The Complexity of Deterministic Guaranteed Adaptive Automatic Algorithms: Cones, Not Balls

Nicholas Clancy, Yuhan Ding, 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

Automatic numerical algorithms are widely used in practice. An algorithm that is automatic attempts to provide an approximate solution that differs from the true solution by no more than a user-specified error tolerance, ε. Furthermore, the computational effort required is typically determined adaptively by the algorithm based on function data, e.g., function values. Ideally, the computational cost should match the difficulty of the problem. Unfortunately, most automatic algorithms lack *rigorous guarantees*, i.e., sufficient conditions on the input function that ensure the success of the algorithm.

This article establishes a framework for automatic, adaptive algorithms that do have rigorous guarantees. Sufficient conditions for success and upper bounds on the computational cost are provided in Theorems 2 and 3. Lower bounds on the complexity of the problem are given in Theorem 6 and conditions are given under which the proposed algorithms attain those lower bounds in Corollary 1. These general theorems are illustrated with automatic algorithms for univariate numerical integration and function recovery. Both algorithms use linear splines to approximate the input function.

The key idea behind these automatic algorithms is that the error analysis should be done for *cones* of input functions rather than balls. The existing literature contains certain negative results about the usefulness and reliability of automatic algorithms. The theory presented does not share the assumptions on which those negative results are based, and so they are irrelevant.

Keywords: adaptive, automatic, cones, function recovery, integration, quadrature
2010 MSC: 65D05, 65D30, 65G20

1. Introduction

Automatic algorithms conveniently determine the computational effort required to obtain an approximate answer that is within a user-supplied error tolerance, ε, of the true answer. Unfortunately, existing guaranteed automatic

algorithms are typically not adaptive, meaning that the computational cost does not depend on the function values sampled by the algorithm. On the other hand, adaptive automatic algorithms are typically not guaranteed to provide answers satisfying the error tolerance. This article provides a way of constructing automatic algorithms that are both guaranteed and adaptive.

1.1. Automatic Algorithms for Balls of Input Functions

Let $\mathcal{F}$ be a linear space of input functions with semi-norm $|\cdot|_{\mathcal{F}}$, $\mathcal{G}$ be a linear space of outputs with norm $\|\cdot\|_{\mathcal{G}}$, and $S : \mathcal{F} \to \mathcal{G}$ be a *solution operator*. Suppose that one has a sequence of algorithms, $\{A_n\}_{n\in\mathcal{I}}$, indexed by the computational cost n, where $\mathcal{I} \subseteq \mathbb{N}_0$, and that these algorithms satisfy some known error bound:

$$\|S(f) - A_n(f)\|_{\mathcal{G}} \leq h(n)\,|f|_{\mathcal{F}}\,, \tag{1a}$$

where $h : \mathcal{I} \to [0, \infty)$ is non-increasing. It is necessary for A_n to be exact if the semi-norm of the input function vanishes, i.e., $S(f) = A_n(f)$ if $|f|_{\mathcal{F}} = 0$. Define the (non-increasing) inverse of h as

$$h^{-1}(\varepsilon) = \min\{n \in \mathcal{I} : h(n) \leq \varepsilon\}, \qquad \varepsilon > 0. \tag{1b}$$

Error bound (1a) allows one to construct an automatic, but non-adaptive algorithm that is guaranteed for input functions in a prescribed $\mathcal{F}$-ball.

Algorithm 1. (Automatic, Non-Adaptive). Let $\mathcal{F}$, $|\cdot|_{\mathcal{F}}$, $\mathcal{G}$, $\|\cdot\|_{\mathcal{G}}$, S, and $\{A_n\}_{n\in\mathbb{N}}$ be defined as above, and let σ be a fixed positive number. Given ε, a positive error tolerance, compute the computational cost needed to satisfy the error tolerance: $n = h^{-1}(\varepsilon/\sigma)$. Then return $A_n(f)$ as the answer.

Theorem 1. *Under the assumptions of Algorithm 1, if f lies in the* ball

$$\mathcal{B}_\sigma = \{f \in \mathcal{F} : |f|_{\mathcal{F}} \leq \sigma\}, \tag{2}$$

then the answer provided by Algorithm 1 must satisfy the error tolerance, i.e., $\|S(f) - A_n(f)\|_{\mathcal{G}} \leq \varepsilon$.

While Algorithm 1 is guaranteed to return an answer with the desired accuracy by Theorem 1, this algorithm has drawbacks. If this algorithm works for $f \in \mathcal{F}$, it may not work for $cf \in \mathcal{F}$, where c is some constant, because cf may fall outside the ball $\mathcal{B}_\sigma$. Moreover, although error bound (1a) depends on $|f|_{\mathcal{F}}$, the computational cost of Algorithm 1 does not depend on $|f|_{\mathcal{F}}$. The cost is the same whether $|f|_{\mathcal{F}} = \sigma$ or $|f|_{\mathcal{F}}$ is much smaller than σ. This is because Algorithm 1 makes no use of the function values sampled to estimate $|f|_{\mathcal{F}}$.

1.2. Adaptive Automatic Algorithms for Cones of Input Functions

Adaptive automatic algorithms are common in numerical software packages. Examples include MATLAB's `quad` and `integral` [18], the quadrature algorithms in the NAG Library [19], and the MATLAB Chebfun toolbox [5]. While these adaptive algorithms work well in practice for many cases, they do not

have rigorous guarantees of success. The methods used to determine the computational cost are heuristics or either asymptotic error estimates that do not necessarily hold for finite sample sizes.

In this article we derive guaranteed adaptive automatic algorithms by using function data to construct a *rigorous* upper bound on the semi-norm $|f|_{\mathcal{F}}$ that appears in (1a). The details are presented in Section 3, but we summarize the argument here. The key idea is to consider functions lying in a *cone*, not a ball. A cone is a set with the property that any positive multiple of an element in the set is also in the set.

Let $|\cdot|_{\widetilde{\mathcal{F}}}$ be some weaker semi-norm than $|\cdot|_{\mathcal{F}}$, i.e., there exists a positive constant $\tau_{\min}$ for which

$$\tau_{\min} |f|_{\widetilde{\mathcal{F}}} \le |f|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}. \tag{3}$$

The $\widetilde{\mathcal{F}}$-semi-norm should be weak enough that $|f|_{\widetilde{\mathcal{F}}}$ can be estimated using function data as accurately as needed with a computational cost depending on $|f|_{\mathcal{F}}$. To assure that $|f|_{\widetilde{\mathcal{F}}}$ can be estimated with reasonable cost, f is assumed to lie in the cone

$$\mathcal{C}_\tau = \{f \in \mathcal{F} : |f|_{\mathcal{F}} \le \tau |f|_{\widetilde{\mathcal{F}}}\}. \tag{4}$$

Note that functions in this cone may have arbitrarily large $\mathcal{F}$- and $\widetilde{\mathcal{F}}$-semi-norms. Since the stronger norm is now bounded above by a constant times the weaker norm, it is possible to construct reliable data-based upper bounds for $|f|_{\widetilde{\mathcal{F}}}$ and $|f|_{\mathcal{F}}$. This latter upper bound allows one to determine the computational cost needed to estimate $S(f)$ via error bound (1a).

An explicit application of these ideas to the problem of evaluating $\int_0^1 f(x)\,\mathrm{d}x$ is provided in Section 5. There $\mathcal{F}$ is the set of all continuous functions whose first derivatives have finite total variation, $|f|_{\mathcal{F}} = \mathrm{Var}(f')$, and $|f|_{\widetilde{\mathcal{F}}} = \|f'\|_1$. The algorithm used is the composite equal-width trapezoidal rule. The cost of the guaranteed adaptive automatic algorithm is no greater than $4 + \tau + \sqrt{\tau\,\mathrm{Var}(f')/(4\varepsilon)}$ (Theorem 7), where $\mathrm{Var}(f')$ is unknown a priori. Here τ represents is related to the minimum sample size, and $1/\tau$ represents a length scale of for possible spikes that one wishes to integrate accurately.

1.3. Scope and Outline of this Article

There are limited theoretical results providing conditions under which adaption is useful. Novak [12] shows the advantage of adaption in for some problems in the average case and randomized settings. Plaskota and Wasilkowski [13, 15] demonstrate the advantage of adaption for integrating or approximating functions with singularities. By contrast, here we consider the deterministic setting, and we are not concerned with functions with singularities.

This article starts with the general setting and then moves to two concrete cases. Section 2 defines the problems to be solved and introduces our notation. Sections 3 and 4 describe the automatic algorithms in detail and provides proofs of their success for cones of input functions. Although the long term goal of

this research is to construct good locally adaptive algorithms, where the sampling density varies according to the function data, here we present only globally adaptive algorithms, where the sampling density is constant, but the number of samples is determined adaptively. Section 5 illustrates the general results in Sections 3 and 4 for the univariate integration problem. Section 6 presents analogous results for function approximation. Common concerns about automatic and adaptive algorithms are answered in Section 7. The article ends with several suggestions for future work.

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 $S : \mathcal{F} \to \mathcal{G}$, where $(\mathcal{F}, |\cdot|_{\mathcal{F}})$ is a semi-normed linear space of possible input functions defined on $\mathcal{X}$, and $(\mathcal{G}, \|\cdot\|_{\mathcal{G}})$ is some other normed linear space of possible outputs or solutions. The solution operator is assumed to be positively homogeneous, i.e.,

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

Examples include the following:

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

The first three examples above are linear problems, but the last example is a nonlinear problem, which nevertheless is positively homogeneous.

An automatic algorithm $A : \mathcal{F} \times (0, \infty) \to \mathcal{G}$ takes as inputs a function, f, and an error tolerance, ε. Our goal is to find efficient A for which $\|S(f) - A(f, \varepsilon)\|_{\mathcal{G}} \leq \varepsilon$. Algorithm 1 is one non-adaptive example that is successful for functions in balls.

Following the definition of algorithms described in [20, Section 3.2], the algorithm takes the form of some function of data derived from the input function:

$$A(f, \varepsilon) = \phi(\boldsymbol{L}(f)), \quad \boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f)) \qquad \forall f \in \mathcal{F}. \tag{5}$$

Here the $L_i \in \Lambda$ are real-valued homogeneous functions defined on $\mathcal{F}$:

$$L(cf) = cL(f) \qquad \forall f \in \mathcal{F},\ c \in \mathbb{R},\ L \in \Lambda. \tag{6}$$

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 ε, and the choice of L_i may depend on $L_1(f), \ldots, L_{i-1}(f)$. The set of all such algorithms is denoted $\mathcal{A}(\mathcal{F}, \mathcal{G}, S, \Lambda)$. In this article, all algorithms are assumed to be deterministic. There is no random element.

In most cases the algorithms we consider are designed for input functions lying in some proper subset of $\mathcal{F}$, such as a ball or a cone. Algorithms that know that the input function comes from $\mathcal{N} \subseteq \mathcal{F}$ are said to lie in $\mathcal{A}(\mathcal{N}, \mathcal{G}, S, \Lambda)$. For example, Algorithm 1 lies $\mathcal{A}(\mathcal{B}_\sigma, \mathcal{G}, S, \Lambda)$.

2.2. Costs of Algorithms

The cost of a possibly adaptive algorithm, A, depends on the function and the error tolerance:

$$\text{cost}(A, f, \varepsilon) = \$(\boldsymbol{L}) = \$(L_1) + \cdots + \$(L_m) \in \mathbb{N}_0,$$

where $\$: \Lambda \to \mathbb{N}$, and $\$(L)$ is the cost of acquiring the datum $L(f)$. The cost of L may be the same for all $L \in \Lambda$, e.g, $\$(L) = 1$. Alternatively, the cost might vary with the choice of L. For example, if f is a function of the infinite sequence of real numbers, $(x_1, x_2, \ldots)$, the cost of evaluating the function with arbitrary values of the first d coordinates, $L(f) = f(x_1, \ldots, x_d, 0, 0, \ldots)$, might be d. This cost model has been used by [7, 8, 10, 11, 14] for integration problems and [22, 23, 24] for function approximation problems. If an algorithm does not require any function data, then its cost is zero.

Although the cost of an adaptive algorithm varies with f, we hope that it does not vary wildly for different input functions with the same $\mathcal{F}$-semi-norm. We define the *maximum* and *minimum* costs of the algorithm $A \in \mathcal{A}(\mathcal{N}, \mathcal{G}, S, \Lambda)$ relative to $\mathcal{B}_\sigma$, the $\mathcal{F}$-semi-norm ball, as follows:

$$\text{maxcost}(A, \mathcal{N}, \varepsilon, \mathcal{B}_s) = \sup\{\text{cost}(A, f, \varepsilon) : f \in \mathcal{N} \cap \mathcal{B}_s\},$$

$$\text{mincost}(A, \mathcal{N}, \varepsilon, \mathcal{B}_s) = \inf\left\{\text{cost}(A, f, \varepsilon) : f \in \mathcal{N} \setminus \bigcup_{s'<s} \mathcal{B}_{s'}\right\}.$$

We stress that A knows that $f \in \mathcal{N}$ but does not necessarily know $|f|_{\mathcal{F}}$. An algorithm is said to have *$\mathcal{B}_\sigma$-stable computational cost* if

$$\sup_{\varepsilon, s>0} \frac{\text{maxcost}(A, \mathcal{N}, \varepsilon, \mathcal{B}_s)}{\max(1, \text{mincost}(A, \mathcal{N}, \varepsilon, \mathcal{B}_s))} < \infty. \tag{7}$$

Analogous definitions of maximum and minimum cost and stability of computational cost can be made in terms of $\widetilde{\mathcal{F}}$-semi-norm balls.

