppose one wants to approximate functions in $\mathcal{H}$ based on function values. The goal is to make the $\mathcal{L}_\infty$ error small, i.e., to find an algorithm, A for which $\|f - A(f)\|_\infty \leq \varepsilon$. A non-adaptive algorithm is described and analyzed first, however, as described in the introduction it is successful only for functions whose size is known. Next an adaptive algorithm is constructed from the non-adaptive one. A measure of guile is defined, and functions with moderate guile are guaranteed to be well-approximated by the non-adpative algorithm.

5.1. Non-Adaptive Algorithms

As a start, consider the algorithm that produces a piecwise constant approximation using function values at $n+1$ equally spaced data sites, $\left\{x_i = \frac{i}{n}\right\}_{i=0}^{n}$:

$$\tilde{A}_n(f) = 1_{\left[0,\frac{1}{2n}\right)}(x)\,f(0) + \sum_{i=1}^{n-1} 1_{\left[\frac{2i-1}{2n},\frac{2i+1}{2n}\right)}(x)\,f\left(x_i\right) + 1_{\left[\frac{2n-1}{2n},1\right]}(x)\,f(1). \quad (30)$$

This algorithm assigns as the value $\tilde{A}_n(f)(x)$ the value of $f\left(x_i\right)$ corresponding to the data site, x_i, nearest x. The algorithm $\tilde{A}_n$ is used to construct an adaptive algorithm to be described later. Note that the cost of this algorithm is $n+1$, not n. Although this algorithm is slightly sub-optimal for fixed sample size,

Define the size of a function as the $\mathcal{L}_\infty$ norm of its first derivative, i.e., $\text{size}(f) = \|f'\|_\infty$. Furthermore, as in the previous section, define the ball of functions $\mathcal{B}_\sigma := \{f \in \mathcal{H} : \|f'\|_\infty \leq \sigma\}$.

Theorem 4. *The problem of approximating functions of bounded size has complexity*

$$\text{comp}(\varepsilon, \mathcal{B}_\sigma, \Lambda^{\text{std}}) = \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil.$$

The algorithm defined in (30) *is nearly optimal, in the sense that it satisfies the error tolerance with only one additional function evaluation than is optimal:*

$$\sup_{f \in \mathcal{B}_\sigma} \left\| f - \tilde{A}_{\lceil \sigma/(2\varepsilon) \rceil}(f) \right\|_\infty \leq \varepsilon, \qquad \text{cost}(\tilde{A}_{\lceil \sigma/(2\varepsilon) \rceil}, \mathcal{B}_\sigma) = \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil + 1.$$

Proof. For any function f lying in the ball $\mathcal{B}_\sigma$, and any $x \in [0,1)$, let x_i be the data site nearest x, and so $|x - x_i| \leq 1/(2\lceil \sigma/(2\varepsilon) \rceil) \leq \varepsilon/\sigma$ and $\tilde{A}_{\lceil \sigma/(2\varepsilon) \rceil}(f)(x) = f(x_i)$. It then follows that

$$\left| f(x) - \tilde{A}_{\lceil \sigma/(2\varepsilon) \rceil}(f)(x) \right| = |f(x) - f(x_i)| = \left| \int_{x_i}^{x} f'(t)\, \mathrm{d}t \right| \leq \|f'\|_\infty |x - x_i| \leq \sigma \frac{\varepsilon}{\sigma} = \varepsilon.$$

This implies that $\tilde{A}_{\lceil \sigma/(2\varepsilon) \rceil}$ that attains the error tolerance with $\left\lceil \frac{\sigma}{2\varepsilon} \right\rceil + 1$ function evaluations. A similar argument using an algorithm based on function evaluations at $x_i = (2i-1)/n$, $i = 1, \ldots, n$ can be used to show that $\text{comp}(\varepsilon, \mathcal{B}_\sigma, \Lambda^{\text{std}}) \leq \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil$.

