The Complexity of Deterministic Guaranteed Automatic Algorithms

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

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

Abstract

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 tolerance, ε. Furthermore, the computational effort is determined adaptively by the algorithm to match the difficulty of the problem. Unfortunately, many 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 providing rigorous guarantees for automatic algorithms. Sufficient conditions for success and upper bounds on the computational cost are provided in Theorems 1 and 2, and lower bounds on the complexity of the problem are given in Theorem 3. Examples of these general theorems are illustrated for univariate numerical integration and function recovery. The key observation is that the error analysis should be done for *cones* of input functions rather than balls. The existing literature contains certain cautions about the usefulness and reliability of automatic algorithms. The theory presented here does not share the assumptions on which those concerns are based, rendering them moot.

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

1. Introduction

Numerical algorithms for many elementary and special functions reliably provide answers to machine precision without user tuning. For example, the MATLAB [15] algorithms

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

automatically use the right number of terms in a polynomial, rational or other suitable expansion needed to provide the corresponding function value with 15

significant digit accuracy. The only exceptions are atypical cases of round-off error, e.g.,

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

whereas the correct answer is `1`.

For more complex numerical problems, such as integration and function recovery, it is desirable to have algorithms that require a minimum of user tuning. Namely, one would like to have algorithms that are

Guaranteed — answers are provided with an error no greater than the user-specified *error tolerance*, ε.

Automatic — the computational cost is determined without user intervention based on ε.

Adaptive — the computational cost for a given input is also determined based on the difficulty of the problem (size of the input function) *as inferred from data, not specified a priori*.

Optimal — the computational cost as a function of the user-specified ε and the unknown problem difficulty is no greater than a constant multiple times that of the best algorithm possible *even if the problem difficulty were known a priori*.

Tunable — parmeters defining the algorithm can be adjusted to change the algorithm's robustness, the maximum allowable computational cost budget, etc.

The checklist in Figure 1 provides an overview of how algorithms currently in use and those proposed here compare for these fiver criteria. This article presents a general framework for constructing algorithms that satisfy all of the above criteria and also provides 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` [15] and those in the NAG Library [16] work well often in practice, but they do not have theoretical guarantees of success. The Chebfun toolbox in MATLAB [6], which approximates functions by Chebyshev polynomial expansions also works weel for many cases, but has no guarantees. All of these algorithms do have the advantage of adjusting the computational effort based on function data, i.e, they are adaptive. The degree to which they are tunable varies.

Most existing theory for numerical problems of interes 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, $\mathcal{L}_\infty(\mathcal{X})$. Given this set-up,

	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.

one may often find an algorithm, A_n, with computational cost n, that provides an approximate solution whose error is bounded as follows:

$$\|S(f) - A_n(f)\|_{\mathcal{H}} \le \frac{C_{\text{up}}\,\|f\|_{\mathcal{F}}}{n^p}, \quad n \in \mathbb{N}. \tag{1a}$$

where C_{up} and p are some known constants. Here it is necessarily assumed that the algorithm is exact if the semi-norm of the input function vanishes, i.e., $S(f) = A(f)$ if $\|f\|_{\mathcal{F}} = 0$. This allows one to compute an approximation that differs from the true solution by no more than ε, provided that the input functions lie inside a *ball*, i.e.,

$$\sup_{f \in \mathcal{B}_\sigma} \|S(f) - A_n(f)\|_{\mathcal{H}} \le \varepsilon, \qquad \mathcal{B}_\sigma = \{f \in \mathcal{F} : \|f\|_{\mathcal{F}} \le \sigma\}, \tag{1b}$$

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

Furthermore, the computational cost of the algorithm, $\text{cost}(A_n)$, which typically corresponds to the number of function data needed, is bounded in terms of ε and the radius of the ball, σ.

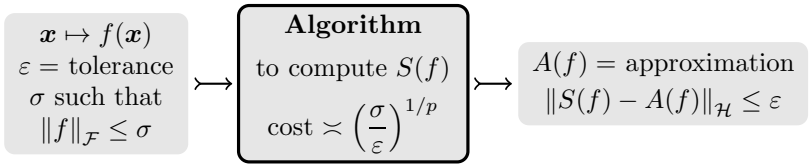

Figure 2: Diagram of a guaranteed, non-adaptive algorithm.

Figure 2 provides a diagram of an automatic algorithm based on these ideas. It 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 ε and σ.

This lack of adaptivity is coupled to the need to provide as input, σ, a known upper bound on $\|f\|_{\mathcal{F}}$, which might be called the *difficulty of the problem*. In

practice, it is not easy to know a priori how large $\|f\|_{\mathcal{F}}$ is, so the requirement of σ is a practical drawback of the algorithm in Figure 2. Moreover, since the computational cost of such this algorithm is tied to σ, it not decrease if $\|f\|_{\mathcal{F}}$ is drastically smaller than σ, which is quite possible in practice.

Adaptive automatics algorithms try to estimate $\|f\|_{\mathcal{F}}$, so that the computational cost can be adjusted accordingly. This is the approach taken here. Many automatic algorithms approximate $\|f\|_{\mathcal{F}}$ heuristically or rely on an error estimate which converges asymptotically as the number of function values tends to infinity. Such algorithms are not guaranteed to provide the answer to within the specified error tolerance. The aim here is to construct from data a *rigorous* upper bound on $\|f\|_{\mathcal{F}}$.

The key idea is to consider functions lying in a *cone* instead of a ball. Let $\mathcal{G}$ be some superspace of $\mathcal{F}$ with a weaker semi-norm, and define the cone

$$\mathcal{C}_\tau = \{f \in \mathcal{F} : \|f\|_{\mathcal{F}} \leq \tau \|f\|_{\mathcal{G}}\}. \tag{2}$$

In essence, functions inside the cone may have large norms, but their $\mathcal{G}$-semi-norm can be estimated with reasonable effort. The argument to obtain an approximation to the true solution, $S(f)$, with a guaranteed error estimate from data is outlined as follows:

- Since the $\mathcal{G}$-semi-norm is weaker than the $\mathcal{F}$-semi-norm, it can be approximated by some algorithm $G(f)$, with $0 \leq \|f\|_{\mathcal{G}} - G(f) \leq C_1 \|f\|_{\mathcal{F}}$ for some known C_1.
- The definition of the cone in (2) implies that $\|f\|_{\mathcal{G}} \leq G(f)/(1 - \tau C_1)$, provided that $C_1 < 1/\tau$.
- Applying the cone condition again leads to an upper bound on the $\mathcal{F}$-semi-norm: $\|f\|_{\mathcal{F}} \leq \tau G(f)/(1 - \tau C_1)$.
- This data-driven upper bound on $\|f\|_{\mathcal{F}}$ can be used with (1c) to guarantee that the error tolerance will be met using A_n with

$$n = \text{cost}(A_n) \leq \left\lceil \left(\frac{C_{\text{up}} \tau G(f)}{\varepsilon(1 - \tau C_1)} \right)^{1/p} \right\rceil.$$

- Furthermore, since $G(f)$ never overestimates the $\mathcal{G}$-semi-norm of f, one can be assured that the above criterion will be met for

$$n = \text{cost}(A_n) \leq \left\lceil \left(\frac{C_{\text{up}} \tau \|f\|_{\mathcal{G}}}{\varepsilon(1 - \tau C_1)} \right)^{1/p} \right\rceil.$$

Figure 3 is a diagram of the automatic algorithm just outlined. As one can see, the computational cost varies with $\|f\|_{\mathcal{G}}$ as one might expect, but $\|f\|_{\mathcal{G}}$ is not an input to the algorithm. Moreover, $\|f\|_{\mathcal{G}}$, which can be estimated directly, acts as a surrogate for $\|f\|_{\mathcal{F}}$.

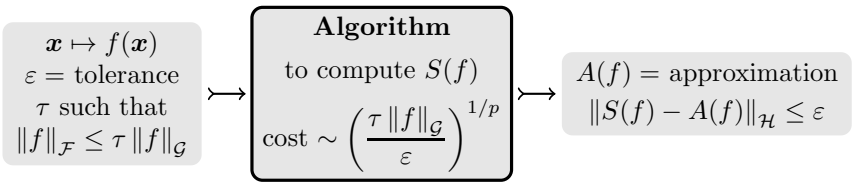

Figure 3: Diagram of a guaranteed, adaptive, automatic algorithm.

