GUARANTEED, ADAPTIVE, AUTOMATIC ALGORITHMS FOR UNIVARIATE INTEGRATION: METHODS, COST, 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
22nd October, 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 withour 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 is motivated by solving the univariate integration problems using numerical methods, including trapezoidal rule and Simpson's rule. Most existing guaranteed algorithms are not adaptive and require too much information. Most existing adaptive algorithms do not valid justification of the results. The goal is to create adaptive algorithms utilizing the two mentioned methods with guarantees. The class of integrands studied in this thesis reside in two cone shaped function spaces with respect to two different methods. The algorithms are analytically proved to success if the input function lies in the cone. The algorithms are adaptive and automatically adjust the computational cost based on input function values. The lower and upper bounds on the computational cost for both algorithms are provided. The lower bounds on the complexity of any algorithms to the problem are provided. By comparing the upper bounds on computational cost and lower bounds on complexity, our algorithms are asymtotically 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 is lack of analytical solution, or the integrand is only known at certain points by sampling. This thesis focus on the univariate integration problem

$$\mathrm{INT}(f) = \int_a^b f(x)\,\mathrm{d}x \in \mathbb{R}. \tag{1.1}$$

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

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

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

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" means that the algorithms automatically expend the right amount of computational effort needed to 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 input function. 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 Input Functions

Fixed-cost algorithms with guarantees usually require input functions to lie in a ball-shaped function space. 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 trapezoidal rule, $p = 1$; for Simpson's rule, $p = 3$. In order to make sure of the success of the algorithms, 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 $n = C^{-1}(\varepsilon/\sigma)$. Fixed-cost algorithms such as trapezoidal rule and Simpson's rule are automatic because the computational effort required is depended 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 obtain properties of the input function. If a moderate σ is selected. The algorithms may work for easy functions but not for functions with sharp peaks. Functions with sharp peaks may has derivatives that change rapidly in a small interval. The variation of the derivative could be high enough to exceed the radius σ of the ball. Secondly, if the algorithms work for f, it 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 integration of cf can be easily calculated as $c\mathrm{INT}(f)$. Moreover, those algorithms have the computa-

tional cost depending on σ, which does not depend on function values. Therefore, the algorithms cannot adjust the computational cost according to the input function. 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 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. These algorithms work well in many cases. However, they 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. 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.

In fact, the challenge of spiky integrands applies to 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 space of the input function as a cone, not a ball. In a cone space, linear multiplications of a function remains in the cone. Moreover, the respective derivatives of the functions cannot change too much in a small interval. The bound on error estimates is also defined and approximated using only function values (to make the algorithms adaptive). In contrast to fixed-cost algorithms, our methods are adaptive and succeed for a cone of integrands. In contrast to the adaptive algorithms, our methods have rigorous proof of how spiky an integrand can be to stay inside the cone.

1.3 Motivation and Outline

In Chapter 2, the definitions and assumptions used in the thesis are introduced. Chapter 3 focuses on finding upper bound on approximation error and rigorous theoretical proof of success. Guaranteed algorithms are presented in Chapter 4 with the lower and upper bound of the computational cost. Chapter 5 studies the lower bound of complexity. In Chapter 6, the results from numerical experiences will be discussed. The guaranteed algorithms derived here are based on 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 introduced the univariate integration problem as (1.1). The building blocks of the adaptive, automatic algorithms are two widely used fixed cost algorithms, 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 we $[a,b]$ are divided into $6n$ intervals is that firstly, 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, 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 as mentioned in (1.4). Here we

give detailed definitions on $C(n)$:

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

$$C(n) = \begin{cases} \frac{(b-a)^2}{8n^2}, p = 1, & \text{trapezoidal rule[1, (7.15)]}, \\ \frac{(b-a)^4}{5832n^4}, p = 3, & \text{Simpson's rule[1, (p.231)]}. \end{cases}$$

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:

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

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

Our algorithms will 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 space $\mathcal{C}^p$ 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

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

for convenience.

With the cone space 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 BOUND

In this chapter, the author derives upper bounds on the approximation errors in terms of function values. By (2.5) in Chapter 2, we know that the approximation errors of 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 Trapezoidal Rule

The backward finite difference are used to approximate f'. Let the width of the interval $h = u_{j+1} - u_j = (b-a)/n$ 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 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 $\{a = x_0, x_1, \cdots, x_n, x_{n+1} = b\}$ 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 $\widehat{V}(f^{(1)},\{x_j\}_{i=0}^{n+1})$ using only function values is defined as:

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

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 composite trapezoidal rule.

Lemma 1. *The approximation error of the composite trapezoidal rule is bounded in term of the function values of the input function as follows:*

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

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 going up, $\mathfrak{C}(2(b-a)/n)$ is non-increasing. 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 input function. With a user provided tolerance ε, as long as n is big enough, the approximation error will eventually decrease below ε. Therefore, the error bound (3.4) of our algorithm using only function values provides guarantees that it will succeed.