To prove a lower bound, one considers *any* (possibly adaptive) algorithm A using $n \leq \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil - 1$ function evaluations. The proof proceeds by constructing a fooling function, $f^* \in \mathcal{B}_\sigma$ for which $\|f^* - A(f^*)\|_\infty > \varepsilon$. Let A be any such algorithm. Since the algorithm may be adaptive, the choice of the point, ξ_i, where the fooling function, f^*, is evaluated, may depend on $(x_1, f^*(\xi_1)), \ldots, (\xi_{i-1}, f^*(\xi_{i-1}))$, for $i = 2, \ldots, n$. By fiat, the fooling function is constructed to vanish at all data sites, ξ_i. The fooling function is a translated and scaled bump. Define a standard triangular bump on $\mathbb{R}$ as

$$\psi(x) := \max(1 - |x|, 0) = \begin{cases} 0, & |x| > 1, \\ 1 - |x|, & |x| \leq 1, \end{cases} \tag{31}$$

and note that $\|\psi\|_\infty = \|\psi'\|_\infty = 1$. Given the algorithm A with data sites $\xi_1, \ldots, \xi_n$, there exists at least one point $\xi_0 \in [0,1)$ with $\min_{1 \leq i \leq n} |\xi_i - \xi_0| \geq 1/(2n)$. Then define the fooling function by

$$f^* : x \mapsto \frac{\sigma}{2n} \psi(2n(x - \xi_0)).$$

By definition, $\pm f^*(\xi_i) = 0$ for $i = 1, \ldots, n$, and $\left\| \pm f^{*\prime} \right\|_\infty = \sigma$, so $\pm f^* \in \mathcal{B}_\sigma$. Moreover, $\pm f^*(\xi_i) = \pm\sigma/(2n)$. Since the two functions $\pm f^*$ provide the same data to algorithm A, it follows that $A(f)(\xi_0) = A(-f)(\xi_0)$. The triangle

inequality and the strict upper bound on n then imply that the approximation error must be greater than the desired tolerance.

$$\begin{aligned}
\sup_{f\in\mathcal{B}} \|f - A(f)\|_\infty &\geq \max\left(\|f^* - A(f^*)\|_\infty, \|-f^* - A(-f^*)\|_\infty\right) \\
&\geq \max\left(|f^*(\xi_0) - A(f)(\xi_0)|, |-f^*(\xi_0) - A(-f^*)(\xi_0)|\right) \\
&\geq \frac{1}{2}\left[|f(\xi_0) - A(f^*)(\xi_0)| + |f^*(\xi_0) + A(-f^*)(\xi_0)|\right] \\
&\geq \frac{1}{2}\left|f^*(\xi_0) - A(f^*)(\xi_0) + f^*(\xi_0) + A(-f^*)(\xi_0)\right| \\
&= |f^*(\xi_0)| = \frac{\sigma}{2n} \geq \frac{\sigma}{2\left(\left\lceil \frac{\sigma}{2\varepsilon}\right\rceil - 1\right)} > \varepsilon.
\end{aligned}$$

Thus, $\mathrm{comp}(\varepsilon, \mathcal{B}_\sigma, \Lambda^{\mathrm{std}}) > \left\lceil \frac{\sigma}{2\varepsilon}\right\rceil - 1$, which completes the proof. □

5.2. Adaptive Algorithms

The complexity result in Theorem 4 requires knowledg of the size of the function to be approximated, i.e, $\mathrm{size}(f) = \|f'\|_\infty$. This is typically not known in advance, however, it can be approximated from the data. Given the data sites $0, 1/n, \ldots, 1$, and the corresponding function values, one may estimate $\mathrm{size}(f)$ in terms of divided differences:

$$\hat{\sigma}_n(f) = \max_{i=1,\ldots,n} n\left|f(i/n) - f((i-1)/n)\right|. \tag{32}$$

Note that $\hat{\sigma}_n(f)$ never overestimates $\mathrm{size}(f)$. The quality of this estimate depends on how much f' changes over a short distance. This motivates the following definition of guile:

$$\mathrm{guile}(f) := \frac{\|f''\|_\infty}{\|f'\|_\infty} = \frac{\|f''\|_\infty}{\mathrm{size}(f)}, \tag{33}$$

with the convention that constant functions have zero guile. The guile of a function is the size of its second derivative relative to the size of its first derivative. This guile remains unchanged when the function is multiplied by any nonzero constant.