The complexity of a problem is defined as the maximum cost of the cheapest algorithm that always satisfies the error tolerance:

$$\begin{aligned}\text{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{G}, S, \Lambda), \mathcal{B}_s) \\ = \inf\{&\text{maxcost}(A, \mathcal{N}, \varepsilon, \mathcal{B}_s) : A \in \mathcal{A}(\mathcal{N}, \mathcal{G}, S, \Lambda), \\ &\|S(f) - A(f, \varepsilon)\|_{\mathcal{G}} \le \varepsilon \ \ \forall \varepsilon \ge 0\} \in \mathbb{N}_0.\end{aligned} \tag{8}$$

Here the infimum of an empty set is defined to be ∞. An adaptive automatic algorithm is *optimal* if its cost tracks that of the best possible possible algorithm, i.e.,

$$\sup_{\varepsilon,\sigma>0} \frac{\operatorname{maxcost}(A,\mathcal{N},\varepsilon,\mathcal{B}_\sigma)}{\max(1,\operatorname{comp}(\varepsilon,\mathcal{A}(\mathcal{N},\mathcal{G},S,\Lambda),\mathcal{B}_\sigma))} < \infty. \tag{9}$$

Algorithm 1 is an example of an algorithm that fits the framework described above. It is defined for input functions lying in the ball $\mathcal{B}_\sigma$. It is automatic, though not adaptive, and takes as its inputs a function, f, and an error tolerance, ε. It has a cost that depends only on ε/σ, and not on the particulars of f:

$$\operatorname{maxcost}(A,\mathcal{B}_\sigma,\varepsilon,\mathcal{B}_s) = \operatorname{mincost}(A,\mathcal{B}_\sigma,\varepsilon,\mathcal{B}_s) = \operatorname{cost}(A,f,\varepsilon) = h^{-1}(\varepsilon/\sigma). \tag{10}$$

The optimality of Algorithm 1 depends on whether h satisfies condition (28) below.

2.3. Non-Automatic Algorithms

Algorithm 1 is built using a sequence of non-automatic, fixed cost algorithms, $\{A_n\}_{n\in\mathcal{I}}$, indexed by their cost and defined for all $f \in \mathcal{F}$. The set of non-automatic algorithms is denoted $\mathcal{A}_{\text{non}}(\mathcal{F},\mathcal{G},S,\Lambda)$. These are algorithms for which neither the choice of the L_i nor the number of function data depend on the input function. Furthermore, such non-automatic algorithms have no dependence on ε, so we will typically write $A_n(f)$ rather than $A_n(f,\varepsilon)$ for convenience sake. Any non-automatic algorithm, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{F},\mathcal{G},S,\Lambda)$ is assumed to be satisfy the following positive homogeneity properties:

$$\boldsymbol{L}(cf) = c\boldsymbol{L}(f), \quad \phi(c\boldsymbol{y}) = c\phi(\boldsymbol{y}), \quad A_n(cf) = cA_n(f) \quad \forall c \geq 0,\ \boldsymbol{y} \in \mathbb{R}^m. \tag{11}$$

These conditions make the error of these algorithms homogeneous, resulting in error bounds involving a non-increasing h as described in (1).

The adaptive, automatic algorithms constructed in the next section use sequences of non-automatic algorithms indexed by their cost that converge to the true answer as the cost increases:

$$\{A_n\}_{n\in\mathcal{I}}, \quad A_n \in \mathcal{A}_{\text{non}}(\mathcal{F},\mathcal{G},S,\Lambda), \quad \operatorname{cost}(A_n) = n, \quad \inf_{n\in\mathcal{I}} h(n) = 0. \tag{12a}$$

Here in this article the positive integer-valued index set, $\mathcal{I} = \{n_1, n_2, \ldots\} \subseteq \mathbb{N}_0$, with $n_i < n_{i+1}$ is assumed to satisfy

$$\sup_i \frac{n_{i+1}}{n_i} < \infty. \tag{12b}$$

Note that the definition of h^{-1} in (1b) implies that

$$h(n) \leq \varepsilon \implies n \geq h^{-1}(\varepsilon), \qquad h(n) > \varepsilon \implies n < h^{-1}(\varepsilon). \tag{12c}$$

Finally, in this article we consider sequences of algorithms for which h satisfies

$$\sup_{\epsilon>0} \frac{h^{-1}(\varepsilon)}{h^{-1}(2\varepsilon)} < \infty. \tag{12d}$$

This means that $h(n)$ may decay polynomially in n^{-1} or faster as n tends to infinity.

3. General Algorithms and Upper Bounds on the Complexity

This section provides rather general theorems about the complexity of automatic algorithms. In some sense, these theorems are a road map because their assumptions are non-trivial and require effort to verify for specific problems of interest. On the other hand, the assumptions are reasonable as is demonstrated by the concrete examples in Sections 5 and 6.

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

As mentioned in Section 1.2 adaptive automatic algorithms require reliable upper bounds on $|f|_{\widetilde{\mathcal{F}}}$ for all f in the cone $\mathcal{C}_\tau$. These can be obtained using any non-automatic algorithm $\widetilde{F}_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\widetilde{\mathcal{F}}}, \Lambda)$ for approximating $|f|_{\widetilde{\mathcal{F}}}$ for $f \in \mathcal{F}$ and having cost n, provided that one has explicit upper bounds on the errors of these algorithms. This means that there exist non-negative valued, non-increasing functions $h_\pm$ such that

$$-h_-(n)\,|f|_{\mathcal{F}} \le |f|_{\widetilde{\mathcal{F}}} - \widetilde{F}_n(f) \le h_+(n)\,|f|_{\mathcal{F}}\,, \qquad \forall f \in \mathcal{F}. \tag{13}$$

This implies that $\widetilde{F}_n(f) = |f|_{\widetilde{\mathcal{F}}}$ for all $f \in \mathcal{F}$ with vanishing $\mathcal{F}$-semi-norm. Applying the two bounds for the $\mathcal{F}$- and $\widetilde{\mathcal{F}}$-semi-norms in (3) and (4) and rearranging the above error bound implies that

$$\widetilde{F}_n(f) \le |f|_{\widetilde{\mathcal{F}}} + h_-(n)\,|f|_{\mathcal{F}} \le \begin{cases} [1+\tau h_-(n)]\,|f|_{\widetilde{\mathcal{F}}} \\ \left[\dfrac{1}{\tau_{\min}} + h_-(n)\right]|f|_{\mathcal{F}} \end{cases} \qquad \forall f \in \mathcal{C}_\tau,$$

$$\widetilde{F}_n(f) \ge |f|_{\widetilde{\mathcal{F}}} - h_+(n)\,|f|_{\mathcal{F}} \ge \begin{cases} [1-\tau h_+(n)]\,|f|_{\widetilde{\mathcal{F}}} \\ \left[\dfrac{1}{\tau} - h_+(n)\right]|f|_{\mathcal{F}} \end{cases} \qquad \forall f \in \mathcal{C}_\tau.$$

Lemma 1. *Any non-automatic algorithm $\widetilde{F}_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\widetilde{\mathcal{F}}}, \Lambda)$ with cost $n = \text{cost}(\widetilde{F}_n)$ and two sided error bounds as in* (13) *yields an approximation to the $\widetilde{\mathcal{F}}$-semi-norm of functions in the cone $\mathcal{C}_\tau$ with the following upper and lower error bounds:*

$$\frac{|f|_{\mathcal{F}}}{\tau\mathfrak{C}_n} \le \frac{|f|_{\widetilde{\mathcal{F}}}}{\mathfrak{C}_n} \le \widetilde{F}_n(f) \le \begin{cases} \tilde{\mathfrak{c}}_n\,|f|_{\widetilde{\mathcal{F}}} \\ \dfrac{\mathfrak{c}_n\,|f|_{\mathcal{F}}}{\tau_{\min}} \end{cases} \qquad \forall f \in \mathcal{C}_\tau, \tag{14}$$

where the $\mathfrak{c}_n$, $\tilde{\mathfrak{c}}_n$, and $\mathfrak{C}_n$ are defined as follows:

$$\tilde{\mathfrak{c}}_n := 1 + \tau h_-(n) \ge \mathfrak{c}_n := 1 + \tau_{\min} h_-(n) \ge 1, \tag{15}$$

$$\mathfrak{C}_n := \frac{1}{1-\tau h_+(n)} \ge 1, \qquad \textit{assuming } h_+(n) < 1/\tau. \tag{16}$$

3.2. Two-Stage Automatic Algorithms

Computing an approximate solution to the problem $S : \mathcal{C}_\tau \to \mathcal{G}$, e.g., integration or function approximation also depends on a sequence of non-automatic algorithms, $\{A_n\}_{n\in\mathcal{I}}$, satisfying (1) and (12). One may then use the upper bound in Lemma (1) to construct a data-generated upper bound on the error:

$$\|S(f) - A_n(f)\|_{\mathcal{G}} \le h(n)\,|f|_{\mathcal{F}} \le \tau\mathfrak{C}_n h(n)\widetilde{F}_n(f) \qquad \forall f \in \mathcal{C}_\tau. \tag{17}$$

Algorithm 2. (Adaptive, Automatic, Two-Stage). Let $\mathcal{F}$, $|\cdot|_{\mathcal{F}}$, $|\cdot|_{\widetilde{\mathcal{F}}}$, $\mathcal{G}$, $\|\cdot\|_{\mathcal{G}}$, and S be as described above. Let τ be a fixed positive number, and let $\widetilde{F}_{n_{\widetilde{F}}}$ be an algorithm as described in Lemma 1 with cost $n_{\widetilde{F}}$ satisfying $h_+(n_{\widetilde{F}}) < 1/\tau$. Moreover, let $\{A_n\}_{n\in\mathcal{I}}$ be a sequence of algorithms as described in (1) and (12). Given a positive error tolerance, ε, and an input function f, do the following:

Step 1. Bound $|f|_{\mathcal{F}}$. First compute $\widetilde{F}_{n_{\widetilde{F}}}(f)$. Define the inflation factor $\mathfrak{C} = \mathfrak{C}_{n_{\widetilde{F}}}$ according to (16). Then $\tau\mathfrak{C}\widetilde{F}_{n_{\widetilde{F}}}(f)$ provides a reliable upper bound on $|f|_{\mathcal{F}}$.

Step 2. Estimate $S(f)$. Choose the sample size need to approximate $S(f)$, namely, $n_A = h^{-1}(\varepsilon/(\tau\mathfrak{C}\widetilde{F}_{n_{\widetilde{F}}}(f)))$. Finally, return $A_{n_A}(f)$ as the approximation to $S(f)$ at a total cost of $n_{\widetilde{F}} + n_A$.

The bounds in Lemma 1 involving $\widetilde{F}_n$ imply bounds on the cost in the algorithm above. Since h^{-1} is a non-increasing function, it follows that for all $f \in \mathcal{C}_\tau$

$$h^{-1}\left(\frac{\varepsilon}{|f|_{\mathcal{F}}}\right) \le h^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\widetilde{\mathcal{F}}}}\right) \le h^{-1}\left(\frac{\varepsilon}{\tau\mathfrak{C}\widetilde{F}_{n_{\widetilde{F}}}(f)}\right) \le \begin{cases} h^{-1}\left(\dfrac{\varepsilon}{\tau\mathfrak{C}\tilde{\mathfrak{c}}\,|f|_{\widetilde{\mathcal{F}}}}\right) \\ h^{-1}\left(\dfrac{\tau_{\min}\varepsilon}{\tau\mathfrak{C}\mathfrak{c}\,|f|_{\mathcal{F}}}\right) \end{cases}.$$

Theorem 2. *Under the assumptions of Algorithm 2, let $\mathfrak{c} = \mathfrak{c}_{n_{\widetilde{F}}}$ be defined as in* (15)*. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (4) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\widetilde{\mathcal{F}}$-semi-norms. Then it follows that Algorithm 2, which lies in $\mathcal{A}(\mathcal{C}_\tau, \mathcal{G}, S, \Lambda)$, is successful, i.e., $\|S(f) - A(f,\varepsilon)\|_{\mathcal{G}} \le \varepsilon$ for all $f \in \mathcal{C}_\tau$. Moreover, the cost of this algorithm is bounded above and below in terms of the unknown $\widetilde{\mathcal{F}}$- and $\mathcal{F}$-semi-norms of any input function in $\mathcal{C}_\tau$ as follows:*

$$n_{\widetilde{F}} + h^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\widetilde{\mathcal{F}}}}\right) \le \mathrm{cost}(A,f,\varepsilon) \le n_{\widetilde{F}} + h^{-1}\left(\frac{\varepsilon}{\tau\mathfrak{C}\tilde{\mathfrak{c}}\,|f|_{\widetilde{\mathcal{F}}}}\right), \tag{18a}$$

$$n_{\widetilde{F}} + h^{-1}\left(\frac{\varepsilon}{|f|_{\mathcal{F}}}\right) \le \mathrm{cost}(A,f,\varepsilon) \le n_{\widetilde{F}} + h^{-1}\left(\frac{\tau_{\min}\varepsilon}{\tau\mathfrak{C}\mathfrak{c}\,|f|_{\mathcal{F}}}\right). \tag{18b}$$

Since h satisfies (12)*, then this algorithm is computationally stable in the sense that the maximum cost is no greater than some constant times the minimum cost, both for $\widetilde{\mathcal{F}}$-balls and $\mathcal{F}$-balls.*

Proof. The choice of n_A in Algorithm 2 ensures that the right hand side of (17) is no greater than the error tolerance, so the algorithm is successful, as claimed in the theorem. The argument preceding the statement of this theorem establishes the maximum and minimum cost bounds in (18). □

There are a several points to note about this result.

Remark 1. Two main conditions must be checked for this theorem to hold.

i. There must be an algorithm, $\widetilde{F}_n$, that approximates the weaker semi-norm, $|\cdot|_{\widetilde{\mathcal{F}}}$, and its error bound, $h_{\pm}$, as defined in (13) must be known explicitly, as described in Section 3.1.
ii. There must be a sequence of nonadaptive algorithms, $\{A_n\}_{n\in\mathcal{I}}$, as described at the beginning of Section 3.2, and the error function h, defined in (1) must be known explicitly.

Sections 5 and 6 provide concrete examples where these conditions are checked.

Remark 2. The maximum and minimum costs of Algorithm 2, as given by (18), depend on the $\mathcal{F}$- and $\widetilde{\mathcal{F}}$-semi-norms of the input function, f. However, the values of these semi-norms are not inputs to the algorithm, but rather are bounded by the algorithm. The number of samples used to obtain the approximation to $S(f)$ is adjusted adaptively based on these bounds.

Remark 3. Instead of choosing τ as an input parameter for Algorithm 2, one may alternatively choose the inflation factor $\mathfrak{C} > 1$. This then implies that

$$\tau = \left(1 - \frac{1}{\mathfrak{C}}\right) \frac{1}{h_+(n_{\widetilde{F}})}, \tag{19}$$

which is equivalent to (16).

