The Complexity of Deterministic Guaranteed 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 5 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

One would like to have automatic algorithms for numerical problems that are *guaranteed* to provide answers whose errors do not exceed the user-specified error tolerance. Such algorithms are generally lacking, even for fundamental

problems such as integration and function recovery. This article shows how to construct such algorithms.

Specifically, we want algorithms with the following properties:

Guaranteed — the errors in the answers provided must be no greater than the user-specified *error tolerance*, ε.

Automatic — the computational cost required depends on ε, but *does not require additional user input* beyond a routine for providing function data.

Adaptive — the computational cost depends on the input function. Specifically, this cost depends on some semi-norm of the input function, *not input by the user, but reliably inferred from function data*.

Optimal — the computational cost of the algorithm, as $\varepsilon \to 0$, does not exceed a constant multiple of the computational complexity of the problem (the computational cost of the best possible algorithm). Furthermore, although this new algorithm is not given the semi-norm of the input function, its computational cost does not exceed a constant multiple of the cost of the best algorithm that *is* given the semi-norm of the input function.

Tunable — parameters defining the algorithm can be adjusted to change the algorithm's robustness, the maximum allowable computational cost budget, etc. These parameters are intended to be changed occasionally, as opposed to inputs, which may change each time that the algorithm is called.

The checklist in Figure 1 rates some state-of-the-art algorithms and those proposed here against these five criteria. Nearly all existing algorithms fail to meet one or more of these criteria. Here we present a general framework for constructing algorithms that satisfy all of the above criteria. We also present two concrete examples, namely, univariate integration via the composite trapezoidal rule and univariate function recovery via linear splines.

Automatic quadrature algorithms such as MATLAB's `quad` and `quadgk` [17] and those in the NAG Library [18] work well in practice for many cases. But, these popular algorithms do not have rigorous guarantees of success. The Chebfun toolbox in MATLAB [5] approximates functions by Chebyshev polynomial expansions and uses those expansions to approximate integrals, solve differential equations, etc. Chebfun also works well for many cases, but has no guarantees. All of these algorithms are adaptive. The degree to which they are tunable varies.

Most existing theory for numerical problems of interest starts with a Banach space, $\mathcal{F}$, of input functions defined on some set $\mathcal{X}$, and having a semi-norm, $|\cdot|_{\mathcal{F}}$. The definition of $(\mathcal{F}, |\cdot|_{\mathcal{F}})$ contains assumptions about smoothness, periodicity or other qualities of the input functions. The mathematical problem of interest is defined by a *solution operator* $S : \mathcal{F} \to \mathcal{H}$, where $\mathcal{H}$ is some other Banach space with its norm $\|\cdot\|_{\mathcal{H}}$. For integration, $\mathcal{H} = \mathbb{R}$, and for function approximation, $\mathcal{H}$ is some superset of $\mathcal{F}$, for example, $\mathcal{L}_\infty(\mathcal{X})$. One often finds

	MATLAB, NAG, and Chebfun	Algorithm in Figure 2	Algorithm in Figure 3
Guaranteed		✓	✓
Automatic	✓	✓	✓
Adaptive	✓		✓
Optimal		✓	✓
Tunable	?	✓	✓

Figure 1: Check list of desired features for an algorithm.

a sequence of algorithms, $\{A_n\}_{n\in\mathbb{N}}$, with computational cost n, which provides an approximate solution whose error can be bounded as

$$\|S(f) - A_n(f)\|_{\mathcal{H}} \leq \frac{C_{\text{up}}\,|f|_{\mathcal{F}}}{n^p}, \tag{1}$$

where C_{up} and p are known positive constants independent of n. 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 $|f|_{\mathcal{F}} = 0$. Error bound (1) allows one to compute an approximation that differs from the true solution by no more than ε, as shown in the following algorithm and theorem.

Algorithm 1. (Automatic, Non-Adaptive). Let $\mathcal{F}$ and $\mathcal{H}$ be Banach spaces as described above, let S be the solution operator, and let σ be a fixed positive number. Let $\{A_n\}_{n\in\mathbb{N}}$ be a sequence of algorithms as described above such that error bound (1) holds. Let ε a positive error tolerance and $f \in \mathcal{F}$ be an input function. Compute the computational cost needed to satisfy the error tolerance:

$$n = \text{cost}(A_n) = \left\lceil \left(\frac{C_{\text{up}}\sigma}{\varepsilon} \right)^{1/p} \right\rceil. \tag{2}$$

Then return $A_n(f)$ as the answer at a computational cost of n.

Theorem 1. *Let $\mathcal{F}$, $\mathcal{H}$, σ, and ε be given as described in Algorithm 1, and assume that f lies in the* ball

$$\mathcal{B}_\sigma = \{f \in \mathcal{F} : |f|_{\mathcal{F}} \leq \sigma\}. \tag{3}$$

It then follows that the answer provided by Algorithm 1 satisfies the error tolerance, i.e., $\|S(f) - A_n(f)\|_{\mathcal{H}} \leq \varepsilon$.

Figure 2 provides a diagram of an automatic algorithm based on these ideas. The parameter σ must be fixed initially by the user or algorithm designer and is typically left unchanged each time the algorithm is called. The algorithm then only requires as input the error tolerance and a routine for evaluating the function at any point desired. This algorithm is guaranteed. It is optimal if any algorithm satisfying the error criterion above must have a cost of at least $\mathcal{O}((\sigma/\varepsilon)^{1/p})$. However, this algorithm is not adaptive. The computational cost

does not depend on function data, but is determined purely by the input ε and the parameter σ. In the terminology of information-based complexity theory, the computational cost depends on global information but not local information [20, p. 11–12].

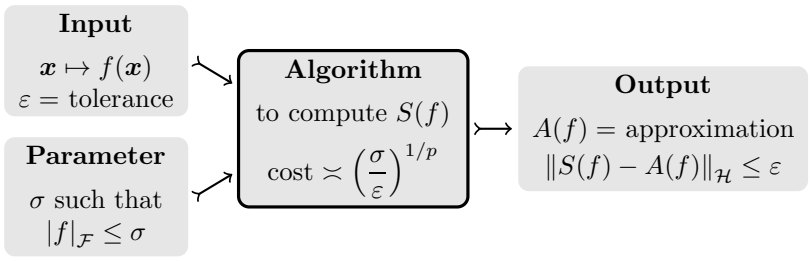

Figure 2: Diagram of the guaranteed, non-adaptive Algorithm 1.

The semi-norm $|f|_{\mathcal{F}}$ may be thought of as the *difficulty of the problem*. In practice, it is difficult to know a priori the maximum possible value of $|f|_{\mathcal{F}}$, so having to determine σ in advance is a practical drawback of the Algorithm 1. Moreover, since the computational cost of this algorithm depends on σ, not $|f|_{\mathcal{F}}$, the cost does not decrease if $|f|_{\mathcal{F}}$ is drastically smaller than σ.

Adaptive algorithms use function data to try to bound $|f|_{\mathcal{F}}$ or to estimate $\|S(f) - A_n(f)\|_{\mathcal{H}}$, so that their computational cost can be adjusted accordingly. Unfortunately, the bounds on $|f|_{\mathcal{F}}$ and the estimates of $\|S(f) - A_n(f)\|_{\mathcal{H}}$ used by adaptive algorithms are typically heuristic or valid only in the limit of $n \to \infty$. These approaches are not guaranteed to provide the answer to within the specified error tolerance for finite n.

Here we show how to construct a *rigorous* upper bound on $|f|_{\mathcal{F}}$ from data, which then leads to a guaranteed automatic and adaptive algorithm. The key idea is to consider functions lying in a *cone* instead of 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 $\mathcal{G}$ be some superspace of $\mathcal{F}$ with a weaker semi-norm, and define the cone

$$\mathcal{C}_\tau = \{f \in \mathcal{F} : |f|_{\mathcal{F}} \leq \tau\, |f|_{\mathcal{G}}\}. \tag{4}$$

A function inside the cone may have a large semi-norm, but its $\mathcal{G}$-semi-norm can be estimated with reasonable effort. Our approach to obtaining a reliable approximation to the true solution is sketched here:

- Since the $\mathcal{G}$-semi-norm is weaker than the $\mathcal{F}$-semi-norm, it can be approximated by some algorithm $G_n(f)$, with $|f|_{\mathcal{G}} - G(f) \leq h_+(n)\, |f|_{\mathcal{F}}$ for some known $h_+(n)$ that vanishes as $n \to \infty$. One chooses a sample size n_G such that $h_+(n_G) \leq 1/\tau$.
- The definition of the cone in (4) implies that

  $$|f|_{\mathcal{G}} \leq G_{n_G}(f) + h_+(n_G)\, |f|_{\mathcal{F}} \leq G_{n_G}(f) + \tau h_+(n_G)\, |f|_{\mathcal{G}}\,,$$

  and so $|f|_{\mathcal{G}} \leq G_{n_G}(f)/(1 - \tau h_+(n_G))$.

- Applying the cone condition again leads to an upper bound on the $\mathcal{F}$-semi-norm, namely, $|f|_{\mathcal{F}} \leq \tau G_{n_G}(f)/(1-\tau h_+(n_G))$.
- This data-driven upper bound on $|f|_{\mathcal{F}}$ can be used with (2) to guarantee that the error tolerance will be met using A_n with

$$n = \text{cost}(A_n) \leq \left\lceil \left(\frac{C_{\text{up}} \tau G_{n_G}(f)}{\varepsilon(1-\tau h_+(n_G))} \right)^{1/p} \right\rceil .$$

- If, as often is the case, $G_{n_G}(f)$ does not overestimate the $\mathcal{G}$-semi-norm of f. Moreover, since $|\cdot|_{\mathcal{G}}$ is a weaker semi-norm, it follows that $\tau_{\min} |f|_{\mathcal{G}} \leq |f|_{\mathcal{F}}$. Thus, one can be assured that the above criterion will be met for

$$n = \text{cost}(A_n) \leq \left\lceil \left(\frac{C_{\text{up}} \tau |f|_{\mathcal{F}})}{\varepsilon \tau_{\min}(1-\tau h_+(n_G))} \right)^{1/p} \right\rceil .$$

  Thus, the cost of this adaptive aglgorithm is at worst a constant multiple times the cost of the non-adaptive Algorithm 1 that requires an a priori upper bound on $|f|_{\mathcal{F}}$.
- On top of the above analysis, in this article we allow for a maximum computational cost budget, $N_{\max}$, that allows the user to specify how long he or she is willing to wait for an answer.

