GUARANTEED, ADAPTIVE, AUTOMATIC ALGORITHMS FOR UNIVARIATE INTEGRATION: METHODS, COSTS AND IMPLEMENTATIONS

BY

YIZHI ZHANG

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Applied Mathematics
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Advisor

Chicago, Illinois
December 2018

ACKNOWLEDGMENT

This thesis is dedicated to my dearest father and late mother. I want to express my appreciation for their support. I miss my mother very much. This thesis could not have been written without Dr. Fred J. Hickernell, who not only did serve as my advisor but also encouraged me, supported me with the greatest patience in the world. I could not have done this without his greatness. I thank them all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis investigates how to solve univariate integration problems using numerical methods, including the trapezoidal rule and the Simpson's rule. Most existing guaranteed algorithms are not adaptive and require too much a priori information. Most existing adaptive algorithms do not have valid justification for their results. The goal is to create adaptive algorithms utilizing the two above-mentioned methods with guarantees. The classes of integrands studied in this thesis are cones. The algorithms are analytically proved to be a success if the integrand lies in the cone. The algorithms are adaptive and automatically adjust the computational costs based on the integrand values. The lower and upper bounds on the computational costs for both algorithms are derived. The lower bounds on the complexity of the problems are derived as well. By comparing the upper bounds on the computational cost and the lower bounds on the complexity, our algorithms are shown to be asymptotically optimal. Numerical experiments are implemented.

CHAPTER 1

INTRODUCTION

In numerical analysis, numerical integration plays an important role. An algorithm is used to calculate the numerical value of a definite integral. Numerical integration is used instead of analytical methods because sometimes the problem lacks an analytical solution, or the integrand is only known at certain points by sampling. This thesis focuses on solving the univariate integration problem

$$\int_a^b f(x)\,\mathrm{d}x. \tag{1.1}$$

Two adaptive automatic algorithms, `integral_t`, and `integral_s` are derived. They use the trapezoidal rule and Simpson's rule respectively, and are guaranteed to provide numerical approximations that differ from the true integral by no more than a specified error tolerance ε, i.e.

$$\left|\int_a^b f(x)\,\mathrm{d}x - \texttt{integral\_t}(f,\varepsilon)\right| \leq \varepsilon, \tag{1.2}$$

$$\left|\int_a^b f(x)\,\mathrm{d}x - \texttt{integral\_s}(f,\varepsilon)\right| \leq \varepsilon, \tag{1.3}$$

assuming that $\forall f \in \mathcal{C}^p, a, b \in \mathbb{R}, a < b, \varepsilon > 0$, where a and b are finite, and $\mathcal{C}^p$ is some set of integrands defined in later in the thesis.

Our algorithms are adaptive. Adaptive algorithms automatically expend the right amount of computational effort needed and provide an approximate solution within the error tolerance. Our algorithms are also guaranteed to satisfy the error tolerance. Other existing guaranteed algorithms are usually not adaptive. They cannot adjust their effort based on the property of the integrand. Details about those algorithms are discussed in section 1.1.

The essence of every adaptive algorithm is a bound on the error of the numerical approximation based on the computations already performed. The error estimates

of existing adaptive algorithms either require too much information from the user or they are based on heuristics and not guaranteed. Details about such algorithms are discussed in section 1.2.

1.1 Fixed-Cost Algorithms for Balls of Integrands

Fixed-cost algorithms with guarantees usually require integrands to lie in a ball of a function space. The traditional trapezoidal rule and Simpson's rule in calculus courses are two typical examples of fixed-cost algorithms. They have upper bounds on approximation error in terms of the total variation of a particular derivative of integrand:

$$\mathrm{err}(f,n) \le \overline{\mathrm{err}}(f,n) := C(n)\,\mathrm{Var}(f^{(p)}), \tag{1.4}$$

for some $C(n)$, where for the trapezoidal rule, $p = 1$; for the Simpson's rule, $p = 3$. To make sure the algorithms work, users need to obtain information about the total variation of the integrand. So the algorithms only provide guarantees when the total variation has an upper bound, namely $\mathrm{Var}(f^{(p)}) < \sigma$. The computational cost is $n = C^{-1}(\varepsilon/\sigma)$. Fixed-cost algorithms such as the trapezoidal rule and Simpson's rule are automatic because the computational effort required depends on the error tolerance. With a determined error tolerance, the computational cost is automatically decided by the algorithms.

Fixed-cost algorithms have drawbacks. Firstly, users may not know enough about the integrand. If a moderate σ is selected, the algorithms may work for easy functions but not for functions with large peaks. Functions with large peaks may have derivatives which change rapidly in a small interval. The variation of the derivative could be high enough to exceed the radius σ of the ball. Secondly, even if the algorithms work for f, they may not work for cf, where c is a constant and $c > 1$, since cf may fall out of the ball. It is such a pity considering that the integral of

cf can be easily calculated as c times the integral of f. An example is created in [11, (10)] such that fixed-cost algorithms fail for large integrand. Moreover, fixed-cost algorithms have the computational cost depending on σ, but which does not depend on the integrand values. Therefore, the algorithms cannot adjust the computational cost according to the integrand. Namely, fixed-cost functions are not adaptive.

1.2 Adaptive Algorithms with No Guarantees

Adaptive, automatic algorithms are commonly used in numerical software packages. MATLAB's `integral` and Chebfun toolbox are two well-known examples. These methods do not require a value of σ. Instead, they estimate the error using only function values and then determine the sample size accordingly. Since these algorithms are adaptive, they work well in many cases including big functions aforementioned in 1.1. However, these algorithms do not have rigorous justification. Namely, one cannot tell whether the answer provided by the algorithm is true or not. Therefore, the asymptotic error estimates from these algorithms are heuristics and do not hold for all cases. Moreover, for another example defined in [11, (13)], a fluky function which resembles the big function, these algorithms can be totally fooled. Not to mention that the fixed-cost algorithms can be fooled by the fluky functions as well.

Not only can these adaptive algorithms be fooled by the fluky integrands, according to [14], users can always fool these methods by giving them a peak-shaped function with the width less than the mesh size. In this case, the algorithms cannot detect the peak and return zero as an answer.

1.3 Guaranteed Algorithms for Integrands in Cones

Our algorithms can correct compute the integration for the fluky integrands. However, the challenge of spiky integrands applies to any algorithm that depends on function values, including ours. The goal of the thesis is to construct adaptive,

automatic algorithms with guarantees of success. The way of achieving the goal is to define the set of integrands as a cone, not a ball. The idea of defining integrands in a cone instead of a ball is originally introduced in [5]. Linear multiplications of a function remain in the cone. Moreover, the respective derivatives of the functions cannot change too much in a small interval. The error bounds are also defined and approximated using only integrand values (to make the algorithms adaptive). Based on the idea of the cone, we have created a MATLAB library named GAIL (Guaranteed Automatic Integration Library)[2] with co-workers. Details about this software can be found in [4]. Different problems are investigated in the cone settings, including integration (see [10] and [13]), function recovery (see [7] and [3]), optimization (see [3]), Monte-Carlo sampling (see [9] and [12]) and etc.. For univariate integration problems, in contrast to fixed-cost algorithms, our methods are adaptive and succeed for cones of integrands. In contrast to typical adaptive algorithms, our methods have rigorous proofs of how spiky an integrand can be to stay inside the cone.

In the original cone paper [5], a cone different from the cone in this thesis is defined. Rigorous proof of guarantees and an automatic algorithm using the trapezoidal rule for univariate integration are provided based on this cone. Study of the upper bound on the computational cost and the lower bound on the complexity shows that the algorithm is asymptotically optimal. Unfortunately, bounds on the error estimates of Simpson's rule algorithms are too complicated to derive in this cone. Therefore, we introduce a new cone which is the one discussed in this thesis in order to derive the Simpson's rule algorithm. The trapezoidal rule algorithms in the new cone are also studied. The theoretical results from the new cone are better than the ones from the old cone. We put the results from the new cone for the trapezoidal rule algorithm in the thesis for the reason that they are better and more uniform with the Simpson's rule results. Readers can refer to [5, Algo 4, Thm 7, Thm 8] for the original results.

1.4 Outline

In Chapter 2, the definitions and assumptions used in the thesis are introduced. Chapter 3 focuses on finding upper bounds on the approximation error and rigorous theoretical proofs of success. Guaranteed algorithms are presented in Chapter 4 with lower and upper bounds on the computational costs. Chapter 5 studies the lower bound of complexity. In Chapter 6, the results from numerical experiences are discussed. The guaranteed algorithms derived here are based on the trapezoidal rule and Simpson's rule. The former is simple but has a lower order of convergence rate The latter has a higher convergence rate but the derivation is more complicated. The derivation of the two algorithms is done in parallel in Chapters 3 to 5 for the trapezoidal rule and Simpson's rule.

CHAPTER 2

PROBLEM STATEMENT AND ASSUMPTIONS

The previous chapter introduces the univariate integration problem as (1.1). The building blocks of the adaptive, automatic algorithms are two widely used fixed cost algorithms, the trapezoidal rule and Simpson's rule. The composite trapezoidal rule can be defined as:

$$T(f,n) = \frac{b-a}{2n}\sum_{j=0}^{n}(f(u_j)+f(u_{j+1})), \tag{2.1}$$

where

$$u_j = a + \frac{j(b-a)}{n}, \qquad j = 0,\ldots,n, \qquad n \in \mathbb{N}. \tag{2.2}$$

It uses $n+1$ function values where those $n+1$ node points are equally spaced between a and b. The composite Simpson's rule can be defined as:

$$S(f,n) = \frac{b-a}{18n}\sum_{j=0}^{3n-1}(f(v_{2j})+4f(v_{2j+1})+f(v_{2j+2})), \tag{2.3}$$

where

$$v_j = a + \frac{j(b-a)}{6n}, \qquad j = 0,\ldots,6n, \qquad n \in \mathbb{N}. \tag{2.4}$$

It uses $6n+1$ equally spaced function values, which form $6n$ intervals/subintervals between a and b. The reason why $[a,b]$ is divided into $6n$ intervals is that firstly, the Simpson's rule requires an even number of intervals. Secondly, the third derivative of f is approximated using a third order finite difference in later chapters. This third order finite difference method requires the number of intervals to be a multiple of 3. Thus, the number of intervals for input data has to be a multiple of 6.

To better define (1.4) with specification, the traditional non-adaptive trapezoidal rule and Simpson's rule have upper bounds on approximation error in terms of the total variation of a particular derivative of integrand. Here we give detailed

definitions of $C(n)$:

$$\mathrm{err}(f,n) \le \overline{\mathrm{err}}(f,n) := C(n)\,\mathrm{Var}(f^{(p)}), \tag{2.5}$$

$$C(n) = \begin{cases} \frac{(b-a)^2}{8n^2}, p = 1, & \text{trapezoidal rule[1, (7.15)]}, \\ \frac{(b-a)^4}{93312n^4}, p = 3, & \text{Simpson's rule[1, (p.231)]}. \end{cases}$$

The error bounds contains the total variation, which is unknown to the users.

To define the total variation, $\mathrm{Var}(f)$, firstly we introduce $\widehat{V}(f, \{x_i\}_{i=0}^{n+1})$ as an under-approximation of the total variation:

$$\widehat{V}(f, \{x_i\}_{i=0}^{n+1}) = \sum_{i=1}^{n-1} |f(x_{i+1}) - f(x_i)|, \tag{2.6}$$

where $\{x_i\}_{i=0}^{n+1}$ is any partition with no necessity of equal spacing, and $a = x_0 \le x_1 < \cdots < x_n \le x_{n+1} = b$. The partition $\{x_i\}_{i=0}^{n+1}$ contains $n+2$ points. The two end points x_0 and x_{n+1} are ignored in (2.6) for convenience later. Then the total variation can be defined as the upper of bound on $\widehat{V}(f, \{x_i\}_{i=0}^{n+1})$:

$$\mathrm{Var}(f) := \sup \left\{ \widehat{V}(f, \{x_i\}_{i=0}^{n+1}), \quad \text{for any } n \in \mathbb{N}, \text{ and } \{x_i\}_{i=0}^{n+1} \right\}. \tag{2.7}$$

To ensure a finite error bound, our algorithms are defined for function spaces with finite total variations of the respective derivatives:

$$\mathcal{V}^p := \{f \in C^{p-1}[a,b] : \mathrm{Var}(f^{(p)}) < \infty\}, \tag{2.8}$$

where for the trapezoidal rule, $p = 1$, and for the Simpson's rule, $p = 3$.