Section 4 presents the concrete example of this approach for the problem of computing the integral $\int_0^1 f(x)\,\mathrm{d}x$ for all integrands whose second derivatives are absolutely integrable, i.e., $\|f\|_{\mathcal{F}} = \|f''\|_1$. The trapezoidal rule computes an approximation with error no greater than ε using $\Theta(\sqrt{\|f''\|_1 / \varepsilon})$ function values. However, except for relatively simple integrands, $\|f''\|_1$ is unknown in practice. One may estimate instead $\|f'\|_1$ by the one-norm of the derivative of the piecewise linear spline for f. The error of approximating $\|f'\|_1$ in this way can be bounded rigorously in terms of $\|f''\|_1$. If one chooses a $\tau > 1$ and assumes that $\|f''\|_1 \leq \tau \|f'\|_1$ (the cone condition), then the reliable numerical upper bound on $\|f'\|_1$ can be used as a surrogate for $\|f''\|_1$. This leads to a guaranteed, automatic (adaptive) algorithm for approximating the integral to the desired accuracy.

This article proceeds from the abstract to the concrete. Section 2 lays out the problems to be solved, including how to define the cost of an adaptive, automatic algorithm and the computational complexity of a problem. 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. Section 3 describes the automatic algorithms in detail and provides proofs of their success for cones of input functions. Their cost is derived in terms of the unknown, but estimated, $\mathcal{G}$-semi-norm of the input function. Lower bounds on the complexity of the problem are derived using fooling functions. Section 4 illustrates these theorems for the univariate integration problem. Section 5 considers function approximation. Common concerns about automatic and adaptive algorithms are answered in Section 6. 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 of possible outputs or solutions with norm $\|\cdot\|_{\mathcal{H}}$. It is assumed that the $\mathcal{F}_0 = \{f \in \mathcal{F} : \|f\|_{\mathcal{F}} = 0\}$, the subspace of $\mathcal{F}$ containing functions with vanishing $\mathcal{F}$-semi-norm, has finite dimension. The solution operator is assumed

to have a scale property, i.e.,

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

Examples include the following:

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

The first three examples above are linear problems, which automatically have the scale property for S, 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 [17, Section 3.2], the algorithm takes the form of some function of data derived from the input function:

$$A(f) = \phi(\boldsymbol{L}(f)), \quad \boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f)) \qquad \forall f \in \mathcal{F}. \tag{3}$$

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

$$L(cf) = cL(f) \qquad \forall f \in \mathcal{F},\ c \in \mathbb{R},\ L \in \Lambda. \tag{4}$$

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

$$\boldsymbol{L}(cf) = c\boldsymbol{L}(f), \quad \phi(c\boldsymbol{y}) = c\phi(\boldsymbol{y}), \quad A(cf) = cA(f) \qquad \forall c \geq 0,\ \boldsymbol{y} \in \mathbb{R}^m. \tag{5}$$

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

$$\mathrm{cost}(A) = \$(\boldsymbol{L}) = \$(L_1) + \cdots + \$(L_m), \tag{6}$$

where $\$: \Lambda \to (0, \infty)$, and $\$(L)$ is the cost of acquiring the datum $L(f)$. The cost of L may be the same for all $L \in \Lambda$. Alternatively, it might vary with the

choice of L. E.g., 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 [8, 9, 11, 12, 13] for integration problems and [18, 19, 20] for function approximation problems.

The error of a non-adaptive algorithm $A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)$ is defined as

$$\text{err}(A, \mathcal{F}, \mathcal{H}, S) = \min\{\delta \geq 0 : \|S(f) - A(f)\|_{\mathcal{H}} \leq \delta \|f\|_{\mathcal{F}} \ \forall f \in \mathcal{F}\}, \quad (7)$$

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

$$\text{err}_{\pm}(A, \mathcal{F}, \mathbb{R}, S) = \min\{\delta \geq 0 : \pm[S(f) - A(f)] \leq \delta \|f\|_{\mathcal{F}} \ \forall f \in \mathcal{F}\} \quad (8)$$

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

The above error criteria are normalized, meaning that the absolute error, $\|S(f) - A(f)\|_{\mathcal{H}}$ is measured with respect to the $\mathcal{F}$-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:

$$\begin{aligned}&\text{comp}(\varepsilon, \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)) \\ &\qquad = \inf\left\{\text{cost}(A) : \text{err}(A, \mathcal{F}, \mathcal{H}, S) \leq \varepsilon, \ A \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda)\right\}. \quad (9)\end{aligned}$$

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

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

This cost does not decrease 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:

$$\{A_n\}_{n \in \mathcal{I}}, \qquad A_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda), \quad (10a)$$

$$\lim_{\substack{n \to \infty \\ n \in \mathcal{I}}} \text{err}(A_n, \mathcal{F}, \mathcal{H}, S) = 0, \qquad \text{cost}(A_n) = n, \quad (10b)$$

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

$$\mathcal{I} = \{n_1, n_2, \ldots\} \quad \text{with } n_i < n_{i+1}, \quad \text{satisfies } \sup_i \frac{n_{i+1}}{n_i} < \infty. \quad (11)$$

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

$$\sup_{0 < \varepsilon \leq 1} \frac{\min\{n \in \mathcal{I} : \text{err}(A_n, \mathcal{F}, \mathcal{H}, S) \leq \varepsilon\} \varepsilon^p}{\text{comp}(\varepsilon, \mathcal{A}_{\text{non}}(\mathcal{F}, \mathcal{H}, S, \Lambda))} < \infty, \qquad \forall p > 0. \quad (12)$$

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

2.3. Automatic, Adaptive Algorithms

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

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

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

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

As in (3), the algorithm takes the form of some function of function data: $A(f; \varepsilon, N_{\max}) = \phi\left(\boldsymbol{L}(f); \varepsilon, N_{\max}\right)$. Now, however, the algorithm is allowed to be adaptive. The choice of L_2 may depend on the value of $L_1(f)$, the choice of L_3 may depend on $L_1(f)$ and $L_2(f)$, etc. The number of function data used by the algorithm, m, may also be determined adaptively. The choice of how many and which function data to use depends on ε and $N_{\max}$. 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 $\|S(f) - A(f; \varepsilon, N_{\max})\|_{\mathcal{H}} \leq \varepsilon$, but this is not a requirement of the definition.

The cost of the algorithm for a specified input function is defined analogously to (6) as the sum of the costs of all function data.

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

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 $\text{cost}(A, f; \varepsilon, N_{\max}) \leq N_{\max}$ for all f and ε. This is a practical consideration since the user does not want to wait indefinitely for an answer.

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

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

Here the set $\mathcal{N}$ is allowed to depend on the algorithm inputs, ε and $N_{\max}$, but not on σ. Moreover, the algorithm, A, does not take σ as an input.

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 for $\mathcal{N} \subseteq \mathcal{G}$, denoted $(A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)$, is deemed successful provided that it meets the prescribed error tolerance and does not raise the warning flag. Specifically, success is defined as

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

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 cost of the cheapest successful algorithm for input functions with $\mathcal{G}$-semi-norm no greater than σ:

$$\begin{aligned}&\operatorname{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda), N_{\max}, \sigma) \\ &\qquad = \inf \{\operatorname{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) : \operatorname{succ}(A, W, \mathcal{N}, \varepsilon, N_{\max}) = \text{true}, \\ &\qquad\qquad\qquad\qquad (A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)\}\,. \quad (13)\end{aligned}$$

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

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 (1) rules out fooling functions with large error by restricting the size of $\|f\|_{\mathcal{F}}$.

As mentioned above, 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 estimated reliably. 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:

$$\min_{f_0 \in \mathcal{F}_0} \|f - f_0\|_{\mathcal{G}} \leq C_{\mathcal{F}} \|f\|_{\mathcal{F}} \qquad \forall f \in \mathcal{F}, \tag{14a}$$

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

$$\mathcal{G}_0 \subseteq \mathcal{F}_0. \tag{14b}$$

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 (2). For any $f \in \mathcal{C}_\tau$ both $\|f\|_{\mathcal{F}}$ and $\|f\|_{\mathcal{G}}$ must be finite, but they can be arbitrarily large. There is no need to assume an upper bound on their sizes, but it is possible to obtain reliable upper bounds for both $\|f\|_{\mathcal{F}}$ and $\|f\|_{\mathcal{G}}$ by sampling f. An upper bound on $\|f\|_{\mathcal{G}}$ for $f \in \mathcal{C}_\tau$ can be computed in terms of the data $\boldsymbol{L}(f) = (L_1(f), \ldots, L_m(f))$ because $\|\cdot\|_{\mathcal{F}}$ is a stronger semi-norm than $\|\cdot\|_{\mathcal{G}}$ in the sense of (14a), 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 1 below).