3.2 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 divided 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 $\{a = x_0, x_1, \cdots, x_n, x_{2n+1} = b\}$ 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

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

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 $\widetilde{V}_1(f,n)$ to $\widehat{V}(f^{(p)},\{x_i\}_{i=0}^{n+1})$ using only function values can be written as follow:

$$\widetilde{V}_3(f,n) = \frac{216n^3}{(b-a)^3}\sum_{j=1}^{2n-1} |f(v_{3j+3}) - 3f(v_{3j+2}) + 3f(v_{3j+1}) \\ -2f(v_{3j}) + 3f(v_{3j-1}) - 3f(v_{3j-2}) + f(v_{3j-3})|, \quad (3.7)$$

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:

$$\begin{aligned}
\overline{\mathrm{err}}_{\mathrm{s}}(f,n) :=& C(n)\,\mathrm{Var}(f'''), && \text{(by (2.5))}\\
\le& C(n)\mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{2n+1}))\widehat{V}(f''',\{x_i\}_{i=0}^{2n+1}), && \text{(by (2.9))}\\
=& C(n)\mathfrak{C}(\mathrm{size}(\{x_j\}_{i=0}^{2n+1}))\widetilde{V}_3(f,n), && \text{(by (3.7))}\\
\le& \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{5832n^4}. && \text{(by (3.6))}
\end{aligned}$$

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

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

$$\overline{\mathrm{err}}_s(f,n) \le \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{5832n^4}. \quad (3.8)$$

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 will eventually decrease below ε. Therefore, this error bound (3.8) of our algorithm using only function values provides guarantees that it will succeed.

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 THEIR COMPUTATIONAL COST

In the previous Chapter, the stopping criterion of our trapezoidal rule and Simpson's rule algorithms using input function values are derived. In the Chapter, the two algorithms are discussed in detail. Our algorithms require the inputs of function values and error tolerance. The algorithms give out an answer adaptively by deciding the number of functions value used, the location of the points, and the shape of the cone by themselves. For practical reasons, the algorithms set a max number of iterations and a max number of input points. If the either of the max 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 input points used will be multiplied by an integer and the function values from the last iteration will be saved and used again.

4.1 Trapezoidal Rule

Our algorithm is guaranteed to work for functions lying in the cone-shaped function space. 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 out 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, the author proposes a method later in this section. This method tries to make the cone function space a more inclusive new one so that it contains more functions than before.

Firstly, the derivation of finding a necessary condition for which $f \in \mathcal{C}^1$ starts from the definition of the variation and the cone space. From (2.7), the approxi-

mation $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ is a lower bound on $\mathrm{Var}(f')$, i.e. $\widehat{V}(f', \{x_j\}_{j=0}^{n+1}) \leq \mathrm{Var}(f')$ for all $n \geq 2$. From the definition of the cone (2.9), the variation of the first derivative of the input function 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. $\mathrm{Var}(f') \leq \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ for all $n \geq 2$. 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, $\mathfrak{C}(2(b-a)/n) \leq \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$. From (3.3), $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ can be approximated by $\widetilde{V}_1(f, n)$ 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 multiplies. But $\mathfrak{C}(2(b-a)/n)$ decreases since it is a non-decreasing function. Therefore, the largest lower bound on $\mathrm{Var}(f')$ is always the one at the current kth loop. The smallest upper bound 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 input function being in the cone function space $\mathcal{C}^1$, the function must satisfy the following condition:

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

Secondly, if the input function does not meet (4.1) and falls out of the cone,the cut-off value $\mathfrak{h}$ is halved to make the cone function space more inclusive. The reason is that the inflation factor defined in (2.11) is decreasing with respect to $\mathfrak{h}$. Thus by halving $\mathfrak{h}$, the inflation factor becomes larger. Intuitively, the cone of function space is more inclusive.

4.1.1 Trapezoidal Rule Algorithm. Following the settings of how to check whether the input function is in the cone, and how to revise the cone if needed, the author now provides the guaranteed automatic integration algorithm using trapezoidal rule.

Algorithm 1 (Trapezoidal Rule Adaptive Algorithm)**.** 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 input function f and error tolerance ε, do the following:

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

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

b) Otherwise, recompute the upper bound on $\mathrm{Var}(f')$, for n_j, such that $j \in \mathfrak{J}$ by the following:

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.

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

a) If this is true, return $T_{n_k}(f)$ in (2.1) and terminate the algorithm.

b) Otherwise, go the Step 6.

Step 6. Increase 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 Algorithm 1 firstly 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 input function 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 input function fits into the cone. 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 start over. Even if the mesh is fine enough, the function may still not lie in the new cone. And the upper bounds on $\mathrm{Var}(f')$ are changed with the new $\mathfrak{h}$. Thus, η_k is recalculated in 4.b). And the algorithm goes to the beginning of step 4 to check the necessary condition again. Step 5 checks whether the error estimates 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. Algorithm 1 always succeeds if the input function is in the cone and the number of nodes is large enough.