A function with small guile has a size that is well approximated by $\hat{\sigma}_n$. Suppose that ζ is some point satisfying $|f'(\zeta)| = \|f'\|_\infty$ and that $(i-1)/n \leq \zeta < i/n$. It then follows that $f(i/n) - f((i-1)/n) = f'(\eta)/n$ for some η between $(i-1)/n$ and i/n. Thus,

$$\begin{aligned}
\|f'\|_\infty = |f'(\zeta)| &= \left|f'(\eta) + \int_\eta^\zeta f''(x)\,\mathrm{d}x\right| \leq |f'(\eta)| + \left|\int_\eta^\zeta f''(x)\,\mathrm{d}x\right| \\
&\leq \hat{\sigma}_n(f) + |\zeta - \eta|\,\|f''\|_\infty \leq \hat{\sigma}_n(f) + \frac{\mathrm{guile}(f)\,\|f'\|_\infty}{n}, \\
\|f'\|_\infty &\leq \frac{\hat{\sigma}_n(f)}{1 - \mathrm{guile}(f)/n}.
\end{aligned}$$

Letting $\mathcal{N}_\tau = \{f \in \mathcal{H} : \text{guile}(f) \le \tau\}$ now carves out a subset of $\mathcal{H}$ which includes functions of arbitrarily size, but for which that size can be easily bounded empirically as

$$\text{size}(f) \le \frac{\hat{\sigma}_n(f)}{1 - \tau/n} \qquad \forall f \in \mathcal{N}_\tau. \tag{34}$$

This then suggests the following adaptive algorithm that is guaranteed to approximate functions in $\mathcal{N}_\tau$ within a tolerance of ε.

Algorithm 2. Given positive numbers ε and τ, let $n_1 = \lceil \tau \rceil + 1$, and $j = 1$.

Step 1. Let $n = n_1 2^{j-1}$. Evaluate $\hat{\sigma}_n(f)$.

Step 2. If

$$\frac{\hat{\sigma}_n(f)}{2(n - \tau)} \le \varepsilon,$$

then output $A_\varepsilon(f) = \tilde{A}_n(f)$ as the approximation. Else, let $j = j + 1$ and to to Step 1.

Theorem 5. Nice *functions may be well approximated at a reasonable cost by A_ε, the adaptive Algorithm 2. In particular,*

$$\sup_{f \in \mathcal{N}_\tau} \|f - A_\varepsilon(f)\|_\infty \le \varepsilon,$$

$$\left\lceil \frac{\sigma}{2\varepsilon} \right\rceil \le \text{comp}(\varepsilon, \mathcal{N}_\tau, \sigma) \le \text{cost}(A_\varepsilon, \mathcal{N}_\tau, \sigma) \le 2\lceil \tau \rceil + \max\left(2, \frac{\sigma}{\varepsilon}\right) + 1.$$

Proof. For any nice function f lying in $\mathcal{N}_\tau$, its size is bounded above by $\hat{\sigma}_n(f)/(1 - \tau/n)$ according to (34). Furthermore, by the proof of Theorem 4 it follows that

$$\left\| f - \tilde{A}_n(f) \right\|_\infty \le \frac{\text{size}(f)}{2n} \le \frac{\hat{\sigma}_n(f)}{2(n - \tau)}.$$

Thus, when the stopping criterion in Step 2 is satisfied, the resulting approximation is within the desired tolerance.