Figure 3 is a diagram of the automatic algorithm just outlined.

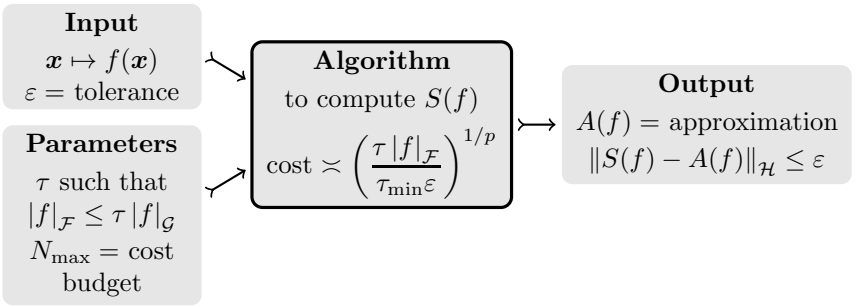

Figure 3: Diagram of a guaranteed, adaptive, automatic algorithm.

Section 5 presents a concrete example of this approach for the problem of computing the integral $\int_0^1 f(x)\,\mathrm{d}x$ for all integrands whose derivatives have finite total variation, i.e., $|f|_{\mathcal{F}} = \text{Var}(f')$. The trapezoidal rule based on n function values, i.e., $n-1$ trapezoids, computes an approximation with error no greater than ε using $2 + \sqrt{\text{Var}(f')/(8\varepsilon)}$ function values. However, except for relatively simple integrands, a bound on $\text{Var}(f')$ is unknown in practice.

Following the procedure outlined above we choose the weaker semi-norm as $|f|_{\mathcal{G}} = \|f' - f(1) + f(0)\|_1$. This semi-norm may be estimated by an algorithm

G_n that takes the $\mathcal{G}$-semi-norm of the piecewise linear spline for f. The error for $G_n(f)$ is bounded by $\mathrm{Var}(f')/(2n-2)$. This implies that $2\,\|f' - f(1) + f(0)\|_1 \leq \mathrm{Var}(f')$ for all f whose derivative has finite total variation. Fixing the parameter $\tau > 2$, and assuming that $\|f''\|_1 \leq \tau\,\|f' - f(1) + f(0)\|_1$ (the cone condition), then leads to a guaranteed, automatic (adaptive) algorithm for approximating the integral to the desired accuracy with a computational cost that is no greater than $4 + \tau + \sqrt{\tau\,\mathrm{Var}(f')/(4\varepsilon)}$ (Theorem 6), where $\mathrm{Var}(f')$ is unknown a priori. Here τ represents a minimum sample size, and $1/\tau$ represents a length scale of for possible spikes that one wishes to integrate accurately.

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 lays out the problems to be solved. Section 2.2 defines non-adaptive algorithms, their costs, and the complexities of numerical problems for which non-adaptive algorithms are suited. The algorithm diagrammed in Figure 2 is essentially this kind of algorithm. Non-adaptive algorithms are the building blocks for adaptive algorithms. Adaptive algorithms, their costs, and the complexity of problems defined on cones of functions are introduced in Section 2.3. The maximum and minimum costs of algorithms are defined in terms of the unknown, but estimated, $\mathcal{G}$-semi-norm of the input function. The problem definition here allows for a maximum computational cost budget to be imposed so that one never needs to wait an arbitrarily large time for the answer.

Sections 3 and 4 describes 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. In particular, the following results are presented:

- Algorithm 2 is a two stage algorithm: $|f|_{\mathcal{G}}$ is bounded above once, and this bound is used to determine the additional computational cost, n, such that $A_n(f)$ achieves the desired accuracy. Algorithm 3 is a multi-stage algorithm, where the algorithm, G_n, used to bound $|f|_{\mathcal{G}}$, and the algorithm, A_n, used to approximate $S(f)$, are based on the same data. Moreover, these are embedded algorithms whose costs increase progressively according to some sequence $n_1, n_2, \ldots$. Once the number of samples is sufficiently large so that $g_{n_i}(f)$ is small enough, the algorithm terminates.

- Theorems 2 and 3 in Section 3 prove that Algorithms 2 and 3, respectively, are guaranteed to produce answers to within the desired error tolerance, provided that they do not exceed the maximum cost budget. These the-

orems also demonstrate that the computational costs of these algorithms do not exceed the costs of algorithms that know $|f|_{\mathcal{F}}$ or $|f|_{\mathcal{G}}$ a priori.

- Theorem 5 in Section 4 provides lower bounds on the computational complexity of the problems defined on cones of input functions by constructing fooling functions. Corollary 1 shows that Algorithms 2 and 3 are optimal if the non-adaptive algorithms on which they are based are optimal for problems defined on balls of input functions.

Section 5 illustrates the general results in Sections 3 and 4 for the univariate integration problem. An explicit automatic algorithm based on the trapezoidal rule is presented and its success and optimality are proved in Theorem 6. Section 6 considers function approximation with analogous results. 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* $S : \mathcal{F} \to \mathcal{H}$, where $\mathcal{F}$ is a Banach space of possible input functions defined on $\mathcal{X}$ with semi-norm $|\cdot|_{\mathcal{F}}$, and $\mathcal{H}$ is some other Banach space of possible outputs or solutions with norm $\|\cdot\|_{\mathcal{H}}$. The solution operator is assumed to have a scale property, i.e.,

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

Examples include the following:

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

The first three examples above are linear problems, but the last example is a nonlinear problem, which nevertheless also has the scale property.

The goal is to find an algorithm $A : \mathcal{F} \to \mathcal{H}$ for which $S(f) \approx A(f)$. Following the definition of algorithms described in [19, Section 3.2], the algorithm takes the form of some function of data derived from the input function:

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

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

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

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 the choice of L_i may depend on $L_1(f), \ldots, L_{i-1}(f)$. In this article, all algorithms are assumed to be deterministic. There is no random element.

2.2. Non-Adaptive Algorithms

The set $\mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$ contains non-adaptive algorithms, i.e., those algorithms for which the choice of the L_i and the number of function data used both independent of the input function. Furthermore, any $A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$ is assumed to satisfy the following scale properties:

$$\boldsymbol{L}(cf) = c\boldsymbol{L}(f), \quad \phi(c\boldsymbol{y}) = c\phi(\boldsymbol{y}), \quad A(cf) = cA(f) \qquad \forall c \geq 0,\ \boldsymbol{y} \in \mathbb{R}^m. \quad (7)$$

The cost of a non-adaptive algorithm, $A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$, is fixed and is defined as the sum of the costs of all the function data:

$$\text{cost}(A) = \$(\boldsymbol{L}) = \$(L_1) + \cdots + \$(L_m), \quad (8)$$

where $\$: \Lambda \to [1, \infty)$, 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 [21, 22, 23] for function approximation problems.

It is assumed that any non-adaptive algorithm $A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$ is exact for all inputs with zero semi-norm, i.e., $S(f) = A(f)$, for all f with $|f|_{\mathcal{F}} = 0$. The error of a non-adaptive algorithm is defined as

$$\text{err}(A, \mathcal{F}, \mathcal{H}, S) = \sup_{\substack{f \in \mathcal{F} \\ |f|_{\mathcal{F}} \neq 0}} \frac{\|S(f) - A(f)\|_{\mathcal{H}}}{|f|_{\mathcal{F}}}. \quad (9)$$

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

$$\text{err}_{\pm}(A, \mathcal{F}, \mathbb{R}, S) = \sup_{\substack{f \in \mathcal{F} \\ |f|_{\mathcal{F}} \neq 0}} \frac{\pm[S(f) - A(f)]}{|f|_{\mathcal{F}}}. \quad (10)$$

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

$$\begin{aligned} &\text{comp}(\varepsilon, \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)) \\ &\qquad = \inf \left\{ \text{cost}(A) : \text{err}(A, \mathcal{F}, \mathcal{H}, S) \leq \varepsilon,\ A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda) \right\}. \end{aligned} \quad (11)$$

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

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

This cost is non-increasing as either ε decreases or $|f|_{\mathcal{F}}$ increases.

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

$$\{A_n\}_{n \in \mathcal{I}}, \qquad A_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda), \tag{12a}$$

$$\lim_{\substack{n \to \infty \\ n \in \mathcal{I}}} \mathrm{err}(A_n, \mathcal{F}, \mathcal{H}, S) = 0, \qquad \mathrm{cost}(A_n) = n, \tag{12b}$$

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

$$\mathcal{I} = \{n_1, n_2, \ldots\} \quad \text{with } n_i < n_{i+1}, \quad \text{satisfies } \sup_i \frac{n_{i+1}}{n_i} < \infty. \tag{13}$$

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

$$\sup_{0 < \varepsilon \le 1} \frac{\min\{n \in \mathcal{I} : \mathrm{err}(A_n, \mathcal{F}, \mathcal{H}, S) \le \varepsilon\}}{\mathrm{comp}(\varepsilon, \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda))} < \infty. \tag{14}$$

2.3. Automatic, Adaptive Algorithms

Non-adaptive algorithms, $A \in \mathcal{A}_{\mathrm{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$ need an upper bound on $|f|_{\mathcal{F}}$ to guarantee that they meet the prescribed error tolerance for the input function f. Adaptive algorithms attempt to estimate $|f|_{\mathcal{F}}$ and then determine the number of function data needed to meet the error tolerance. The definitions of automatic, adaptive algorithms given in this subsection, differ somewhat from the definitions of non-adaptive algorithms in the previous subsection. However, automatic algorithms use non-adaptive algorithms as building blocks.