Remark 4. In some cases it is possible to find a lower bound on the $\mathcal{F}$-norm of the input function. This means that there exists an algorithm F_n using the same function values as $\widetilde{F}_n$, such that

$$F_n(f) \le |f|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}.$$

When such an F_n is known, Lemma 1 can be used to derive a necessary condition that f is in the cone $\mathcal{C}_\tau$:

$$\begin{aligned} f \in \mathcal{C}_\tau &\implies F_n(f) \le |f|_{\mathcal{F}} \le \frac{\tau \widetilde{F}_n(f)}{1 - \tau h_+(n)} \\ &\implies \tau_{\min,n} := \frac{F_n(f)}{\widetilde{F}_n(f) + h_+(n) F_n(f)} \le \tau. \end{aligned} \tag{20}$$

For Algorithm 2 the relevant value of n is $n_{\widetilde{F}}$, whereas for Algorithm 3 the relevant value of n is n_i. We stress that this is not a sufficient condition for f to lie in $\mathcal{C}_\tau$, so it is possible to get an incorrect answer from Algorithm 2 or 3 even if (20) is satisfied but $f \notin \mathcal{C}_\tau$. However, in light of this argument one might modify these algorithms by increasing τ to, say, $2\tau_{\min,n}$ whenever $\tau_{\min,n}$ rises above τ.

Remark 5. It may also be possible to obtain an error upper bound in terms of weaker semi-norm, i.e., $\tilde{h}$ such that $\|S(f) - A_n(f)\|_{\mathcal{G}} \leq \tilde{h}(n)\,|f|_{\widetilde{\mathcal{F}}}$. By following the argument leading to (17), it is clear that such an error bound is no use if

$$\tau h(n) \leq \tilde{h}(n) \quad \forall n \in \mathcal{I}. \tag{21}$$

This condition is observed to hold for the examples of Sections 5 and 6. A sufficient condition for (21) is

$$h(n) \leq \tilde{h}(n) h_+(n) \quad \forall n \in \mathcal{I}, \tag{22}$$

since the Algorithms 2 and 3 both require that $h_+(n) < 1/\tau$, and since h is a non-increasing function.

3.3. Automatic Algorithms Based on Embedded 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 composite trapezoidal rules for integration where the number of trapezoids is a power of two. Furthermore, suppose that the data used by $\widetilde{F}_{n_{\widetilde{F}}}$, the algorithm used to estimate the $\widetilde{\mathcal{F}}$-semi-norm of f, is the same data used by A_{n_1}, and so $n_1 = n_{\widetilde{F}}$. Then the total cost of Algorithm 2 can be reduced to simply n_A, as given in Step 2, instead of $n_{\widetilde{F}} + n_A$.

Again suppose that $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{G}, S, \Lambda)$, consists of algorithms as described in (1) and (12), but some of which are embedded in others. An example would be all possible composite trapezoidal rules for integration that use trapezoids of equal widths. Moreover, suppose that there exists some fixed $r > 1$ such that for all $n \in \mathcal{I}$, there exists a $\tilde{n} \in \mathcal{I}$ with $n < \tilde{n} \leq rn$, such that the data for A_n is embedded in the data for $A_{\tilde{n}}$. One may think of r as the minimum cost multiple that one must incur when moving to the next more costly algorithm. For trapezoidal rules, where the cost is the number of trapezoids plus one, one may take $\tilde{n} - 1 = 2(n - 1)$, so one may choose $r = 2$.

Suppose also that there exists a sequence of algorithms for approximating the $\widetilde{\mathcal{F}}$-semi-norm, $\{\widetilde{F}_n\}_{n\in\mathcal{I}}$, $\widetilde{F}_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\widetilde{\mathcal{F}}}, \Lambda)$, such that for each $n \in \mathcal{I}$, A_n and $\widetilde{F}_n$ use exactly the same data. Since $h_\pm$ are non-increasing functions, the quantities $\mathfrak{C}_n$ and $\mathfrak{c}_n$ do not increase as n increases. These embedded algorithms suggest the following iterative algorithm.

Algorithm 3. Let $\mathcal{F}$, $|\cdot|_{\mathcal{F}}$, $|\cdot|_{\widetilde{\mathcal{F}}}$, $\mathcal{G}$, $\|\cdot\|_{\mathcal{G}}$, S, and τ be as described above. Let the sequences of algorithms, $\{A_n\}_{n\in\mathcal{I}}$ and $\{\widetilde{F}_n\}_{n\in\mathcal{I}}$ be as described above. Set $i = 1$, and $n_1 = \min\{n \in \mathcal{I} : h_+(n) < 1/\tau\}$. For any positive error tolerance ε and any input function f, do the following:

Step 1. Estimate $|f|_{\widetilde{\mathcal{F}}}$. Compute $\widetilde{F}_{n_i}(f)$ and $\mathfrak{C}_{n_i}$ as defined in (16).

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

$$\tau \mathfrak{C}_{n_i} h(n_i) \widetilde{F}_{n_i}(f) \leq \varepsilon. \tag{23}$$

If this is true, return $A_{n_i}(f)$ and terminate the algorithm.

Step 3. Compute n_{i+1}. Otherwise, if (23) fails to hold, compute $\mathfrak{c}_{n_i}$ according to (15), and choose n_{i+1} as the smallest number exceeding n_i and not less than $h^{-1}(\varepsilon \tilde{\mathfrak{c}}_{n_i} / [\tau \widetilde{F}_{n_i}(f)])$ such that A_{n_i} is embedded in $A_{n_{i+1}}$. Increment i by 1, and return to Step 1.

This iterative algorithm is guaranteed to converge also, and its cost can be bounded. Define

$$h_1(n) := \mathfrak{C}_n \tilde{\mathfrak{c}}_n h(n) \geq h(n), \quad h_2(n) := \mathfrak{C}_n \mathfrak{c}_n h(n) \geq h(n) \qquad n \in \mathcal{I}. \tag{24}$$

and note that h_1 and h_2 are non-increasing functions. Let h_1^{-1} and h_2^{-1} be defined analogously to h^{-1} as in (1b). These definitions imply that the quantity appearing on the left hand side of (23), has the following upper and lower bounds based on (14) for $f \in \mathcal{C}_\tau$:

$$h(n) \left| f \right|_{\mathcal{F}} \leq \tau h(n) \left| f \right|_{\widetilde{\mathcal{F}}} \leq \tau \mathfrak{C}_n h(n) \widetilde{F}_n(f) \leq \begin{cases} \tau h_1(n) \left| f \right|_{\widetilde{\mathcal{F}}} \\ \dfrac{\tau h_2(n) \left| f \right|_{\mathcal{F}}}{\tau_{\min}} \end{cases}, \quad n \in \mathcal{I}, \tag{25a}$$

$$\begin{aligned} h^{-1}\left(\frac{\varepsilon}{\left| f \right|_{\mathcal{F}}}\right) \leq h^{-1}\left(\frac{\varepsilon}{\tau \left| f \right|_{\widetilde{\mathcal{F}}}}\right) &\leq \min\{n : \tau \mathfrak{C}_n h(n) \widetilde{F}_n(f) \leq \varepsilon\} \\ &\leq \begin{cases} h_1^{-1}\left(\dfrac{\varepsilon}{\tau \left| f \right|_{\widetilde{\mathcal{F}}}}\right) \\ h_2^{-1}\left(\dfrac{\tau_{\min} \varepsilon}{\tau \left| f \right|_{\mathcal{F}}}\right) \end{cases}, \qquad \varepsilon > 0. \end{aligned} \tag{25b}$$

These inequalities may be used to prove the following theorem about Algorithm 3, which is analogous to Theorem 2.

Theorem 3. *Under the assumptions of Algorithm 3, let r be the minimum cost multiple described in the paragraphs preceding that Algorithm 3. Let $\mathcal{C}_\tau$ be the cone of functions defined in (4) whose $\mathcal{F}$-semi-norms are no larger than τ times their $\widetilde{\mathcal{F}}$-semi-norms. Then it follows that Algorithm 3, which lies in $\mathcal{A}(\mathcal{C}_\tau, \mathcal{G}, S, \Lambda)$, is successful, i.e., $\|S(f) - A(f, \varepsilon)\|_{\mathcal{G}} \leq \varepsilon$ for all $f \in \mathcal{C}_\tau$. Moreover, the cost of this algorithm is bounded above and below in terms of the unknown $\widetilde{\mathcal{F}}$- and $\mathcal{F}$-semi-norms of any input function in $\mathcal{C}_\tau$ as follows:*

$$\begin{aligned} \max\left(n_1, h^{-1}\left(\frac{\varepsilon}{\tau \left| f \right|_{\widetilde{\mathcal{F}}}}\right)\right) &\leq \operatorname{cost}(A, f, \varepsilon) \\ &\leq \max\left(n_1, r h_1^{-1}\left(\frac{\varepsilon}{\tau \left| f \right|_{\widetilde{\mathcal{F}}}}\right)\right), \end{aligned} \tag{26a}$$

$$\max\left(n_1, h^{-1}\left(\frac{\varepsilon}{|f|_{\mathcal{F}}}\right)\right) \leq \mathrm{cost}(A, f, \varepsilon)$$
$$\leq \max\left(n_1, rh_2^{-1}\left(\frac{\tau_{\min}\varepsilon}{\tau\,|f|_{\mathcal{F}}}\right)\right). \quad (26b)$$

If h satisfies (12)*, then this algorithm is computationally stable in the sense that the maximum cost is no greater than some constant times the minimum cost, both for $\widetilde{\mathcal{F}}$-balls and $\mathcal{F}$-balls.*

Proof. Let $n_1, n_2, \ldots$ be the sequence of n_i generated by Algorithm 3. We shall prove that under the hypotheses of the theorem, in particular $f \in \mathcal{C}_\tau$, that

i) if the convergence criterion (23) is satisfied for i, then the algorithm stops and $A_{n_i}(f)$ is returned as the answer and it meets the error tolerance, and
ii) if the convergence criterion (23) is not satisfied for i, then n_{i+1} does not exceed the cost upper bounds in (26).

Proposition i) holds because of the bounds in (17) and in Lemma 1.

If (23) is not satisfied for i, then it follows from the inequality in (25) that

$$n_i \leq \begin{cases} h_1^{-1}\left(\dfrac{\varepsilon}{\tau\,|f|_{\widetilde{\mathcal{F}}}}\right) \\ h_2^{-1}\left(\dfrac{\tau_{\min}\varepsilon}{\tau\,|f|_{\mathcal{F}}}\right) \end{cases} .$$

The algorithm then considers the candidate $n_{i+1}^* = h^{-1}(\varepsilon\tilde{\mathfrak{c}}_{n_i}/[\tau\widetilde{F}_{n_i}(f)])$ as a possible choice for n_{i+1}. If $n_i \geq n_{i+1}^*$, then Step 3 chooses n_i to be the smallest element of $\mathcal{I}$ that exceeds n_{i-1} and for which A_{n_i} is embedded in $A_{n_{i+1}}$. By the definition of r it follows that $n_{i+1} \leq rn_i$, and so by the above inequalities for n_i, it follows that n_{i+1} is bounded above by the right hand sides of the inequalities in (26).

If, on the other hand, $n_i < n_{i+1}^*$, then Step 3 chooses n_{i+1} to be the smallest element of $\mathcal{I}$ that is no less than n_{i+1}^* and for which $A_{n_{i+1}}$ is embedded in A_{n_i}. By the definition of r, (14), and the inequalities in (25), it follows that

$$n_{i+1} < rn_{i+1}^* = rh^{-1}\left(\frac{\varepsilon\tilde{\mathfrak{c}}_{n_{j-1}}}{\tau\widetilde{F}_{n_{j-1}}(f)}\right) \leq rh^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\widetilde{\mathcal{F}}}}\right) \leq \begin{cases} rh_1^{-1}\left(\dfrac{\varepsilon}{\tau\,|f|_{\widetilde{\mathcal{F}}}}\right) \\ rh_2^{-1}\left(\dfrac{\tau_{\min}\varepsilon}{\tau\,|f|_{\mathcal{F}}}\right) \end{cases} .$$

Again, n_{i+1} is bounded above by the right hand sides of the inequalities in (26).

Since Proposition ii) now holds, it means the the right hand side inequalities in (26) also hold for cost of the algorithm. The lower bounds on the computational cost also follow from (26). □

4. Lower Complexity Bounds for the Problems

Lower complexity bounds are typically proved by constructing fooling functions. First, a lower bound is derived for the complexity of problems defined on an $\mathcal{F}$-semi-norm ball of input functions, $\mathcal{B}_\sigma$. This technique is generally known, see for example [21, p. 11–12]. Then it is shown how to extend this idea for the cone $\mathcal{C}_\tau$.

Consider fixed (semi-)normed linear spaces $\mathcal{F}$ and $\mathcal{G}$, as described above, and now assume that the solution operator $S : \mathcal{F} \to \mathcal{G}$ is *linear*. Let Λ be the set of bounded linear functionals that can be used as data. Let $\mathcal{I}$ be a subset of $\mathbb{N}_0$ satisfying (12b). Suppose that for any $n \in \mathcal{I}$, 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 zero data, unit $\mathcal{F}$-semi-norm, and solution known lower bound, namely,

$$|f_1|_{\mathcal{F}} = 1, \qquad \boldsymbol{L}(f_1) = \boldsymbol{0}, \qquad \|S(f_1)\|_{\mathcal{G}} \ge g(n), \tag{27}$$

for some positive, non-increasing function g defined on $\mathcal{I}$ with $\inf_{n\in\mathcal{I}} g(n) = 0$. Let g^{-1} be defined analogously to h^{-1} in (1b). For example, one might have $g(n) = an^{-p}$ with positive a and p.

4.1. Problems Defined on Balls

Suppose that A is any successful automatic algorithm for the ball $\mathcal{B}_\sigma$, i.e., $A \in \mathcal{A}(\mathcal{B}_\sigma, \mathcal{G}, S, \Lambda)$, and $\|S(f) - A(f)\|_{\mathcal{G}} \le \varepsilon$ for all $f \in \mathcal{B}_\sigma$. For a fixed $\varepsilon > 0$ let $n = \operatorname{maxcost}(A, \mathcal{B}_\sigma, \varepsilon, \mathcal{B}_\sigma)$. Let $\boldsymbol{L}$ be the design used by the algorithm for the zero function, and let f_1 be constructed according to (27) for this $\boldsymbol{L}$. Since the data for the functions $\pm s f_1$ are all zero, it follows that $A(sf_1) = A(-sf_1)$. Also note that $\pm s f_1 \in \mathcal{B}_s$.