2.5. Results that One Wishes to Prove

The previous subsections define the problem to be approximated and the notation describing the difficulty of the problem and the efficiency of the algorithms. This subsection summarizes the results that are proved in general in the next section and illustrated for specific cases in the Sections 4 and 5.

i. *Upper bound on the complexity.* One wishes to bound the complexity of solving the problem successfully, $\mathrm{comp}(\mathcal{N}, \varepsilon, N_{\max}, \sigma, \Lambda)$, in terms of some power of ε/σ for $\mathcal{N}$ suitably defined as a subset of the cone $\mathcal{C}_\tau$. This is done in Theorems 1 and 2.

ii. *An algorithm that achieves the upper bound.* Upper bounds on the complexity are sometimes found in a non-constructive way. However, it is desirable to identify an explicit successful algorithm, $(A, W) \in \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda)$, that achieves these upper bounds. This is done in Algorithms 1 and 2.

iii. *Penalty for not knowing the $\mathcal{F}$- and $\mathcal{G}$-semi-norms of f.* The optimal successful algorithm must find an upper bound on $\|f\|_{\mathcal{F}}$ or $\|f\|_{\mathcal{G}}$ rather than assuming such an upper bound. One hopes that the extra cost relative to the situation of knowing a priori bounds on these semi-norms is not too great. Positive results are given in Theorems 1 and 2.

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

3. General Theorems

This section provides rather general theorems about the complexity of automatic algorithms. In some sense, these theorems are a roadmap or an outline because their assumptions are non-trivial and require effort to be verify for specific problems of interest. On the other hand, the assumptions are reasonable as is demonstrated in the Sections 4 and 5 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 non-adaptive algorithm $G_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$, where $n = \text{cost}(G_n)$, provided that one has explicit upper bounds on the errors of these algorithms. Namely, there should be non-increasing functions $h_\pm$ defined on some subset of the non-negative numbers such that

$$\text{err}_-(G_n, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}) = \sup_{f \neq 0} \frac{G_n(f) - \|f\|_{\mathcal{G}}}{\|f\|_{\mathcal{F}}} \leq h_-(n) \tag{15a}$$

$$\text{err}_+(G_n, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}) = \sup_{f \neq 0} \frac{\|f\|_{\mathcal{G}} - G_n(f)}{\|f\|_{\mathcal{F}}} \leq h_+(n). \tag{15b}$$

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

Lemma 1. *Any nonadaptive algorithm $G_n \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$ with cost $n = \text{cost}(G_n)$ and two sided error bounds as in* (15) *yields an approximation to the $\mathcal{G}$-semi-norm of functions in the cone $\mathcal{C}_\tau$ with the following upper and lower error bounds:*

$$\frac{G_n(f)}{\mathfrak{c}_n} \leq \|f\|_{\mathcal{G}} \leq \mathfrak{C}_n G_n(f) \qquad \forall f \in \mathcal{C}_\tau. \tag{16}$$

where the deflation factor, $1/\mathfrak{c}_n$, and the inflation factor, $\mathfrak{C}_n$ are defined as follows:

$$\mathfrak{c}_n = 1 + \tau h_-(n) \geq 1 \tag{17}$$

$$\mathfrak{C}_n = \frac{1}{1 - \tau h_+(n)} \geq 1, \qquad \textit{assuming } h_+(n) < 1/\tau. \tag{18}$$

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 (10), and for which upper error bounds are known for both the spaces $\mathcal{G}$ and $\mathcal{F}$:

$$\text{err}(A_n, \mathcal{G}, \mathcal{H}, S) \leq \tilde{h}(n), \qquad \text{err}(A_n, \mathcal{F}, \mathcal{H}, S) \leq h(n), \tag{19}$$

for some *non-increasing* functions $\tilde{h}$ and h. The definitions of these errors in (7) then implies upper bounds on the error of $A_n(f)$ in terms of the $\mathcal{G}$-semi-norm of f:

$$\begin{aligned} \|S(f) - A_n(f)\|_{\mathcal{H}} &\leq \min(\text{err}(A_n, \mathcal{G}, \mathcal{H}, S) \|f\|_{\mathcal{G}}, \text{err}(A_n, \mathcal{F}, \mathcal{H}, S) \|f\|_{\mathcal{F}}) \\ &\leq \min(\tilde{h}(n), \tau h(n)) \|f\|_{\mathcal{G}} \qquad \forall f \in \mathcal{C}_\tau. \end{aligned} \tag{20}$$

Algorithm 1. (Automatic, Adaptive, Two-Stage). Let $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ be Banach spaces as described above, let S be the solution operator, let ε be a positive error tolerance, and let $N_{\max}$ be the maximum cost allowed. Let τ be a fixed positive number, and let $G_{n_G} \in \mathcal{A}_{\text{non}}(\mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$ be an algorithm as described in Lemma 1 with cost n_G satisfying $h_+(n_G) < 1/\tau$. Moreover, let $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, be a sequence of algorithms as described in (10) and (19). Given 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 (18). 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, $n_A = N_A(\varepsilon/(\mathfrak{C}G_{n_G}(f)))$, where

$$N_A(a) = \min\left\{n \in \mathcal{I} : \min(\tilde{h}(n), \tau h(n)) \le a\right\}, \quad a \in (0, \infty). \tag{21}$$

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. Compute $A_{n_A}(f)$ as the approximation to $S(f)$.

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

Theorem 1. *Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, ε, $N_{\max}$, $\widetilde{N}_{\max}$, $\mathfrak{C}$, and τ be given as described in Algorithm 1, and assume that $\mathcal{F}$ satisfies* (14). *Let $\mathfrak{c} = \mathfrak{c}_{n_G}$ be defined as in* (17). *Let $\mathcal{C}_\tau$ be the cone of functions defined in* (2) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\mathcal{G}$-semi-norms. Let*

$$\begin{aligned}\mathcal{N} &= \left\{f \in \mathcal{C}_\tau : N_A\left(\frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\,\|f\|_{\mathcal{G}}}\right) \le \widetilde{N}_{\max}\right\} \\ &= \left\{f \in \mathcal{C}_\tau : \|f\|_{\mathcal{G}} \le \frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\min(\tilde{h}(\widetilde{N}_{\max}), \tau h(\widetilde{N}_{\max}))}\right\}\end{aligned} \tag{22}$$

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

$$\text{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) \le n_G + N_A\left(\frac{\varepsilon}{\mathfrak{C}\mathfrak{c}\sigma}\right). \tag{23}$$

The upper bound on the cost of this specific algorithm provides an upper bound on the complexity of the problem, $\text{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda), N_{\max}, \sigma)$.

Consider the limit of infinite cost budget, i.e., $N_{\max} \to \infty$. If the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is nearly optimal for the problems

$(\mathcal{G}, \mathcal{H}, S, \Lambda)$ *and* $(\mathcal{F}, \mathcal{H}, S, \Lambda)$ *as defined in* (12)*, then Algorithm 1 does not incur a significant penalty for not knowing* $\|f\|_{\mathcal{G}}$ *a priori, i.e., for all* $p > 0$*,*

$$\sup_{0<\varepsilon/\sigma\le 1} \frac{\operatorname{cost}(A, \mathcal{C}_\tau, \varepsilon, \infty, \sigma)}{\operatorname{comp}(\varepsilon/\sigma, \mathcal{A}_{\text{non}}(\mathcal{J}, \mathcal{H}, S, \Lambda))} \left(\frac{\varepsilon}{\sigma}\right)^p < \infty, \qquad \mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}. \tag{24}$$

Proof. The definition of $\mathfrak{C}$ in (18) implies that the true $\mathcal{G}$-semi-norm of f is bounded above by $\mathfrak{C}G_{n_G}(f)$ according to Lemma 1. The upper bound on the error of the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$ in (20) then implies that

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

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

To ensure that the algorithm does not attempt to overrun the cost budget, one must limit the $\mathcal{G}$-semi-norm of the input function. The definition of $\mathfrak{c}$ in (17) implies that $G_{n_G}(f) \le \mathfrak{c}\,\|f\|_{\mathcal{G}}$ according to Lemma 1. This means that for any function, f, with actual $\mathcal{G}$-semi-norm $\sigma = \|f\|_{\mathcal{G}}$, the upper bound on its $\mathcal{G}$-semi-norm computed via Lemma 1 is no greater than $\mathfrak{C}\mathfrak{c}\sigma$. Thus, after using N_G samples to estimate $\|f\|_{\mathcal{G}}$, functions in $\mathcal{N}$ as defined in (22) never need more than $N_{\max} - n_G$ additional samples to estimate $S(f)$ with the desired accuracy. This establishes that Algorithm 1 must be successful for all $f \in \mathcal{N}$. It furthermore establishes an upper bound on the cost of the algorithm as given in (23).