Our algorithms does not work for all $f \in \mathcal{V}^p$ because f may have changes in $f^{(p)}$ on small intervals that the algorithms cannot detect. In order to build adaptive automatic and guaranteed algorithms, the following assumptions are set up so that all functions to be integrated must lie in a cone of integrands for which $\widehat{V}(f^{(p)}, \{x_i\}_{i=0}^{n+1})$

does not underestimate $\mathrm{Var}(f^{(p)})$ too much:

$$\mathcal{C}^p := \Big\{ f \in \mathcal{V}^p, \mathrm{Var}(f^{(p)}) \le \mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{n+1}))\widehat{V}(f^{(p)}, \{x_i\}_{i=0}^{n+1}), \\ \text{for all choices of } n \in \mathbb{N}, \text{and } \{x_i\}_{i=0}^{n+1} \text{ with } \mathrm{size}(\{x_i\}_{i=0}^{n+1}) < \mathfrak{h} \Big\}. \quad (2.9)$$

The cone $\mathcal{C}^p$ is a subset of $\mathcal{V}^p$. Here $\mathrm{size}(\{x_i\}_{i=0}^{n+1})$ is the largest possible width between any adjacent x_i:

$$\mathrm{size}(\{x_i\}_{i=0}^{n+1}) := \max_{i=0,\cdots,n} x_{i+1} - x_i. \quad (2.10)$$

The cut-off value $\mathfrak{h} \in (0, b-a)$ and the inflation factor $\mathfrak{C} : [0, \mathfrak{h} \to [1, \infty)$ define the cone. The choice of $\mathfrak{C}$ is flexible, but it must be non-decreaing. In this thesis, we choose

$$\mathfrak{C}(h) := \frac{\mathfrak{C}(0)}{1 - h/\mathfrak{h}}, \quad \mathfrak{C}(0) > 1, \quad (2.11)$$

again for convenience.

With the cone defined, we can start constructing our adaptive trapezoidal rule and Simpson's rule algorithms for functions in $\mathcal{C}^p$, with appropriate value of p. To begin with, Chapter 3 shows how the algorithms confidently estimate the error and provide data-driven error bounds.

CHAPTER 3

DATA-DRIVEN ERROR BOUNDS

In this chapter, we derive upper bounds on the approximation errors in terms of function values. By (2.5) in Chapter 2, the approximation errors of the trapezoidal rule and Simpson's rule have an upper bound in terms of the total variation of the appropriate derivatives. The definition of the cone suggests a way of bounding $\mathrm{Var}(f^{(p)})$ in terms of $\widehat{V}(f^{(p)}, \{x_i\}_{i=0}^{n+1})$. However, our algorithms are based on function values and $\widehat{V}(f^{(p)}, \{x_i\}_{i=0}^{n+1})$ is defined in terms of derivative values. Thus finite differences are used to express derivative values in terms of function values.

3.1 The Trapezoidal Rule

Backward finite differences are used to approximate f'. Define the width of the subintervals as $h = u_{j+1} - u_j = (b-a)/n$ and

$$f[u_j] = f(u_j), \text{ for } j = 0, \cdots, n,$$
$$f[u_j, u_{j-1}] = \frac{f(u_j) - f(u_{j-1})}{h}, \text{ for } j = 1, \cdots, n.$$

According to the Mean Value Theorem for finite differences, for all $j = 1, 2, \cdots, n$, there exists $x_j \in (u_{j-1}, u_j)$ such that

$$f'(x_j) = f[u_j, u_{j-1}],$$

for $j = 1, 2, \cdots, n$. This implies that

$$f'(x_j) = \frac{f(u_j) - f(u_{j-1})}{h} = \frac{n}{b-a}[f(u_j) - f(u_{j-1})], \tag{3.1}$$

for some $x_j \in (u_{j-1}, u_j)$. Let $\{a = x_0, x_1, \ldots, x_n, x_{n+1} = b\}$ be a partition as was introduced just below (2.6). Note that no matter how the x_j's are located, the largest possible width between two adjacent x_i's cannot be larger than the width of two intervals. So

$$\mathrm{size}(\{x_j\}_{j=0}^{n+1}) \le 2h = 2(b-a)/n < \mathfrak{h}. \tag{3.2}$$

Since (2.6) is true for all partition $\{x_i\}_{i=0}^{n+1}$, it is true for (3.1) with this particular partition $\{x_j\}_{i=0}^{n+1}$. Thus the approximation $\widetilde{V}_1(f,n)$ to $\widehat{V}(f',\{x_j\}_{i=0}^{n+1})$ using only integrand values is defined as:

$$\begin{aligned}\widehat{V}(f',\{x_j\}_{j=0}^{n+1}) &= \sum_{j=1}^{n-1} |f'(x_{j+1}) - f'(x_j)|\,, \\ &= \sum_{j=1}^{n-1} \left| \frac{n}{b-a}[f(u_{j+1}) - f(u_j) - f(u_j) + f(u_{j-1})] \right| \\ &= \frac{n}{b-a} \sum_{j=1}^{n-1} |f(u_{j+1}) - 2f(u_j) + f(u_{j-1})| =: \widetilde{V}_1(f,n). \end{aligned} \tag{3.3}$$

Therefore by combining the deduction above together, the error bound on error estimate for our trapezoidal rule algorithm using only function values can be written as follow:

$$\begin{aligned}\overline{\mathrm{err}}_{\mathrm{t}}(f,n) :=& C(n)\,\mathrm{Var}(f'), && \text{(by (2.5))} \\ \le& C(n)\mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{n+1}))\widehat{V}(f',\{x_i\}_{i=0}^{n+1}), && \text{(by (2.9))} \\ =& C(n)\mathfrak{C}(\mathrm{size}(\{x_j\}_{i=0}^{n+1}))\widetilde{V}_1(f,n), && \text{(by (3.3))} \\ \le& \frac{(b-a)^2\mathfrak{C}(2(b-a)/n)\widetilde{V}_1(f,n)}{8n^2}. && \text{(by (3.2))} \end{aligned}$$

Lemma 1 gives the upper bound on the estimation error of the composite trapezoidal rule.

Lemma 1. *The approximation error of the composite trapezoidal rule is bounded in terms of the integrand values as follows:*

$$\overline{\mathrm{err}}_{\mathrm{t}}(f,n) \le \frac{(b-a)^2\mathfrak{C}(2(b-a)/n)\widetilde{V}_1(f,n)}{8n^2}, \text{ for all } f \in \mathcal{C}^1. \tag{3.4}$$

This upper bound can be used as the factor of determining whether the output of our trapezoidal rule algorithm meets the error tolerance. Since $\mathfrak{C}(\cdot)$ is a non-decreasing function, a, b is fixed, as the value of n increases, $\mathfrak{C}(2(b-a)/n)$ is does

not increase. According to the definition of $\mathrm{Var}(f')$ in (2.7) and the definition of $\widetilde{V}_1(f,n)$ in (3.3), $\widetilde{V}_1(f,n)$ is always less than the variation of the first derivative of the integrand. With a user provided tolerance ε, as long as n is big enough, the approximation error eventually decreases below ε. Therefore, the error bound (3.4) of our algorithm using only function values provides guarantees that it succeeds.

3.2 The Simpson's Rule

Similar to the trapezoidal rule, the third order backward finite difference is used to approximate f'''. Let $h = v_{j+1} - v_j = (b-a)/6n$ be the width of the interval and

$$\begin{aligned}
f[v_j] &= f(v_j), && \text{for } j = 0, \cdots, 6n,\\
f[v_j, v_{j-1}] &= \frac{f(v_j) - f(v_{j-1})}{h}, && \text{for } j = 1, \cdots, 6n,\\
f[v_j, v_{j-1}, v_{j-2}] &= \frac{f(v_j) - 2f(v_{j-1}) + f(v_{j-2})}{2h^2}, && \text{for } j = 2, \cdots, 6n,\\
f[v_j, v_{j-1}, v_{j-2}, v_{j-3}] &= \frac{f(v_j) - 3f(v_{j-1}) + 3f(v_{j-2}) - f(v_{j-3})}{6h^3}, && \text{for } j = 3, \cdots, 6n.
\end{aligned}$$

According to Mean Value Theorem for finite differences, for all $j = 1, 2, \cdots, 2n$, there exists $x_j \in (v_{3j-3}, v_{3j})$ such that

$$\frac{f'''(x_j)}{6} = f[v_{3j}, v_{3j-1}, v_{3j-2}, v_{3j-3}],$$

for $j = 1, 2, \cdots, 2n$. This implies that

$$\begin{aligned}
f'''(x_j) &= \frac{f(v_{3j}) - 3f(v_{3j-1}) + 3f(v_{3j-2}) - f(v_{3j-3})}{h^3},\\
&= \frac{216n^3}{(b-a)^3}[f(v_{3j}) - 3f(v_{3j-1}) + 3f(v_{3j-2}) - f(v_{3j-3})] \quad (3.5)
\end{aligned}$$

for some $x_j \in (v_{3j-3}, v_{3j})$. Let $\{a = x_0, x_1, \ldots, x_n, x_{2n+1} = b\}$ be a partition as was introduced just below (2.6). Note that no matter how the x_j's are located, the largest possible width between two adjacent x_i's cannot be larger than the width of six intervals. So

$$\mathrm{size}(\{x_j\}_{i=0}^{2n+1}) \le 6h = (b-a)/n < \mathfrak{h}. \quad (3.6)$$

Since (2.6) is true for all partition $\{x_i\}_{i=0}^{2n+1}$, it is true for (3.5) with $\{x_j\}_{i=0}^{2n+1}$. Then the approximation $\widetilde{V}_3(f,n)$ to $\widehat{V}(f''', \{x_i\}_{i=0}^{n+1})$ using only integrand values can be written as follow:

$$\begin{aligned}
\widehat{V}(f''', \{x_j\}_{j=0}^{n+1}) &= \sum_{j=1}^{n-1} \left| f'''(x_{j+1}) - f'''(x_j) \right|, \\
&= \sum_{j=1}^{2n-1} \left| \frac{216n}{(b-a)^3} [(f(v_{3j+3}) - 3f(v_{3j+2}) + 3f(v_{3j+1}) - f(v_{3j})) \right. \\
&\qquad \left. -(f(v_{3j}) - 3f(v_{3j-1}) + 3f(v_{3j-2}) - f(v_{3j-3}))] \right| \\
&= \frac{216n^3}{(b-a)^3} \sum_{j=1}^{2n-1} |f(v_{3j+3}) - 3f(v_{3j+2}) + 3f(v_{3j+1}) \\
&\qquad -2f(v_{3j}) + 3f(v_{3j-1}) - 3f(v_{3j-2}) + f(v_{3j-3})| =: \widetilde{V}_3(f,n). \quad (3.7)
\end{aligned}$$

Therefore by combining the relative equations together, the error bound on error estimate for our Simpson's rule algorithm using only function values is obtained as:

$$\begin{aligned}
\overline{\mathrm{err}}_{\mathrm{s}}(f,n) :=& C(n)\,\mathrm{Var}(f'''), && \text{(by (2.5))} \\
\leq& C(n)\mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{2n+1}))\widehat{V}(f''', \{x_i\}_{i=0}^{2n+1}), && \text{(by (2.9))} \\
=& C(n)\mathfrak{C}(\mathrm{size}(\{x_j\}_{i=0}^{2n+1}))\widetilde{V}_3(f,n), && \text{(by (3.7))} \\
\leq& \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{93312n^4}. && \text{(by (3.6))}
\end{aligned}$$

Lemma 2 describes the upper bound on the estimation error of the Simpson's rule.