Now since A must be successful for $\pm\sigma f_1$, it follows that

$$\begin{aligned}
\varepsilon &\ge \max(\|S(sf_1) - A(sf_1)\|_{\mathcal{G}}, \|S(-sf_1) - A(-sf_1)\|_{\mathcal{G}}) \\
&\ge \frac{s}{2}\left[\|S(f_1) - A(f_1)\|_{\mathcal{G}} + \|S(f_1) + A(f_1)\|_{\mathcal{G}}\right] \\
&\ge \frac{s}{2}\|[S(f_1) - A(f_1)] + [S(f_1) + A(f_1)]\|_{\mathcal{G}} = s\,\|S(f_1)\|_{\mathcal{G}} \\
&\ge sg(n).
\end{aligned}$$

This implies lower bounds on the complexity of algorithms defined for $\mathcal{F}$-balls.

Theorem 4. *The computational complexity of the problem for a ball of input functions $\mathcal{B}_\sigma$ is bounded below by*

$$\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{B}_\sigma, \mathcal{G}, S, \Lambda), \mathcal{B}_s) \ge g^{-1}(\varepsilon/\min(\sigma, s)).$$

Thus, the cost of solving the problem within error tolerance ε for input functions in a $\mathcal{F}$-semi-norm ball of radius σ is at least $g^{-1}(\varepsilon/\sigma)$. This leads to a simple check on whether a sequence of non-adaptive algorithms is optimal.

Theorem 5. *Suppose that there exist non-automatic algorithms* $\{A_n\}_{n\in\mathcal{I}}$*, with* $A_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{G}, S, \Lambda)$*, for which the upper error bounds satisfy* (1) *for known* h*. If*

$$\sup_{n\in\mathcal{I}} \frac{h(n)}{g(n)} < \infty, \tag{28}$$

then Algorithm 1 is optimal in the sense that for fixed σ *and* s*,*

$$\sup_{\varepsilon>0} \frac{\text{maxcost}(A, \mathcal{B}_\sigma, \varepsilon, \mathcal{B}_s)}{\max(1, \text{comp}(\varepsilon, \mathcal{A}(\mathcal{B}_\sigma, \mathcal{G}, S, \Lambda), \mathcal{B}_s))} < \infty.$$

Proof. Choose a number $C \geq h(n)/g(n)$ for all $n \in \mathcal{I}$, which is possible by assumption (28). It then follows that

$$\begin{aligned} g^{-1}(\varepsilon) = \min\{n \in \mathbb{N} : g(n) \leq \varepsilon\} &\geq \min\{n \in \mathcal{I} : g(n) \leq \varepsilon\} \\ &\geq \min\left\{n \in \mathcal{I} : \frac{h(n)}{C} \leq \varepsilon\right\} = h^{-1}(C\varepsilon). \end{aligned}$$

Thus, it follows from the expression for the maximum cost in (10) and the condition on h in (12d) that the error of $\{A_n\}_{n\in\mathcal{I}}$ is no worse than a constant times the best possible non-adaptive algorithm:

$$\begin{aligned} &\sup_{0<\varepsilon\leq 1} \frac{\text{maxcost}(A, \mathcal{B}_\sigma, \varepsilon, \mathcal{B}_s)}{\max(1, \text{comp}(\varepsilon, \mathcal{A}(\mathcal{B}_\sigma, \mathcal{G}, S, \Lambda), \mathcal{B}_s))} \\ &\quad\leq \sup_{0<\varepsilon\leq 1} \frac{h^{-1}(\varepsilon/\sigma)}{\max(1, g^{-1}(\varepsilon/\min(\sigma, s)))} \\ &\quad\leq \sup_{0<\varepsilon\leq 1} \frac{h^{-1}(\varepsilon/\sigma)}{\max(1, h^{-1}(C\varepsilon/\min(\sigma, s)))} < \infty. \end{aligned}$$

□

4.2. Problems Defined on Cones

Now we turn to the solving the numerical problem where the input functions lie in the the cone $\mathcal{C}_\tau$. The cone condition makes the complexity lower bound a bit more difficult to derive. Note that condition (27) does not require the fooling function f_1 to lie *inside* this cone. To remedy this shortcoming, the fooling function is constructed as a linear combination of f_1 and another function, f_0, lying in the interior of the cone. Specifically, f_0 is assumed to satisfy

$$|f_0|_{\widetilde{\mathcal{F}}} = 1, \qquad |f_0|_{\mathcal{F}} \leq \tau_{\min} |f_0|_{\widetilde{\mathcal{F}}} = \tau_{\min} < \tau. \tag{29}$$

Linear combinations of f_0 and f_1 may be used to derive the following lower bound on the complexity of solving the problem S.

Theorem 6. *Assume that* $\tau > \tau_{\min}$ *and let* s *be some positive number. Suppose that functions* f_0 *and* f_1 *can be found that satisfy conditions* (27) *and* (29)*. It*

then follows that the complexity of the problem for cones of input function is bounded below by

$$\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{G}, S, \Lambda), \infty, \mathcal{B}_s) \geq g^{-1}\left(\frac{2\tau\varepsilon}{s(\tau - \tau_{\min})}\right).$$

Proof. Let $A \in \mathcal{A}(\mathcal{C}_\tau, \mathcal{G}, S, \Lambda)$ be an arbitrary successful, possibly adaptive, algorithm. Given an error tolerance, ε, and a positive σ, let f_0 be a function satisfying (29) and choose

$$c_0 = \frac{s(\tau + \tau_{\min})}{2\tau\tau_{\min}} > 0. \tag{30a}$$

Provide the algorithm A with the input function $c_0 f_0$, and let $\boldsymbol{L}(c_0 f_0)$ 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 f_1 satisfying conditions (27) with c_1 satisfying

$$c_1 = \frac{s(\tau - \tau_{\min})}{2\tau} > 0. \tag{30b}$$

Both fooling functions have $\mathcal{F}$-semi-norms no greater than s, since

$$\begin{aligned} |f_\pm|_{\mathcal{F}} &\leq c_0 \, |f_0|_{\mathcal{F}} + c_1 \, |f_1|_{\mathcal{F}} \\ &= \frac{s}{2\tau}\left[\frac{\tau + \tau_{\min}}{\tau_{\min}}\tau_{\min} + (\tau - \tau_{\min})\right] = s \qquad \text{by (30).} \end{aligned}$$

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

$$\begin{aligned} |f_\pm|_{\mathcal{F}} - \tau \, |f_\pm|_{\widetilde{\mathcal{F}}} &\leq s - \tau(c_0 \, |f_0|_{\widetilde{\mathcal{F}}} - c_1 \, |f_1|_{\widetilde{\mathcal{F}}}) \\ &\qquad \text{by the triangle inequality} \\ &\leq s - \tau c_0 + \frac{\tau}{\tau_{\min}} c_1 \qquad \text{by (27), (29)} \\ &= s - \frac{s(\tau + \tau_{\min})}{2\tau_{\min}} + \frac{s(\tau - \tau_{\min})}{2\tau_{\min}} = 0 \qquad \text{by (30).} \end{aligned}$$

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,

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

Since A is successful for these two fooling functions, it follows that n the cost of this arbitrarily algorithm, must be bounded below by

$$g^{-1}\left(\frac{\varepsilon}{c_1}\right) = g^{-1}\left(\frac{2\tau\varepsilon}{s(\tau - \tau_{\min})}\right).$$

This then implies the lower bound on the complexity of the problem. □

Minimum cost of the **best** algorithm that knows that $f \in \mathcal{C}_\tau$ and $\|f\|_{\mathcal{F}}$	$\geq g^{-1}\left(\dfrac{2\tau\varepsilon}{\|f\|_{\mathcal{F}}(\tau-\tau_{\min})}\right)$
Minimum cost of the **best** algorithm that knows that $f \in \mathcal{C}_\tau$, but not $\|f\|_{\mathcal{F}}$	$\geq g^{-1}\left(\dfrac{2\tau\varepsilon}{\|f\|_{\mathcal{F}}(\tau-\tau_{\min})}\right)$
Cost of a **nonadaptive** algorithm (like Algorithm 1) that knows $\|f\|_{\mathcal{F}}$	$h^{-1}\left(\dfrac{\varepsilon}{\|f\|_{\mathcal{F}}}\right)$
Minimum cost of **adaptive** Algorithm 3 that knows $f \in \mathcal{C}_\tau$, but not $\|f\|_{\mathcal{F}}$	$\max\left(n_1, h^{-1}\left(\dfrac{\varepsilon}{\|f\|_{\mathcal{F}}}\right)\right)$
Maximum cost of **adaptive** Algorithm 3 that knows $f \in \mathcal{C}_\tau$, but not $\|f\|_{\mathcal{F}}$	$\max\left(n_1, rh_2^{-1}\left(\dfrac{\tau_{\min}\varepsilon}{\tau\|f\|_{\mathcal{F}}}\right)\right)$

Table 1: Costs of various algorithms, A, guaranteed to satisfy the tolerance, i.e., $\|S(f) - A(f)\|_{\mathcal{G}} \leq \varepsilon$.

Table 1 summarizes the lower and upper bounds on the computational complexity of computing $S(f)$ to within an absolute error of ε. The results summarized here are based on Theorems 1, 3, and 6.

Comparing the lower bound on the computational complexity in Theorem 6 to the upper bounds on the cost of the adaptive algorithms in Theorems 2 and 3, one can see that these adaptive algorithms are optimal for solving the problem for a cone of input functions if their non-automatic, non-adaptive building blocks satisfy condition (28). In fact, under this condition all the algorithms mentioned in Table 1, from the best ideal ones to the actual ones, have essentially the same computational cost.

Corollary 1. *Suppose that the functions g and h satisfy the hypotheses of Theorem 5, i.e., conditions* (28)*, which means that Algorithm 1 is optimal for solving the problem on $\mathcal{F}$-balls of input functions. It then follows that Algorithms 2 and 3 are both optimal for solving the problem on for input functions lying in the cone $\mathcal{C}_\tau$.*

The next two sections illustrate the theorems of Section 3 by looking at the problems of integration and approximation. Each section identifies

- the semi-normed space of functions, $(\mathcal{F}, |\cdot|_{\mathcal{F}})$, with the weaker semi-norm $|\cdot|_{\widetilde{\mathcal{F}}}$, the normed space of outputs, $(\mathcal{G}, \|\cdot\|_{\mathcal{G}})$, and the solution operator, S,
- a set of non-automatic algorithms, $\{A_n\}_{n\in\mathcal{I}}$, which approximate S, are indexed by their cost, n, and have the property that some lower cost algorithms are embedded in higher cost algorithms, the minimum cost multiple, r, defined just before Algorithm 3, and the upper bound error function, h, defined in (1), and satisying (12),

- a set of non-automatic algorithms, $\{\widetilde{F}_n\}_{n\in\mathcal{I}}$, which approximate $|\cdot|_{\widetilde{\mathcal{F}}}$, such that $\widetilde{F}_n$ uses the same function data as A_n, the error bounds $h_{\pm}(n)$ and the deflation and inflation factors, $\mathfrak{c}_n$ and $\mathfrak{C}_n$, which are defined in (15) and (16) respectively, and
- the fooling functions f_0 and f_1, along with the function g, all of which satisfy (27) and (29).

This allows one to apply Algorithms 2 and 3 with the guarantees provided by Theorems 2 and 3, the lower bound on complexity provided by Theorem 6, and the optimality given by Corollary 1.

Fix this here and below.

5. Approximation of One-Dimensional Integrals

The algorithms used in this section on integration and the next section on function recovery are all based on linear splines. The node set and the linear spline algorithm using n function values are defined for $n \in \mathcal{I} = \{2, 3, \ldots\}$ as follows:

$$x_i = \frac{i-1}{n-1}, \qquad i = 1, \ldots, n, \tag{31a}$$

$$A_n(f)(x) = (n-1)\left[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)\right] \quad \text{for } x_i \le x \le x_{i+1}. \tag{31b}$$

The cost of each function value is one and so the cost of A_n is n.

The first example we consider is univariate integration on the unit interval, $S(f) = \mathrm{INT}(f) := \int_0^1 f(x)\,\mathrm{d}x \in \mathcal{H} := \mathbb{R}$. The space of input functions is $\mathcal{F} = \mathcal{V}^1$, the space of functions whose first derivatives have finite total variation. The general definition of these relevant norms and spaces are as follows:

$$\mathrm{Var}(f) := \sup_{\substack{n\in\mathbb{N} \\ 0=x_0<x_1<\cdots<x_n=1}} \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})|, \tag{32a}$$

$$\|f\|_p := \begin{cases} \left[\displaystyle\int_0^1 |f(x)|^p \,\mathrm{d}x\right]^{1/p}, & 1 \le p < \infty, \\ \displaystyle\sup_{0\le x\le 1} |f'(x)|, & p = \infty, \end{cases} \tag{32b}$$

$$\mathcal{V}^k := \mathcal{V}^k[0,1] = \{f \in C[0,1] : \mathrm{Var}(f^{(k)}) < \infty\}, \tag{32c}$$

$$\mathcal{W}^{k,p} = \mathcal{W}^{k,p}[0,1] = \{f \in C[0,1] : \|f^{(k)}\|_p < \infty\}. \tag{32d}$$

The stronger semi-norm is $|f|_{\mathcal{F}} := \mathrm{Var}(f')$, while the weaker semi-norm is

$$|f|_{\widetilde{\mathcal{F}}} := \|f' - A_2(f)'\|_1 = \|f' - f(1) + f(0)\|_1 = \mathrm{Var}(f - A_2(f)),$$

where that $A_2(f) : x \mapsto f(0)(1-x) + f(1)x$ is the linear interpolant of f using the two endpoints of the integration interval. The reason for defining $|f|_{\widetilde{\mathcal{F}}}$ this way is that $|f|_{\widetilde{\mathcal{F}}}$ vanishes if f is a linear function, and of course linear functions are integrated exactly by the trapezoidal rule. The cone of integrands is defined as

$$\mathcal{C}_\tau = \{f \in \mathcal{V}^1 : \mathrm{Var}(f') \le \tau \|f' - f(1) + f(0)\|_1\}. \tag{33}$$

The non-adaptive building blocks to construct the adaptive integration algorithm are the composite trapezoidal rules based on $n-1$ trapezoids:

$$T_n(f) = \int_0^1 A_n(f)\,\mathrm{d}x = \frac{1}{2n-2}[f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)],$$

with $\mathrm{cost}(T_n) = n$. The algorithm T_n is imbedded in the algorithm T_{2n-1}, which uses $2n-2$ trapezoids. Thus, any trapezoidal rule is embedded in some other trapezoidal rule with cost no more than $r = 2$ times the original one. This is the minimum cost multiple as described just before Algorithm 3.