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

$$(A, W) : \mathcal{F} \times (0, \infty) \times [1, \infty] \to \mathcal{H} \times \{\mathrm{false}, \mathrm{true}\},$$

for which one hopes that $S(f) \approx A(f; \varepsilon, N_{\max})$. Here $\varepsilon \in (0, \infty)$ is a user-supplied error tolerance, $N_{\max} \in [1, \infty]$ is a user-supplied maximum cost budget, and $W(f; \varepsilon, N_{\max})$ is a Boolean warning flag that is false if the algorithm completed its calculations without attempting to exceed the cost budget, and is true otherwise.

As in (5), the algorithm is some function of function data: $A(f; \varepsilon, N_{\max}) = \phi(\boldsymbol{L}(f); \varepsilon, N_{\max})$. Now, however, the algorithm is allowed to be adaptive. The choice of L_2 may depend on the value of $L_1(f)$, the choice of L_3 may depend on $L_1(f)$ and $L_2(f)$, etc. The number of function data used by the algorithm, m, may also be determined adaptively. The choice of how many and which function data to use depends on ε and $N_{\max}$. Thus, $\boldsymbol{L}(c\boldsymbol{y})$ might not equal

$c\boldsymbol{L}(\boldsymbol{y})$ since the length of the information vector depends on the data recorded. The goal of the algorithm is to make $\|S(f) - A(f;\varepsilon,N_{\max})\|_{\mathcal{H}} \le \varepsilon$, but this is not a requirement of the definition.

The cost of the algorithm for a specified input function is defined analogously to (8) as the sum of the costs of all function data.

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

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

The automatic, adaptive algorithms developed here are valid for subsets of $\mathcal{F}$ that may be unbounded, e.g., cones of functions, $\mathcal{C}_\tau$, as described in Section 2.4, or their subsets, $\mathcal{N}$. The design of the algorithm depends on the definition of $\mathcal{N}$. The computational cost of the adaptive algorithm for f in $\mathcal{N}$ may be quite large or even arbitrarily large, depending on the $\mathcal{F}$-semi-norm of f. We want to find upper and lower bounds on the cost of the algorithm that take into account more information about f, say that it lies in some ball. However, the definition of the ball is not used by the algorithm, but only as a device to measure computational cost. Let $\mathcal{B}_\sigma$ denote the $\mathcal{F}$-semi-norm ball defined in (3). For automatic algorithms defined with respect to some set of functions $\mathcal{N} \subset \mathcal{F}$, we define the *maximum* cost of the algorithm relative to $\mathcal{B}_\sigma$ as follows:

$$\mathrm{maxcost}(A,\mathcal{N},\varepsilon,N_{\max},\mathcal{B}_\sigma) = \sup\{\mathrm{cost}(A,f;\varepsilon,N_{\max}) : f \in \mathcal{N} \cap \mathcal{B}_\sigma\}.$$

Here the set $\mathcal{N}$ is allowed to depend on the algorithm inputs, ε and $N_{\max}$, but not on σ. Again we stress that A knows $\mathcal{N}$ but not $\mathcal{B}_\sigma$.

One would prefer an algorithm where the costs for different input functions with the same size or norm to be similar. To make this precise we define the *minimum* cost:

$$\mathrm{mincost}(A,\mathcal{N},\varepsilon,N_{\max},\mathcal{B}_\sigma) = \inf\left\{\mathrm{cost}(A,f;\varepsilon,N_{\max}) : f \in \mathcal{N} \setminus \bigcup_{\sigma'<\sigma} \mathcal{B}_{\sigma'}\right\}.$$

An algorithm is said to have *stable computational cost* if

$$\sup_{\substack{\varepsilon,\sigma>0\\ 0<\varepsilon<\sigma}} \frac{\mathrm{maxcost}(A,\mathcal{N},\varepsilon,N_{\max},\mathcal{B}_\sigma)}{\mathrm{mincost}(A,\mathcal{N},\varepsilon,N_{\max},\mathcal{B}_\sigma)} < \infty.$$

For automatic algorithms, returning an approximation with the desired error is not enough. One also wants the algorithm to be confident that the answer is correct. An algorithm $(A,W) \in \mathcal{A}(\mathcal{N},\mathcal{H},S,\Lambda)$, is deemed successful provided that it meets the prescribed error tolerance and does not raise the warning flag.

Specifically, success is defined as

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

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

The complexity of a problem is defined as the maximum cost of the cheapest successful algorithm for input functions with $\mathcal{G}$-semi-norm no greater than σ:

$$\begin{aligned}&\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda), N_{\max}, \sigma)\\ &\quad = \inf\{\operatorname{maxcost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) : \operatorname{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = \text{true},\\ &\qquad\qquad (A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)\}\,. \quad (15)\end{aligned}$$

Here the infimum of an empty set is defined to be ∞. The optimality of adaptive algorithms is defined analogously to (14).

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

2.4. Cones of Functions

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

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

$$\tau_{\min}\,|f|_{\mathcal{G}} \le |f|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}, \tag{16a}$$

where $\tau_{\min}$ is some *positive* constant. Here, the subspace of functions in $\mathcal{G}$ with vanishing $\mathcal{G}$-semi-norm, is assumed to be a finite dimensional subspace of $\mathcal{F}$ as well, and by the above condition must be contain the subspace of of functions with vanishing $\mathcal{F}$-semi-norm, i.e.,

$$\mathcal{F}_0 := \{f \in \mathcal{F} : |f|_{\mathcal{F}} = 0\} \subseteq \mathcal{G}_0 := \{f \in \mathcal{G} : |f|_{\mathcal{G}} = 0\} \subset \mathcal{F}. \tag{16b}$$

Given $\tau > 0$, let $\mathcal{C}_\tau \subset \mathcal{F}$ denote a *cone* of functions whose $\mathcal{F}$-semi-norms are no greater than τ times their $\mathcal{G}$-semi-norms, as defined in (4). This means that

$$\tau_{\min} |f|_{\mathcal{G}} \leq |f|_{\mathcal{F}} \leq \tau |f|_{\mathcal{G}} \qquad \forall f \in \mathcal{C}_\tau \tag{17}$$

Note that one must choose $\tau \geq \tau_{\min}$ for $\mathcal{C}_\tau$ to contain functions outside $\mathcal{G}_0$. Moreover, $\mathcal{F}_0 = \mathcal{G}_0 \cap \mathcal{C}_\tau$.

For any $f \in \mathcal{C}_\tau$ both $|f|_{\mathcal{F}}$ and $|f|_{\mathcal{G}}$ must be finite, but they can be arbitrarily large. There is no need to assume an upper bound on their sizes, but it is possible to obtain reliable upper bounds for both $|f|_{\mathcal{F}}$ and $|f|_{\mathcal{G}}$ by sampling f. This argument appears to be circular, but is in fact valid. An upper bound on $|f|_{\mathcal{G}}$ for $f \in \mathcal{C}_\tau$ can be computed in terms of the data $\boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f))$ because $|\cdot|_{\mathcal{F}}$ is a stronger semi-norm than $|\cdot|_{\mathcal{G}}$ in the sense of (16a), and because $|f|_{\mathcal{F}}$ is no larger than a multiple of $|f|_{\mathcal{G}}$ (see Lemma 1 below). This upper bound on $|f|_{\mathcal{G}}$ then automatically implies an upper bound on $|f|_{\mathcal{F}}$ from the definition of the cone. These reliable bounds on both $|f|_{\mathcal{F}}$ and $|f|_{\mathcal{G}}$ may be used to obtain a bound on the error of the algorithm for estimating $S(f)$ (see Theorem 2 below).

2.5. Results that We Prove

The previous subsections define the problem to be approximated and the notation describing the difficulty of the problem and the efficiency of the algorithms. This subsection summarizes the results that are proved in general in the next two sections and illustrated for specific cases in the Sections 5 and 6.

i. *Upper bound on the complexity.* Theorems 2, 3, 6, and 8 provide upper bounds on the complexity of solving the problem, $\mathrm{comp}(\mathcal{N}, \varepsilon, N_{\max}, \sigma, \Lambda)$, in terms of some function of ε/σ and for the set $\mathcal{N} \subset \mathcal{C}_\tau$.

ii. *An algorithm that achieves the upper bound.* Algorithms 2 and 3 provide explicit, constructive frameworks for two kinds of successful algorithms, $(A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)$, that achieve these upper bounds. Algorithms 4 and 5 provide concrete cases.

iii. *Penalty for not knowing the $\mathcal{F}$- and $\mathcal{G}$-semi-norms of f.* The optimal successful algorithm must find an upper bound on $|f|_{\mathcal{F}}$ or $|f|_{\mathcal{G}}$ rather than assuming such an upper bound. One hopes that the extra cost relative to the situation of knowing a priori bounds on these semi-norms is not too great. Positive results are given in Theorems 2, 3, 6, and 8.

iv. *Stable Computational Cost.* The cost of solving the problem for a given tolerance, ε, and an input function with a given value of the $\mathcal{G}$-semi-norm, $|f|_{\mathcal{G}}$, varies somewhat depending on the particulars of f. However, we do not want too much variation. Theorems 2, 3, 6, and 8 provide positive results on the stability of the computational cost.

v. *Lower bound on the complexity and optimality of algorithms.* Lower bounds on the complexity of the problem, i.e., the problem's difficulty, are given in Theorem 5. Conditions under which Algorithms 2 and 3 attain these lower bounds are given in Corollary 1. Specific examples for univariate integration and function recovery are given in Theorems 7 and 9.

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 in Sections 5 and 6 where concrete cases are are discussed.