Now consider the penalty for not knowing $\|f\|_{\mathcal{G}}$ in advance. If the sequence of nonadaptive algorithms, $\{A_n\}_{n\in\mathcal{I}}$, used to construct Algorithm 1 are nearly optimal for solving the problem on both $\mathcal{F}$ and $\mathcal{G}$, as defined in (12), then it follows that for $\mathcal{J} \in \{\mathcal{F}, \mathcal{G}\}$,

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

The first of these suprema is finite by comparing the convergence rates of the sequence algorithms, $\{A_n\}_{n\in\mathcal{I}}$, in (19) with the cost of the automatic algorithm given by (23). The second of these suprema is finite for all $p > 0$ by the near optimality of $\{A_n\}_{n\in\mathcal{I}}$. □

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

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

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

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

Remark 2. If $\tilde{h}$ is unknown, then one may take $\tilde{h}(n) = \infty$, and the algorithm still satisfies the error tolerance with a cost upper bound given in (23). The optimality result in (24) then only holds for $\mathcal{F}$, and not $\mathcal{G}$. The analogy holds if h is unknown. However, at least one of these two functions $\tilde{h}$ or h, must be known for this theorem to have a meaningful result.

Remark 3. The cost of Algorithm 1, as given by (23), depends on σ, which is essentially the $\mathcal{G}$-semi-norm of the input function, f. Thus, if σ is smaller, the cost will correspondingly be smaller. Moreover, σ is not an input parameter for the algorithm. Rather the algorithm reliably estimates $\|f\|_{\mathcal{G}}$, and then adjusts the number of samples used (the cost) accordingly.

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

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

$$\tau = \left(1 - \frac{1}{\mathfrak{C}}\right) \frac{1}{h_+(n_G)}, \tag{25}$$

which is equivalent to (18).

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

$$\tau h(n) \le \tilde{h}(n) \quad \forall n \in \mathcal{I}. \tag{26}$$

or equivalently, $\min(\tilde{h}(n), \tau h(n)) = \tau h(n)$. This then simplifies Algorithm 1 in the computation of the sample size for A in (21) and also in Theorem 1 in the definition of $\mathcal{N}$ in (22) and the upper bound on the cost in (23).

A sufficient condition for (26) is

$$h(n) \le \tilde{h}(n) h_+(n) \quad \forall n \in \mathcal{I}, \tag{27}$$

since the Algorithms 1 and 2 both require that $h_+(n) < 1/\tau$, and since h is a non-increasing function. Another sufficient condition for (26) is finding some

sequence of functions $\{f_n^* \in \mathcal{F}\}_{n\in\mathcal{I}}$ with non-zero $\mathcal{F}$- and $\mathcal{G}$-semi-norms such that

$$h(n) \le \frac{\|S(f_n^*) - A_n(f_n^*)\|_{\mathcal{H}}}{\|f_n^*\|_{\mathcal{G}}} \frac{\|f_n^*\|_{\mathcal{G}} - G_n(f_n^*)}{\|f_n^*\|_{\mathcal{F}}} \quad \forall n \in \mathcal{I}, \tag{28}$$

since the right hand side above is no greater than the right hand side in (27). An advantage of (28) is that one does not need to find $\tilde{h}$ explicitly. Finally, in the case where the data used for G_n and A_n are the same, and where the f_n^* are constructed such that all their data all vanish, then the sufficient condition above reduces to

$$h(n) = \frac{\|S(f_n^*)\|_{\mathcal{H}}}{\|f_n^*\|_{\mathcal{F}}} \quad \forall n \in \mathcal{I}. \tag{29}$$

The inequality now becomes an equality because the left hand side cannot be smaller than the right hand side by the definition of h.

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, the algorithm used to estimate the $\mathcal{G}$-semi-norm of f, is the same data used by A_{n_1}, and so $n_1 = n_G$. Then the total cost of Algorithm 1 can be reduced; it is simply n_A, as given in Stage 2, instead of $n_G + n_A$. Moreover, $\widetilde{N}_{\max}$ may then be taken to be $N_{\max}$, and the cost bound of the automatic algorithm in (23) does not need the term n_G.

Again suppose that $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$, consists of algorithms as described in (10) and (19), but some of which are embedded in others. Specifically, suppose that for some fixed $r > 1$, for all $n \in \mathcal{I}$, there exists at least one $\tilde{n} \in \mathcal{I}$ with $n < \tilde{n} \le rn$, such that the data for A_n is embedded in the data for $A_{\tilde{n}}$. An example would be all possible composite trapezoidal rules for integration that use trapezoids of equal widths. Suppose also that there exists a sequence of algorithms for approximating the $\mathcal{G}$-sem-inorm, $\{G_n\}_{n\in\mathcal{I}}$, $G_n \in \mathcal{A}_{\text{non}}(\mathcal{G}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}}, \Lambda)$, such that for each $n \in \mathcal{I}$, A_n and G_n use exactly the same data. Moreover, define $\mathfrak{c}_n$ and $\mathfrak{C}_n$ in terms of G_n as in (17) and (18) for all $n \in \mathcal{I}$. It is assumed that $\text{err}_{\pm}(G_n, \mathcal{F}, \mathbb{R}_+, \|\cdot\|_{\mathcal{G}})$ are non-increasing functions of n, which implies that $\mathfrak{C}_n$ and $\mathfrak{c}_n$ do not increase as n increases. These embedded algorithms suggest the following iterative algorithm.

Algorithm 2. Let the Banach spaces $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$, the solution operator S, and the error tolerance ε, the maximum cost budget $N_{\max}$, and the positive constant τ be as described in Algorithm 1. 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 be the smallest number $n \in \mathcal{I}$ satisfying $h_+(n) < 1/\tau$. For any input function $f \in \mathcal{F}$, do the following:

Stage 1. Estimate $\|f\|_{\mathcal{G}}$. Compute $G_{n_i}(f)$ and $\mathfrak{C}_{n_i} \leq \infty$ as defined in (18).

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

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

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 the inequality above fails to hold, compute $\mathfrak{c}_{n_i}$ according to (17) using G_{n_i}. Choose n_{i+1} as the smallest number exceeding n_i and not less than $N_A(\varepsilon \mathfrak{c}_{n_i}/G_{n_i}(f))$ 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. The following theorem is analogous to Theorem 1.

Theorem 2. *Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, ε, $N_{\max}$, τ, and n_1 be given as described in Algorithm 2. Assume that $n_1 \leq N_{\max}$. Let r be the number described in the paragraph preceding that algorithm. Define*

$$\widetilde{N}_A(a) = \min\left\{n \in \mathcal{I} : \min(\tilde{h}(n), \tau h(n))\mathfrak{C}_n \mathfrak{c}_n \leq a\right\}, \quad a \in (0, \infty).$$

Let $\mathcal{C}_\tau$ be the cone of functions defined in (2) *whose $\mathcal{F}$-semi-norms are no larger than τ times their $\mathcal{G}$-semi-norms. Let*

$$\begin{aligned}\mathcal{N} &= \left\{f \in \mathcal{C}_\tau : r\widetilde{N}_A\left(\frac{\varepsilon}{\|f\|_{\mathcal{G}}}\right) \leq N_{\max}\right\} \\ &= \left\{f \in \mathcal{C}_\tau : \|f\|_{\mathcal{G}} \leq \frac{\varepsilon}{\mathfrak{C}_{N_{\max}/r}\mathfrak{c}_{N_{\max}/r}\min(\tilde{h}(N_{\max}/r), \tau h(N_{\max}/r))}\right\}\end{aligned} \tag{31}$$

be the nice subset of the cone $\mathcal{C}_\tau$. Then it follows that Algorithm 2 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 in terms of the $\mathcal{G}$-semi-norm of the input function as follows:*

$$\mathrm{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) \leq \max\left(n_1, r\widetilde{N}_A\left(\frac{\varepsilon}{\sigma}\right)\right). \tag{32}$$

The upper bound on the cost of this specific algorithm provides an upper bound on the complexity of the problem, $\mathrm{comp}(\varepsilon, \mathcal{A}(\mathcal{N}, \mathcal{H}, S, \Lambda), N_{\max}, \sigma)$.