The algorithm terminates for the first positive integer j satisfying the inequality in Step 2. Since $\hat{\sigma}(f)$ never overestimates size(f) for all $f \in \mathcal{H}$, this means that

$$\begin{aligned} n \ge \tau + \frac{\hat{\sigma}_n(f)}{2\varepsilon} &\Longleftarrow (\lceil \tau \rceil + 1)2^{j-1} = n_1 2^{j-1} = n \ge \lceil \tau \rceil + \frac{\text{size}(f)}{2\varepsilon} \\ &\Longleftarrow j \ge 2 + \log_2 \left(\frac{\lceil \tau \rceil + \frac{\text{size}(f)}{2\varepsilon}}{\lceil \tau \rceil + 1} \right) \\ &\Longleftarrow n \ge 2\lceil \tau \rceil + \max\left(2, \frac{\text{size}(f)}{\varepsilon}\right). \end{aligned}$$

This establishes an upper bound on the cost of the algorithm.

The proof of the lower bound is similar to that in Theorem 4. Consider *any* (possibly adaptive) algorithm A using $n \le \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil - 1$ function evaluations. A

fooling function $f^* \in \mathcal{N}_\tau$ is constructed to vanish at all data sites, ξ_i. Unlike Theorem 4, the fooling function must be continuously differentiable. The bump function in Theorem 4 is replaced by

$$\phi(x) := [\max(0, 1 - x^2)]^2 = \begin{cases} (1-x^2)^2, & |x| \le 1, \\ 0, & |x| > 1, \end{cases} \tag{35}$$

which has first and second derivatives

$$\phi'(x) = \begin{cases} -4x(1-x^2), & |x| \le 1, \\ 0, & |x| > 1, \end{cases} \qquad \phi''(x) = \begin{cases} -4(1-3x^2), & |x| \le 1, \\ 0, & |x| > 1, \end{cases} \tag{36}$$

and note that $\|\phi\|_\infty = 1$, $\|\phi'\|_\infty = 8/(3\sqrt{3})$, and $\|\phi''\|_\infty = 8$. Given the algorithm A with data sites $\xi_1, \ldots, \xi_n$, there exists at least one point $\xi_0 \in [0, 1)$ with $\min_{1 \le i \le n} |\xi_i - \xi_0| \ge 1/(2n)$. Then define the fooling function by

$$f^* : x \mapsto \frac{\sigma}{2n} \psi(2n(x - \xi_0)).$$

By definition, $\pm f^*(\xi_i) = 0$ for $i = 1, \ldots, n$, and $\left\| \pm f^{*\prime} \right\|_\infty = \sigma$, so $\pm f^* \in \mathcal{B}_\sigma$. Moreover, $\pm f^*(\xi_i) = \pm\sigma/(2n)$. Since the two functions $\pm f^*$ provide the same data to algorithm A, it follows that $A(f)(\xi_0) = A(-f)(\xi_0)$. The triangle inequality and the strict upper bound on n then imply that the approximation error must be greater than the desired tolerance.

$$\begin{aligned} \sup_{f \in \mathcal{B}} \|f - A(f)\|_\infty &\ge \max\left(\|f^* - A(f^*)\|_\infty, \|-f^* - A(-f^*)\|_\infty \right) \\ &\ge \max\left(|f^*(\xi_0) - A(f)(\xi_0)|, |-f^*(\xi_0) - A(-f^*)(\xi_0)| \right) \\ &\ge \frac{1}{2} \left[|f(\xi_0) - A(f^*)(\xi_0)| + |f^*(\xi_0) + A(-f^*)(\xi_0)| \right] \\ &\ge \frac{1}{2} |f^*(\xi_0) - A(f^*)(\xi_0) + f^*(\xi_0) + A(-f^*)(\xi_0)| \\ &= |f^*(\xi_0)| = \frac{\sigma}{2n} \ge \frac{\sigma}{2\left(\left\lceil \frac{\sigma}{2\varepsilon} \right\rceil - 1\right)} > \varepsilon. \end{aligned}$$

Thus, $\mathrm{comp}(\varepsilon, \mathcal{B}_\sigma, \Lambda^{\mathrm{std}}) > \left\lceil \frac{\sigma}{2\varepsilon} \right\rceil - 1$, which completes the proof. □

[1] J. F. Traub, G. W. Wasilkowski, H. Woźniakowski, Information-Based Complexity, Academic Press, Boston, 1988.