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

As mentioned in Section 2.4 automatic algorithms require reliable upper bounds on $|f|_{\mathcal{G}}$ for all f in the cone $\mathcal{C}_\tau$. These can be obtained using any non-adaptive algorithm $G_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\mathcal{G}}, \Lambda)$ for approximating $|f|_{\mathcal{G}}$ for $f \in \mathcal{F}$ and having cost n, provided that one has explicit upper bounds on the errors of these algorithms as in (10). Namely, $G_n(f) = |f|_{\mathcal{G}}$ for all $f \in \mathcal{F}$ with vanishing $\mathcal{F}$-semi-norm, and there exist non-negative valued, non-increasing functions $h_\pm$ such that

$$\operatorname{err}_\pm(G_n, \mathcal{F}, \mathbb{R}_+, |\cdot|_{\mathcal{G}}) = \sup_{\substack{f \in \mathcal{F} \\ |f|_{\mathcal{F}} \neq 0}} \frac{\pm[|f|_{\mathcal{G}} - G_n(f)]}{|f|_{\mathcal{F}}} \leq h_\pm(n). \tag{18}$$

Applying the two-sided bound for the $\mathcal{F}$- and $\mathcal{G}$-semi-norms (17) and rearranging the above error bound implies that

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

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

Lemma 1. *Any nonadaptive algorithm $G_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\mathcal{G}}, \Lambda)$ with cost $n = \operatorname{cost}(G_n)$ and two sided error bounds as in* (18) *yields an approximation to the $\mathcal{G}$-semi-norm of functions in the cone $\mathcal{C}_\tau$ with the following upper and lower error bounds:*

$$\frac{|f|_{\mathcal{F}}}{\tau \mathfrak{C}_n} \leq \frac{|f|_{\mathcal{G}}}{\mathfrak{C}_n} \leq G_n(f) \leq \begin{cases} \tilde{\mathfrak{c}}_n\,|f|_{\mathcal{G}} \\ \dfrac{\mathfrak{c}_n\,|f|_{\mathcal{F}}}{\tau_{\min}} \end{cases} \qquad \forall f \in \mathcal{C}_\tau, \tag{19}$$

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

$$\tilde{\mathfrak{c}}_n := 1 + \tau h_-(n) \geq \mathfrak{c}_n := 1 + \tau_{\min} h_-(n) \geq 1, \tag{20}$$

$$\mathfrak{C}_n := \frac{1}{1 - \tau h_+(n)} \geq 1, \qquad \textit{assuming } h_+(n) < 1/\tau. \tag{21}$$

3.2. Two-Stage Automatic Algorithms

Computing an approximate solution to the problem $S : \mathcal{C}_\tau \to \mathcal{H}$, e.g., integration or function approximation, depends on non-adaptive algorithms. Suppose that there is a sequence of such of algorithms, $\{A_n\}_{n\in\mathcal{I}}$, with $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, indexed by their cost as defined in (12), and for which upper error bounds are known $\mathcal{F}$:

$$\text{err}(A_n, \mathcal{F}, \mathcal{H}, S) \le h(n), \tag{22a}$$

for some *non-increasing* function h. Define the (non-increasing) inverse of h as

$$h^{-1}(\varepsilon) = \min\{n \in \mathcal{I} : h(n) \le \varepsilon\}. \tag{22b}$$

In this article we consider function h satisfying

$$\sup_{\epsilon>0} \frac{h^{-1}(\varepsilon)}{h^{-1}(2\varepsilon)} < \infty \tag{22c}$$

This means that $h(n)$ may decay polynomially in n^{-1} or faster as n tends to infinity. The definitions of the errors in (9) then implies an upper bounds on the error of $A_n(f)$ in term of the $\mathcal{F}$-semi-norm of f:

$$\|S(f) - A_n(f)\|_{\mathcal{H}} \le \text{err}(A_n, \mathcal{F}, \mathcal{H}, S)\,|f|_{\mathcal{F}} \le \tau h(n)\,|f|_{\mathcal{G}} \qquad \forall f \in \mathcal{C}_\tau. \tag{23}$$

Algorithm 2. (Automatic, Adaptive, Two-Stage). Let $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ be Banach spaces as described above, let S be the solution operator, and let $N_{\max}$ be the maximum cost allowed. Let τ be a fixed positive number, and let $G_{n_G} \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, |\cdot|_{\mathcal{G}}, \Lambda)$ be an algorithm as described in Lemma 1 with cost n_G satisfying $h_+(n_G) < 1/\tau$. Moreover, let $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$, be a sequence of algorithms as described in (12) and (22). Given a positive error tolerance, ε, and an input function f, do the following:

Stage 1. Estimate $|f|_{\mathcal{G}}$. First compute $G_{n_G}(f)$. Define the inflation factor $\mathfrak{C} = \mathfrak{C}_{n_G}$ according to (21). Then $\mathfrak{C}G_{n_G}(f)$ provides a reliable upper bound on $|f|_{\mathcal{G}}$.

Stage 2. Estimate $S(f)$. Choose the sample size need to approximate $S(f)$, namely, $n_A = h^{-1}(\varepsilon/(\tau\mathfrak{C}G_{n_G}(f)))$. If $n_A \le N_{\max} - n_G$, then $S(f)$ may be approximated within the desired error tolerance and within the cost budget. Set the warning flag, W, to false.

Otherwise, recompute n_A to be within budget, $n_A = \widetilde{N}_{\max} := \max\{n \in \mathcal{I} : n \le N_{\max} - n_G\}$, and set the warning flag, W to true. Finally, compute $A_{n_A}(f)$ as the approximation to $S(f)$.

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

The bounds in Lemma 1 involving G_n and $|\cdot|_{\mathcal{G}}$ plus the bounds in (17) involving $|\cdot|_{\mathcal{G}}$ and $|\cdot|_{\mathcal{F}}$ imply bounds on the desired number of samples in the

algorithm above. Since h^{-1} is a non-increasing function, it follows that for all $f \in \mathcal{C}_\tau$

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

This means that proposed computational cost to compute $A(f)$ is reasonable, i.e., $h^{-1}(\varepsilon/(\tau\mathfrak{C}G_{n_G}(f))) \leq \widetilde{N}_{\max}$ if either of the following conditions holds

$$|f|_{\mathcal{G}} \leq \varepsilon\tilde{\sigma}_{\max}, \qquad \tilde{\sigma}_{\max} := \frac{1}{\tau\mathfrak{C}\tilde{\mathfrak{c}}h(\widetilde{N}_{\max})},$$
$$|f|_{\mathcal{F}} \leq \varepsilon\sigma_{\max}, \qquad \sigma_{\max} := \frac{\tau_{\min}}{\tau\mathfrak{C}\mathfrak{c}h(\widetilde{N}_{\max}))}.$$

Theorem 2. *Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, τ, $N_{\max}$, $\widetilde{N}_{\max}$, $\mathfrak{C}$, and ε be given as described in Algorithm 2, and assume that $\mathcal{F}$ satisfies* (16a). *Let $\mathfrak{c} = \mathfrak{c}_{n_G}$ be defined as in* (20). *Let $\mathcal{C}_\tau$ be the cone of functions defined in* (4) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\mathcal{G}$-semi-norms. Let $\mathcal{N}$ be the subset of the cone $\mathcal{C}_\tau$ that lies inside either the $\mathcal{G}$-semi-norm ball of radius $\varepsilon\tilde{\sigma}_{\max}$, i.e., $\widetilde{\mathcal{B}}_{\varepsilon\tilde{\sigma}_{\max}}$, or the $\mathcal{F}$-semi-norm ball of radius $\varepsilon\sigma_{\max}$, i.e., $\mathcal{B}_{\varepsilon\sigma_{\max}}$:*

$$\mathcal{N} = \mathcal{C}_\tau \cap (\widetilde{\mathcal{B}}_{\varepsilon\tilde{\sigma}_{\max}} \cup \mathcal{B}_{\varepsilon\sigma_{\max}}). \tag{24}$$

Since $\widetilde{N}_{\max}$ is assumed to be huge, it follows that radii $\tilde{\sigma}_{\max}$ and $\sigma_{\max}$ are also huge.

Then it follows that Algorithm 2 is successful for all functions in this set of nice *functions $\mathcal{N}$, i.e.,* $\mathrm{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = \mathit{true}$. *Moreover, the cost of this algorithm is bounded above and below in terms of the unknown $\mathcal{G}$- and $\mathcal{F}$-semi-norms of any input function in $\mathcal{N}$ as follows:*

$$n_G + h^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\mathcal{G}}}\right) \leq \mathrm{cost}(A, f; \varepsilon, N_{\max}) \leq n_G + h^{-1}\left(\frac{\varepsilon}{\tau\tilde{\mathfrak{C}}\,|f|_{\mathcal{G}}}\right), \tag{25a}$$
$$n_G + h^{-1}\left(\frac{\varepsilon}{|f|_{\mathcal{F}}}\right) \leq \mathrm{cost}(A, f; \varepsilon, N_{\max}) \leq n_G + h^{-1}\left(\frac{\tau_{\min}\varepsilon}{\tau\mathfrak{C}\mathfrak{c}\,|f|_{\mathcal{F}}}\right). \tag{25b}$$

If h satisfies (22), *then this algorithm is computationally stable in the sense that the maximum cost is no greater than some constant times the minimum cost.*

Proof. The definition of $\mathfrak{C}$ in (21) implies that the true $\mathcal{G}$-semi-norm of f is bounded above by $\mathfrak{C}G_{n_G}(f)$ according to Lemma 1. The upper bound on the error of the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$ in (23) then implies that

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

This error upper bound may be made no greater than the error tolerance, ε, by choosing the algorithm cost, n, to satisfy the condition in Stage 2 of Algorithm

2, provided that this can be done within the maximum cost budget. In this case, the algorithm is successful, as claimed in the theorem.

The argument preceding the statement of this theorem establishes that Algorithm 2 does not attempt to overrun the cost budget if $f \in \widetilde{\mathcal{B}}_{\varepsilon\tilde{\sigma}_{\max}} \cup \mathcal{B}_{\varepsilon\sigma_{\max}}$. This argument also establishes the maximum and minimum cost bounds in (25). □

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, G_n, as described in Section 3.1 that approximates the weaker semi-norm, $|\cdot|_{\mathcal{G}}$, and its error bound, $h_{\pm}$, as defined in (18) must be known explicitly.
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 satisfying condition (22), and the error function h, defined in (22) 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 (25), depend on the $\mathcal{F}$- and $\mathcal{G}$-semi-norms of the input function, f. However, the values of these semi-norms are not required as inputs to the algorithm, but rather are conservatively estimated by the algorithm. The number of samples used to obtain the approximation to $S(f)$ is adjusted adaptively based on these bounds on the semi-norms.