Lemma 2. *The approximation error of the composite Simpson's rule is bounded in terms of the integrand values as follows:*

$$\overline{\mathrm{err}}_s(f,n) \leq \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{93312n^4}. \quad (3.8)$$

This upper bound can be used for our Simpson's algorithm as the factor of determining whether the output meets the error tolerance. Since $\mathfrak{C}(\cdot)$ is a non-decreasing

function, a, b is fixed, as the value of n going up, $\mathfrak{C}((b-a)/n)$ is non-increasing. The definition of $\mathrm{Var}(f''')$ in (2.7) and $\widetilde{V}_3(f,n)$ in (3.7) provides that $\widetilde{V}_3(f,n)$ is always less than the variation of the function's third derivative. With a user provided tolerance ε, as long as n is big enough, the approximation error eventually decreases below ε. Therefore, this error bound (3.8) of our algorithm using only function values provides guarantees that it will succeeds.

As mentioned earlier, the error bounds (3.4) and (3.8) of our trapezoidal rule and Simpson's rule algorithms can be treated as the stopping criterion. In the next chapter, details about the algorithms are introduced. The lower bounds and upper bounds of the computational cost are discussed as well.

CHAPTER 4

ADAPTIVE, AUTOMATIC ALGORITHMS WITH GUARANTEES AND COMPUTATIONAL COST

In the previous chapter, the stopping criterion of our trapezoidal rule and Simpson's rule algorithms using the integrand values are derived. In this chapter, the two algorithms are discussed in detail. Our algorithms require the inputs of a function value black-box and an error tolerance. The algorithms output an answer adaptively by deciding the number of function values to be used, the location of the points, and an input of the cone by themselves. For practical reasons, the algorithms set a maximum number of iterations and a maximum number of input points. If the either maximum number is reached, the algorithms quit and provide a best possible output. In order to save the computation time, embedded input points are used to compute function values. This means that if the algorithms enter the next iteration, the number of subintervals used are multiplied by an integer and the integrand values from the last iteration are saved and used again.

4.1 The Trapezoidal Rule

Our algorithm is guaranteed to work for functions lying in the cone. However, one may not be able to check the sufficient condition such that a particular integrand is in the cone. In order to decide whether one integrand is in the cone or not, it is important to find a necessary condition for which $f \in \mathcal{C}^1$. This condition can be used as a threshold for the algorithm to rule out all the functions such that $f \notin \mathcal{C}^1$. For those functions not in the cone, we propose a method later in this section. This method tries to make the cone more inclusive so that it contains more functions than before.

The derivation of finding a necessary condition for which $f \in \mathcal{C}^1$ starts from the definition of the variation and the cone. From (2.7), the approximation $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$

is a lower bound on $\mathrm{Var}(f')$, i.e. $\widehat{V}(f', \{x_j\}_{j=0}^{n+1}) \le \mathrm{Var}(f')$ for all $n \ge 2$. From the definition of the cone (2.9), the variation of the first derivative of the integrand has an upper bound in terms of the approximation $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ times an appropriate inflation factor $\mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$, i.e. $\mathrm{Var}(f') \le \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ for all $n \ge 2$. In the inequality above, the size of the partition is unknown. From the definition of the cut-off value in (3.2), $\mathrm{size}(\{x_j\}_{j=0}^{n+1}) < 2(b-a)/n$. Since that $\mathfrak{C}(\cdot)$ is non-decreasing, $\mathfrak{C}(2(b-a)/n) \le \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$. From (3.3), $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ can be expressed as $\widetilde{V}_1(f, n)$ using only function values. The nodes used in the algorithm are nested, for $\widetilde{V}_1(f, n_k) = \widehat{V}(f', \{x_j\}_{j=0}^{n+1})$. And $\widetilde{V}_1(f, n_k)$ is non-decreasing as n_k increases. But $\mathfrak{C}(2(b-a)/n)$ decreases with increasing n. Therefore, the largest lower bound on $\mathrm{Var}(f')$ is always the one at the current kth loop. For the smallest upper bound this is not always the case. In the algorithm, the lower and upper bounds are updated as the number of nodes multiplies. Thus, in order to make two sides of the inequality hold, namely, for an integrand lying in the cone $\mathcal{C}^1$, the following condition must be satisfied:

$$\widetilde{V}_1(f, n_k) \le \mathfrak{C}(2(b-a)/n_j)\widetilde{V}_1(f, n_j), \quad \text{for all } j \le k. \tag{4.1}$$

If (4.1) does not meet, the cut-off value $\mathfrak{h}$ is halved to make the cone more inclusive. The reason is that the inflation factor defined in (2.11) is decreasing with respect to $\mathfrak{h}$. By halving $\mathfrak{h}$, the inflation factor becomes larger. Intuitively the cone is more inclusive. Integrands have higher chances of lying in the new cone than the old one, thus have higher chancec to be accurately calculated.

4.1.1 The trapezoidal rule algorithm. Following the settings of how to check whether the integrand is in the cone, and how to revise the cone if needed, now we provide the guaranteed automatic integration algorithm using the trapezoidal rule.

Algorithm 1 (Trapezoidal Rule Adaptive Algorithm `integral_t`)**.** Let the sequence

of algorithms $\{T_n\}_{n\in\mathbb{N}}$, and $\widetilde{V}_1(\cdot,\cdot)$ be as described above. Let $\mathfrak{h} \le (b-a)$. Set $k=0$, $n_k = 1$. For any integrand f and error tolerance ε, do the following:

Step 1. (Re)set the upper bound on $\mathrm{Var}(f')$**; increase number of nodes.** Let $\eta_k = \infty$ and $n_{k+1} = n_k \times (\lfloor 2(b-a)/(\mathfrak{h} n_k)\rfloor + 1)$.

Step 2. Compute the largest lower bound on $\mathrm{Var}(f')$**.** Let $k = k+1$. Compute $\widetilde{V}_1(f, n_k)$ in (3.3).

Step 3. Compute the smallest upper bound on $\mathrm{Var}(f')$**.** Compute

$$\eta_k = \min\left(\eta_{k-1}, \mathfrak{C}(2(b-a)/n_k)\widetilde{V}_1(f,n_k)\right).$$

Step 4. Check the necessary condition for $f \in \mathcal{C}^1$**.** If $\widetilde{V}_1(f,n_k) \le \eta_k$, then go to Step 5. Otherwise, set $\mathfrak{h} = \mathfrak{h}/2$.

a) Let $\mathfrak{J} = \{j \in \{1,\cdots,k\} : n_j > 2(b-a)/\mathfrak{h}\}$. If $\mathfrak{J}$ is empty, go to Step 1.

b) Otherwise, recompute the upper bound on $\mathrm{Var}(f')$, for all n_j, with $j \in \mathfrak{J}$ as follows:

For $j' = \min \mathfrak{J}$, let $\eta_{j'} = \mathfrak{C}(2(b-a)/n_{j'})\widetilde{V}_1(f,n_{j'})$,

Compute $\eta_j = \min\{\eta_{j-1}, \mathfrak{C}(2(b-a)/n_j)\widetilde{V}_1(f,n_j)\}$, for $j = j'+1,\cdots,k$.

Go to the beginning of Step 4.

Step 5. Check for convergence. Check whether n_k is large enough to satisfy the error tolerance, i.e.

$$n_k^2 \ge \frac{\eta_k (b-a)^2}{8\varepsilon}.$$

a) If this is true, return $\texttt{integral\_t}(f,\varepsilon) = T_{n_k}(f)$ and terminate the algorithm.

b) Otherwise, go the Step 6.

Step 6. Increase the number of nodes and loop again. Choose

$$n_{k+1} = n_k \times \max\left(\left\lceil \frac{(b-a)}{n_k}\sqrt{\frac{\widetilde{V}_1(f,n_k)}{8\varepsilon}}\right\rceil, 2\right).$$

Go to Step 2.

Step 1 in `integral_t` sets the memory of the upper bound on $\mathrm{Var}(f')$ as infinity. Then the number of inputs is updated for the case where the integrand is out of the cone, and the interval width of nodes is larger than the new cuff-off value $\mathfrak{h}$. In Step 2, the lower bound on the variation $\mathrm{Var}(f')$ is calculated.The number of loops is increased by 1. In Step 3, the upper bound on $\mathrm{Var}(f')$ is updated by comparing the new $\mathfrak{C}(2(b-a)/n_k)\widetilde{V}_1(f,n_k)$ to the previously largest one. Step 4 checks whether the integrand satisfies the necessary condition. If true, the algorithm goes to Step 5. Otherwise the cut-off value $\mathfrak{h}$ is halved to make the cone more inclusive as described earlier in this section. Note that after $\mathfrak{h}$ is halved, the mesh size in previous loops may not be fine enough to provide an answer. In this case, the algorithm goes back to Step 1 and starts over. Even if the mesh is fine enough, the function may still not lie in the new cone. Also the upper bounds on $\mathrm{Var}(f')$ are changed with the new $\mathfrak{h}$. Thus, η_k is recalculated in 4.b). Then, the algorithm goes to the beginning of Step 4 to check the necessary condition again. Step 5 checks whether the error bound is small enough to meet the tolerance according to (3.4) in Lemma 1. If the error is small enough, the algorithm calculates the integrand according to (2.1) and terminates. Otherwise the algorithm goes to Step 6 to multiply the number of nodes and start over.

The algorithm `integral_t` always succeeds if the integrand is in the cone $\mathcal{C}^1$ and the maximum number of nodes allowed is large enough.

Theorem 1. *For $f \in \mathcal{C}^1$,* `integral_t` *is successful, i.e.,*

$$\left|\int_a^b f(x)dx - \texttt{integral\_t}(f,\varepsilon)\right| \le \varepsilon, \qquad \forall f \in \mathcal{C}^1.$$

Proof. From Lemma 1 and the description below it, `integral_t` is guaranteed to provide an approximation of the integral (1.1) with the approximation error less than the tolerance. □

4.1.2 Computational cost of `integral_t`. Having created the multistage adaptive algorithm using the trapezoidal rule, we continue to find the computational cost of the algorithm. The computational cost of the trapezoidal algorithm has upper and lower bounds as described in the following theorem.

Theorem 2. *Let $N(f,\varepsilon)$ denote the final number n_k in Step 5 when* `integral_t` *terminates. Then this number is bounded below and above in terms of the true, yet unknown,* $\mathrm{Var}(f')$.

$$\begin{aligned}\max\left(\left\lfloor\frac{2(b-a)}{\mathfrak{h}}\right\rfloor+1,\left\lceil 2(b-a)\sqrt{\frac{\mathrm{Var}(f')}{8\varepsilon}}\right\rceil\right) &\le N(f,\varepsilon)\\ &\le 2\min\left\{n\in\mathbb{N}: n\ge\left\lfloor\frac{2(b-a)}{\mathfrak{h}}\right\rfloor+1,\zeta(n)\,\mathrm{Var}(f')\le\varepsilon\right\}\\ &\le 2\min_{0<\alpha\le 1}\max\left(\left\lfloor\frac{2(b-a)}{\alpha\mathfrak{h}}\right\rfloor+1,(b-a)\sqrt{\frac{\mathfrak{C}(\alpha\mathfrak{h})\,\mathrm{Var}(f')}{8\varepsilon}}\right), \quad (4.2)\end{aligned}$$

where $\zeta(n)=(b-a)^2\mathfrak{C}(2(b-a)/n)/(8\varepsilon n^2)$. *The number of function values, namely, the computational cost required by the algorithm, is* $N(f,\varepsilon)+1$.

Proof. No matter what inputs f and ε are provided, $N(f,\varepsilon)\ge n_1=\lfloor 2(b-a)/\mathfrak{h}\rfloor+1$. Then the number of intervals increases until $\overline{\mathrm{err}}(f,n)\le\varepsilon$. From the error bound defined in (2.5), $C(N(f,\varepsilon))\,\mathrm{Var}(f')\le\varepsilon$. Thus $N(f,\varepsilon)\ge\left\lceil 2(b-a)\sqrt{\mathrm{Var}(f')/(8\varepsilon)}\right\rceil$. This implies the lower bound on $N(f,\varepsilon)$.