Consider the limit of infinite cost budget, i.e., $N_{\max} \to \infty$. If the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, $A_n \in \mathcal{A}_{\mathrm{non}}(\mathcal{G}, \mathcal{H}, S, \Lambda)$ is nearly optimal for the problems

$(\mathcal{G}, \mathcal{H}, S, \Lambda)$ *and* $(\mathcal{F}, \mathcal{H}, S, \Lambda)$ *as defined in* (12)*, then Algorithm 2 does not incur a significant penalty for not knowing* $\|f\|_{\mathcal{G}}$ *a priori, i.e., for all* $p > 0$*,*

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

Proof. Let $n_1, \ldots, n_j$ be the sequence of n_i generated by Algorithm 2, j being the number of the iterate where the algorithm either

i) terminates 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 2 guarantees that $n_1 < \cdots < n_j$. It is shown that under the hypotheses of this theorem, the algorithm does terminate without a warning and the error tolerance is satisfied.

First, consider possibility i) and recall inequality (16). 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 1. 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 $n_{j-1} < \widetilde{N}_A(\varepsilon/\|f\|_{\mathcal{G}})$, since $\tilde{h}(n), h(n), \mathfrak{C}_n$, and $\mathfrak{c}_n$ all do not decrease as n increases. 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

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

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, (16), and the definition of $\widetilde{N}_A$, it follows that

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

In both cases, the algorithm chooses $n_j < r\widetilde{N}_A(\varepsilon/\|f\|_{\mathcal{G}})$. Thus, the cost of the algorithm is bounded as in (32).

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

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

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

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

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 1. This completes the proof. □

3.4. Lower Complexity Bounds for the Algorithms

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

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

$$\|S(f_1)\|_{\mathcal{H}} = 1, \quad \boldsymbol{L}(f_1) = \mathbf{0}, \quad \|f_1\|_{\mathcal{G}} \le g(n), \quad \|f_1\|_{\mathcal{F}} \le \tilde{g}(n)\,\|f_1\|_{\mathcal{G}}\,, \tag{33}$$

for some positive, non-decreasing functions g and $\tilde{g}$ defined in $(0, \infty)$. For example, one might have $g(n) = an^p$ and $\tilde{g}(n) = bn^q$ with positive a, b, p, and q. Since the data for the function f_1 are all zero, it follows that $A(f_1) = A(-f_1)$ for any algorithm, A, adaptive or not, that is based on the information $\boldsymbol{L}(f_1)$. Then, by the triangle inequality and (33) the error for one of the fooling functions $\pm f_1$ must be at least one:

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

Furthermore, applying (33), 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:

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

This implies lower bounds on the complexity of nonadaptive algorithms, as defined in (9):

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

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

Turning to the problem of solving functions in the cone $\mathcal{C}_\tau$, the lower bound on the complexity becomes a bit more difficult to derive. Note that condition (33) allows the fooling function f_1 to lie *outside* this cone for $bn^q > \tau$. Thus, when considering the cone of input functions, the fooling function must be modified as described below.

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

$$\|f_0\|_{\mathcal{G}} > 0, \qquad \|f_0\|_{\mathcal{F}} \leq \tau_0 \|f_0\|_{\mathcal{G}}, \qquad \tau_0 < \tau. \tag{34}$$

Furthermore, suppose that for each $n > 0$, and for all $\boldsymbol{L} \in \Lambda^m$, satisfying $\$(\boldsymbol{L}) \leq n$, there exists f_1 as described above in (33). Under these assumptions, one may show the following lower bound on the complexity of solving the problem S for functions in the cone $\mathcal{C}_\tau$.

Theorem 3. *Suppose that functions f_0 and f_1 can be found that satisfy conditions* (33) *and* (34)*. It then follows that the complexity of the problem, defined by* (13)*, assuming infinite cost budget, over the cone of functions $\mathcal{C}_\tau$ is*

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

Proof. Let A be a successful, possibly adaptive, algorithm for all functions lying in the cone $\mathcal{C}_\tau$. Given an error tolerance, ε, and a positive σ, let f_0 be a function satisfying (34) and choose

$$c_0 = \frac{\sigma\tau}{\|f_0\|_{\mathcal{G}} (2\tau - \tau_0)}. \tag{35a}$$

Provide the algorithm A with the input function $c_0 f_0$, and let $\boldsymbol{L}(c_0 f_0) = c_0 \boldsymbol{L}(f_0)$ be the data vector extracted by A to obtain the estimate $A(c_0 f_0)$. Let $n = \$(\boldsymbol{L})$ denote the cost of this algorithm for the function $c_0 f_0$, and define two fooling functions, $f_\pm = c_0 f_0 \pm c_1 f_1$, in terms of the f_0 and f_1 satisfying conditions (33) with c_1 satisfying

$$c_1 = \frac{(\tau - \tau_0) c_0 \|f_0\|_{\mathcal{G}}}{\|f_1\|_{\mathcal{G}} [\tilde{g}(n) + \tau]} = \frac{\sigma\tau(\tau - \tau_0)}{\|f_1\|_{\mathcal{G}} (2\tau - \tau_0)[\tilde{g}(n) + \tau]}. \tag{35b}$$

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

$$\begin{aligned}
\|f_\pm\|_{\mathcal{F}} - \tau \|f_\pm\|_{\mathcal{G}} &\le c_0 \|f_0\|_{\mathcal{F}} + c_1 \|f_1\|_{\mathcal{F}} - \tau(c_0 \|f_0\|_{\mathcal{G}} - c_1 \|f_1\|_{\mathcal{G}}) \\
&\qquad\qquad \text{by the triangle inequality} \\
&\le c_1[\tilde{g}(n) + \tau] \|f_1\|_{\mathcal{G}} - (\tau - \tau_0) c_0 \|f_0\|_{\mathcal{G}} \qquad \text{by (33), (34)} \\
&= 0 \qquad\qquad \text{by (35).}
\end{aligned}$$

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

$$\begin{aligned}
\|f_\pm\|_{\mathcal{G}} &\le c_0 \|f_0\|_{\mathcal{G}} + c_1 \|f_1\|_{\mathcal{G}} \\
&= \frac{\sigma\tau}{2\tau - \tau_0} \left[1 + \frac{\tau - \tau_0}{\tilde{g}(n) + \tau}\right] \qquad \text{by (35)} \\
&\le \frac{\sigma\tau}{2\tau - \tau_0} \left[1 + \frac{\tau - \tau_0}{\tau}\right] = \sigma.
\end{aligned}$$

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,

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

Since A is successful for these two fooling functions, c_1, as defined in (35), must be no larger than the error tolerance, which implies by (33) that

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

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

Remark 7. Now suppose that the sequence of nonadaptive algorithms used to construct the adaptive, automatic Algorithm 1, and the fooling functions in Theorem 3 have comparable powers, namely $p_1 = p$ and $p_2 = p + q$. It then follows by comparing the upper bound on the cost in Theorem 1 to the lower bound in Theorem 3 that Algorithm 1 is optimal.

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 upper bound error functions, $\tilde{h}$ and h, defined in (19),
- 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 deflation and inflation factors, $\mathfrak{c}_n$ and $\mathfrak{C}_n$, which are defined in (17) and (18) 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 (33) and (34).