Remark 3. The set of input functions for which the Algorithm 2 is guaranteed to work, $\mathcal{N}$, is limited somewhat by the error tolerance, but only because of the practical constraint of a cost budget of $N_{\max}$. This dependence disappears if one lifts this constraint by taking $N_{\max} \to \infty$. The primary constraint determining the success of the algorithm is that f lies in the cone $\mathcal{C}_\tau$.

Remark 4. 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

$$\tau = \left(1 - \frac{1}{\mathfrak{C}}\right) \frac{1}{h_+(n_G)}, \tag{26}$$

which is equivalent to (21).

Remark 5. In some cases it is possible to find a lower bound on the $\mathcal{F}$-norm of the input function, i.e., there exists an algorithm F_n using the same function values as G_n, such that

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

In this case one can use Lemma 1 to derive a necessary condition that f is in the cone $\mathcal{C}_\tau$:

$$\begin{aligned} f \in \mathcal{C}_\tau &\implies F_n(f) \le |f|_{\mathcal{F}} \le \frac{\tau G_n(f)}{1 - \tau h_+(n)} \\ &\implies \tau_{\min,n} := \frac{F_n(f)}{G_n(f) + h_+(n) F_n(f)} \le \tau. \end{aligned} \tag{27}$$

For Algorithm 2 the relevant value of n is n_G, 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 the condition above is satisfied. However, in light of this argument one might modify these algorithms by increasing τ to, say, $2\tau_{\min,n}$ in case $\tau_{\min,n}$ ever falls above τ.

Remark 6. It may also be possible to obtain an error upper bound, $\tilde{h}$ such that $\mathrm{err}(A_n, \mathcal{G}, \mathcal{H}, S) \leq \tilde{h}(n)$. By following the argument leading to (23), it is clear that such an error bound is no use if

$$\tau h(n) \leq \tilde{h}(n) \quad \forall n \in \mathcal{I}. \tag{28}$$

This condition is observed to hold for the examples of Sections 5 and 6. A sufficient condition for (28) is

$$h(n) \leq \tilde{h}(n) h_+(n) \quad \forall n \in \mathcal{I}, \tag{29}$$

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, it is supposed that the data used by G_{n_G}, the algorithm used to estimate the $\mathcal{G}$-semi-norm of f, is the same data used by A_{n_1}, and so $n_1 = n_G$. Then the total cost of Algorithm 2 can be reduced; it is simply n_A, as given in Stage 2, instead of $n_G + n_A$. Moreover, $\widetilde{N}_{\max}$ may then be taken to be $N_{\max}$, and the cost bound of the automatic algorithm in (25) does not need the term n_G.

Again suppose that $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, consists of algorithms as described in (12) and (22), 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 $\mathcal{G}$-semi-norm, $\{G_n\}_{n\in\mathcal{I}}$, $G_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{G}, \mathbb{R}_+, |\cdot|_{\mathcal{G}}, \Lambda)$, such that for each $n \in \mathcal{I}$, A_n and G_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 the Banach spaces $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$, the solution operator S, the maximum cost budget $N_{\max}$, and the positive constant τ be as described in Algorithm 2. Let the sequences of algorithms, $\{A_n\}_{n\in\mathcal{I}}$ and $\{G_n\}_{n\in\mathcal{I}}$ be as described above. Set $i = 1$, 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:

Stage 1. Estimate $\|f\|_{\mathcal{G}}$. Compute $G_{n_i}(f)$ and $\mathfrak{C}_{n_i} \leq \infty$ as defined in (21).

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

$$\tau\mathfrak{C}_{n_i}h(n_i)G_{n_i}(f) \leq \varepsilon. \tag{30}$$

If this is true, then set W to be false, return $(A_{n_i}(f), W)$ and terminate the algorithm.

Stage 3. Compute n_{i+1}. Otherwise, if (30) fails to hold, compute $\mathfrak{c}_{n_i}$ according to (20) using G_{n_i}. Choose n_{i+1} as the smallest number exceeding n_i and not less than $h^{-1}(\varepsilon\mathfrak{c}_{n_i}/G_{n_i}(f))$ such that A_{n_i} is embedded in $A_{n_{i+1}}$. 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} 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.

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

$$h_1(n) = \mathfrak{C}_n\tilde{\mathfrak{c}}_n h(n), \quad h_2(n) = \mathfrak{C}_n\mathfrak{c}_n h(n) \qquad n \in \mathcal{I}. \tag{31}$$

and note that the left hand side of the convergence check, (30), has the following upper and lower bounds based on (19) for $f \in \mathcal{C}_\tau$:

$$h(n_i)\,|f|_{\mathcal{F}} \leq \tau h(n_i)\,|f|_{\mathcal{G}} \leq \tau\mathfrak{C}_{n_i}h(n_i)G_{n_i}(f) \leq \begin{cases} \tau h_1(n_i)\,|f|_{\mathcal{G}} \\ \dfrac{\tau h_2(n_i)\,|f|_{\mathcal{F}}}{\tau_{\min}} \end{cases}. \tag{32}$$

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

Theorem 3. *Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, $N_{\max}$, τ, n_1, and ε be given as described in Algorithm 3. Let r be the minimum cost multiple described in the paragraphs preceding that Algorithm 3. Assume that $n_1 \leq N_{\max}/r$. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (4) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\mathcal{G}$-semi-norms. Let*

$$\mathcal{N} = \mathcal{C}_\tau \cup (\widetilde{\mathcal{B}}_{\varepsilon\sigma_{1,\max}} \cup \mathcal{B}_{\varepsilon\sigma_{2,\max}}).$$

be the nice subset of the cone $\mathcal{C}_\tau$, where the parameters determining the radii of the two balls are

$$\sigma_{1,\max} = \frac{\varepsilon}{\tau h_1(N_{\max}/r)}, \qquad \sigma_{2,\max} = \frac{\varepsilon}{\tau h_2(N_{\max}/r)}.$$

Then it follows that Algorithm 3 is successful for all functions in this set of nice *functions $\mathcal{N}$, i.e.,* $\mathrm{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = true$. *Moreover, the cost of this algorithm is bounded above and below in terms of the unknown $\mathcal{G}$- and $\mathcal{F}$-seminorms of any input function in $\mathcal{N}$ as follows:*

$$\max\left(n_1, h^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\mathcal{G}}}\right)\right) \leq \mathrm{cost}(A, f; \varepsilon, N_{\max}) \leq \max\left(n_1, rh_1^{-1}\left(\frac{\varepsilon}{\tau\,|f|_{\mathcal{G}}}\right)\right), \quad (33a)$$

$$\max\left(n_1, h^{-1}\left(\frac{\varepsilon}{|f|_{\mathcal{F}}}\right)\right) \leq \mathrm{cost}(A, f; \varepsilon, N_{\max}) \leq \max\left(n_1, rh_1^{-1}\left(\frac{\tau_{\min}\varepsilon}{\tau\,|f|_{\mathcal{F}}}\right)\right). \quad (33b)$$

If h satisfies (22)*, then this algorithm is computationally stable in the sense that the maximum cost is no greater than some constant times the minimum cost.*

Proof. Let $n_1, \ldots, n_j$ be the sequence of n_i generated by Algorithm 3, j being the number of the iterate where the algorithm either

i) terminates successfully because the convergence criterion, (30), is satisfied for $i = j$, or
ii) terminates with a warning because (30) is not satisfied for $i = j$, but the the proposed n_{j+1} exceeds the cost budget, $N_{\max}$.

Here j may be any positive integer. The design of Algorithm 3 guarantees that $n_1 < \cdots < n_j$. It is shown that under the hypotheses of this theorem, that under condition i), the error tolerance is met and that the cost of the algorithm satisfies upper bound (33). It is also shown that under condition ii), an unsuccessful termination of the algorithm, $f \in \mathcal{N}$, is impossible.

First, consider possibility i) and recall inequality (19). Since (30) is satisfied, it follows that $\left\|S(f) - A_{n_j}(f)\right\|_{\mathcal{H}} \leq \varepsilon$ by the same argument as given in the proof of Theorem 2. In this case the algorithm terminates without warning and the approximate value is within the required tolerance.

If $j = 1$, then the cost of the algorithm is n_1. If $j > 1$, then the convergence criterion was not satisfied for $i = j - 1$. Thus, it follows that $n_{j-1} < h_1^{-1}(\varepsilon/(\tau\,|f|_{\mathcal{G}}))$ by the inequality in (32). If $n_{j-1} \geq N_A(\varepsilon\mathfrak{c}_{n_{j-1}}/G_{n_{j-1}}(f))$, then Stage 3 chooses n_j to be the smallest element of $\mathcal{I}$ that exceeds n_{j-1} and for which $A_{n_{j-1}}$ is embedded in A_{n_j}. By the definition of r it follows that

$$n_j \leq rn_{j-1} < r\widetilde{N}_A(\varepsilon/\,|f|_{\mathcal{G}}),$$

and the upper bound on the cost in (33) is satisfied. If, on the other hand, $n_{j-1} < N_A(\varepsilon\mathfrak{c}_{n_{j-1}}/G_{n_{j-1}}(f))$, then Stage 3 chooses n_j to be the smallest element of $\mathcal{I}$ that is no less than $N_A(\varepsilon\mathfrak{c}_{n_{j-1}}/G_{n_{j-1}}(f))$ and for which $A_{n_{j-1}}$ is

embedded in A_{n_j}. By the definition of r, (19), and the definition of $\widetilde{N}_A$, it follows that

$$n_j < rN_A(\varepsilon \mathfrak{c}_{n_{j-1}}/G_{n_{j-1}}(f)) \le rN_A(\varepsilon/|f|_{\mathcal{G}}) \le r\widetilde{N}_A(\varepsilon/|f|_{\mathcal{G}}).$$

Again, the maximum cost of the algorithm is bounded as in (33).