The algorithm for approximating $\|f' - f(1) + f(0)\|_1$ is the $\widetilde{\mathcal{F}}$-semi-norm of the linear spline, $A_n(f)$:

$$\begin{aligned}\widetilde{F}_n(f) := |A_n(f)|_{\widetilde{\mathcal{F}}} &= \big\|A_n(f)' - A_2(f)'\big\|_1 \\ &= \sum_{i=1}^{n-1} \left| f(x_{i+1}) - f(x_i) - \frac{f(1) - f(0)}{n-1} \right|. \end{aligned} \tag{34}$$

The total variation of the first derivative of the linear spline of f,

$$F_n(f) := \mathrm{Var}(A_n(f)') = (n-1)\sum_{i=1}^{n-2} \big|f(x_i) - 2f(x_{i+1}) + f(x_{i+2})\big|, \tag{35}$$

provides a lower bound on $\mathrm{Var}(f')$, and can be used in the necessary condition that f lies in $\mathcal{C}_\tau$ as described in Remark 4. It follows using the mean value theorem that

$$\begin{aligned} F_n(f) &= (n-1)\sum_{i=1}^{n-1} \big|[f(x_{i+2}) - f(x_{i+1})] - [f(x_{i+1}) - f(x_i)]\big| \\ &= \sum_{i=1}^{n-1} |f'(\xi_{i+1}) - f'(\xi_i)| \le \mathrm{Var}(f'), \end{aligned}$$

where ξ_i is some point in $[x_i, x_{i+1}]$.

5.1. Adaptive Algorithm and Upper Bound on the Cost

Constructing the adaptive algorithm for integration requires upper bounds on the errors of the T_n and $\widetilde{F}_n$. Note that $\widetilde{F}_n(f)$ never overestimates $|f|_{\widetilde{\mathcal{F}}}$

because

$$|f|_{\widetilde{\mathcal{F}}} = \left\|f' - A_2(f)'\right\|_1 = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} |f'(x) - A_2(f)'(x)| \, \mathrm{d}x$$
$$\geq \sum_{i=1}^{n-1} \left| \int_{x_i}^{x_{i+1}} [f'(x) - A_2(f)'(x)] \, \mathrm{d}x \right| = \|A_n(f)' - A_2(f)'\|_1 = \widetilde{F}_n(f).$$

Thus, $h_-(n) = 0$ and $\mathfrak{c}_n = \tilde{\mathfrak{c}}_n = 1$.

To find an upper bound on $|f|_{\widetilde{\mathcal{F}}} - \widetilde{F}_n(f)$, note that

$$|f|_{\widetilde{\mathcal{F}}} - \widetilde{F}_n(f) = |f|_{\widetilde{\mathcal{F}}} - \left|A_n(f)\right|_{\widetilde{\mathcal{F}}} \leq \left|f - A_n(f)\right|_{\widetilde{\mathcal{F}}} = \left\|f' - A_n(f)'\right\|_1,$$

since $(f - A_n(f))(x)$ vanishes for $x = 0, 1$. Moreover,

$$\left\|f' - A_n(f)'\right\|_1 = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} |f'(x) - (n-1)[f(x_{i+1}) - f(x_i)]| \, \mathrm{d}x. \tag{36}$$

Now we bound each integral in the summation. For $i = 1, \ldots, n-1$, let $\eta_i(x) = f'(x) - (n-1)[f(x_{i+1}) - f(x_i)]$, and let p_i denote the probability that $\eta_i(x)$ is non-negative:

$$p_i = (n-1) \int_{x_i}^{x_{i+1}} \mathbb{1}_{[0,\infty)}(\eta_i(x)) \, \mathrm{d}x,$$

and so $1-p_i$ is the probability that $\eta_i(x)$ is negative. Since $\int_{x_i}^{x_{i+1}} \eta_i(x) \, \mathrm{d}x = 0$, we know that η_i must take on both non-positive and non-negative values. Invoking the mean value theorem, it follows that

$$\frac{p_i}{n-1} \sup_{x_i \leq x \leq x_{i+1}} \eta_i(x) \geq \int_{x_i}^{x_{i+1}} \max(\eta_i(x), 0) \, \mathrm{d}x$$
$$= \int_{x_i}^{x_{i+1}} \max(-\eta_i(x), 0) \, \mathrm{d}x \leq \frac{-(1-p_i)}{n-1} \inf_{x_i \leq x \leq x_{i+1}} \eta_i(x).$$

These bounds allow us to derive bounds on the integrals in (36):

$$\int_{x_i}^{x_{i+1}} |\eta_i(x)| \, \mathrm{d}x = \int_{x_i}^{x_{i+1}} \max(\eta_i(x), 0) \, \mathrm{d}x + \int_{x_i}^{x_{i+1}} \max(-\eta_i(x), 0) \, \mathrm{d}x$$
$$= 2(1-p_i) \int_{x_i}^{x_{i+1}} \max(\eta_i(x), 0) \, \mathrm{d}x$$
$$+ 2p_i \int_{x_i}^{x_{i+1}} \max(-\eta_i(x), 0) \, \mathrm{d}x$$
$$\leq \frac{2p_i(1-p_i)}{n-1} \left[\sup_{x_i \leq x \leq x_{i+1}} \eta_i(x) - \inf_{x_i \leq x \leq x_{i+1}} \eta_i(x) \right]$$
$$\leq \frac{1}{2(n-1)} \left[\sup_{x_i \leq x \leq x_{i+1}} f'(x) - \inf_{x_i \leq x \leq x_{i+1}} f'(x) \right],$$

since $p_i(1-p_i) \le 1/4$.

Plugging this bound into (36) yields

$$\begin{aligned}\left\|f' - f(1) + f(0)\right\|_1 - \widetilde{F}_n(f) &= |f|_{\widetilde{\mathcal{F}}} - \widetilde{F}_n(f)\\ &\le \left\|f' - A_n(f)'\right\|_1\\ &\le \frac{1}{2n-2}\sum_{i=1}^{n-1}\left[\sup_{x_i \le x \le x_{i+1}} f'(x) - \inf_{x_i \le x \le x_{i+1}} f'(x)\right]\\ &\le \frac{\operatorname{Var}(f')}{2n-2} = \frac{|f|_{\mathcal{F}}}{2n-2},\end{aligned}$$

and so

$$h_+(n) = \frac{1}{2n-2}, \qquad \mathfrak{C}_n = \frac{1}{1-\tau/(2n-2)} \qquad \text{for } n > 1 + \tau/2. \tag{37}$$

Since $\widetilde{F}_2(f) = 0$ by definition, the above inequality for $|f|_{\widetilde{\mathcal{F}}} - \widetilde{F}_2(f)$ implies that

$$2\left\|f' - f(1) + f(0)\right\|_1 = 2\,|f|_{\widetilde{\mathcal{F}}} \le |f|_{\mathcal{F}} = \operatorname{Var}(f'), \qquad \tau_{\min} = 2. \tag{38}$$

The errors of the trapezoidal rule for integrands in the spaces $\mathcal{V}^1$ and $\mathcal{W}^{1,1}$ are given in [1, (7.14) and (7.15)]:

$$\left|\int_0^1 f(x)\,dx - T_n(f)\right| \le \begin{cases} h(n)\operatorname{Var}(f') \\ \tilde{h}(n)\left\|f' - f(1) + f(0)\right\|_1 \end{cases} \tag{39a}$$

$$h(n) = \frac{1}{8(n-1)^2}, \qquad h^{-1}(\varepsilon) = \left\lceil \sqrt{\frac{1}{8\varepsilon}} \right\rceil + 1, \qquad \tilde{h}(n) = \frac{1}{2(n-1)}. \tag{39b}$$

By condition (22), it follows that the error bound on the trapezoidal rule is of no use for the adaptive algorithm as noted in Remark 5.

Given the above definitions of $h, \mathfrak{C}_n, \mathfrak{c}_n$, and $\tilde{\mathfrak{c}}_n$, it is now possible to also specify

$$h_1(n) = h_2(n) = \mathfrak{C}_n h(n) = \frac{1}{4(n-1)(2n-2-\tau)}, \tag{40a}$$

$$h_1^{-1}(\varepsilon) = h_2^{-1}(\varepsilon) = 1 + \left\lceil \sqrt{\frac{\tau}{8\varepsilon} + \frac{\tau^2}{16}} + \frac{\tau}{4} \right\rceil \le 2 + \frac{\tau}{2} + \sqrt{\frac{\tau}{8\varepsilon}}. \tag{40b}$$

Moreover, the left side of (23), the stoping criterion inequality in the multistage algorithm, becomes

$$\tau h(n_i)\mathfrak{C}_{n_i}\widetilde{F}_{n_i}(f) = \frac{\tau \widetilde{F}_{n_i}(f)}{4(n_i-1)(2n_i-2-\tau)}. \tag{40c}$$

With these preliminaries, Algorithm 3 and Theorem 3 may be applied directly to yield the following automatic, adaptive integration algorithm and its guarantee.

Algorithm 4 (Univariate Integration)**.** Let the error tolerance ε, the maximum cost budget $N_{\max}$ and the cone constant $\tau > 2$ be given inputs. Let the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}, \{\widetilde{F}_n\}_{n\in\mathcal{I}}$, and $\{F_n\}_{n\in\mathcal{I}}$ be described above. Set $i = 1$. Let $n_1 = \lceil(\tau+1)/2\rceil + 1$. For any input function $f \in \mathcal{W}^{2,1}$, do the following:

Stage 1. Estimate $\|f' - f(1) + f(0)\|_1$ and bound $\mathrm{Var}(f')$. Compute $\widetilde{F}_{n_i}(f)$ in (34) and $F_{n_i}(f)$ in (35).

Stage 2. Check the necessary condition for $f \in \mathcal{C}_\tau$. Compute

$$\tau_{\min,n_i} = \frac{F_{n_i}(f)}{\widetilde{F}_{n_i}(f) + F_{n_i}(f)/(2n_i - 2)}.$$

If $\tau \geq \tau_{\min,n_i}$, then go to stage 3. Otherwise, set $\tau = 2\tau_{\min,n_i}$. If $n_i \geq (\tau+1)/2$, then go to stage 3. Otherwise, choose

$$n_{i+1} = 1 + (n_i - 1)\left\lceil \frac{\tau+1}{2n_i - 2} \right\rceil.$$

Go to Stage 4.

Stage 3. Check for convergence. Check whether n_i is large enough to satisfy the error tolerance, i.e.

$$\widetilde{F}_{n_i}(f) \leq \frac{4\varepsilon(n_i - 1)(2n_i - 2 - \tau)}{\tau}.$$

If this is true, then set W to be false, return $(T_{n_i}(f), W)$ and terminate the algorithm. If this is not true, choose

$$n_{i+1} = 1 + (n_i - 1)\max\left\{2, \left\lceil \frac{1}{(n_i - 1)}\sqrt{\frac{\tau\widetilde{F}_{n_i}(f)}{8\varepsilon}} \right\rceil\right\}.$$

Go to Stage 4.

Stage 4. Check whether n_{i+1} is within budget. If $n_{i+1} \leq N_{\max}$, increment i by 1, and return to Stage 1. Otherwise, if $n_{i+1} > N_{\max}$, choose

$$n_{i+1} = 1 + (n_i - 1)\left\lfloor \frac{N_{\max} - 1}{(n_i - 1)} \right\rfloor,$$

and set W to be true. Return $(T_{n_{i+1}}(f), W)$ and terminate the algorithm.

Theorem 7. *Let (T, W) be the automatic trapezoidal rule defined by Algorithm 4, and let $N_{\max}$, τ, n_1, and ε be the inputs and parameters described there. Assume that $n_1 \leq N_{\max}$. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (33)*. Let*

$$\mathcal{N} := \left\{ f \in \mathcal{C}_\tau : \|f' - f(1) + f(0)\|_1 \leq \frac{2\varepsilon(N_{\max} - 2)(N_{\max} - 2 - \tau)}{\tau} \right\}$$
$$\supset \left\{ f \in \mathcal{C}_\tau : \mathrm{Var}(f') \leq \frac{\varepsilon(N_{\max} - 2)(N_{\max} - 2 - \tau)}{\tau} \right\}$$

be the nice subset of the cone $\mathcal{C}_\tau$. Then it follows that Algorithm 4 is successful for all functions in $\mathcal{N}$, i.e., $\mathrm{succ}(T, W, \mathcal{N}, \varepsilon, N_{\max}) = 1$. *Moreover, the cost of this algorithm is bounded above as follows:*

$$\begin{aligned}
&\max\left(\left\lceil\frac{\tau+1}{2}\right\rceil, \left\lceil\sqrt{\frac{\mathrm{Var}(f')}{8\varepsilon}}\right\rceil\right)+1 \\
&\qquad \max\left(\left\lceil\frac{\tau+1}{2}\right\rceil, \left\lceil\sqrt{\frac{\tau\left\|f'-f(1)+f(0)\right\|_1}{8\varepsilon}}\right\rceil\right)+1 \\
&\qquad\qquad \le \mathrm{cost}(A, f; \varepsilon, N_{\max}) \\
&\qquad \le \sqrt{\frac{\tau\left\|f'-f(1)+f(0)\right\|_1}{2\varepsilon}}+\tau+4 \le \sqrt{\frac{\tau\,\mathrm{Var}(f')}{4\varepsilon}}+\tau+4. \quad (41)
\end{aligned}$$

The algorithm is computationally stable, meaning that the minimum and maximum costs for all integrands, f, with fixed $\|f'-f(1)+f(0)\|_1$ or $\mathrm{Var}(f')$ are an ε-dependent constant of each other.