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

4. Approximation of One-Dimensional Integrals

In the following cases, we focus on approximation of upper bounds for truncation errors. The input function is in Sobolev space:

$$\begin{aligned}\mathcal{F} = W^{2,1} = W^{2,1}[0,1] &= \{f \in C^1[0,1] : \|f''\|_1 < \infty\}, \text{ and}\\ \mathcal{G} = W^{1,1} = W^{1,1}[0,1] &= \{f \in C^1[0,1] : \|f'\|_1 < \infty\},\end{aligned}$$

which have the semi-norm $\|f\|_{\mathcal{F}} = \|f''\|_1$ and $\|f\|_{\mathcal{G}} = \|f'\|_1$ respectively. The space of outputs $\mathcal{H}$ is the real space $\mathbb{R}$. Given $\tau > 0$, we have the cone $\mathfrak{C}_\tau \subset \mathcal{F}$ defined as

$$\mathfrak{C}_\tau = \{f \in W^{2,1} : \|f''\|_1 \le \tau\|f'\|_1\}.$$

To approximate the integrations of functions in the cone based on the function values, Define the solution operator S as $S(f) = \int_0^1 f(x)dx$. The goal is to make the error small, i.e. to find an algorithm A, for which $|\int_0^1 f(x)dx - A(f)| \le \varepsilon$.

We use the trapezoidal rule to approximate the integral. Consider a sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, where $\mathcal{I} = \{2, 3, \cdots\}$ with $\text{cost}(A_n) = n$, given the data $f(x_i)$, where $x_i = (i-1)/(n-1), i = 1, \cdots, n$ are uniformly distributed between $[0, 1]$, define $A_n(f)$ as:

$$\begin{aligned}A_n(f) &= \frac{1}{2n}\sum_{i=1}^{n}(f(x_{i+1}) + f(x_i)),\\ &= \frac{1}{2n}[f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)].\end{aligned}$$

According to (7.14) and (7.15) from [1], the upper bound error function can be expressed as $h(n) = 1/[2(n-1)]$, $\tilde{h}(n) = 1/[8(n-1)^2]$.

4.1. Upper Bounds

Fix the algorithm, trapezoidal rule, A_n, with $\text{cost}(A_n) = n$. Suppose that the data sites $x_0, x_1, \cdots, x_{n-1}$ and the corresponding function values. One may estimate $\|f'\|_1$ in the following way:

$$G_n(f) = \sum_{i=0}^{n-1} |f(x_{i+1}) - f(x_i)| .$$

Consider (16), in this case, the deflation factor, $\mathfrak{c}_n$, and the inflation factor, $\mathfrak{C}_n$ are defined as follows:

$$\mathfrak{c}_n = 1$$
$$\mathfrak{C}_n = \frac{1}{1 - \tau(n-1)} \geq 1.$$

The upper bound implies that $\tau < n-1$. Thus it is shown that in the specific case, the worst error in $W^{2,1}$ will be always smaller than the one in $W^{1,1}$, namely, $\tau\tilde{h}(n) \leq h(n)$. Then for those two non-decreasing functions, $h(n)$ and $\tau\tilde{h}(n)$ the upper bounds on the error of $A_n(f)$ in terms of the $\mathcal{G}$-semi-norm of f is:

$$\begin{aligned}
\left| \int_0^1 f(x)dx - A_n(f) \right| &\leq \min(h(n), \tau\tilde{h}(n)) \|f\|_{\mathcal{G}}, \quad \forall f \in \mathfrak{C}_\tau \\
&= \min\left(1, \frac{\tau}{4(n-1)}\right) \frac{\|f'\|_1}{2(n-1)}, \\
&\leq \frac{\tau\|f'\|_1}{8(n-1)^2}, \\
&\leq \frac{\tau\mathfrak{C}_n G_n(f)}{8(n-1)^2} \leq \varepsilon.
\end{aligned}$$

Theorem 4. *Let ε, $\mathfrak{C}_n$, and τ be given. Let $\mathcal{C}_\tau$ be the cone of functions defined above. The cost of the algorithm is bounded above in terms of $\|f'\|_1$ of the input function as follows:*

$$\textit{cost}(A, \varepsilon, \sigma) \leq \min\left\{ n \in \mathcal{I} : n \geq \left(\frac{\sigma\tau}{8\varepsilon(1 - \tau/(n-1))} \right) \right\}.$$

where $\sigma = \|f'\|_1$.

Algorithm 3. Let ε be the positive error tolerance, $N_{\max}$ be the maximum cost budget. Let τ be a fixed positive constant and 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 be the smallest number in $\mathcal{I}$ such that $\tau < n - 1$. Then do the following:

Stage 1. Estimate $\|f'\|_1$. First compute $G_{n_i}(f)$.

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

$$\frac{1}{8}\frac{\tau}{(n-1)^2-\tau(n_i-1)}G_{n_i}(f)\le\varepsilon.$$

If this is true, then set $W=0$, 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} as the smallest number exceeding n_i and not less than $N_A(\varepsilon/G_{n_i}(f))$ such that A_n is embedded in A_{n+1}. That is

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

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} be the largest number not exceeding $N\max$ such that A_n is embedded in A_{n+1} and set $W=1$. Return $(A_{n_{i+1}}(f),W)$ and terminate the algorithm.

4.2. Lower Bounds

Now we consider the lower bound. We define the fooling function as $f_\pm=c_0f_0\pm c_1f_1$. The first piece of the fooling function is defined by $f_0=x$. In this case,

$$\begin{aligned}\|f_0\|_{W^{1,1}}&=\|f_0'\|_1=1,\\ \|f_0\|_{W^{2,1}}&=\|f_0''\|_1=0=\tau_0.\end{aligned}$$

The second part depends on the sample points. Given any n sample points $\{\xi_i\}_{i=1}^n$ between 0 and 1 such that $0=\xi_0\le\xi_1<\cdots<\xi_n\le\xi_{n+1}=1$, let l and u be the two consecutive points with maximum distance between them, i.e. $\exists j\in\{1,2,\cdots,n-1\}$ such that $l=\xi_j,u=\xi_{j+1}$ and $u-l=\max_{1\le i\le n}(\xi_{i+1}-\xi_i)$. Then let

$$f_1(x)=\begin{cases}\dfrac{30(x-l^2)(x-u)^2}{(u-l)^5}, & l\le x\le u,\\ 0, & \text{otherwise.}\end{cases}.$$

Note that

$$\begin{aligned}\|S(f_1)\|_{W^{1,1}}&=\int_0^1 f_1(x)dx=1,\\ \|f_1\|_{W^{1,1}}&=\|f_1'\|_1=\frac{15}{4(u-l)}\le\frac{15}{4}(n+1)=g(n),\\ \|f_1\|_{W^{2,1}}&=\|f_1''\|_1=\frac{40}{\sqrt{3}(u-l)^2}\le\frac{40}{\sqrt{3}}(n+1)^2=\frac{32}{3\sqrt{3}}(n+1)\|f_1'\|_1\end{aligned}$$

We can easily get that $g^{-1}(n) = \frac{4}{15}x - 1$ and $(\tilde{g}g)^{-1}(x) = \sqrt{\frac{\sqrt{3}}{40}x} - 1$. So according to Theorem 2 in the paper,

$$\text{comp} \geq \min\left(\frac{4}{15}(\frac{\sigma}{4\varepsilon}) - 1, \sqrt{\frac{\sqrt{3}}{40}\frac{\sigma\tau}{4\varepsilon}} - 1\right) = \min\left(\frac{\sigma}{15\varepsilon}, \sqrt{\frac{\sqrt{3}\sigma\tau}{160\varepsilon}}\right) - 1$$

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

In this part, we consider the function recovery $S(f) = f$ and $\mathcal{H} = \mathcal{L}_\infty$. Then $\|S(f) - A(f)\|_{\mathcal{H}} = \|f - A(f)\|_\infty$. Let $\mathcal{G} \subset \mathcal{W}^{1,\infty}[0,1]$ and $\mathcal{F} \subset \mathcal{W}^{2,\infty}[0,1]$, where

$$\mathcal{W}^{k,\infty}[0,1] = \{u \in \mathcal{L}_\infty[0,1] : D^\alpha u \in \mathcal{L}_\infty[0,1], \forall |\alpha| \leq k\}.$$

Given $\tau > 0$, then by the (2), we have $\mathcal{C}_\tau \subset \mathcal{F}$ as

$$\mathcal{C}_\tau = \{f \in \mathcal{W}^{2,\infty}[0,1] : \|f''\|_\infty \leq \tau \|f'\|_\infty\}, \tag{36}$$