Let K be the value of k for which `integral_t` terminates. Since $n_1=\lfloor 2(b-a)/\mathfrak{h}\rfloor+1$ satisfies the upper bound, one may assume that $K\ge 2$. Let m be the multiplication integer found in Step 6. Note that $\zeta((m-1)n_{K-1})\,\mathrm{Var}(f')>\varepsilon$. For $m=2$, this is true because $\zeta(n_{K-1})\,\mathrm{Var}(f')\ge\zeta(n_{K-1})\widetilde{V}_1(f,n_{K-1})>\varepsilon$. For

$m > 2$ it is true because of the definition of m. Since ζ is a decreasing function, it follows that

$$(m-1)n_{K-1} < n^* := \min\left\{n \in \mathbb{N} : n \geq \left\lfloor \frac{2(b-a)}{n} \right\rfloor + 1, \zeta(n)\operatorname{Var}(f') \leq \varepsilon\right\}.$$

Therefore $n_L = mn_{L-1} < m\frac{n^*}{m-1} = \frac{m}{m-1}n^* \leq 2n^*$.

To prove the latter part of the upper bound, we need to prove that

$$n^* \leq \max\left(\left\lfloor \frac{2(b-a)}{\alpha\mathfrak{h}} \right\rfloor + 1, (b-a)\sqrt{\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f')}{8\varepsilon}}\right), \quad 0 < \alpha < 1.$$

For fixed $\alpha \in (0,1]$, we only need to consider that case where $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor$. This implies that $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor \geq 2(b-a)/(\alpha\mathfrak{h})$ thus $\alpha\mathfrak{h} \geq 2(b-a)/n^*$. Also by the definition of n^*, ζ, and $\mathfrak{C}$ is non-decreasing:

$$\begin{aligned}
&\zeta(n^*)\operatorname{Var}(f') > \varepsilon,\\
\Rightarrow\ & 1 < \sqrt{\frac{\zeta(n^*)\operatorname{Var}(f')}{\varepsilon}},\\
\Rightarrow\ & n^* < n^*\sqrt{\frac{\zeta(n^*-1)\operatorname{Var}(f')}{\varepsilon}}\\
&= n^*\sqrt{\frac{(b-a)^2\mathfrak{C}(2(b-a)/n^*)\operatorname{Var}(f')}{8(n^*)^2\varepsilon}}\\
&\leq (b-a)\sqrt{\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f')}{8\varepsilon}}.
\end{aligned}$$

This completes the proof of latter part of the upper bound. □

4.2 The Simpson's Rule

Similar to the trapezoidal algorithm, firstly we provide the necessary condition for the integrand lying in the cone. From (2.7), the approximation $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$ is a lower bound on $\operatorname{Var}(f''')$, i.e. $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1}) \leq \operatorname{Var}(f''')$ for all $n \geq 6$. From the definition of the cone in (2.9), the variation of the third derivative can be bounded by the approximation $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$ times an appropriate inflation factor $\mathfrak{C}(\text{size}(\{x_j\}_{j=0}^{n+1}))$,

namely $\mathrm{Var}(f''') \leq \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$, for all $n \geq 6$. Following a similar derivation as the trapezoidal rule case, the necessary condition for an integrand being in the cone $\mathcal{C}^3$ at the kth loop is:

$$\widetilde{V}_3(f, n_k) \leq \mathfrak{C}((b-a)/n)\widetilde{V}_3(f, n_j), \quad \text{for all } j \leq k. \tag{4.3}$$

The same mechanism is used if this condition is violated, namely halving the cut-off value $\mathfrak{h}$ is implemented to make the cone more inclusive in the Simpson's rule case.

4.2.1 The Simpson's rule algorithm. Now we provide the guaranteed automatic integration algorithm using the Simpson's rule.

Algorithm 2 (Simpson's Rule Adaptive Algorithm `integral_s`)**.** Let the sequence of algorithms $\{S_n\}_{n\in\mathbb{N}}$, and $\widetilde{V}_3(\cdot,\cdot)$ be as described above. Let $\mathfrak{h} \leq (b-a)/6$. Set $k = 0$, $n_k = 1$. For any integrand f and error tolerance ε, do the following:

Step 1. (Re)set the upper bound on $\mathrm{Var}(f''')$**; increase the number of nodes.** Let $\eta_k = \infty$ and $n_{k+1} = n_k \times (\lfloor (b-a)/\mathfrak{h}n_k \rfloor + 1)$.

Step 2. Compute the largest lower bound on $\mathrm{Var}(f''')$**.** Let $k = k+1$. Compute $\widetilde{V}_3(f, n_k)$ in (3.7).

Step 3. Compute the smallest upper bound on $\mathrm{Var}(f''')$**.** Compute

$$\eta_k = \min\left(\eta_{k-1}, \mathfrak{C}((b-a)/n_k)\widetilde{V}_3(f, n_k)\right).$$

Step 4. Check the necessary condition for $f \in \mathcal{C}^3$**.** If $\widetilde{V}_3(f, n_k) \leq \eta_k$, go to Step 5. Otherwise, set $\mathfrak{h} = \mathfrak{h}/2$.

a) Let $\mathfrak{J} = \{j \in \{1, \cdots, k : n_j\} > (b-a)/\mathfrak{h}\}$. If $\mathfrak{J}$ is empty, go to Step 1.

b) Otherwise, recompute the upper bound on $\mathrm{Var}(f''')$ for all n_j, with $j \in \mathfrak{J}$

as follows:

For $j' = \min \mathfrak{J}$, let $\eta_{j'} = \mathfrak{C}((b-a)/n_{j'})\widetilde{V}_3(f, n_{j'})$,

Compute $\eta_j = \min\{\eta_{j-1}, \mathfrak{C}((b-a)/n_j)\widetilde{V}_3(f, n_j)\}$, for $j = j'+1, \cdots, k$.

Go to the beginning of Step 4.

Step 5. Check for convergence. Check whether n_k is large enough to satisfy the error tolerance, i.e.

$$n_k^4 \geq \frac{\eta_k (b-a)^4}{93312\varepsilon}.$$

a) If true, return $\texttt{integral\_s}(f, \varepsilon) = S_{n_k}(f)$ and terminate the algorithm.

b) Otherwise, go the Step 6.

Step 6. Increase the number of nodes and loop again. Choose

$$n_{k+1} = n_k \times \max\left(\left\lceil \frac{(b-a)}{n_k} \left(\frac{\widetilde{V}_3(f, n_k)}{93312\varepsilon} \right)^{1/4} \right\rceil, 2 \right).$$

Go to Step 2.

The process of `integral_s` is similar to `integral_t`. Moreover, `integral_s` always succeeds if the integrand is in the cone $\mathcal{C}^3$ and the number of nodes is large enough.

Theorem 3. *For integrands in $\mathcal{C}^3$,* `integral_s` *is successful, i.e.,*

$$\left| \int_a^b f(x)dx - \texttt{integral\_s}(f, \varepsilon) \right| \leq \varepsilon, \qquad \forall f \in \mathcal{C}^3.$$

Proof. From Lemma 2 and the description below it, `integral_s` is guaranteed to provide an approximation of integral (1.1) with the approximation error less than the tolerance using the Simpson's rule. □

4.2.2 Computational cost of `integral_s`. Having created the multistage adaptive algorithm using the Simpson's rule, we now continue to find the computational cost of the algorithm. The computational cost of the Simpson's algorithm can be bounded by the following theorem.

Theorem 4. *Let $N(f,\varepsilon)$ denote the final number of n_k in Step 5 when* `integral_s` *terminates. Then this number is bounded below and above in terms of the true, yet unknown,* $\mathrm{Var}(f''')$.

$$\begin{aligned}\max\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor+1,\left\lceil(b-a)\left(\frac{\mathrm{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right\rceil\right)&\le N(f,\varepsilon)\\ &\le 2\min\left\{n\in\mathbb{N}:n\ge\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor+1,\zeta(n)\,\mathrm{Var}(f''')\le\varepsilon\right\}\\ &\le 2\min_{0<\alpha\le1}\max\left(\left\lfloor\frac{(b-a)}{\alpha\mathfrak{h}}\right\rfloor+1,(b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\,\mathrm{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right),\end{aligned}\tag{4.4}$$

where $\zeta(n)=(b-a)^4\mathfrak{C}((b-a)/n)/(93312n^4)$. *The number of function values, namely the computational cost, required by the algorithm is* $6N(f,\varepsilon)+1$.

Proof. No matter what inputs f and ε are provided, $N(f,\varepsilon)\ge n_1=\lfloor(b-a)/\mathfrak{h}\rfloor+1$. Then the number of intervals increases until $\overline{\mathrm{err}}(f,n)\le\varepsilon$. From the error bound defined in (2.5), $C(N(f,\varepsilon))\,\mathrm{Var}(f''')\le\varepsilon$. Thus $N(f,\varepsilon)\ge\left\lceil(b-a)\left(\frac{\mathrm{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right\rceil$. This implies the lower bound on $N(f,\varepsilon)$.

Let K be the value of k for which `integral_s` terminates. Since n_1 satisfies the upper bound, one may assume that $K\ge2$. Let m be the multiplication integer found in Step 6. Note that $\zeta((m-1)n_{K-1})\,\mathrm{Var}(f''')>\varepsilon$. For $m=2$, this is true because $\zeta(n_{K-1})\,\mathrm{Var}(f''')\ge\zeta(n_{K-1})\widetilde{V}_3(f,n_{K-1})>\varepsilon$. For $m>2$ it is true because of the definition of m. Since ζ is a decreasing function, it follows that

$$(m-1)n_{K-1}<n^*:=\min\left\{n\in\mathbb{N}:n\ge\left\lfloor\frac{2(b-a)}{n}\right\rfloor+1,\zeta(n)\,\mathrm{Var}(f''')\le\varepsilon\right\}.$$

Therefore $n_K=m^*n_{K-1}<m^*\frac{n^*}{m-1}=\frac{m}{m-1}n^*\le2n^*$.

To prove the latter part of the upper bound, we need to prove that

$$n^* \leq \max\left(\left\lfloor \frac{2(b-a)}{\alpha\mathfrak{h}} \right\rfloor + 1, (b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right), \quad 0 < \alpha < 1.$$

For fixed $\alpha \in (0,1]$, we only need to consider that case where $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor$. This implies that $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor \geq 2(b-a)/(\alpha\mathfrak{h})$ thus $\alpha\mathfrak{h} \geq 2(b-a)/(n^*)$. Also by the definition of n^*, ζ, and $\mathfrak{C}$ is non-decreasing:

$$\begin{aligned}
&\zeta(n^*)\operatorname{Var}(f''') > \varepsilon,\\
\Rightarrow 1 &< \left(\frac{\zeta(n^*)\operatorname{Var}(f''')}{\varepsilon}\right)^{1/4},\\
\Rightarrow n^* &< n^*\left(\frac{\zeta(n^*)\operatorname{Var}(f''')}{\varepsilon}\right)^{1/4}\\
&= n^*\left(\frac{(b-a)^4\mathfrak{C}(2(b-a)/(n^*))\operatorname{Var}(f''')}{93312(n^*)^4\varepsilon}\right)^{1/4}\\
&\leq (b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{93312\varepsilon}\right)^{1/4}.
\end{aligned}$$

This completes the proof of latter part of the upper bound. □

Having obtained the upper bound on the computational cost, it is important to discover a lower bound of complexity for univariate integration problems. By studying the lower bound on the complexity, one can decide whether the cost of our algorithms are optimal among all possible algorithms using function values for the same problem. In the next chapter, the complexity of the problem is discussed.

CHAPTER 5

LOWER BOUNDS ON COMPLEXITY

In the previous Chapter, we discuss the algorithms and their computational cost. In this Chapter, the lower bound of complexity steps into the spotlight. The computational complexity of a certain problem is the lowest cost required by any possible algorithm for this problem, including unknown algorithms. If the upper bound of the computational cost of our algorithm is asymptotically the same order to the lower bound on the complexity, then our algorithm asymptotically optimal. The technique of finding the lower bound of complexity was brought up by Traub and Werschulz in [16, p. 11–12]. That is, to build a fooling function to show that no matter how good an algorithms is, it can always be fooled by the function and provide a wrong answer, if the mesh is not fine enough. In this case, when the number of nodes is less than a certain number n, any algorithm fails. This means that in order for any algorithm to be successful, the number of nodes should not be less than this certain n, namely, a lower bound on the complexity. The derivation starts from defining $\texttt{int}(\cdot, \varepsilon)$ as any algorithm that solves univariate integration problem. Two fooling functions are built for the trapezoidal rule and Simpson's rule respectively following by discussion of how to find the lowest number of nodes required to solve the problem.