Also note that under possibility i) the convergence condition (30) implies that $n_j \ge n_1$ and $n_j \ge N_A(\varepsilon/|f|_{\mathcal{G}})$ since $|f|_{\mathcal{G}} \le \mathfrak{C}_{n_i} G_{n_i}(f)$. This establishes the lower bound on the minimum computational cost.

Second, consider possibility ii), meaning that the convergence criterion, (30), is not satisfied for $i = j$. Then Stage 3 tries to choose n_{j+1} to satisfy this criterion. Using similar arguments as in the previous paragraph, it follows that $n_j < \widetilde{N}_A(\varepsilon/|f|_{\mathcal{G}})$. If $n_j \ge N_A(\varepsilon \mathfrak{c}_{n_j}/G_{n_j}(f))$, then the proposed n_{j+1} satisfies

$$n_{j+1} \le rn_j < r\widetilde{N}_A(\varepsilon/|f|_{\mathcal{G}}).$$

If, on the other hand, $n_j < N_A(\varepsilon \mathfrak{c}_{n_j}/G_{n_j}(f))$, then the proposed n_{j+1} satisfies

$$n_{j+1} < rN_A(\varepsilon \mathfrak{c}_{n_j}/G_{n_j}(f)) \le rN_A(\varepsilon/|f|_{\mathcal{G}}) \le r\widetilde{N}_A(\varepsilon/|f|_{\mathcal{G}}).$$

In both cases, the n_{j+1} proposed by the algorithm satisfies $n_{j+1} < r\widetilde{N}_A(\varepsilon/|f|_{\mathcal{G}})$, which does not exceed the cost budget by the by the definition of $\mathcal{N}$. Thus, possibility ii) cannot happen.

The proof of the optimality of the multistage algorithm follows the same line of argument used to prove the optimality of the two-stage algorithm in Theorem 2. This completes the proof. □

4. Lower Complexity Bounds for the Algorithms

Lower complexity bounds are typically proved by constructing fooling functions. First, a lower bound is derived for the complexity of problems defined on $\mathcal{F}$- and $\mathcal{G}$-semi-norm balls of input functions. This technique is generally known, see for example [20, p. 11–12]. Then it is shown how to extend this idea for the cone $\mathcal{C}_\tau$.

Consider the Banach spaces $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, and the *linear* solution operator $S : \mathcal{G} \to \mathcal{H}$. Let Λ be the set of bounded linear functionals that can be used as data. Suppose that for any $n \ge 1$, 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, solution norm one, and $\mathcal{F}$ and $\mathcal{G}$ semi-norms with known upper bounds, namely,

$$\|S(f_1)\|_{\mathcal{H}} = 1, \quad \boldsymbol{L}(f_1) = \boldsymbol{0}, \quad |f_1|_{\mathcal{G}} \le \tilde{g}(n), \quad |f_1|_{\mathcal{F}} \le g(n)\,|f_1|_{\mathcal{G}}\,, \tag{34}$$

for some positive, unbounded, non-decreasing functions $\tilde{g}$ and g defined on $[1, \infty)$. For example, one might have $\tilde{g}(n) = an^p$ and $g(n) = bn^q$ with positive a, b, p, and q. Since the data for the function f_1 are all zero, it follows that $A(f_1) = A(-f_1)$ for any algorithm, A, adaptive or not, that is based on the

information $\boldsymbol{L}(f_1)$. Then, by the triangle inequality and (34) the error for one of the fooling functions $\pm f_1$ must be at least one:

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

Furthermore, applying (34), for $\mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}$, it follows that any nonadaptive algorithm satisfying the error tolerance ε must have a cost n satisfying the following inequality:

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

This implies lower bounds on the complexity of nonadaptive algorithms, as defined in (11):

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

where $\tilde{g}^{-1}$ and $(g\tilde{g})^{-1}$ denote the inverse functions of $\tilde{g}$ and $g\tilde{g}$, respectively:

$$\tilde{g}^{-1}(x) = \inf\{n \geq 1 : \tilde{g}(n) \geq x\}, \qquad (g\tilde{g})^{-1}(x) = \inf\{n \geq 1 : g(n)\tilde{g}(n) \geq x\}.$$

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

Proposition 4. *Suppose that there exist non-adaptive algorithms $\{A_n\}_{n\in\mathcal{I}}$, with $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, for which the upper error bounds satisfy (22), for known $\tilde{h}$ and h. If*

$$\sup_{m \geq 1} \inf\{j : jm \in \mathcal{I},\ \tilde{g}(m)\tilde{h}(jm) \leq 1\} < \infty, \tag{35a}$$

then these algorithms are optimal in the sense of (14) for the problem defined on $\mathcal{G}$. If

$$\sup_{m \geq 1} \inf\{j : jm \in \mathcal{I},\ g(m)\tilde{g}(m)h(jm) \leq 1\} < \infty, \tag{35b}$$

then these algorithms are optimal for the problem defined on $\mathcal{F}$.

Proof. For the case of input functions defined on $\mathcal{G}$ it follows that

$$\begin{aligned}
&\sup_{0<\varepsilon\leq 1} \frac{\min\{n\in\mathcal{I}:\operatorname{err}(A_n,\mathcal{G},\mathcal{H},S)\leq\varepsilon\}}{\operatorname{comp}(\varepsilon,\mathcal{A}_{\text{non}}(\mathcal{G},\mathcal{H},S,\Lambda))}\\
&\quad\leq \sup_{0<\varepsilon\leq 1} \frac{\min\{n\in\mathcal{I}:\tilde{h}(n)\leq\varepsilon\}}{\tilde{g}^{-1}(\varepsilon^{-1})}\\
&\quad\leq \sup_{m\geq 1} \frac{\min\{n\in\mathcal{I}:\tilde{g}(m)\tilde{h}(n)\leq 1\}}{m} \qquad (\varepsilon=\tilde{g}(m))\\
&\quad\leq \sup_{m\geq 1} \frac{m\inf\{j:jm\in\mathcal{I},\ \tilde{g}(m)\tilde{h}(jm)\leq 1\}}{m} < \infty.
\end{aligned}$$

The proof for $\mathcal{F}$ follows similarly. □

Turning to the problem of solving functions in the cone $\mathcal{C}_\tau$, the lower bound on the complexity becomes a bit more difficult to derive. Note that condition (34) allows the fooling function f_1 to lie *outside* this cone for $g(n)>\tau$. Thus, when considering the cone of input functions, the fooling function must be constructed as a linear combination of f_1 and a function lying in the interior of the cone.

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

$$|f_0|_{\mathcal{G}}>0, \qquad |f_0|_{\mathcal{F}}\leq\tau_0\,|f_0|_{\mathcal{G}}, \qquad \tau_0<\tau. \tag{36}$$

Furthermore, suppose that for each $n\geq 1$, and for all $\boldsymbol{L}\in\Lambda^m$, satisfying $\$(\boldsymbol{L})\leq n$, there exists f_1 as described above in (34). 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 for functions in the cone $\mathcal{C}_\tau$.

Theorem 5. *Suppose that functions f_0 and f_1 can be found that satisfy conditions* (34) *and* (36)*. It then follows that the complexity of the problem, defined by* (15)*, assuming infinite cost budget, over the cone of functions $\mathcal{C}_\tau$ is*

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

Proof. Let A be a successful, possibly adaptive, algorithm for all functions lying in the cone $\mathcal{C}_\tau$. Given an error tolerance, ε, and a positive σ, let f_0 be a function satisfying (36) and choose

$$c_0=\frac{\sigma\tau}{|f_0|_{\mathcal{G}}\,(2\tau-\tau_0)}>0. \tag{37a}$$

Provide the algorithm A with the input function c_0f_0, and let $\boldsymbol{L}(c_0f_0)$ be the data vector extracted by A to obtain the estimate $A(c_0f_0)$. 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 (34) with c_1 satisfying

$$c_1 = \frac{(\tau - \tau_0) c_0 \left| f_0 \right|_{\mathcal{G}}}{\left| f_1 \right|_{\mathcal{G}} [g(n) + \tau]} = \frac{\sigma\tau(\tau - \tau_0)}{\left| f_1 \right|_{\mathcal{G}} (2\tau - \tau_0)[g(n) + \tau]} > 0. \tag{37b}$$

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

$$\begin{aligned}
\left| f_\pm \right|_{\mathcal{F}} - \tau \left| f_\pm \right|_{\mathcal{G}} &\leq c_0 \left| f_0 \right|_{\mathcal{F}} + c_1 \left| f_1 \right|_{\mathcal{F}} - \tau(c_0 \left| f_0 \right|_{\mathcal{G}} - c_1 \left| f_1 \right|_{\mathcal{G}}) \\
&\qquad\qquad \text{by the triangle inequality} \\
&\leq c_1[g(n) + \tau] \left| f_1 \right|_{\mathcal{G}} - (\tau - \tau_0) c_0 \left| f_0 \right|_{\mathcal{G}} \qquad \text{by (34), (36)} \\
&= 0 \qquad\qquad \text{by (37).}
\end{aligned}$$

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

$$\begin{aligned}
\left| f_\pm \right|_{\mathcal{G}} &\leq c_0 \left| f_0 \right|_{\mathcal{G}} + c_1 \left| f_1 \right|_{\mathcal{G}} \\
&= \frac{\sigma\tau}{2\tau - \tau_0} \left[1 + \frac{\tau - \tau_0}{g(n) + \tau} \right] \qquad \text{by (37)} \\
&\leq \frac{\sigma\tau}{2\tau - \tau_0} \left[1 + \frac{\tau - \tau_0}{\tau} \right] = \sigma.
\end{aligned}$$

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

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

Since A is successful for these two fooling functions, c_1, as defined in (37), must be no larger than the error tolerance, which implies by (34) that

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

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

Comparing the lower bound on the computational complexity in Theorem 5 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-adaptive building blocks are also optimal for solving the problem for balls of input functions.