5.2. Lower Bound on the Computational Cost

Next, we derive a lower bound on the cost of approximating functions in the cone $\mathcal{C}_\tau$ by constructing fooling functions. Following the arguments of Section 4, we choose the triangle shaped function $f_0 : x \mapsto 1 - |1 - 2x|$. Then

$$\begin{aligned}
|f_0|_{\widetilde{\mathcal{F}}} &= \left\|f_0' - f_0(1) + f_0(0)\right\|_1 = \int_0^1 |2\,\mathrm{sign}(1-2x)|\,\mathrm{d}x = 2, \\
|f_0|_{\mathcal{F}} &= \mathrm{Var}(f_0') = 4 = 2\left\|f_0' - f_0(1) + f_0(0)\right\|_1, \qquad \tau_{\min} = 2.
\end{aligned}$$

For any $n \in \mathbb{N}$, suppose that the one has the data $L_i(f) = f(\xi_i)$, $i = 1, \ldots, n$ for arbitrary ξ_i, where $0 = \xi_0 \le \xi_1 < \cdots < \xi_n \le \xi_{n+1} = 1$. There must be some $j = 0, \ldots, n$ such that $\xi_{j+1} - \xi_j \ge 1/(n+1)$. The function f_1 is defined as a triangle function on the interval $[\xi_j, \xi_{j+1}]$:

$$f_1(x) := \begin{cases} \dfrac{2[\xi_{j+1} - \xi_j - |\xi_{j+1} + \xi_j - 2x|]}{(\xi_{j+1} - \xi_j)^2} & \xi_j \le x \le \xi_{j+1}, \\ 0 & \text{otherwise}, \end{cases}$$

which satisfies $\int_0^1 f_1(x)\,\mathrm{d}x = 1$ and $f_1(\xi_i) = 0$ for $i = 0, \ldots, n+1$. Moreover, the total variation of the first derivative of f_1 is bounded by

$$|f_1|_{\mathcal{F}} = \mathrm{Var}(f_1') \le \frac{16}{(\xi_{j+1} - \xi_j)^2},$$

with inequality holding if $0 < j < n$. The inequality $\xi_{j+1} - \xi_j \ge 1/(n+1)$ is used to define g as

$$g(n) := \frac{1}{16(n+1)^2}, \qquad |f_1|_{\mathcal{F}} = \mathrm{Var}(f_1') \le \frac{1}{g(n)}.$$

Using these choices of f_0 and f_1, along with the corresponding $\tilde{g}$ and g above, one may invoke Proposition 5, Theorem 6, and Corollary 1 to obtain the following theorem.

Theorem 8. *Assuming an infinite cost budget and for $\tau > 2$, the complexity of the integration problem, over the cone of functions $\mathcal{C}_\tau$ defined in* (33) *is bounded below as*

$$\begin{aligned}&\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{H}, \mathrm{INT}, \Lambda), \infty, \mathcal{B}_\sigma)\\ &\qquad \geq \operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau \cap \mathcal{B}_\sigma, \mathcal{H}, \mathrm{INT}, \Lambda), \infty, \mathcal{B}_\sigma) \geq \sqrt{\frac{(\tau-2)\sigma}{32\tau\varepsilon}} - 1.\end{aligned}$$

The adaptive trapezoidal Algorithm 4 is optimal for integration of functions in $\mathcal{C}_\tau$ in the sense that the maximum cost of the algorithm is only an ε and σ independent constant multiple of the computational complexity of the problem.

FIX BELOW with new $\widetilde{\mathcal{F}}$- and $\mathcal{F}$-semi-norms, A_n, T_n

5.3. Numerical Example

Consider the family of test functions defined by

$$f(x) = b\mathrm{e}^{-a^2(x-z)^2}, \tag{42}$$

where $z \sim \mathcal{U}[1/\sqrt{2}a, 1 - 1/\sqrt{2}a]$, $\log_{10}(a) \sim \mathcal{U}[1, 4]$, and b is chosen to make $\int_0^1 f(x)\,\mathrm{d}x = 1$. If a is large, then f is spiky, and it is difficult to approximate the integral. From the numerical results, we can find the success rate for our algorithm. Straightforward calculations yield the norms of the first two derivatives of this test function:

$$\begin{aligned}\|f'\|_1 &= 2 - \mathrm{e}^{-a^2z^2} - \mathrm{e}^{-a^2(1-z)^2},\\ \|f''\|_1 &= 2a\left\{2\sqrt{\frac{2}{\mathrm{e}}} - a\left[z\mathrm{e}^{-a^2z^2} + (1-z)\mathrm{e}^{-a^2(1-z)^2}\right]\right\}.\end{aligned}$$

Monte Carlo simulation is used to determine the probability that $f \in \mathcal{C}_\tau$ for $\tau = 10, 100$, and 1000 (see Table 2).

	Algorithm 4					
τ	10	100	1000	`quad`	`quadgk`	`chebfun`
Probability of $f \in \mathcal{C}_\tau$	0%	25%	58%			
Observed Success Rate	37%	69%	97%	30%	77%	68%

Table 2: Performance of multistage, adaptive Algorithm 4 for various values of τ, and also of other automatic quadrature algorithms. The test function (42) with random parameters, a and z, was used.

Ten thousand random instances of this test function are integrated by Algorithm 4 and three existing automatic algorithms with an absolute error tolerance

of $\varepsilon = 10^{-8}$. The observed success rates for these algorithms are shown. As expected, the success rate of Algorithm 4 increases as τ is increased. All of the integrands lying inside $\mathcal{C}_\tau$ are integrated to within the error tolerance, and a significant number of integrands lying outside the cone are integrated accurately as well.

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

Now we consider the problem of $\mathcal{L}_\infty$ recovery of functions defined on $[0,1]$, i.e.,

$$S(f) = \mathrm{APP}(f) := f, \qquad \mathcal{H} = \mathcal{L}_\infty, \qquad \|S(f) - A(f)\|_{\mathcal{H}} = \|f - A(f)\|_\infty .$$

The spaces of functions to be recovered are the Sobolev spaces $\mathcal{G} = \mathcal{W}^{1,\infty}$ and $\mathcal{F} = \mathcal{W}^{2,\infty}$, where $\mathcal{W}^{k,\infty}$ was defined in (32). The cone of functions for which our adaptive algorithms will be shown to work well is defined as

$$\|f\|_{\mathcal{G}} = \|f' - f(1) + f(0)\|_\infty , \qquad |f|_{\mathcal{F}} = \|f''\|_\infty , \tag{43a}$$

$$\mathcal{C}_\tau = \{f \in \mathcal{W}^{2,\infty} : \|f''\|_\infty \le \tau \, \|f' - f(1) + f(0)\|_\infty\}. \tag{43b}$$

Again, we consider algorithms based on function values, and the cost of a function value is one. The basic non-adaptive algorithm used to approximate functions in the cone $\mathcal{C}_\tau$ is the linear spline algorithm given in (31). The cost of A_n, is the number of function evaluations, n. Since A_n, which uses $n-1$ line segments, is embedded in A_{2n-1}, which uses $2(n-1)$ line segments, the minimum cost multiple is $r = 2$.

Using this same data one may approximate the $\mathcal{L}_\infty$ norm of $f' - f(1) + f(0)$ by the algorithm

$$\begin{aligned} G_n(f) &:= \|A_n(f)' - A_2(f)'\|_\infty \\ &= \sup_{i=1,\ldots,n-1} \big|(n-1)[f(x_{i+1}) - f(x_i)] - f(1) + f(0)\big|. \end{aligned} \tag{44}$$

Moreover, a lower bound on $\|f'\|_\infty$ can be derived similarly to the previous section. Specifically,

$$F_n(f) := \frac{1}{2}(n-1)^2 \sup_{i=1,\ldots,n-2} |f(x_i) - 2f(x_{i+1}) + f(x_{i+2})| . \tag{45}$$

It follows using the mean value theorem that

$$\begin{aligned} F_n(f) &= \frac{1}{2}(n-1)^2 \sup_{i=1,\ldots,n-2} \big|[f(x_{i+2}) - f(x_{i+1})] - [f(x_{i+1}) - f(x_i)]\big| \\ &= \frac{1}{2}(n-1) \sup_{i=1,\ldots,n-2} |f'(\xi_{i+1}) - f'(\xi_i)| \le \|f''\|_\infty , \end{aligned}$$

where ξ_i is some point in $[x_i, x_{i+1}]$, and so $\xi_{i+1} - \xi_i \le 2(n-1)^{-1}$.

6.1. Adaptive Algorithm and Upper Bound on the Cost

Given the algorithms G_n and A_n, we now turn to deriving the worst case error bounds, $h_\pm$ defined in (13) and $\tilde{h}$ and h defined in (**??**). Note that $G_n(f)$ never overestimates $\|f\|_{\mathcal{G}}$ because

$$\begin{aligned}
\|f\|_{\mathcal{G}} &= \left\|f' - A_2(f)'\right\|_\infty = \sup_{i=1,\dots,n-1} \sup_{x_i \le x \le x_{i+1}} |f'(x) - A_2(f)'(x)| \,\mathrm{d}x \\
&\ge \sup_{i=1,\dots,n-1} (n-1) \int_{x_i}^{x_{i+1}} |f'(x) - f(1) + f(0)| \,\mathrm{d}x \\
&\ge \sup_{i=1,\dots,n-1} (n-1) \left| \int_{x_i}^{x_{i+1}} [f'(x) - f(1) + f(0)] \,\mathrm{d}x \right| \\
&= \sup_{i=1,\dots,n-1} (n-1) \left| f'(x_{i+1}) - f(x_i) - \frac{f(1) - f(0)}{n-1} \right| = G_n(f).
\end{aligned}$$

Thus, $h_-(n) = 0$ and $\mathfrak{c}_n = \tilde{\mathfrak{c}}_n = 1$.

To find an upper bound on $\|f\|_{\mathcal{G}} - G_n(f)$, note that

$$\|f\|_{\mathcal{G}} - G_n(f) = \|f\|_{\mathcal{G}} - \left|A_n(f)\right|_{\mathcal{G}} \le \left|f - A_n(f)\right|_{\mathcal{G}} = \left\|f' - A_n(f)'\right\|_\infty,$$

since $(f - A_n(f))(x)$ vanishes for $x = 0, 1$. The next step is to bound the difference between

Moreover,

$$\begin{aligned}
&\left\|f' - A_n(f)'\right\|_\infty \\
&\qquad = \sup_{i=1,\dots,n-1} \sup_{x_i \le x \le x_{i+1}} |f'(x) - (n-1)[f(x_{i+1}) - f(x_i)]| \,\mathrm{d}x. \quad (46)
\end{aligned}$$

Using integration by parts, one can write the difference between f and its linear spline in terms of an integral kernel. For $x \in [x_i, x_{i+1}]$,

$$\begin{aligned}
f(x) - A_n(f)(x) &= f(x) - (n-1)\left[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)\right] \\
&= (n-1) \int_{x_i}^{x_{i+1}} v_i(t,x) f'(t) \,\mathrm{d}t \quad (47) \\
&= (n-1) \int_{x_i}^{x_{i+1}} u_i(t,x) f''(t) \,\mathrm{d}t, \quad (48) \\
f'(x) - A_n(f)'(x) &= (n-1) \int_{x_i}^{x_{i+1}} w_i(t,x) f''(t) \,\mathrm{d}t, \quad (49)
\end{aligned}$$

where the kernels are defined as

$$\begin{aligned}
u_i(t,x) &= \begin{cases} (x_{i+1} - x)(x_i - t), & x_i \le t \le x, \\ (x - x_i)(t - x_{i+1}), & x < t \le x_{i+1}, \end{cases} \\
v_i(t,x) &= -\frac{\partial u_i}{\partial t}(t,x) = \begin{cases} x_{i+1} - x, & x_i \le t \le x, \\ x_i - x, & x < t \le x_{i+1}, \end{cases} \\
w_i(t,x) &= \frac{\partial u_i}{\partial x}(t,x) = \begin{cases} t - x_i, & x_i \le t \le x, \\ t - x_{i+1}, & x < t \le x_{i+1}. \end{cases}
\end{aligned}$$

This implies the following upper bound on a piece of $\|f'\|_1$:

$$\begin{aligned}
&\int_{x_i}^{x_{i+1}} |f'(x) - A_n(f)'(x)| \,\mathrm{d}x \\
&\quad \le (n-1) \int_{x_i}^{x_{i+1}} \int_{x_i}^{x_{i+1}} |w(t,x)||f''(t)| \mathrm{d}t \,\mathrm{d}x \\
&\quad \le (n-1) \int_{x_i}^{x_{i+1}} 2(t-x_i)(x_{i+1}-t)|f''(t)| \,\mathrm{d}t \\
&\quad \le (n-1) \max_{x_i \le t \le x_{i+1}} |2(t-x_i)(x_{i+1}-t)| \int_{x_i}^{x_{i+1}} |f''(t)| \,\mathrm{d}t \\
&\quad \le \frac{1}{2(n-1)} \int_{x_i}^{x_{i+1}} |f''(t)| \,\mathrm{d}t.
\end{aligned}$$

Applying this inequality for $i = 1, \ldots, n-1$ leads to

$$\begin{aligned}
\|f\|_{\mathcal{G}} - G_{1,n}(f) &\le \left\|f' - A_n(f)'\right\|_1 \\
&= \sum_{i=1}^{n-1} \left\{ \int_{x_i}^{x_{i+1}} |f'(x) - A_n(f)'(x)| \,\mathrm{d}x \right\} \\
&\le \frac{1}{2(n-1)} \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} |f''(t)| \,\mathrm{d}t = \frac{\|f''\|_1}{2(n-1)}.
\end{aligned}$$

According to (13) and , we have $h_+(n) = 1/[2(n-1)]$ and an inflation factor of

$$\mathfrak{C}_n = \frac{1}{1 - \tau/(2n-2)} \qquad \text{for } n > 1 + \tau/2. \tag{50}$$

For $x_i \le x \le x_{i+1}$ note from (**??**) that the difference between $f'(x)$ and its linear spline approximation is bounded by

$$\begin{aligned}
&|f'(x)| - (n-1)\,|f(x_{i+1}) - f(x_i)| \\
&\quad \le (n-1) \left| \int_{x_i}^{x_{i+1}} v(t,x) f''(t) \,\mathrm{d}t \right| \\
&\quad \le (n-1)\,\|f''\|_\infty \int_{x_i}^{x_{i+1}} |v(t,x)| \,\mathrm{d}t \\
&\quad = (n-1)\,\|f''\|_\infty \left\{ \frac{1}{2(n-1)^2} - (x-x_i)(x_{i+1}-x) \right\} \\
&\quad \le \frac{\|f''\|_\infty}{2(n-1)}.
\end{aligned}$$

Applying the above argument for $i = 1, \ldots, n-1$ and noting that $G_n(f)$ never overestimates $\|f'\|_\infty$, we have the following two-sided inequality:

$$0 \le \|f'\|_\infty - G_n(f) \le \frac{\|f''\|_\infty}{2(n-1)}.$$

Thus, the error bounds on G_n and the inflation and deflation factors defined in (15) and (16) may be taken to be

$$h_-(n) = 0, \quad h_+(n) = \frac{1}{2(n-1)}, \qquad \mathfrak{c}_n = 1, \quad \mathfrak{C}_n = \frac{1}{1 - \tau/(2n-2)}, \tag{51}$$

provided that $n > 1 + \tau/2$.