5.1 The Trapezoidal Rule Complexity

The construction of fooling function for the trapezoidal rule starts from creating a triangle shaped function $\text{peak}(x; t, \delta)$ on (a, b) that starts at point t, with the width of 2δ and zero value elsewhere than $t, t + 2\delta$. Considering the B-Spline to the

first order:

$$\mathrm{b}_{j,0}(x) := \begin{cases} 1, & t_j \le x < t_{j+1}, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathrm{b}_{j,1}(x) := \begin{cases} \dfrac{x - t_j}{t_{j+1} - t_j}, & t_j \le x < t_{j+1}, \\ \dfrac{t_{j+2} - x}{t_{j+2} - t_{j+1}}, & t_{j+1} \le x < t_{j+2}, \\ 0, & \text{otherwise.} \end{cases}$$

The function $\mathrm{b}_{j,1}(x)$ is a triangle shaped function that starts at point t_j, with the width of $t_{j+2} - t_j$ and zero value elsewhere than (t_j, t_{j+2}). Assuming that $\{t_j\}_{j=1}^n$ is equally space between (a, b), $t_{j+1} - t_j = \delta$, and $t = t_j$. Our peak function can be defined as

$$\mathrm{peak}(x; t, \delta) = \delta \mathrm{b}_{j,1} = \begin{cases} x - t, & t \le x < t + \delta, \\ t + 2\delta - x, & t + \delta \le x < t + 2\delta, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Picture 5.1 shows the shape of the function.

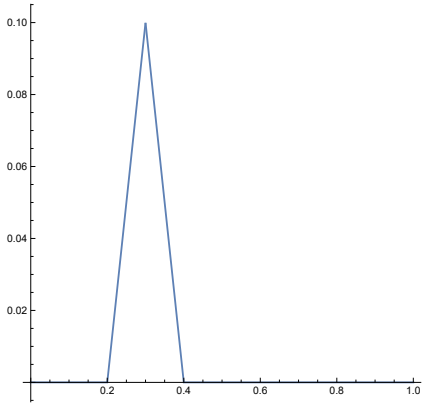

Figure 5.1. A peak function where $a = 0$, $b = 1$, $t = 0.2$, $\delta = 0.1$.

Other properties of the peak function are displayed below:

$$\text{peak}'(x;t,\delta) := \begin{cases} 1, & t \le x < t+\delta, \\ -1, & t+\delta \le x < t+2\delta, \\ 0, & \text{otherwise}, \end{cases}$$

$$\text{Var}(\text{peak}'(\cdot;t,\delta)) \le 4 \text{ with equality if } a < t < t+2\delta < b,$$

$$\int_a^b \text{peak}(x;t,\delta)dx = \delta^2.$$

The peak function above is continuous and with finite variation of the first derivative of the function, namely $\text{peak}(x;t,\delta) \in \mathcal{V}^1$. The picture of the peak function is shown in 5.2: Starting from this peak function $\text{peak}(x;t,\delta)$, the following double-peak functions

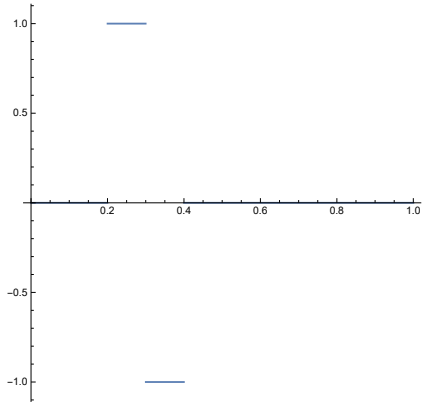

Figure 5.2. First derivative of peak function 5.1.

are good candidates as fooling functions:

$$\text{twopk}(x;t,\delta,\pm) := \text{peak}(x;a,\delta) \pm \frac{3[\mathfrak{C}(\delta)-1]}{4}\text{peak}(x;t,\delta)$$
$$a+3\delta \le t \le b-3\delta, 0 \le \delta < \mathfrak{h}. \quad (5.2a)$$

$$\text{Var}(\text{twopk}'(x;t,\delta,\pm)) = 3 + 4\frac{3[\mathfrak{C}(\delta)-1]}{4} = 3\mathfrak{C}(\delta). \quad (5.2b)$$

Picture 5.3 shows the shape of the double-peak function $\text{twopk}(x;t,\delta,+)$. Note that

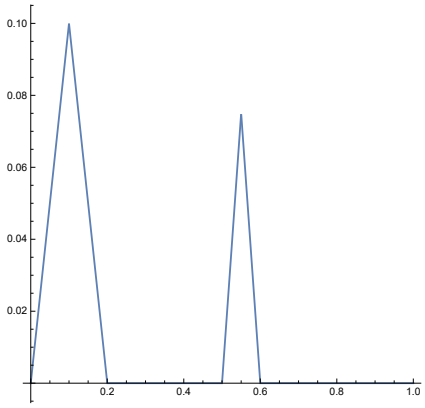

Figure 5.3. A fooling function that can fool the trapezoidal rule algorithms, where $a = 0$, $b = 1$, $t = 0.5$, $\delta = 0.05$ and $\mathfrak{h} = 0.1$.

twopk$(x; t, \delta, \pm)$ are always in $\mathcal{C}^1$. From the definition of the peak function, it follows that

$$\begin{aligned}
&\mathfrak{C}(\text{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(\text{twopk}'(x; t, \delta, \pm), \{x_j\}_{j=0}^{n+1}) \\
&\geq \begin{cases} 3\mathfrak{C}(\delta) = \text{Var}(\text{twopk}'(x; t, \delta, \pm)), \delta \leq \text{size}(\{x_j\}_{j=0}^{n+1})) < \mathfrak{h} \\ \mathfrak{C}(0)\,\text{Var}(\text{twopk}'''(x; t, \delta, \pm)), 0 \leq \text{size}(\{x_j\}_{j=0}^{n+1})) < \delta \end{cases} \\
&\geq \text{Var}(\text{twopk}'(x; t, \delta, \pm))
\end{aligned}$$

The functions twopk$(x; t, \delta, \pm)$ have two peaks. The first always starts from the left end, a, and has the width of $2\mathfrak{h}$. The second starts from any point t that is at least one interval right to the first peak and have a smaller width of 2δ. Although twopk$(x; t, \delta, \pm)$ may have a bump with arbitrarily small width 2δ, the height is small enough for twopk$(x; t, \delta, \pm)$ to lie in the cone. These two functions can be used as our fooling functions.

The fooling functions are used to prove a lower bound on the complexity of the trapezoidal rule algorithm. The key to the proof is to find one appropriate δ. This δ makes the smaller peak resides between data sites. Thus the small peak cannot be detected by the algorithm. Therefore, the algorithm cannot distinguish

between $\text{peak}(x;t,\delta)$ and $\text{twopk}(x;t,\delta,\pm)$. Assume that the algorithm $\texttt{int}(\cdot,\varepsilon)$ uses n equally spaced nodes to successfully return and answer. Moreover, assume that $\{x_j\}_{j=0}^{m}$, $m < n$, is a subinterval of $\{x_j\}_{j=0}^{n}$ where the smaller peak resides. Notice that average width of intervals is $(x_{m+1} - x_0)(m+1)$ and $x_{m+1} - x_0 = b - a - 3\mathfrak{h} - \delta$. Choosing one interval such that the width is less than 2δ, the width of small bump can make the fooling functions fool the algorithm. In order to do that, one can choose interval wider than average and 2δ narrower than average.

$$\frac{x_{j+1} - x_j}{2} \geq \frac{x_{m+1} - x_0}{2(m+1)} \geq \frac{x_{m+1} - x_0}{2(n+1)} = \frac{b - a - 3\mathfrak{h} - \delta}{2n+2} = \delta,$$
$$\Rightarrow \delta = \frac{b - a - 3\mathfrak{h}}{2n+3}$$

Therefore, the lower bound of the complexity for the integration problem using the trapezoidal rule can be represented as:

Theorem 5. *Let* $\texttt{int}(\cdot,\varepsilon)$ *be any (possibly adaptive) algorithm that succeeds for all integrands in* $\mathcal{C}^1$*, and only uses function values. For any error tolerance* $\varepsilon > 0$ *and any arbitrary value of* $\text{Var}(f')$*, there are some* $f \in \mathcal{C}^1$ *for which* $\texttt{int}(\cdot,\varepsilon)$ *must use at least*

$$-\frac{3}{2} + \frac{b - a - 3\mathfrak{h}}{4}\sqrt{\left\lfloor \frac{[\mathfrak{C}(0) - 1]\,\text{Var}(f')}{\varepsilon} \right\rfloor} \tag{5.3}$$

function values. As $\text{Var}(f')/\varepsilon \to \infty$ *the asymptotic rate of increase is the same as the computational cost of* `integral_t`*. The adaptive trapezoidal* `integral_t` *has optimal order for integration of functions in* $\mathcal{C}^1$*.*

Proof. For any positive α, suppose that $\texttt{int}(\cdot,\varepsilon)$ evaluates integrand $\alpha\text{peak}'(\cdot;t,\delta)$ at n nodes before returning to an answer. Let $\{x_j\}_{j=1}^{m})$ be the $m < n$ ordered nodes used by $\texttt{int}(\cdot,\varepsilon)$ that fall in the interval (x_0, x_{m+1}) where $x_0 := a + 2\mathfrak{h}$, $x_{m+1} := b - \delta$ and $\delta := (b - a - 3\mathfrak{h})/(2n+3)$. There must exit at least one of these x_j with $i = 0, \cdots, m$

for which

$$\frac{x_{j+1}-x_j}{2} \geq \frac{x_{m+1}-x_0}{2(m+1)} \geq \frac{x_{m+1}-x_0}{2(n+1)} = \frac{b-a-3\mathfrak{h}-\delta}{2n+2} = \delta.$$

Choose one such x_j and call it t. The choice of t and δ ensures that $\texttt{int}(\cdot,\varepsilon)$ cannot distinguish between $\alpha\text{peak}(\cdot;t,\delta)$ and $\alpha\text{twopk}(\cdot;t,\delta,\pm)$. Thus

$$\texttt{int}(\alpha\text{twopk}(\cdot;t,\delta,\pm),\varepsilon) = \texttt{int}(\alpha\text{peak}(\cdot;t,\delta),\varepsilon)$$

Moreover, $\alpha\text{peak}(\cdot;t,\delta)$ and $\alpha\text{twopk}(\cdot;t,\delta,\pm)$ are all in the cone $\mathcal{C}^1$. This means that $\texttt{int}(\cdot,\varepsilon)$ is successful for all of the functions.

$$\begin{aligned}
\varepsilon &\geq \frac{1}{2}\Big[\left|\int_a^b \alpha\text{twopk}(x;t,\delta,-)dx - \texttt{int}(\alpha\text{twopk}(\cdot;t,\delta,-),\varepsilon)\right| \\
&\qquad + \left|\int_a^b \alpha\text{twopk}(x;t,\delta,+)dx - \texttt{int}(\alpha\text{twopk}(\cdot;t,\delta,+),\varepsilon)\right|\Big] \\
&\geq \frac{1}{2}\Big[\left|\texttt{int}(\alpha\text{peak}(\cdot;t,\delta,-),\varepsilon) - \int_a^b \alpha\text{twopk}(x;t,\delta,-)dx\right| \\
&\qquad + \left|\int_a^b \alpha\text{twopk}(x;t,\delta,+)dx - \texttt{int}(\alpha\text{peak}(\cdot;t,\delta,+),\varepsilon)\right|\Big] \\
&\geq \frac{1}{2}\left|\int_a^b \alpha\text{twopk}(x;t,\delta,+)dx - \int_a^b \alpha\text{twopk}(x;t,\delta,-)dx\right| \\
&= \int_a^b \alpha\text{peak}(x;t,\delta)dx \\
&= \frac{3\alpha[\mathfrak{C}(\delta)-1]\delta^2}{4} \\
&= \frac{[\mathfrak{C}(\delta)-1]\delta^2 \operatorname{Var}(\alpha\text{peak}'(\cdot;a,\mathfrak{h}))}{4}
\end{aligned}$$

Substituting δ in terms of n:

$$\begin{aligned}
2n+3 = \frac{b-a-3\mathfrak{h}}{\delta} &\geq (b-a-3\mathfrak{h})\left[\frac{[\mathfrak{C}(\delta)-1]\operatorname{Var}(\alpha\text{peak}'(\cdot;a,\mathfrak{h})))}{4\varepsilon}\right]^{1/2}, \\
&\geq \frac{b-a-3\mathfrak{h}}{2}\left[\frac{[\mathfrak{C}(0)-1]\operatorname{Var}(\alpha\text{peak}'(\cdot;a,\mathfrak{h}))}{\varepsilon}\right]^{1/2}.
\end{aligned}$$

Since α is an arbitrary positive number, the value of $\operatorname{Var}(\alpha\text{peak}'(\cdot;a,\mathfrak{h}))$ is arbitrary.