Corollary 1. *Suppose that the functions $g, \tilde{g}, h,$ and $\tilde{h}$ satisfy the hypotheses of Proposition 4, i.e., conditions* (35)*, which means that the non-adaptive algorithms are optimal for solving the problem on $\mathcal{F}$- and $\mathcal{G}$-balls of input functions. It then follows that Algorithms 2 and 3 are both optimal for solving the problem on 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 Banach spaces of input functions, $\mathcal{F}$ and $\mathcal{G}$, and their semi-norms,
- the Banach space of outputs, $\mathcal{H}$, and its norm,
- the solution operator, S,
- a set of non-adaptive 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,
- the upper bound error functions, $\tilde{h}$ and h, defined in (22),
- a set of non-adaptive algorithms, $\{G_n\}_{n\in\mathcal{I}}$, which approximate $|\cdot|_{\mathcal{G}}$, such that G_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 (20) and (21) respectively, and
- the fooling functions f_0 and f_1, along with the associated parameter τ_0 and the functions g and $\tilde{g}$, all of which satisfy (34) and (36).

This allows one to use Algorithms 2 and 3 with the guarantees provided by Theorems 2 and 3, the lower bound on complexity provided by Theorem 5, and the optimality given by Corollary 1.

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:

$$x_i = \frac{i-1}{n-1}, \qquad i = 1, \ldots, n, \tag{38a}$$

$$A_n(f)(x) = (n-1)\,[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)] \quad \text{for } x_i \le x \le x_{i+1}. \tag{38b}$$

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, $S(f) = \mathrm{INT}(f) = \int_0^1 f(x)\,\mathrm{d}x$. The two spaces of input functions are Sobolev spaces of different degrees of smoothness:

$$\mathcal{F} = \mathcal{W}^{2,1} \quad \text{and} \quad \mathcal{G} = \mathcal{W}^{1,1},$$

where the general Sobolev spaces with smoothness of degree k are defined as

$$\mathcal{W}^{k,p} = \mathcal{W}^{k,p}[0,1] = \{f \in C[0,1] : \|f^{(k)}\|_p < \infty\}. \tag{39}$$

The $\mathcal{F}$-semi-norm is defined as $|f|_{\mathcal{F}} = \|f''\|_1$, which corresponds to the total variation of f'. Note 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 interval. The $\mathcal{G}$-semi-norm is defined as $|f|_{\mathcal{G}} = \|f' - A_2(f)'\|_1 = \|f' - f(1) + f(0)\|_1$, which corresponds to the total variation of $f - A_2(f)$. The reason for defining $|f|_{\mathcal{G}}$ in this way, rather than as $\|f'\|_1$, is that $|f|_{\mathcal{G}}$ 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

$$\mathcal{C}_\tau = \{f \in \mathcal{W}^{2,1} : \|f''\|_1 \le \tau\|f' - f(1) + f(0)\|_1\}. \tag{40}$$

The space of outputs $\mathcal{H}$ is $\mathbb{R}$.

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

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

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 $|f|_{\mathcal{G}} = \|f' - f(1) + f(0)\|_1$ is the $\mathcal{G}$-semi-norm of the linear spline, $A_n(f)$:

$$\begin{aligned} G_n(f) = |A_n(f)|_{\mathcal{G}} &= \|A_n(f)' - A_2(f)'\|_1 \\ &= \sum_{i=0}^{n-2} \left| f(x_{i+1}) - f(x_i) - \frac{f(1) - f(0)}{n-1} \right|. \end{aligned} \tag{41}$$

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 G_n. Note that $G_n(f)$ never overestimates $|f|_{\mathcal{G}}$ because

$$\begin{aligned} |f|_{\mathcal{G}} = \|f' - A_2(f)'\|_1 &= \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} |f'(x) - A_2(f)'(x)|\,\mathrm{d}x \\ &\ge \sum_{i=1}^{n-1} \left| \int_{x_i}^{x_{i+1}} [f'(x) - A_2(f)'(x)]\,\mathrm{d}x \right| = \|A_n(f)' - A_2(f)'\|_1 = G_n(f). \end{aligned}$$

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

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

$$|f|_{\mathcal{G}} - G_n(f) = |f|_{\mathcal{G}} - \big|A_n(f)\big|_{\mathcal{G}} \le \big|f - A_n(f)\big|_{\mathcal{G}} = \big\|f' - A_n(f)'\big\|_1,$$

since $(f - A_n(f))(x)$ vanishes for $x = 0, 1$. Thus, we need bounds on $f - A_n(f)$.

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

$$\begin{aligned} f(x) - A_n(f)(x) &= f(x) - (n-1)\left[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)\right] \\ &= (n-1)\int_{x_i}^{x_{i+1}} v_i(t,x) f'(t)\,\mathrm{d}t \qquad (42) \\ &= (n-1)\int_{x_i}^{x_{i+1}} u_i(t,x) f''(t)\,\mathrm{d}t, \qquad (43) \\ f'(x) - A_n(f)'(x) &= (n-1)\int_{x_i}^{x_{i+1}} w_i(t,x) f''(t)\,\mathrm{d}t, \qquad (44) \end{aligned}$$

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

$$\mathfrak{C}_n = \frac{1}{1 - \tau/(2n-2)} \qquad \text{for } n > 1 + \tau/2. \tag{45}$$

An algorithm that provides a lower bound on $|f|_{\mathcal{F}} = \|f''\|_1$ is now derived to use as a necessary condition that f lies in $\mathcal{C}_\tau$ as described in Remark 5. Given $n = 2, 3, \ldots$, define

$$F_n(f) = (n-1) \sum_{i=1}^{n-2} |f(x_i) - 2f(x_{i+1}) + f(x_{i+2})| \tag{46}$$

It follows using the mean value theorem that

$$\begin{aligned}F_n(f) &= (n-1) \sum_{i=1}^{n-1} |[f(x_{i+2}) - f(x_{i+1})] - [f(x_{i+1}) - f(x_i)]| \\ &= \sum_{i=1}^{n-1} |f'(\xi_{i+1}) - f'(\xi_i)| = \sum_{i=1}^{n-1} \left| \int_{\xi_i}^{\xi_{i+1}} f''(x) \, \mathrm{d}x \right| \le \|f''\|_1,\end{aligned}$$

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

The errors of the trapezoidal rule for integrands in the spaces $\mathcal{W}^{2,1}$ and $\mathcal{W}^{1,1}$ given in [1, (7.14) and (7.15)]. The derivations are repeated here for completeness. Integrating (43) leads to the formula for $h(n)$ defined in (22)

$$\begin{aligned}\int_0^1 f(x) \, dx - T_n(f) &= \int_0^1 [f - A_n(f)](x) \, dx \\ &= (n-1) \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} \int_{x_i}^{x_{i+1}} u_i(t,x) f''(t) \, \mathrm{d}t \, \mathrm{d}x \\ &= \frac{1}{2} \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} (t - x_i)(t - x_{i+1}) f''(t) \, \mathrm{d}t \, \mathrm{d}x \\ \left| \int_0^1 f(x) \, dx - T_n(f) \right| &\le \frac{1}{2} \sum_{i=1}^{n} \left[\sup_{x_i \le t \le x_{i+1}} |(t - x_i)(t - x_{i+1})| \int_{x_i}^{x_{i+1}} |f''(t)| \, \mathrm{d}t \, \mathrm{d}x \right] \\ &= \frac{1}{8(n-1)^2} \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} |f''(t)| \, \mathrm{d}t \, \mathrm{d}x = \frac{\|f''\|_1}{8(n-1)^2}, \\ h(n) &= \frac{1}{8(n-1)^2}.\end{aligned}$$

On the other hand, integrating (42) leads to the formula for $\tilde{h}(n)$ defined in (22):

$$\begin{aligned}
\int_0^1 f(x)\,dx - T_n(f) &= \int_0^1 [f - A_n(f)](x)\,dx \\
&= (n-1)\sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} \int_{x_i}^{x_{i+1}} v_i(t,x) f'(t)\,\mathrm{d}t\,\mathrm{d}x \\
&= \frac{1}{2}\sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} (x_i - 2t + x_{i+1}) f'(t)\,\mathrm{d}t\,\mathrm{d}x \\
\left|\int_0^1 f(x)\,dx - T_n(f)\right| &\le \frac{1}{2}\sum_{i=1}^{n} \left[\sup_{x_i \le t \le x_{i+1}} |x_i - 2t + x_{i+1}| \int_{x_i}^{x_{i+1}} |f'(t)|\,\mathrm{d}t\,\mathrm{d}x\right] \\
&= \frac{1}{2(n-1)}\sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} |f'(t)|\,\mathrm{d}t\,\mathrm{d}x = \frac{\|f'\|_1}{2(n-1)}.
\end{aligned}$$

The right hand side is not quite what is needed, but what we want can be obtained by replacing the integrand by the difference between it and its linear approximation.

$$\begin{aligned}
\left|\int_0^1 f(x)\,dx - T_n(f)\right| &= \left|\int_0^1 (f - A_2(f))(x)\,dx - T_n(f - A_2(f))\right| \\
&\le \frac{\|f' - A_2(f)'\|_1}{2(n-1)} = \frac{|f|_{\mathcal{G}}}{2(n-1)}, \\
\tilde{h}(n) &= \frac{1}{2(n-1)}.
\end{aligned}$$

By condition (29), it follows that $\min(\tilde{h}(n), \tau h(n)) = \tau h(n)$, which then simplifies some of the expressions in Algorithm 3 and Theorem 3. Specifically, the left side of the inequality in (30) becomes

$$\min(\tilde{h}(n_i), \tau h(n_i))\mathfrak{C}_{n_i} G_{n_i}(f) = \frac{\tau G_{n_i}(f)}{4(n_i - 1)(2n_i - 2 - \tau)}. \tag{47a}$$

The function N_A defined in (**??**) is

$$N_A(a) = \min\{n \in \mathcal{I} : \tau h(n) \le a\} = 1 + \left\lceil \sqrt{\tau/(8a)} \right\rceil. \tag{47b}$$

The denominator in the definition of the set of integrands $\mathcal{N}$ in (**??**) is

$$\begin{aligned}
&\mathfrak{C}_{N_{\max}/r}\mathfrak{c}_{N_{\max}/r} \min(\tilde{h}(N_{\max}/r), \tau h(N_{\max}/r)) \\
&\qquad = \frac{\tau}{2(N_{\max} - 2)(N_{\max} - 2 - \tau)}.
\end{aligned} \tag{47c}$$

The function $\widetilde{N}_A$ defined in (**??**) is

$$\begin{aligned}\widetilde{N}_A(a) &= \min\left\{n \in \mathcal{I} : \min(\tilde{h}(n), \tau h(n))\mathfrak{C}_n\mathfrak{c}_n \le a\right\} \\ &= 1 + \left\lceil \sqrt{\frac{\tau}{8a} + \frac{\tau^2}{16}} + \frac{\tau}{4} \right\rceil \le 2 + \frac{\tau}{2} + \sqrt{\frac{\tau}{8a}}. \end{aligned} \tag{47d}$$

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 τ be given inputs. Let the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, $\{G_n\}_{n\in\mathcal{I}}$ 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 $\|f''\|_1$. Compute $G_{n_i}(f)$ in (41) and $F_{n_i}(f)$ in (46).