To derive the error bounds for $A_n(f)$, note that for $x_i \le x \le x_{i+1}$ the error can be bounded in terms of an integral involving f', or, using integration by parts, an integral involving f'':

$$\begin{aligned}
&|f(x) - A_n(f)(x)| \\
&\quad = |f(x) - (n-1)\left[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)\right]| \\
&\quad = (n-1)\left|(x_{i+1} - x)\int_{x_i}^{x} f'(t)\,\mathrm{d}t - (x - x_i)\int_{x}^{x_{i+1}} f'(t)\,\mathrm{d}t\right| \\
&\quad = (n-1)\left|(x_{i+1} - x)\int_{x_i}^{x} f''(t)(t - x_i)\,\mathrm{d}t \right. \\
&\qquad\qquad \left. + (x - x_i)\int_{x}^{x_{i+1}} f''(t)(x_{i+1} - t)\,\mathrm{d}t\right|.
\end{aligned}$$

The expression involving f' yields the bound

$$\begin{aligned}
|f(x) - A_n(f)(x)| &\le 2(n-1)(x - x_i)(x_{i+1} - x)\,\|f'\|_\infty \\
&\le \frac{\|f'\|_\infty}{2(n-1)},
\end{aligned}$$

while the expression involving f'' yields the bound

$$\begin{aligned}
|f(x) - A_n(f)(x)| &\le (n-1)\left[(x_{i+1} - x)\frac{(x - x_i)^2}{2} + (x - x_i)\frac{(x_{i+1} - x)^2}{2}\right]\|f''\|_\infty \\
&= \frac{(x - x_i)(x_{i+1} - x)}{2}\,\|f''\|_\infty \le \frac{\|f''\|_\infty}{8(n-1)^2}.
\end{aligned}$$

This implies that we may take

$$\tilde{h}(n) = \frac{1}{2(n-1)}, \qquad h(n) = \frac{1}{8(n-1)^2} \le \frac{1}{4(n-1)^2} = \tilde{h}(n)h_+(n). \tag{52}$$

Since $h_\pm(n), \tilde{h}(n)$, and $h(n)$, are the same as in the previous section for integration, the simplifications in 40 apply here as well. Then Algorithm 3 and Theorem 3 may be applied directly to yield the following algorithm for function approximation and its guarantee.

Algorithm 5. Let the error tolerance ε, the maximum cost budget $N_{\max}$, and the cone constant τ be given inputs. Let the sequences of algorithms, $\{A_n\}_{n\in\mathcal{I}}$ and $\{G_n\}_{n\in\mathcal{I}}$ be as described above. Set $i = 1$. Let $n_1 = \lceil(\tau+1)/2\rceil + 1$. For any input function $f \in \mathcal{F}$, do the following:

Stage 1. Estimate $\|f'\|_\infty$. Compute $G_{n_i}(f)$ in (44).

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

$$G_{n_i}(f) \le \frac{4\varepsilon(n_i-1)(2n_i-2-\tau)}{\tau}.$$

If this is true, then set W to be false, return $(A_{n_i}(f), W)$ and terminate the algorithm. If this is not true, go to Stage 3.

Stage 3. Compute n_{i+1}. Choose

$$n_{i+1} = 1 + (n_i - 1)\max\left\{2, \left\lceil \frac{1}{(n_i-1)}\sqrt{\frac{\tau G_{n_i}(f)}{8\varepsilon}} \right\rceil\right\}.$$

If $n_{i+1} \le N_{\max}$, increment i by 1, and return to Stage 1.

Otherwise, if $n_{i+1} > N_{\max}$, choose n_{i+1} to be the largest number not exceeding $N_{\max}$ such that A_{n_i} is embedded in $A_{n_{i+1}}$, and set W to be true. Return $(A_{n_{i+1}}(f), W)$ and terminate the algorithm.

Theorem 9. *Let ε, $N_{\max}$, τ and n_1 be given as described in Algorithm 5. Assume that $n_1 \le N_{\max}$. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (**??**). *Let*

$$\mathcal{N} = \left\{ f \in \mathcal{C}_\tau : \|f'\|_\infty \le \frac{2\varepsilon(N_{\max}-2)(N_{\max}-2-\tau)}{\tau} \right\}$$

be the nice subset of the cone $\mathcal{C}_\tau$. Then it follows that Algorithm 5 is successful for all functions in $\mathcal{N}$, i.e., $\mathrm{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = 1$. *Moreover, the cost of this algorithm is bounded above as follows:*

$$\begin{aligned}\mathrm{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \|f'\|_\infty) &\le 2 + 2\left\lceil \sqrt{\frac{\tau\|f'\|_\infty}{8\varepsilon} + \frac{\tau^2}{16}} + \frac{\tau}{4} \right\rceil \\ &\le 2 + 2\left\lceil \sqrt{\frac{\tau\|f'\|_\infty}{8\varepsilon}} + \frac{\tau}{2} \right\rceil.\end{aligned}$$

6.2. Lower Bound on the Computational Cost

Next, we derive a lower bound on the cost of approximating functions in the cone $\mathcal{C}_\tau$ by constructing fooling functions. Following the arguments of Section 4, we choose $f_0(x) = x$. Then

$$\|f_0'\|_\infty = 1, \qquad \|f_0''\|_\infty = 0, \qquad \tau_0 = 0.$$

For any $n \in \mathbb{N}$, suppose that the one has the data $L_i(f) = f(\xi_i)$, $i = 1, \ldots, n$ for arbitrary ξ_i, where $0 = \xi_0 \le \xi_1 < \cdots < \xi_n \le \xi_{n+1} = 1$. There must be some $j = 0, \ldots, n$ such that $\xi_{j+1} - \xi_j \ge 1/(n+1)$. The function f_1, defined as

$$f_1(x) := \begin{cases} \dfrac{16(x-\xi_j)^2(\xi_{j+1}-x)^2}{(\xi_{j+1}-\xi_j)^4} & \xi_j \le x \le \xi_{j+1}, \\ 0 & \text{otherwise,} \end{cases}$$

has $\|f_1\|_\infty = 1$ and $f_1(\xi_i) = 0$ for $i = 0, \ldots, n+1$. Moreover, the norms of the first and second derivatives of f_1 are

$$\|f_1'\|_\infty = \frac{16}{3\sqrt{3}(\xi_{j+1} - \xi_j)} \le \tilde{g}(n),$$
$$\|f_1''\|_\infty = \frac{32}{(\xi_{j+1} - \xi_j)^2} = \frac{6\sqrt{3}\,\|f_1'\|_\infty}{(\xi_{j+1} - \xi_j)} \le g(n)\,\|f_1'\|_\infty ,$$

where the inequality $\xi_{j+1} - \xi_j \ge 1/(n+1)$ is used to define $\tilde{g}$ and g as

$$\tilde{g}(n) = \frac{16(n+1)}{3\sqrt{3}}, \qquad g(n) = 6\sqrt{3}(n+1), \qquad (g\tilde{g})(n) = 32(n+1)^2.$$

Using these choices of f_0 and f_1, along with the corresponding $\tilde{g}$ and g above, one may invoke Proposition 5, Theorem 6, and Corollary 1 to obtain the following theorem.

Theorem 10. *The linear spline algorithm is optimal for $\mathcal{L}_\infty$ approximation of functions in $\mathcal{W}^{1,\infty}$ and $\mathcal{W}^{2,\infty}$. Assuming an infinite cost budget, the complexity of the function recovery problem problem, over the cone of functions $\mathcal{C}_\tau$ defined in (??) is*

$$\mathrm{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{H}, S, \Lambda), \infty, \|f'\|_\infty) \ge \min\left(\frac{3\sqrt{3}\,\|f'\|_\infty}{64\varepsilon}, \sqrt{\frac{\tau\,\|f'\|_\infty}{128\varepsilon}}\right) - 1$$
$$\sim \sqrt{\frac{\tau\,\|f'\|_\infty}{128\varepsilon}} \quad \textit{as } \frac{\|f'\|_\infty}{\varepsilon} \to \infty.$$

Moreover, Algorithm 5 is optimal for approximating functions in $\mathcal{C}_\tau$.

6.3. Numerical Example

To illustrate Algorithm 5 we choose the same family of test functions as in (42), but now with $z \sim \mathcal{U}[0,1]$, $\log_{10}(a) \sim \mathcal{U}[0,4]$, and $b = 1$. When a is large this function is very spiky and hard to approximate since the sampled function values may miss the spike. Since $\|f'\|_\infty = a\sqrt{2/\mathrm{e}}$ and $\|f''\|_\infty = 2a^2$, the probability that $f \in \mathcal{C}_\tau$ is $\min\left(\max(0, \log_{10}\left(\tau/\sqrt{2\mathrm{e}}\right)/4), 1\right)$. For $\tau = 10, 100, 1000$ we choose a sample of 10000 functions and an error tolerance of $\varepsilon = 10^{-8}$. The empirical probability that Algorithm 5 succeeds in returning an answer within the error tolerance is given in Table 3. Our algorithm succeeds for all functions lying in the cone plus some others. It is conservative, but not overly so.

7. Addressing Questions and Concerns About Automatic Algorithms

Automatic algorithms are popular, especially for univariate integration problems. As mentioned in the introduction, MATLAB [5, 17] and the NAG [19] library all have one or more automatic integration routines. In spite of this popularity certain questions, concerns, or even objections have been raised about automatic algorithms. This section attempts to address them.

τ	10	100	1000
Probability of $f \in \mathcal{C}_\tau$	16%	41%	66%
Observed Success Rate	25%	49%	74%

Table 3: Comparison between the probability of the input function lying in the cone and the empirical success rate of Algorithm 5.

7.1. Automatic Algorithms Can Be Fooled

In claiming to construct guaranteed automatic algorithms, it must be understood that the guarantees still require certain assumptions on the input functions. Any algorithm that solves a problem for involving an infinite-dimensional space of input functions can be fooled by a spiky function, i.e., one that yields zero data where probed by the algorithm, but is nonzero elsewhere. A guaranteed automatic algorithm specifies sufficient conditions for success, and functions that are too spiky violate these conditions. Thus, the aim is not to have an algorithm that works for all functions, but to know for which functions it is guaranteed to work, and preferably to be able to adjust the algorithm to broaden or narrow that set depending on the relevant application and computational cost budget.

7.2. Why Cones?

Traditional error estimates are derived for balls of input functions, $\mathcal{B}_\sigma$, as defined in (2) with radius σ. The analysis here uses cones, $\mathcal{C}_\tau$, of the form (4), instead. Automatic algorithms proceed by bounding, perhaps conservatively, the approximation error and then increasing the number of function data until that error bound is no larger than the prescribed tolerance. These error bounds are constructed in such a way that if they are reliable for f, then they are also reliable for af, where a is any real number. Thus, if an algorithm is successful for the input function f, then it is also successful for any input of the form af. By definition a cone is just that set that contains af for all numbers a if it contains f.

One might wonder whether the definition of $\mathcal{C}_\tau$ in (4) is too strict. Ignoring the cost budget, an alternative would be to define the cone of success, $\mathcal{C}_{\text{success}}$, consisting of all input functions for which the automatic algorithm successfully approximates the solution within the error tolerance. Clearly $\mathcal{C}_{\text{success}}$ contains $\mathcal{C}_\tau$, and likely $\mathcal{C}_{\text{success}}$ also contains many functions not in $\mathcal{C}_\tau$. However, the definition of $\mathcal{C}_{\text{success}}$ does not provide any insight into what kinds of input functions the automatic algorithm can successfully handle. Moreover, the optimality results that one can often prove, including those in Sections 5 and 6, show that the cost of the automatic algorithm is within a constant multiple of the cost of the best algorithm for $\mathcal{C}_\tau$.

The automatic algorithms described here require that the width of the cone, τ, be specified. This requires the user's judgement, and its value cannot be determined from the function data. The choice of τ reflects how cautious the

user wants to be, since a larger τ leads to an adaptive algorithm with a higher cost.

The condition defining a ball $\mathcal{B}_\sigma$ involves only one semi-norm, whereas the condition defining the cone $\mathcal{C}_\tau$ involves two semi-norms. One may wonder whether this makes specifying τ more difficult than choosing σ. As mentioned in the introduction, automatic algorithms based on balls are not adaptive and have fixed cost. There is no savings in computational cost when the actual semi-norm of the input function is much smaller than the radius of the ball. For automatic algorithms designed for balls of input functions, as diagramed in Figure **??**, σ is a measure of the largest input function that the algorithm is designed to handle. This size is not necessarily something that is easy to bound a priori. On the other hand, for automatic algorithms designed for cones of input functions, as diagramed in Figure **??**, the parameter τ is a measure of the nastiest input function that the algorithm is designed to handle. For the examples of univariate integration in Section 5 and univariate function recovery in Section 6, τ may be interpreted as a minimum sample size, and $1/\tau$ may be interpreted the length scale of the spikiest function that the algorithm can handle. These interpretations facilitate the choice of τ.

7.3. Automatic Algorithms that Stop When $A_{n_i}(f) - A_{n_{i-1}}(f)$ Is Small

Many automatic algorithms, especially those for univariate integration, are based on a sequence of non-adaptive algorithms $\{A_{n_i}\}_{i=1}^{\infty}$, and a stopping rule that returns $A_{n_i}(f)$ as the answer for the first i where $A_{n_i}(f) - A_{n_{i-1}}(f)$ is small enough. Fundamental texts in numerical algorithms advocate such stopping rules, e.g. [2, p. 223–224], [3, p. 233], and [16, p. 270]. Unfortunately, such stopping rules are problematic.