Finally, comparing the upper bound on the computational cost of `integral_t` in (4.2) with the lower bound on the computational cost of the best algorithm in (5.3), both of them increase as $\mathcal{O}((\mathrm{Var}(f')/\varepsilon))^{1/2}$ as $(\mathrm{Var}(f')/\varepsilon)^{1/2} \to \infty$. Thus `integral_t` is asymptotically optimal. □

5.2 The Simpson's Rule Complexity

To build the fooling functions for the Simpson's rule, the B-Spline expansion in the previous section continuous to the third order as follows:

$$\mathrm{b}_{j,0}(x) := \begin{cases} 1, & t_j \le x < t_{j+1}, \\ 0, & \text{otherwise}, \end{cases}$$

$$\mathrm{b}_{j,1}(x) := \begin{cases} \dfrac{x - t_j}{t_{j+1} - t_j}, & t_j \le x < t_{j+1}, \\ \dfrac{t_{j+2} - x}{t_{j+2} - t_{j+1}}, & t_{j+1} \le x < t_{j+2}, \\ 0, & \text{otherwise}, \end{cases}$$

$$\mathrm{b}_{j,2}(x) := \begin{cases} \dfrac{(x - t_j)^2}{2(t_{j+1} - t_j)^2}, & t_j \le x < t_{j+1}, \\ \dfrac{(t_{j+2} - x)(x - t_j)}{2(t_{j+2} - t_{j+1})^2} + \dfrac{(t_{j+3} - x)(x - t_{j+2})}{2(t_{j+2}t_{j+1})^2}, & t_{j+1} \le x < t_{j+2}, \\ \dfrac{(t_{j+3} - x)^2}{2(t_{j+3} - t_{j+2})^2}, & t_{j+2} \le x < t_{j+3}, \\ 0, & \text{otherwise}, \end{cases}$$

$$
\mathrm{b}_{j,3}(x) := \begin{cases}
\dfrac{(x-t_j)^3}{6(t_{j+1}-t_j)^3}, & t_j \le x < t_{j+1}, \\
\dfrac{(t_{j+2}-x)(x-t_j)^2}{6(t_{j+2}-t_{j+1})^3} + \dfrac{(t_{j+3}-x)(x-t_j)(x-t_{j+1})}{6(t_{j+2}-t_{j+1})^3} & \\
\quad + \dfrac{(t_{j+4}-x)(x-t_{j+1})^2}{6(t_{j+2}-t_{j+1})^3}, & t_{j+1} \le x < t_{j+2}, \\
\dfrac{(t_{j+3}-x)^2(x-t_j)}{6(t_{j+3}-t_{j+2})^3} + \dfrac{(t_{j+4}-x)(t_{j+3}-x)(x-t_{j+1})}{6(t_{j+3}-t_{j+2})^3} & \\
\quad + \dfrac{(t_{j+4}-x)^2(x-t_{j+2})}{6(t_{j+3}-t_{j+2})^3}, & t_{j+2} \le x < t_{j+3}, \\
\dfrac{(t_{j+4}-x)^3}{6(t_{j+4}-t_{j+3})^3}, & t_{j+3} \le x < t_{j+4}, \\
0, & \text{otherwise},
\end{cases}
$$

The function $\mathrm{b}_{j,3}(x)$ is a bump shaped function that starts at point t_j, with the width of $t_{j+4} - t_j$ and zero value elsewhere than (t_j, t_{j+4}). Assuming that $\{t_j\}_{j=1}^n$ is equally space between (a, b), $t_{j+1} - t_j = \delta$, and $t = t_j$. The bump function is defined as

$$
\mathrm{bump}(x; t, h) := \begin{cases}
(x-t)^3/6, & t \le x < t+\delta, \\
[-3(x-t)^3 + 12\delta(x-t)^2 & \\
\qquad -12\delta^2(x-t) + 4\delta^3]/6, & t+\delta \le x < t+2\delta, \\
[3(x-t)^3 - 24\delta(x-t)^2 & \\
\qquad +60\delta^2(x-t) - 44\delta^3]/6, & t+2\delta \le x < t+3\delta, \\
(t+4\delta-x)^3/6, & t+3\delta \le x < t+4\delta, \\
0, & \text{otherwise},
\end{cases} \tag{5.5}
$$

Picture 5.4 shows the shape of the bump function. The bump function is continuous.

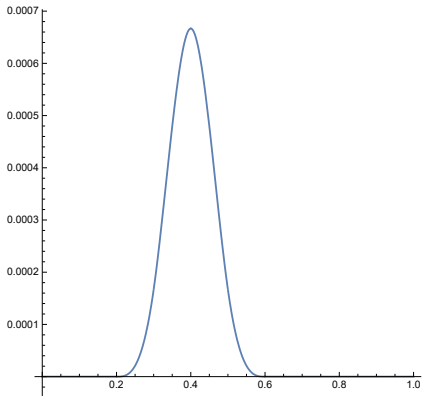

Figure 5.4. A bump function, where $a = 0$, $b = 1$, $t = 0.2$, $\delta = 0.1$ and $\mathfrak{h} = 0.1$.

Other properties of the bump function are displayed below.

$$\text{bump}'(x;t,\delta) := \begin{cases} (x-t)^2/2, & t \le x < t+\delta, \\ -[3(x-t)^2 - 8\delta(x-t) + 4\delta^2]/2, & t+\delta \le x < t+2\delta, \\ [3(x-t)^2 - 16\delta(x-t) + 20\delta^2]/2, & t+2\delta \le x < t+3\delta, \\ -(x-t-4\delta)^2/2, & t+3\delta \le x < t+4\delta, \\ 0, & \text{otherwise}, \end{cases} \tag{5.6a}$$

$$\text{bump}''(x;t,\delta) := \begin{cases} (x-t), & t \le x < t+\delta, \\ -3(x-t) + 4\delta, & t+\delta \le x < t+2\delta, \\ 3(x-t) - 8\delta, & t+2\delta \le x < t+3\delta, \\ -(x-t-4\delta), & t+3\delta \le x < t+4\delta, \\ 0, & \text{otherwise}, \end{cases} \tag{5.6b}$$

$$\text{bump}'''(x;t,\delta) := \begin{cases} 1, & t \le x < t+\delta, \\ -3, & t+\delta \le x < t+2\delta, \\ 3, & t+2\delta \le x < t+3\delta, \\ -1, & t+3\delta \le x < t+4\delta, \\ 0, & \text{otherwise}, \end{cases} \tag{5.6c}$$

$$\text{Var}(\text{bump}'''(\cdot;t,\delta)) \le 16 \text{ with equality if } a < t < t+4\delta < b, \tag{5.6d}$$

$$\int_a^b \text{bump}(x;t,\delta)dx = \delta^4. \tag{5.6e}$$

The bump function above is continuous to the second derivative with finite variation of the third derivatives. Note that the following double-bump function that is always

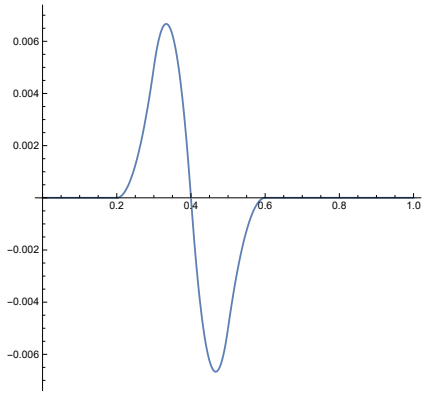

Figure 5.5. First derivative of bump function 5.4.

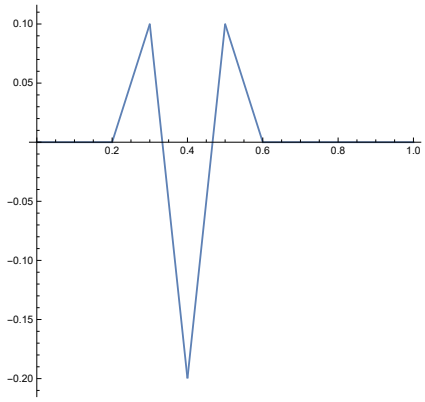

Figure 5.6. Second derivative of bump function 5.4.

in $\mathcal{C}^3$:

$$\text{twobp}(x;t,\delta,\pm) := \text{bump}(x;a,\mathfrak{h}) \pm \frac{15[\mathfrak{C}(\delta)-1]}{16}\text{bump}(x;t,\delta)$$

$$a+5\mathfrak{h} \le x \le b-5\delta, 0 \le \delta < \mathfrak{h}. \quad (5.7a)$$

$$\text{Var}(\text{twobp}'''(x;t,\delta,\pm)) = 15 + 16\frac{15[\mathfrak{C}(\delta)-1]}{16} = 15\mathfrak{C}(\delta). \quad (5.7b)$$

Picture 5.8 shows the shape of the double-peak function.

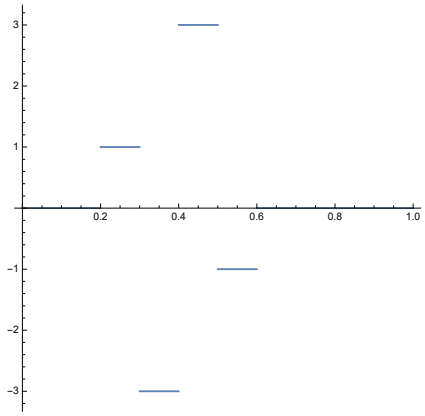

Figure 5.7. Third derivative of bump function 5.4.

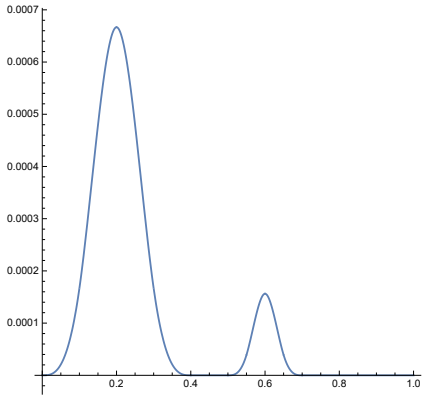

Figure 5.8. A fooling function that can fool the Simpson's rule algorithm, where $a = 0$, $b = 1$, $t = 0.5$, $\delta = 0.05$ and $\mathfrak{h} = 0.1$.

From this properties of the bump function, it follows that

$$\begin{aligned}
&\mathfrak{C}(\text{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(\text{twobp}'''(x;t,\delta,\pm),\{x_j\}_{j=0}^{n+1}) \\
&\geq \begin{cases} 15\mathfrak{C}(\delta) = \text{Var}(\text{twobp}'''(x;t,\delta,\pm)), \delta \leq \text{size}(\{x_j\}_{j=0}^{n+1})) < \mathfrak{h} \\ \mathfrak{C}(0)\,\text{Var}(\text{twobp}'''(x;t,\delta,\pm)), 0 \leq \text{size}(\{x_j\}_{j=0}^{n+1})) < \delta \end{cases} \\
&\geq \text{Var}(\text{twobp}'''(x;t,\delta,\pm))
\end{aligned}$$

Although $\text{twobp}'''(x;t,\delta,\pm)$ may have a bump with arbitrarily small width 4δ, the height is small enough for $\text{twobp}'''(x;t,\delta,\pm)$ to lie in the cone.