Suppose one wants to approximate functions in $\mathcal{C}_\tau$ based on function values. The goal is to find an algorithm, A for which $\|f - A(f)\|_\infty \leq \varepsilon$. Here we use piecewise linear interpolation to approximate. Thus $\forall f \in \mathcal{C}_\tau$, consider the sequence of algorithms $\{A_n\}_{n\in\mathcal{I}}$, where $\mathcal{I} = \{2, 3, \dots\}$ and given the data $f(x_i)$, where $x_i = \frac{i}{n-1}, i = 0, \dots, n-1$, and $\text{cost}(A_n) = n$. For $x_{i-1} \leq x \leq x_i,\ i = 1, \dots, n-1$, define $A_n(f)$

$$A_n(f)(x) = f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}}(f(x_i) - f(x_{i-1})).$$

5.1. Upper Bound

Given the data sites $x_i = (i-1)/(n-1)$, $i = 0, \dots, n$, let the $\mathcal{L}_\infty$ norm of the f' be approximated by

$$G_n(f) = (n-1) \sup_{i=1,\dots,n-1} |f(x_{i+1}) - f(x_i)| \tag{37}$$

Note that $G_n(f)$ never overestimates $\|f'\|_\infty$. Thus, we have $h_-(n) = 0$. For any $x \in [x_i, x_{i+1}]$, note that

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

Furthermore, this inequality is tight if f'' is constant second derivative and f' does not change sign over $[x_i, x_{i+1}]$, and $x = x_i$ or x_{i+1}. Applying the above argument for $i = 0, \ldots, n-1$ implies that

$$\|f'\|_\infty - G_n(f) \leq \frac{\|f''\|_\infty}{2(n-1)} \Rightarrow h_+(n) = \frac{1}{2(n-1)}$$

with equality holding when has a constant second derivative and its first derivative does not change sign over $[0, 1]$. Then by (17) and (18), we can obtain

$$\mathfrak{c}_n = 1, \quad \mathfrak{C}_n = \frac{1}{1 - \tau/(2(n-1))}. \tag{38}$$

Here we use divided difference. Denote

$$f[x, y] := \frac{f(y) - f(x)}{y - x}, f[x, y, z] := \frac{f[y, z] - f[x, y]}{z - x}.$$

Note that for all $x, y \in [0, 1]$ distinct, we have

$$|f(y) - f(x)| = \left| \int_x^y f'(t)dt \right| \leq |y - x| \|f'\|_\infty.$$

Therefore we have $|f[x, y]| \leq \|f'\|_\infty$. For $x_{i-1} \leq x \leq x_i,\ i = 1, \ldots, n-1$, we can rewrite $A_n(f)(x) = f(x_{i-1}) + f[x_{i-1}, x_i](x - x_{i-1})$. Then by Eq(2.67) in [5], we have $f(x) - A_n(f)(x) = f[x_{i-1}, x_i, x] \cdot (x - x_{i-1})(x - x_i)$. As we can rewrite

$$|f[x_{i-1}, x_i, x]| = \left| \frac{f[x_i, x] - f[x_{i-1}, x]}{x_i - x_{i-1}} \right| \leq 2(n-1) \|f'\|_\infty.$$

And as $x \in [x_{i-1}, x_i]$, thus we have $|(x - x_{i-1})(x - x_i)| \leq \frac{1}{4(n-1)^2}$. Then,

$$|f(x) - A_n(f)(x)| \leq \frac{1}{4(n-1)^2} \cdot 2(n-1) \|f'\|_\infty = \frac{1}{2(n-1)} \|f'\|_\infty.$$

Similarly, we can obtain that for $x \in [0, 1]$

$$\|f - A_n(f)\|_\infty \leq \frac{\|f'\|_\infty}{2(n-1)} \Rightarrow \mathrm{err}(A_n, \mathcal{G}, \mathcal{H}, S) \leq \frac{1}{2}(n-1)^{-1} \Rightarrow \tilde{h}(n) = \frac{1}{2}(n-1)^{-1}.$$

On the other hand, by Eq(2.68) in [5], we know $f[x_{i-1}, x_i, x] = \frac{1}{2}f''(\xi)$, where $\xi \in [x_{i-1}, x_i]$. Then we know $|f[x_{i-1}, x_i, x]| \leq \frac{1}{2}\|f''\|_\infty$. And as $x \in [x_{i-1}, x_i]$, thus we have $|(x - x_{i-1})(x - x_i)| \leq \frac{1}{4(n-1)^2}$. Then,

$$|f(x) - A_n(f)(x)| \leq \frac{1}{4(n-1)^2} \frac{\|f''\|_\infty}{2} = \frac{1}{8(n-1)^2} \|f''\|_\infty.$$

Hence, we can obtain that for $\forall x \in [0, 1]$

$$\|f - A_n(f)\|_\infty \leq \frac{1}{8(n-1)^2} \|f''\|_\infty \Rightarrow \mathrm{err}(A_n, \mathcal{F}, \mathcal{H}, S) \leq \frac{1}{8}(n-1)^{-2} \Rightarrow h(n) = \frac{1}{8}(n-1)^{-2}.$$

And we can find $h(n) \leq \tilde{h}(n)h_+(n), \forall n \in \mathcal{I}$, then $\tau h(n) < \tilde{h}(n)\ \forall n \in \mathcal{I}$. Thus we only need to find n such that

$$\frac{\tau \mathfrak{C}_n G_n(f)}{8(n-1)^2} \leq \varepsilon.$$

If we consider about the embedded algorithms, by Algorithm 2, we obtain

Algorithm 4. Let the error tolerance ε, the maximum cost budget $N_{\max}$, and the positive constant τ be as described in (36). 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 be the smallest number in $n \in \mathcal{I}$ satisfying $\mathrm{err}_+(G_n, \mathcal{F}, \|\cdot\|_{\mathcal{G}}, \mathbb{R}_+) \leq 1/\tau$. For any input function $f \in \mathcal{F}$, do the following:

Stage 1. Estimate $\|f'\|_\infty$. Compute $G_{n_i}(f)$ in (37) and $\mathfrak{C}_n$ in (38).

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

$$\frac{\tau \mathfrak{C}_{n_i} G_{n_i}(f)}{8(n_i-1)^2} \leq \varepsilon.$$

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 the inequality above fails to hold, choose n_{i+1} as the smallest number exceeding n_i and not less than $N_A(\varepsilon/G_{n_i}(f))$ such that A_{n_i} is embedded in $A_{n_{i+1}}$. That is

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

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.

Then we can have the complexity of the upper bound.

Theorem 5. *Let ε, $N_{\max}$, τ and n_1 be given as described in Algorithm 4. Assume that $n_1 \leq N_{\max}$. Let r be the number described in the paragraph preceding that algorithm. Let $\mathcal{C}_\tau$ be the cone of functions defined in* (36)*. Define*

$$\widetilde{N}_A(a) = \min\left\{ n \in \mathcal{I} : \frac{\tau \mathfrak{C}_n}{8(n-1)^2} \leq a \right\}, \quad a \in (0, \infty).$$

Let $\mathcal{C}_\tau$ be the cone of functions defined in (36)*. Let*

$$\mathcal{N} = \left\{ f \in \mathcal{C}_\tau : r\widetilde{N}_A\left(\frac{\varepsilon}{\sigma}\right) \leq N_{\max} \right\} = \left\{ f \in \mathcal{C}_\tau : \sigma \leq \frac{8(N_{\max}/r - 1)^2 \varepsilon}{\tau \mathfrak{C}_{N_{\max}/r}} \right\}$$

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

$$\mathrm{cost}(A, \mathcal{N}, \varepsilon, N_{\max}, \sigma) \leq \min\left\{ n \in \mathcal{I} : n \geq \left(\frac{\sigma\tau}{8\varepsilon(1 - \tau/(2(n_1 - 1)))} \right)^{\frac{1}{2}} + 1 \right\} \quad (39)$$

where $\sigma = \|f'\|_\infty$.