For instance, consider the univariate integration problem and the trapezoidal rule algorithm, A_{n_i}, based on $n_i = 2^i + 1$ points, i.e., $n_i - 1 = 2^i$ trapezoids. It is taught that the trapezoidal rule has the following error estimate:

$$\widehat{E}_i(f) := \frac{A_{n_i}(f) - A_{n_{i-1}}(f)}{3} \approx \int_0^1 f(x)\,\mathrm{d}x - A_{n_i}(f) =: E_i(f).$$

Since $A_{n_i}(f) + \widehat{E}_i(f)$ is exactly Simpson's rule, the quality of $\widehat{E}_i(f)$ as an error estimate is equivalent to the quality of Simpson's rule. Since $E_i(f) - \widehat{E}_i(f) = \Theta(16^{-i} \left\|f^{(4)}\right\|_1)$, the error estimate will often be good even for moderate i, but it can only be guaranteed with some a priori knowledge of $\left\|f^{(4)}\right\|_1$.

In his provocatively titled SIAM Review article, *When Not to Use an Automatic Quadrature Routine* [9, p. 69], James Lyness makes the following claim.

> While prepared to take the risk of being misled by chance alignment of zeros in the integrand function, or by narrow peaks which are "missed," the user may wish to be reassured that for "reasonable" integrand functions which do not have these characteristics all will be well. It is the purpose of the rest of this section to demonstrate by example that he cannot be reassured on this point. In fact the

> routine is likely to be unreliable in a significant proportion of the problems it faces (say 1 to 5%) and there is no way of predicting in a straightforward way in which of any set of apparently reasonable problems this will happen.

The following is a summary of Lyness's argument using the notation of the present article. To fool an automatic algorithm one constructs an integrand f_λ, which is parameterized by λ, such that

- f_λ is "reasonable" for all $\lambda \in [0,1]$,
- $A_n(f_\lambda)$ is continuous in λ, and
- for some moderate n_i, $A_{n_i}(f_\lambda) - A_{n_{i-1}}(f_\lambda)$ has different signs for $\lambda = 0, 1$.

It then follows that $A_{n_i}(f_{\lambda_*}) - A_{n_{i-1}}(f_{\lambda_*}) = 0$ for some $\lambda_* \in [0,1]$. Then this algorithm will likely fail for integrands f_λ with λ nearby λ_* since the stopping criterion is satisfied, but n_i is not large enough to satisfy the desired error tolerance.

Lyness's argument, with its pessimistic conclusion, is correct for algorithms that use the size of $A_{n_i}(f) - A_{n_{i-1}}(f)$ as a stopping criterion. Figure 1 depicts an integrand constructed in a manner similarly to that described by Lyness. We would call this integrand "fluky". MATLAB's `quad`, which is based on adaptive Simpson's rule [4], gives the answer ≈ 0.1733, for an absolute error tolerance of $\varepsilon = 10^{-14}$, but the true answer is ≈ 0.1925. The `quad` routine splits the interval of integration into three separate intervals and initially calculates Simpson's rule with one and two parabolas for each of the three intervals. The data taken are denoted by $\bullet$ in Figure 1. Since this fluky integrand is designed so that the two Simpson's rules match exactly for each of the three intervals, `quad` is fooled into thinking that it knows the correct value of the integral and terminates immediately.

While Lyness's argument is correct, it does not apply to the algorithms proposed in this article because their stopping criteria are not based on $A_{n_i}(f) - A_{n_{i-1}}(f)$. Algorithm 4 presented in Section 5, always succeeds for "reasonable" integrands because such integrands lie in the cone $\mathcal{C}_\tau$. The analogous statement is true for function approximation Algorithm 5, as well as for the general Algorithms 2 and 3. All of these algorithms succeed for reasonable input functions.

Lyness's warning in [9] should not be interpreted as an objection to automatic algorithms. It is a valid objection to stopping criteria that are based on the value of a single functional, in this case, $A_{n_i} - A_{n_{i-1}}$. What Lyness clearly demonstrates is that one may easily make a functional vanish even though the error of the algorithm is significant.

7.4. No Advantage in Adaption

Although there are some positive results on adaption in [12, 13, 15], there are also sweeping, rigorous results from information based complexity theory stating that adaptive algorithms have no significant advantage over non-adaptive

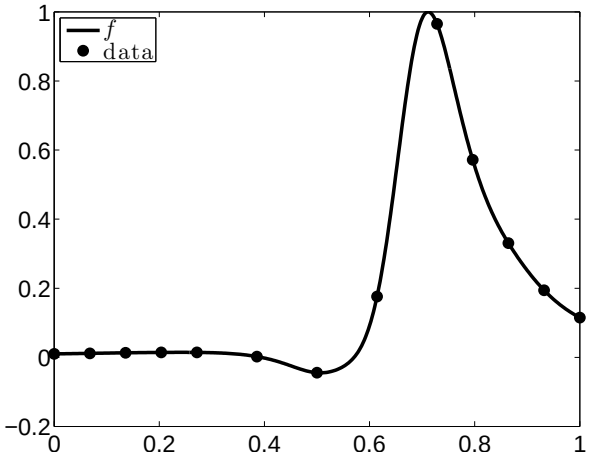

Figure 1: Integrand designed to fool MATLAB's `quad` along with the data used by `quad`.

algorithms (e.g., see [20, Chapter 4, Theorem 5.2.1] and [12]). The automatic algorithms presented here do adaptively determine the total number of samples required based on the function data collected. The reason that adaption can help in this context is that the cone of input functions, $\mathcal{C}_\tau$, is not a convex set. This violates one of the assumptions required to prove the negative results that adaption does not help.

To see why $\mathcal{C}_\tau$ is not convex, let f_{in} and f_{out} be functions in $\mathcal{F}$ with nonzero $\widetilde{\mathcal{F}}$-semi-norms, where f_{in} lies in the interior of this cone, and f_{out} lies outside the cone. This means that

$$\frac{|f_{\text{in}}|_{\mathcal{F}}}{|f_{\text{in}}|_{\widetilde{\mathcal{F}}}} = \tau_{\text{in}} < \tau < \tau_{\text{out}} = \frac{|f_{\text{out}}|_{\mathcal{F}}}{|f_{\text{out}}|_{\widetilde{\mathcal{F}}}}.$$

Next define two functions in terms of f_{in} and f_{out} as follows:

$$f_\pm = (\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,f_{\text{out}} \pm (\tau + \tau_{\text{out}})\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}}\,f_{\text{in}},$$

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

$$\begin{aligned}
\frac{|f_\pm|_{\mathcal{F}}}{|f_\pm|_{\widetilde{\mathcal{F}}}} &= \frac{\left\|(\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,f_{\text{out}} \pm (\tau + \tau_{\text{out}})\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}}\,f_{\text{in}}\right\|_{\mathcal{F}}}{\left\|(\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,f_{\text{out}} \pm (\tau + \tau_{\text{out}})\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}}\,f_{\text{in}}\right\|_{\widetilde{\mathcal{F}}}} \\
&\le \frac{(\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,|f_{\text{out}}|_{\mathcal{F}} + (\tau + \tau_{\text{out}})\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}}\,|f_{\text{in}}|_{\mathcal{F}}}{-(\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}} + (\tau + \tau_{\text{out}})\,|f_{\text{out}}|_{\widetilde{\mathcal{F}}}\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}} \\
&= \frac{(\tau - \tau_{\text{in}})\tau_{\text{out}} + (\tau + \tau_{\text{out}})\tau_{\text{in}}}{-(\tau - \tau_{\text{in}}) + (\tau + \tau_{\text{out}})} = \frac{\tau(\tau_{\text{out}} + \tau_{\text{in}})}{\tau_{\text{out}} + \tau_{\text{in}}} = \tau.
\end{aligned}$$

On the other hand, the average of $f_\pm$, which is also a convex combination is

$$\frac{1}{2}f_- + \frac{1}{2}f_+ = (\tau - \tau_{\text{in}})\,|f_{\text{in}}|_{\widetilde{\mathcal{F}}}\,f_{\text{out}}.$$

Since $\tau > \tau_{\text{in}}$, this is a nonzero multiple of f_{out}, and it lies outside $\mathcal{C}_\tau$. Thus, this cone is not a convex set.

8. Discussion and Further Work

When we consider the landscape of numerical algorithms, it is clear that relatively simple problems have well-developed automatic algorithms that we use without questioning their reliability. For example, the MATLAB [17] algorithms

`cos`, `sin`, `exp`, `log`, `airy`, `besselj`, `gamma`, `gammainc`, `ellipj`, `erf`, and `erfinv`

all automatically use the correct number of terms in a suitable expansion needed to provide the corresponding function value with 15 significant digit accuracy. The only exceptions are cases of unavoidable round-off error, e.g.,

$$\texttt{cos(1e47 * pi)} = \texttt{-5.432862626006405e-01},$$

whereas the correct answer is `1`.

We believe that more complex problems, whose inputs are functions, also deserve to have automatic algorithms with guarantees of their success. Users ought to be able to integrate functions, approximate functions, optimize functions, etc., without needing to manually tune the sample size. Here we have shown how this might be done in general, as well as specifically for two case studies. We hope that this will inspire further research in this direction.

The results presented here suggest a number of other interesting open problems. Here is a summary.

- Here we consider an absolute error tolerance. This analysis should be extended to relative error, which is work in progress.

- The univariate integration and function approximation algorithms in Sections 5 and 6 have low order convergence. Guaranteed automatic algorithms with *higher order convergence rates* for smoother input functions are needed. These might be based on higher degree piecewise polynomial approximations.

- There are other types of problems, e.g., differential equations and nonlinear optimization, which fit the general framework presented here. These problems also have automatic algorithms, but without guarantees. It would be helpful to develop guaranteed automatic algorithms in these other areas.

- The algorithms developed here are *globally adaptive*, in the sense that the function data determines the sample size, but not the kinds of data collected or the locations of the sample points, which depend only on the sample size. Some existing automatic algorithms are *locally adaptive* in that they collect more data in regions of special interest, say where the function has a spike. Such algorithms need guarantees like the ones that

are provided here for globally adaptive algorithms. The appropriate kinds of spaces $\widetilde{\mathcal{F}}$ and $\mathcal{F}$, and their semi-norms, need to be identified for locally adaptive algorithms.

- For some numerical problems the error bound of the non-adaptive algorithm involves a $\widetilde{\mathcal{F}}$- or $\mathcal{F}$-semi-norm that is very hard to approximate because of its complexity. An example is multivariate quadrature using quasi-Monte Carlo algorithms, where the error depends on the *variation* of the integrand. The definition of the variation depends on the definition of $\widetilde{\mathcal{F}}$ or $\mathcal{F}$, but it is essentially some semi-norm of a mixed partial derivative of the integrand. To obtain guaranteed automatic algorithms one must either find an efficient way to approximate the variation of the function or find other suitable conservative estimates for the error that can be reliably obtained from the function data.

- This article considers only the worst case error of deterministic algorithms. There are many random algorithms, and they must be analyzed by somewhat different methods. A guaranteed Monte Carlo algorithm for estimating the mean of a random variable, which includes multivariate integration as a special case, has been proposed in [6].

9. Acknowledgements

The authors are grateful for discussions with a number of colleagues. This research is supported in part by grant NSF-DMS-1115392.

References

[1] H. Brass, K. Petras, Quadrature theory: the theory of numerical integration on a compact interval, American Mathematical Society, Rhode Island, first edition, 2011.

[2] R.L. Burden, J.D. Faires, Numerical Analysis, Cengage Brooks/Cole, Belmont, CA, ninth edition, 2010.

[3] W. Cheney, D. Kincaid, Numerical Mathematics and Computing, Brooks/Cole, Boston, seventh edition, 2013.

[4] W. Gander, W. Gautschi, Adaptive quadrature — revisited, BIT 40 (2000) 84–101.

[5] N. Hale, L.N. Trefethen, T.A. Driscoll, Chebfun Version 4, 2012.

[6] F.J. Hickernell, L. Jiang, Y. Liu, A.B. Owen, Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling, in: J. Dick, F.Y. Kuo, G.W. Peters, I.H. Sloan (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2012, Springer-Verlag, Berlin, 2014. To appear, arXiv:1208.4318 [math.ST].

[7] F.J. Hickernell, T. Müller-Gronbach, B. Niu, K. Ritter, Multi-level Monte Carlo algorithms for infinite-dimensional integration on $\mathbb{R}^{\mathbb{N}}$, J. Complexity 26 (2010) 229–254.

[8] F.Y. Kuo, I.H. Sloan, G.W. Wasilkowski, H. Woźniakowski, Liberating the dimension, J. Complexity 26 (2010) 422–454.

[9] J.N. Lyness, When not to use an automatic quadrature routine, SIAM Rev. 25 (1983) 63–87.

[10] B. Niu, F.J. Hickernell, Monte Carlo simulation of stochastic integrals when the cost of function evaluation is dimension dependent, in: P. L'Ecuyer, A. Owen (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer-Verlag, Berlin, 2010, pp. 545–560.

[11] B. Niu, F.J. Hickernell, T. Müller-Gronbach, K. Ritter, Deterministic multi-level algorithms for infinite-dimensional integration on $\mathbb{R}^{\mathbb{N}}$, J. Complexity 27 (2011) 331–351.

[12] E. Novak, On the power of adaption, J. Complexity 12 (1996) 199–237.

[13] L. Plaskota, G.W. Wasilkowski, Adaption allows efficient integration of functions with unknown singularities, Numer. Math. (2005) 123–144.

[14] L. Plaskota, G.W. Wasilkowski, Tractability of infinite-dimensional integration in the worst case and randomized settings, J. Complexity 27 (2011) 505–518.

[15] L. Plaskota, G.W. Wasilkowski, Y. Zhao, The power of adaption for approximating functions with singularities, Math. Comput. (2008) 2309–2338.

[16] T. Sauer, Numerical Analysis, Pearson, 2012.

[17] The MathWorks, Inc., MATLAB 7.12, Natick, MA, 2012.

[18] The MathWorks, Inc., MATLAB 8.1, Natick, MA, 2013.

[19] The Numerical Algorithms Group, The NAG Library, Oxford, Mark 23 edition, 2012.

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

[21] J.F. Traub, A.G. Werschulz, Complexity and Information, Cambridge University Press, Cambridge, 1998.

[22] G.W. Wasilkowski, Average case tractability of approximating ∞-variate functions, submitted for publication, 2012.

[23] G.W. Wasilkowski, H. Woźniakowski, Liberating the dimension for function approximation, J. Complexity 27 (2011) 86–110.

[24] G.W. Wasilkowski, H. Woźniakowski, Liberating the dimension for function approximation: Standard information, J. Complexity 27 (2011) 417–440.