Similar to the trapezoidal case, the average width of intervals is $(x_{m+1} -$

$x_0)/(m+1)$ and $x_{m+1} - x_0 = b - a - 5\mathfrak{h} - \delta$. Choose one interval such that width less bigger than 4δ (the width of small bump). In order to do that, we can choose interval wider than average and 4δ narrower than average:

$$\frac{x_{j+1} - x_j}{4} \geq \frac{x_{m+1} - x_0}{4(m+1)} \geq \frac{x_{m+1} - x_0}{4(n+1)} = \frac{b - a - 5\mathfrak{h} - \delta}{4n+4} = \delta,$$
$$\Rightarrow \delta = \frac{b - a - 5\mathfrak{h}}{4n+5}$$

With the fooling functions, the lower bound of complexity of the Simpson's algorithms can be expressed as follow.

Theorem 6. *Let* ***int*** *be any (possibly adaptive) algorithm that succeeds for all integrands in* $\mathcal{C}^3$*, and only uses integrand values. For any error tolerance* $\varepsilon > 0$ *and any arbitrary value of* $\mathrm{Var}(f''')$*, there are some* $f \in \mathcal{C}^3$ *for which* ***int*** *must use at least*

$$-\frac{5}{4} + \frac{b - a - 5\mathfrak{h}}{8} \left[\frac{[\mathfrak{C}(0) - 1]\,\mathrm{Var}(f''')}{\varepsilon} \right]^{1/4} \tag{5.8}$$

function values. As $\mathrm{Var}(f''')/\varepsilon \to \infty$ *the asymptotic rate of increase is the same as the computational cost of* `integral_s`*. Thus* `integral_s` *has optimal order for integration of functions in* $\mathcal{C}^3$*.*

Proof. For any positive α, suppose that $\mathtt{int}(\cdot, \varepsilon)$ evaluates integrand $\alpha\mathrm{bump}'''(\cdot; t, \delta)$ at n nodes before returning to an answer. Let $\{x_j\}_{j=1}^m$) be the $m < n$ ordered nodes used by $\mathtt{int}(\cdot, \varepsilon)$ that fall in the interval (x_0, x_{m+1}) where $x_0 := a + 3\mathfrak{h}$, $x_{m+1} := b - \delta$ and $\delta := (b - a - 5\mathfrak{h})/(4n+5)$. There must exits at least one of these x_j with $i = 0, \cdots, m$ for which

$$\frac{x_{j+1} - x_j}{4} \geq \frac{x_{m+1} - x_0}{4(m+1)} \geq \frac{x_{m+1} - x_0}{4(n+1)} = \frac{b - a - 5\mathfrak{h} - \delta}{4n+4} = \delta.$$

Choose one such x_j and call it t. The choice of t and δ ensures that $\mathtt{int}(\cdot, \varepsilon)$ cannot distinguish between $\alpha\mathrm{bump}(\cdot; t, \delta)$ and $\alpha\mathrm{twobp}(\cdot; t, \delta, \pm)$. Thus

$$\mathtt{int}(\alpha\mathrm{twobp}(\cdot; t, \delta, \pm), \varepsilon) = \mathtt{int}(\alpha\mathrm{bump}(\cdot; t, \delta), \varepsilon)$$

Moreover, $\alpha\text{bump}(\cdot;t,\delta)$ and $\alpha\text{twobp}(\cdot;t,\delta,\pm)$ are all in the cone $\mathcal{C}^3$. This means that $\texttt{int}(\cdot,\varepsilon)$ is successful for all of the functions.

$$\begin{aligned}
\varepsilon &\geq \frac{1}{2}\,[\,\left|\int_a^b \alpha\text{twobp}(x;t,\delta,-)dx - \texttt{int}(\alpha\text{twobp}(\cdot;t,\delta,-),\varepsilon)\right| \\
&\qquad\qquad + \left|\int_a^b \alpha\text{twobp}(x;t,\delta,+)dx - \texttt{int}(\alpha\text{twobp}(\cdot;t,\delta,+),\varepsilon)\right|\,] \\
&\geq \frac{1}{2}\,[\,\left|\texttt{int}(\alpha\text{bump}(\cdot;t,\delta,-),\varepsilon) - \int_a^b \alpha\text{twobp}(x;t,\delta,-)dx\right| \\
&\qquad\qquad + \left|\int_a^b \alpha\text{twobp}(x;t,\delta,+)dx - \texttt{int}(\alpha\text{bump}(\cdot;t,\delta,+),\varepsilon)\right|\,] \\
&\geq \frac{1}{2}\left|\int_a^b \alpha\text{twobp}(x;t,\delta,+)dx - \int_a^b \alpha\text{twobp}(x;t,\delta,-)dx\right| \\
&= \int_a^b \alpha\text{bump}(x;t,\delta)dx \\
&= \frac{15\alpha[\mathfrak{C}(\delta)-1]\delta^4}{16} \\
&= \frac{[\mathfrak{C}(\delta)-1]\delta^4\,\text{Var}(\alpha\text{bump}'''(\cdot;a,\mathfrak{h}))}{16}
\end{aligned}$$

Substituting δ in terms of n:

$$\begin{aligned}
4n+5 = \frac{b-a-5\mathfrak{h}}{\delta} &\geq (b-a-5\mathfrak{h})\left[\frac{[\mathfrak{C}(\delta)-1]\,\text{Var}(\alpha\text{bump}'''(\cdot;a,\mathfrak{h})))}{16\varepsilon}\right]^{1/4}, \\
&\geq \frac{b-a-5\mathfrak{h}}{2}\left[\frac{[\mathfrak{C}(0)-1]\,\text{Var}(\alpha\text{bump}'''(\cdot;a,\mathfrak{h}))}{\varepsilon}\right]^{1/4}.
\end{aligned}$$

Since α is an arbitrary positive number, the value of $\text{Var}(\alpha\texttt{bump}'''(\cdot;a,\mathfrak{h}))$ is arbitrary.

Finally, comparing the upper bound on the computational cost of `integral_s` in (4.4) with the lower bound on the computational cost of the best algorithm in (5.8), both of them increase as $\mathcal{O}((\text{Var}(f''')/\varepsilon))^{1/4}$ as $(\text{Var}(f''')/\varepsilon)^{1/4} \to \infty$. Thus `integral_s` is optimal. □

Having proved that our algorithms using the trapezoidal rule and the Simpson's rule are optimal, we continue to test the performance of the two algorithms.

In the next chapter, the numerical results from `integral_t` and `integral_s` are discussed.

CHAPTER 6

NUMERICAL EXPERIMENTS

The bump function $\text{bump}(x; t, \delta)$ defined in (5.5) is used as our test function to check the performance of our algorithms. Consider the family of bump test functions on interval $(0, 1)$ defined by

$$f(x, t, \delta) = \text{bump}(x; t, \delta)/\delta^4 \tag{6.1}$$

with $t \sim \mathcal{U}[0, 1-4\delta]$, $\log_{10}(\delta) \sim \mathcal{U}[-4, -1]$. The integration $\int_0^1 f(x, t, \delta)\mathrm{d}x = 1$. Input t controls the starting position of the bump. The starting point is randomly chosen between 0 and $1 - 4\delta$. Input δ controls the quarter width of the bump. The random quarter width expands from 0.0001 to 0.1. As an experiment, we chose 10000 random test functions and applied `integral_t` and `integral_s` with an error tolerance of $\varepsilon = 10^{-8}$. The initial values of $\mathfrak{h}$ are set as 0.1, 0.01, and 0.001. Both algorithms impose a cost budget of $N_{\max} = 10^7$. If either of the proposed n_{k+1} in Step 1 or Step 6 of `integral_t` or `integral_s` exceeds $N_{\max}$, then the algorithms return with a warning and fall back to the largest possible n_{k+1} which does not exceed $N_{\max}$ and is a multiple of n_k. The algorithm is considered successful for a particular f if the exact and approximate integrals have a difference no greater than ε. Our algorithms give warnings when the integrand is not in the original cone. Table 6.1 shows the test results. The first column indicates different algorithms. The second column indicates the pre-determined cut-off values of `integral_t` and `integral_s`. Smaller cut-off value gives the algorithms larger number of starting points. The third column shows the total success rates of `integral_t`, `integral_s`, `integral` and `chebfun`. The fourth column shows the success rates of `integral_t` and `integral_s` when the integrand is out of the original cone. The fifth column shows the average number of data sites used for each integrand of `integral_t` and `integral_s` with different $\mathfrak{h}$. The last column shows the average time elapsed in seconds for each integrand of

`integral_t`, `integral_s`, `integral` and `chebfun`.

Table 6.1. The success rates, average costs and average time (sec) of `integral_t` and `integral_s` plus the success rates of other common quadrature algorithms.

	$\hbar$	Success	Success Warning	Average Cost	Average Time
	0.1	24.76%	9.43%	397425	0.0775
`integral_t`	0.01	57.36%	3.70%	2202134	0.433
	0.001	87.38%	0.81%	5156884	0.967
	0.1	25.94%	14.28%	2759	0.00117
`integral_s`	0.01	57.63%	4.45%	44458	0.0445
	0.001	94.09%	0.87%	583474	0.583
`integral`		29.68 %			0.0006990
`chebfun`		18.91%			0.009456

Some commonly available numerical algorithms in MATLAB are `integral` [15] and the MATLAB Chebfun toolbox [8]. We applied these two routines to the random family of test functions as well. Their success and failure rates are also recorded in Table 6.1. They do not give warnings of possible failure.

The test results in Table 6.1 shows that both `integral_t` and `integral_s` return higher success rates as the initial cut-off value $\hbar$ being smaller. This makes sense because higher cut-off value provides more inclusive cones. The chance of a bump function lie in the cone is higher. In a other word, smaller cut-off value provides finer mesh, so that the chances of detecting narrow peaks are higher. For the same reason, for both `integral_t` and `integral_s`, the success rates with cone change warning decreases as $\hbar$ becoming smaller. Since when the cut-off value is high, there

is not as many out-of-cone integrands as if the cut-off value is low. Both `integral_t` and `integral_s` outperform `integral` and Chebfun toolbox since the fact that the test functions are defined in order to fool the algorithm. Our algorithms has the mechanism of changing $\mathfrak{h}$ to enhance the success rate. The two comparing algorithms do not have similar settings.

When $\mathfrak{h}$ is low, the average data sites used and the average time elapsed becomes larger for both `integral_t` and `integral_s`,. It is because both algorithms starts with more initial number of points and therefore require longer computational time. If we compare the average cost and average time between `integral_t` and `integral_s`, the performance of the latter is significantly better than the former because the Simpson's rule has higher convergence rate than the trapezoidal rule. Comparing the computational time of both algorithms with `integral` and `chebfun`, out algorithms spend more time to solve the problem. Future works of improving the method are discussed in the next chapter.

Another test are implemented with a family of test functions similar to the first one. In this case, the test functions on interval (0.1) are defined by (6.1) with $t \sim \mathcal{U}[0, 1-4\delta]$ and $\log_{10}(\delta) \sim \mathcal{U}[-3,-1]$. A total number of 1000 random functions are tested with the same $N_{\max}$ and error tolerance. The smallest width of the peak is 0.004. For both `integral_t` and `integral_s`, when the cut-off value is set to be 0.001, the mesh is always fine enough to detect the peak. Thus the algorithms is guaranteed to succeed for all test functions. When the cut-off value is either 0.1 or 0.01, the algorithms may not detect all possible peaks. The results shown in Table 6.2 support the theoretical results shown in the previous chapters that the algorithms are guaranteed to work for all the integrands in the cone.

Convergence rate for both algorithms are also important to study. We use the test function $f(x,t,\delta) = \text{bump}(x;t,\delta)/\delta^4$ where $t = 0.2$; $\delta = 0.1$. Both `integral_t`

Table 6.2. Another test for `integral_t` and `integral_s` with wider random peaks.

	$\mathfrak{h}$	Success	Success Warning	Average Cost	Average Time
	0.1	33.60%	13.60%	489605	0.09729
`integral_t`	0.01	82.00%	3.60%	3110154	0.6318
	0.001	100.00%	0.00%	4942823	0.9657
	0.1	35.60%	23.20%	3961	0.001501
`integral_s`	0.01	86.20%	7.50%	56955	0.01144
	0.001	100.00%	0.00%	110109	0.02145
`integral`		41.70 %			0.0008824
`chebfun`		26.40%			0.01380

and `integral_s` use $\mathfrak{h} = 0.1$. The following picture shows the convergence rate for the two algorithms.