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

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

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

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

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

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

Go to Stage 4.

Stage 4. Check whether n_{i+1} is within budget. If $n_{i+1} \le N_{\max}$, 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 6. *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 $n_1 \le N_{\max}$. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (40)*. Let*

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

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

$$\begin{aligned}
&\operatorname{maxcost}(A, \mathcal{N}, \varepsilon, N_{\max}, \|f' - f(1) + f(0)\|_1) \\
&\quad \le 2 + 2\left\lceil \sqrt{\frac{\tau \|f' - f(1) + f(0)\|_1}{8\varepsilon} + \frac{\tau^2}{16}} + \frac{\tau}{4} \right\rceil \\
&\quad \le 4 + \tau + \sqrt{\frac{\tau \|f' - f(1) + f(0)\|_1}{2\varepsilon}}, \\
&\operatorname{mincost}(A, \mathcal{N}, \varepsilon, N_{\max}, \|f' - f(1) + f(0)\|_1) \\
&\quad \ge 1 + \max\left(\frac{\tau + 1}{2}, \sqrt{\frac{\tau \|f' - f(1) + f(0)\|_1}{8\varepsilon}} \right).
\end{aligned}$$

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 $f_0(x) = x(1 - x)$. Then

$$|f_0|_{\mathcal{G}} = \|f_0' - f_0(1) + f_0(0)\|_1 = \int_0^1 |1 - 2x| \, \mathrm{d}x = \frac{1}{2},$$
$$|f_0|_{\mathcal{F}} = \|f_0''\|_1 = 2, \qquad \tau_0 = 4.$$

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

$$f_1(x) := \begin{cases} \dfrac{30(x - \xi_j)^2(\xi_{j+1} - x)^2}{(\xi_{j+1} - \xi_j)^5} & \xi_j \le x \le \xi_{j+1}, \\ 0 & \text{otherwise}, \end{cases}$$

has $\int_0^1 f_1(x) \, \mathrm{d}x = 1$ 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

$$|f_1|_{\mathcal{G}} = \|f_1' - f_1(1) + f_1(0)\|_1 = \|f_1'\|_1 = \frac{15}{4(\xi_{j+1} - \xi_j)} \le \tilde{g}(n),$$
$$|f_1|_{\mathcal{F}} = \|f_1''\|_1 = \frac{40}{\sqrt{3}(\xi_{j+1} - \xi_j)^2} = \frac{32 \|f_1'\|_1}{3\sqrt{3}(\xi_{j+1} - \xi_j)} \le g(n) \|f_1'\|_1,$$

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

$$\tilde{g}(n) = \frac{15(n+1)}{4}, \qquad g(n) = \frac{32(n+1)}{3\sqrt{3}}, \qquad (g\tilde{g})(n) = \frac{40(n+1)^2}{\sqrt{3}}.$$

Using these choices of f_0 and f_1, along with the corresponding $\tilde{g}$ and g above, one may invoke Proposition 4, Theorem 5, and Corollary 1 to obtain the following theorem.

Theorem 7. *The adaptive trapezoidal algorithm is optimal for integration of functions in $\mathcal{W}^{1,1}$ and $\mathcal{W}^{2,1}$. Assuming an infinite cost budget and for $\tau > 4$, the complexity of the integration problem, over the cone of functions $\mathcal{C}_\tau$ defined in* (40) *is bounded below as*

$$\begin{aligned}
&\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{H}, \mathrm{INT}, \Lambda), \infty, \|f' - f(1) + f(0)\|_1) \\
&\quad \geq -1 + \min\left(\frac{(\tau-4)\,\|f' - f(1) + f(0)\|_1}{15(\tau-2)\varepsilon}, \right. \\
&\qquad\qquad \left. \sqrt{\frac{\sqrt{3}\tau(\tau-4)\,\|f' - f(1) + f(0)\|_1}{160(\tau-2)\varepsilon}} \right) \\
&\quad \sim \sqrt{\frac{\sqrt{3}\tau(\tau-4)\,\|f' - f(1) + f(0)\|_1}{160(\tau-2)\varepsilon}} \quad \textit{as } \frac{\|f' - f(1) + f(0)\|_1}{\varepsilon} \to \infty.
\end{aligned}$$

Moreover, Algorithm 4 is optimal for approximating functions in $\mathcal{C}_\tau$.

FIX BELOW with new $\mathcal{G}$-semi-norm, A_n, T_n

5.3. Numerical Example

Consider the family of test functions defined by

$$f(x) = b\mathrm{e}^{-a^2(x-z)^2}, \tag{48}$$

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 1).

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

	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 1: Performance of multistage, adaptive Algorithm 4 for various values of τ, and also of other automatic quadrature algorithms. The test function (48) with random parameters, a and z, was used.

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.,

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

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 (39). The cone of functions for which our adaptive algorithms will be shown to work well is defined as

$$\mathcal{C}_\tau = \{f \in \mathcal{W}^{2,\infty} : \|f''\|_\infty \le \tau \|f'\|_\infty\}. \tag{49}$$

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 linear interpolation on evenly spaced points. Letting $\mathcal{I} = \{2, 3, \ldots\}$ as in the previous section, the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, is defined by

$$x_i = \frac{i-1}{n-1}, \qquad i = 1, \ldots, n, \tag{50a}$$

$$A_n(f)(x) = (n-1)\left[f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)\right], \qquad x_i \le x \le x_{i+1}. \tag{50b}$$

The cost of A_n, is the number of function evaluations, n. Using this same data one may approximate the $\mathcal{L}_\infty$ norm of f' by the algorithm

$$G_n(f) = (n-1) \sup_{i=1,\ldots,n-1} |f(x_{i+1}) - f(x_i)| . \tag{51}$$

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 $r = 2$.

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 (18) and $\tilde{h}$ and h defined in (22). 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 (20) and (21) may be taken to be

$$h_-(n) = 0, \quad h_+(n) = \frac{1}{2(n-1)}, \qquad \mathfrak{c}_n = 1, \quad \mathfrak{C}_n = \frac{1}{1 - \tau/(2n-2)}, \tag{52}$$

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'':

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

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

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

This implies that we may take

$$\tilde{h}(n) = \frac{1}{2(n-1)}, \qquad h(n) = \frac{1}{8(n-1)^2} \le \frac{1}{4(n-1)^2} = \tilde{h}(n)h_+(n). \tag{53}$$

Since $h_\pm(n), \tilde{h}(n)$, and $h(n)$, are the same as in the previous section for integration, the simplifications in 47 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 (51).

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 8. *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* (49)*. 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:*

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

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 $0 = \xi_0 \leq \xi_1 < \cdots < \xi_n \leq \xi_{n+1} = 1$. There must be some $j = 0, \ldots, n$ such that $\xi_{j+1} - \xi_j \geq 1/(n+1)$. The function f_1, defined as

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

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

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

where the inequality $\xi_{j+1} - \xi_j \geq 1/(n+1)$ 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 4, Theorem 5, and Corollary 1 to obtain the following theorem.

Theorem 9. *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 (49) *is*

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

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 (48), 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 2. Our algorithm succeeds for all functions lying in the cone plus some others. It is conservative, but not overly so.

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

Table 2: Comparison between the probability of the input function lying in the cone and the empirical success rate of Algorithm 5.

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 [18] 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.

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 (3) 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 cf, where c 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 cf. By definition a cone is just that set that contains cf for all numbers c 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 2, 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 3, 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

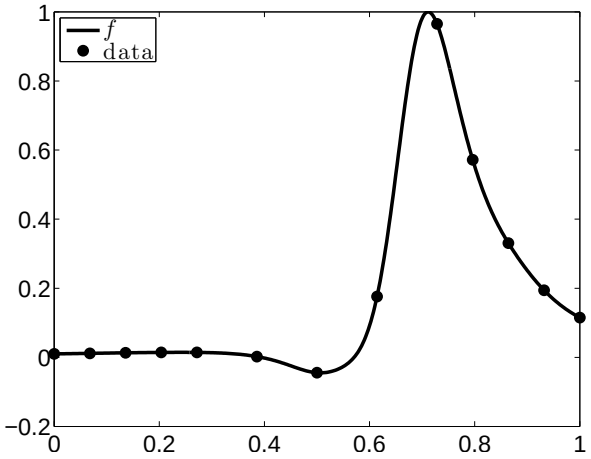

Figure 4: Integrand designed to fool MATLAB's `quad` along with the data used by `quad`.

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 4 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 4. 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 algorithms (e.g., see [19, 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 $\mathcal{G}$-semi-norms, where f_{in} lies in the interior of this cone, and f_{out} lies outside the cone. This means that

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

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

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

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

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

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

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

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.,

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

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 $\mathcal{G}$ 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 $\mathcal{G}$- 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 $\mathcal{G}$ 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 Numerical Algorithms Group, The NAG Library, Oxford, Mark 23 edition, 2012.

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

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

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

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

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