Proof. Applied Theorem 2, then we can have the above result. □

5.2. Lower Bound

Next, we will derive the lower bound. Suppose that for any $n > 0$, for any algorithm which cost is less than n. We want to build fooling function at all data sites $\xi_i,\ i = 0, \ldots, n-1$. There exists at least one i such that $\xi_{i+1} - \xi_i \geq \frac{1}{n-1}$. Then we can define f_1

$$f_1(x) := \begin{cases} A_1(x - \xi_j)^2(x - \xi_{j+1})^2 & \xi_j < x \leq \xi_{j+1}, \\ 0 & \text{otherwise}, \end{cases},$$

where $A_1 \geq 0$. As we need $\|f_1\|_\infty = 1$, we can have $A_1 = \frac{16}{(\xi_{i+1} - \xi_i)^4}$. Then we can obtain

$$\|f_1'\|_\infty = \frac{16}{3\sqrt{3}(\xi_{j+1} - \xi_j)} \leq \frac{16}{3\sqrt{3}}(n-1) \Rightarrow g(n) = \frac{16}{3\sqrt{3}}(n-1).$$

$$\|f_1''\|_\infty = \frac{16}{(\xi_{j+1} - \xi_j)^2} = \frac{3\sqrt{3}}{(\xi_{j+1} - \xi_j)} \|f_1'\|_\infty \leq 3\sqrt{3}(n-1)\|f_1'\|_\infty \Rightarrow \tilde{g}(n) = 3\sqrt{3}(n-1).$$

Let $f_0 = x$. Then we have $\|f_0'\|_\infty = 1, \|f_0''\|_\infty = 0 \Rightarrow \tau_0 = 0$.

Theorem 6. *Suppose that functions f_0 and f_1 can be found that satisfy conditions* (33) *and* (34). *It then follows that the complexity of the problem, defined by* (13), *assuming infinite cost budget, over the cone of functions $\mathcal{C}_\tau$ is*

$$\mathrm{comp}(\varepsilon, \mathcal{A}(\mathcal{C}_\tau, \mathcal{H}, S, \Lambda), \infty, \sigma) \geq \min\left(\frac{3\sqrt{3}\sigma}{64\varepsilon} + 1, \left(\frac{\sigma\tau}{64\varepsilon}\right)^2 + 1 \right).$$

Proof. $g(n)$, $\tilde{g}(n)$ and τ_0 is obtained above, then by Theorem 3 we can have the above results. □

5.3. Numerical Example

We want to testify the success rate for our algorithm. So we choose test function $f(x) = e^{-(a(x-\mu))^2}$, where $\mu \sim U[0,1]$ and $\log_{10} a \sim U[0,4]$. And $\mu \sim U[0,1]$ means μ is uniform on [0,1]. Also for this kind of function, if we

have large a, we can approximate the function hardly because the function has a sharper bump. And we know

$$\|f'\|_\infty = \sqrt{\frac{2}{e}}a,\ \|f''\|_\infty = 2a^2.$$

If $f \in \mathcal{C}_\tau$ then we should have

$$a \leq \frac{\tau}{\sqrt{2e}} \Rightarrow \text{Expected Success Rate } = \frac{1}{4}\ln\left(\frac{\tau}{\sqrt{2}}\right) - \frac{1}{8}.$$

We use $\tau = 10, 25, 100$ separately to test 10000 times with $\varepsilon = 1e-7$ to obtain the observed success rate.

τ	10	25	100
Expected Success Rate	$\leq 16\%$	$\leq 26\%$	$\leq 41\%$
Observed Success Rate	25%	34%	49%

Table 1: Comparison between expected and observed success rate

From the above table, we can find our algorithm works very well, as the observed success rate is always larger than the expected success rate.

6. Addressing Questions and Concerns About Automatic Algorithms

Automatic algorithms are popular, especially for univariate integration problems. As mentioned in Section 4, MATLAB [6, 15], Mathematica [21], and the NAG [16] 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.

6.1. 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 continuous can be fooled by a function that yields zero data where probed by the algorithm, but is nonzero elsewhere. A guaranteed automatic algorithm specifies sufficient conditions for success that rule out those functions that would fool it. 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 widen or narrow that class depending on the relevant application and computational cost budget.

6.2. Why Cones?

Traditional error estimates are derived for balls of input functions, $\mathcal{B}_\sigma$, as defined in (1b) with radius σ (often chosen to be one). The analysis here uses cones, $\mathcal{C}_\tau$, of the form (2), 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 a set that contains cf for all numbers c if it contains f.

One might wonder whether the definition of $\mathcal{C}_\tau$ in (2) 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 quite likely $\mathcal{C}_{\text{success}}$ also contains 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 4 and 5, 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 it's value cannot be determined from the function data. The choice of τ reflects how cautious the user is willing 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 norm, whereas the condition defining the cone $\mathcal{C}_\tau$ involves two 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 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 4 and univariate function recovery in Section 5 τ may be interpreted as in inverse length scale or a minimum sample size. These interpretations facilitate the choice of τ.

6.3. 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 [14, 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* [10, 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 i with moderate n_i, $A_{n_i}(f_\lambda) - A_{n_{i-1}}(f_\lambda)$ has different signs for $\lambda = 0, 1$.

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

Lyness's argument, with its pessimistic conclusion, is correct for algorithms that use the size of $A_{n_i}(f) - A_{n_{i-1}}(f)$ as a stopping criterion. For example, Figure 4 depicts an integrand constructed in a manner similarly to that described by Lyness for which MATLAB's standard numerical integration method, `quad`,

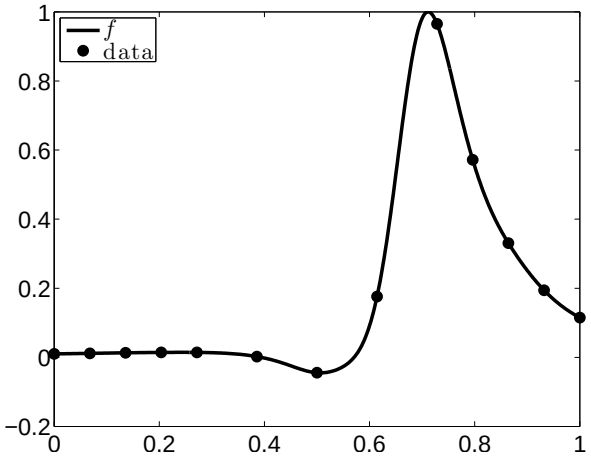

Figure 4: Integrand designed to fool MATLAB's `quad` along with the data used by `quad`

which is based on adaptive Simpson's rule [4]. MATLAB's `quad`, gives the answer $0.1733\ldots$, based on an absolute error tolerance of $\varepsilon = 10^{-14}$, but the true answer is $0.1925\ldots$. 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 the 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 they do not use a stopping criterion based on $A_{n_i}(f) - A_{n_{i-1}}(f)$. Algorithm **??** presented in Section 4, always succeeds for "reasonable" integrands because such integrands lie in the cone $\mathcal{C}_\tau$. The analogous statement is true for function approximation Algorithm **??**, as well as for the general Algorithms 1 and 2. All of these algorithms succeed for reasonable input functions.

Lyness's warning in [10] should not be interpreted as an objection to automatic algorithms. It should be an 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 it one may easily make a functional vanish even though the error of the algorithm is significant.

6.4. No Advantage in Adaption

There are rigorous results from information based complexity theory dating to ???? stating that adaption does not help, namely adaptive algorithms have no significant advantage over non-adaptive algorithms. The automatic algorithms presented here are by definition adaptive in determining the total number of function data required based on an initial sample of the input function. The

reason that adaption can help in this context is that the cone, $\mathcal{C}_\tau$, of input functions is not a convex set. This violates one of the assumptions required to prove the negative result 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

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

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.

7. Further Work

The results presented here suggest a number of other interesting open problems that need to be addressed. Here is a summary.

- The univariate integration and function approximation algorithms in Sections 4 and 5 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 equation initial value problems 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 lie the ones that are provided here for globally adaptive algorithms. The right kinds of spaces $\mathcal{G}$ and $\mathcal{F}$, and their semi-norms, need to be identified that are appropriate 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 may vary somewhat with the definition of $\mathcal{G}$ or $\mathcal{F}$, but it is essentially some 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 by [7].

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] W. Gautschi, Numerical Analysis: An Introduction, Birkhäuser, Boston, 1997.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

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

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

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

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

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

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

[21] Wolfram Research Inc., Mathematica 8, 2011.