From the picture we can see that `integral_s` converges faster than `integral_t`. Moreover, `integral_t` converges at the rate of $-1/2$ to the logarithm. And `integral_s` converges at the rate of $-1/4$ to the logarithm. The results support the results of the upper bound on the computational cost.

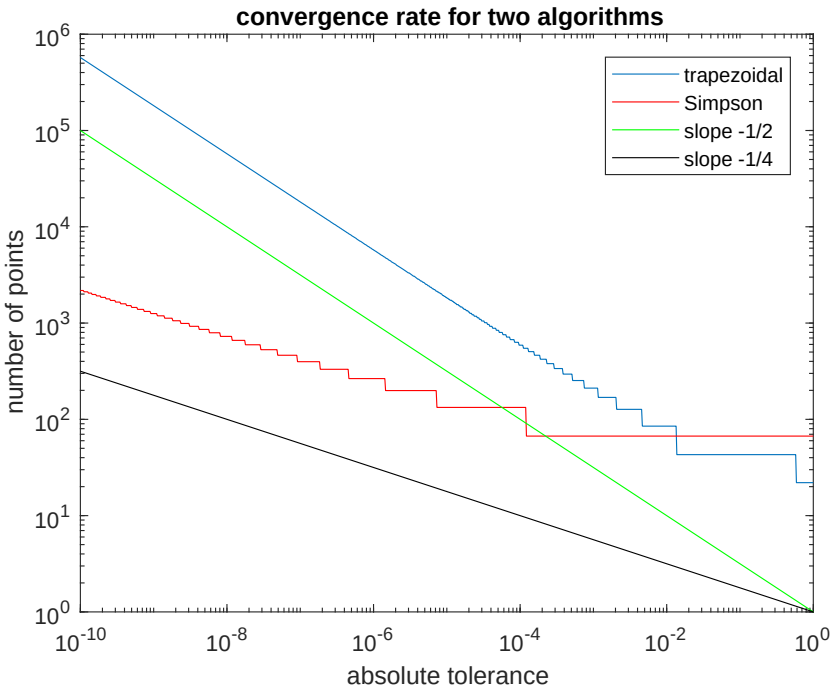

Figure 6.1. The convergence curves for `integral_t` and `integral_s`.

CHAPTER 7

CONCLUSION

In this thesis, we study the algorithms for univariate integration problems. Both algorithms using the trapezoidal rule and Simpson's rule are constructed. The approximation error for them are analyzed in terms of function values. The computational cost of the algorithms is studied in the sense of lower and upper bound of function value required for a specific error tolerance. The complexity of any algorithms using the trapezoidal rule and Simpson's rule are derived. This chapter summarizes the main results and brings out the future work to be pursued.

7.1 Summary

Numerical approximation motivated by univariate integration problems can be categorized into two typical types. The first category is fixed-cost algorithms with guarantees. These algorithms are automatic but not adaptive since the computational cost does not depend on the integrand. The second category is adaptive, automatic algorithms such as MATLAB's integral and Chebfun. These algorithms do not have rigorous justification. The user does not know whether the returned results from the algorithm is true or not. This thesis proposed two algorithms, using the trapezoidal rule and Simpson's rule respectively, where the integrands lie in cone $\mathcal{V}^1$ and $\mathcal{V}^3$. For the trapezoidal rule, $\mathcal{V}^1 := \{f \in C^1[a,b] : \mathrm{Var}(f') < \infty\}$ and the function space is

$$\mathcal{C}^1 := \Big\{ f \in \mathcal{V}^1, \mathrm{Var}(f') \le \mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{n+1}))\widehat{V}(f', \{x_i\}_{i=0}^{n+1}), \\ \text{for all choices of } n \in \mathbb{N}, \text{and } \{x_i\}_{i=0}^{n+1} \text{ with } \mathrm{size}(\{x_i\}_{i=0}^{n+1}) < \mathfrak{h} \Big\}.$$

For the Simpson's rule, $\mathcal{V}^3 := \{f \in C^3[a,b] : \mathrm{Var}(f''') < \infty\}$ and the function space

is

$$\mathcal{C}^3 := \Big\{ f \in \mathcal{V}^3, \mathrm{Var}(f''') \le \mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{n+1}))\widehat{V}(f''', \{x_i\}_{i=0}^{n+1}), \\ \text{for all choices of } n \in \mathbb{N}, \text{and } \{x_i\}_{i=0}^{n+1} \text{ with } \mathrm{size}(\{x_i\}_{i=0}^{n+1}) < \mathfrak{h} \Big\}.$$

Using backward finite difference, the error estimate of the trapezoidal rule, $\overline{\mathrm{err}}_{\mathrm{t}}(f,n)$, can be bounded as follows using function values.

$$\overline{\mathrm{err}}_{\mathrm{t}}(f,n) \le \frac{(b-a)^2\mathfrak{C}(2(b-a)/n)\widetilde{V}_1(f,n)}{8n^2}.$$

The error estimate of the Simpson's rule, $\overline{\mathrm{err}}_{\mathrm{s}}(f,n)$, can be bounded as follows using function values.

$$\overline{\mathrm{err}}_{\mathrm{s}}(f,n) \le \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{93312n^4}.$$

Detailed processes of the two new algorithms are provided. The two algorithm are proven to be successful if the integrand lies into the appropriate cone. The computational cost of the trapezoidal algorithm has lower and upper bound as

$$\begin{aligned} &\max\left(\left\lfloor \frac{2(b-a)}{\mathfrak{h}} \right\rfloor, \left\lceil 2(b-a)\sqrt{\frac{\mathrm{Var}(f')}{8\varepsilon}} \right\rceil\right) \le N(f,\varepsilon) \\ &\le 2\min\left\{ n \in \mathbb{N} : n \ge \left(\left\lfloor \frac{2(b-a)}{\mathfrak{h}} \right\rfloor\right), \zeta(n)\,\mathrm{Var}(f') \le \varepsilon \right\} \\ &\le 2\min_{0<\alpha\le 1} \max\left(\left(\left\lfloor \frac{2(b-a)}{\alpha\mathfrak{h}} \right\rfloor\right), (b-a)\sqrt{\frac{\mathfrak{C}(\alpha\mathfrak{h})\,\mathrm{Var}(f')}{8\varepsilon}}\right), \end{aligned}$$

where $\zeta(n) = (b-a)^2\mathfrak{C}(2(b-a)/n)/(8\varepsilon n^2)$. The number of function values, namely, the computational cost required by the algorithm, is $N(f,\varepsilon)+1$. The computational

cost of the Simpson's algorithm has lower and upper bound as

$$\begin{aligned}
&\max\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor, \left\lceil(b-a)\left(\frac{\operatorname{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right\rceil\right) \le N(f,\varepsilon) \\
&\le 2\min\left\{n\in\mathbb{N}: n\ge\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor\right), \zeta(n)\operatorname{Var}(f''')\le\varepsilon\right\} \\
&\le 2\min_{0<\alpha\le 1}\max\left(\left(\left\lfloor\frac{(b-a)}{\alpha\mathfrak{h}}\right\rfloor\right), (b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{93312\varepsilon}\right)^{1/4}\right),
\end{aligned}$$

where $\zeta(n) = (b-a)^4\mathfrak{C}((b-a)/n)/(93312n^4)$. The number of function values, namely the computational cost, required by the algorithm is $6N(f,\varepsilon)+1$.

The lower bounds on the complexity for any algorithms using the trapezoidal rule and Simpson's rule are studied by constructing fooling functions. For the trapezoidal rule, the complexity has a lower bound of

$$-\frac{3}{2}+\frac{b-a-3\mathfrak{h}}{4}\sqrt{\left\lfloor\frac{[\mathfrak{C}(0)-1]\operatorname{Var}(f')}{\varepsilon}\right\rfloor}.$$

For the Simpson's rule, the complexity has a lower bound of

$$-\frac{5}{4}+\frac{b-a-5\mathfrak{h}}{8}\left\lfloor\frac{[\mathfrak{C}(0)-1]\operatorname{Var}(f''')}{\varepsilon}\right\rfloor^{1/4}.$$

The upper bounds of algorithms using the trapezoidal rule and Simpson's rule is asymptotically to the same order of the complexity of the problem, respectively. It is shown that our algorithms are optimal.

Results from numerical experiments of testing a family of bump functions in $\mathcal{V}^3$ show that finer mesh provides higher success rates for both `integral_t` and `integral_s`. Both algorithms has better success rate than MATLAB's `integral` and Chebfun. The computational cost and time of the Simpson's rule are significantly better than the ones of the trapezoidal rule. When the mesh is fine enough, the test results supports the theoretical results that the algorithms are guaranteed to work.

Results also show that the Simpson's rule algorithm has faster convergence rate than the trapezoidal rule algorithm.

7.2 Future Work

Starting from our new algorithms which are designed to solve the univariate integration problems, there are more to explore. The algorithms use the trapezoidal rule and Simpson's rule and are globally adaptive. For future work, new methods can be proposed to solve multivariate integration problems. The input points can be locally adaptive for the trapezoidal rule, where additional points are only added to the necessary locations according to the shape of the integrand. Making the algorithms locally adaptive may significantly save the computational cost. The error implemented in the derivations and algorithms is the absolute error. Relative error may provide more stable results when the integral is very small and be worthy to study. The convergence rates of the trapezoidal rule and Simpson's rule are not fast enough. New algorithms can be created using methods with faster convergence rate. For integration problems, some of the aforementioned works are finished by the GAIL group. Some of them are undergoing. But more can be investigated to make the idea of the cone shine brighter.

BIBLIOGRAPHY

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

[2] S.-C. T. Choi, Y. Ding, F. J. Hickernell, L. Jiang, Ll. A. Jiménez Rugama, D. Li, R. Jagadeeswaran, X. Tong, K. Zhang, Y. Zhang, and X. Zhou. GAIL: Guaranteed Automatic Integration Library (versions 1.0–2.2). MATLAB software, 2013–2017.

[3] S.-C. T. Choi, Y. Ding, F. J. Hickernell, and X. Tong. Local adaption for approximation and minimization of univariate functions. *J. Complexity*, 40:17–33, 2017.

[4] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Ll. A. Jiménez Rugama, Xin Tong, Yizhi Zhang, and Xuan Zhou. GAIL—Guaranteed Automatic Integration Library in MATLAB: Documentation for version 2.1. *arXiv:1503.06544*, 2015.

[5] N. Clancy, Y. Ding, C. Hamilton, F. J. Hickernell, and Y. Zhang. The cost of deterministic, adaptive, automatic algorithms: Cones, not balls. *J. Complexity*, 30:21–45, 2014.

[6] R. Cools and D. Nuyens, editors. *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, volume 163 of *Springer Proceedings in Mathematics and Statistics*. Springer-Verlag, Berlin, 2016.

[7] Y. Ding, F. J. Hickernell, and H. Woźniakowski. Adaptive piecewise polynomial function recovery, 2016+. in preparation.

[8] N. Hale, L. N. Trefethen, and T. A. Driscoll. *Chebfun Version 4*, 2012.

[9] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen. Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling. In J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2012*, volume 65 of *Springer Proceedings in Mathematics and Statistics*, pages 105–128. Springer-Verlag, Berlin, 2013.

[10] F. J. Hickernell and Ll. A. Jiménez Rugama. Reliable adaptive cubature using digital sequences. In Cools and Nuyens [6], pages 367–383. arXiv:1410.8615 [math.NA].

[11] F. J. Hickernell, M. Razo, and S. Yun. Reliable adaptive numerical integration, 2015+. submitted for publication.

[12] L. Jiang and F. J. Hickernell. Guaranteed conservative confidence intervals for means of Bernoulli random variables. arXiv:1411.1151, 2014. submitted for publication.

[13] Ll. A. Jiménez Rugama and F. J. Hickernell. Adaptive multidimensional integration based on rank-1 lattices. In Cools and Nuyens [6], pages 407–422. arXiv:1411.1966.

[14] J. N. Lyness. When not to use an automatic quadrature routine. *SIAM Rev.*, 25:63–87, 1983.

[15] The MathWorks, Inc. *MATLAB 9.3*. Natick, MA, 2017.

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