Theorem 1. *Algorithm 1 is successful, i.e.,*

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

Proof. From Lemma 1 and the description below it, Algorithm 1 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 Trapezoidal Algorithm. Having created the multistage adaptive algorithm using trapezoidal rule, the author continues to find the computational cost of the algorithm. The computational cost of trapezoidal 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 the algorithm terminates. Then this number is bounded below and above in terms of the true, yet unknown,* $\operatorname{Var}(f')$.

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

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, $N(f,\varepsilon)\ge n_1=(\lfloor 2(b-a)/\mathfrak{h}\rfloor)$. Then the number of intervals increases until $\overline{\operatorname{err}}(f,n)\le\varepsilon$. From the error bound defined in (2.5), $C(N(f,\varepsilon))\operatorname{Var}(f')\le\varepsilon$. Thus $N(f,\varepsilon)\ge\left\lceil 2(b-a)\sqrt{\operatorname{Var}(f')/(8\varepsilon)}\right\rceil$ This implies the lower bound on $N(f,\varepsilon)$.

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

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

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

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

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

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:

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

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

4.2 Simpson's rule

Similar to the trapezoidal algorithm, firstly the author provides the necessary condition for the input function lying in 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. $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1}) \leq \mathrm{Var}(f''')$ 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 $\mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$, 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 $n \geq 6$. After similar derivation as the trapezoidal case, the necessary condition for an input function being in the cone function space $\mathcal{C}^3$ at the kth loop is:

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

Same mechanism, namely halving the cuf-off value $\mathfrak{h}$ is implemented to make the cone function space more inclusive in the Simpson's rule case.

4.2.1 Simpson's Rule Algorithm. Now the author provides the guaranteed automatic integration algorithm using Simpson's rule.

Algorithm 2 (Simpson's Rule Adaptive Algorithm)**.** 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 $k = 0$, $n_k = 0$. For any input function f and error tolerance ε, do the following:

Step 1. Reset upper bound and increase number of nodes. Let $\eta_k = \infty$ and $n_{k+1} = n_k \times \lceil (b-a)/\mathfrak{h}n_k \rceil$.

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 $\widetilde{V}_3(f, n_k) \leq \eta_k$, then go to Step 5. Otherwise, set $\mathfrak{h} = \mathfrak{h}/2$.

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

b) Otherwise, recompute the upper bound on $\mathrm{Var}(f''')$ for n_j, such that $j \in \mathfrak{J}$ by the following:

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.

$$\eta_k \leq \frac{5832\varepsilon n_k^4}{(b-a)^4}.$$

a) If true, return $S_{n_k}(f)$ in (2.1) and terminate the algorithm.

b) Otherwise, go the Step 6.

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

$$n_{k+1} = n_k \times \max\left(\left\lceil \frac{(b-a)}{3n_k}\left(\frac{\widetilde{V}_3(f,n_k)}{72\varepsilon}\right)^{1/4}\right\rceil, 2\right).$$

Go to Step 2.

The process of Algorithm 2 is similar to Algorithm 1. Algorithm 2 always succeeds if the input function is in the cone $\mathcal{C}^3$ and the number of nodes is large enough.

Theorem 3. *Algorithm 2 is successful, i.e.,*

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

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

4.2.2 Computational Cost of Simpson's Algorithm. Having created the multistage adaptive algorithm using Simpson's rule, we now continue to find the computational cost of the algorithm. The computational cost of 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 the algorithm terminates. Then this number is bounded below and above in terms of the true, yet unknown,* $\operatorname{Var}(f''')$.

$$\begin{aligned}\max\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor,\left\lceil(b-a)\left(\frac{\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}\right\rceil\right) &\le N(f,\varepsilon)\\ &\le 2\min\left\{n\in\mathbb{N}:n\ge\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor\right),\zeta(n)\operatorname{Var}(f''')\le\varepsilon\right\}\\ &\le 2\min_{0<\alpha\le1}\max\left(\left(\left\lfloor\frac{(b-a)}{\alpha\mathfrak{h}}\right\rfloor\right),(b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}\right), \quad (4.4)\end{aligned}$$

where $\zeta(n)=(b-a)^4\mathfrak{C}((b-a)/n)/(5832n^4)$. 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, $N(f,\varepsilon)\ge n_1=(\lfloor(b-a)/\mathfrak{h}\rfloor)$. Then the number of intervals increases until $\overline{\operatorname{err}}(f,n)\le\varepsilon$. This implies the lower bound on $N(f,\varepsilon)$.

Let K be the value of k for which Algorithm 2 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 $\zeta((m-1)n_{K-1})\operatorname{Var}(f''')>\varepsilon$. For $m=2$, this is true because $\zeta(n_{K-1})\operatorname{Var}(f''')\ge\zeta(n_{K-1})\widetilde{V}_{n_{K-1}}(f)>\varepsilon$. For $m>2$ it is true because of the definition of m. Since ζ is a decreasing function, it follows that

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

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

$$n^*\le\max\left(\left\lfloor\frac{2(b-a)}{\alpha\mathfrak{h}}\right\rfloor+1,(b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}+1\right),\quad 0<\alpha<1.$$

For fixed $\alpha\in(0,1]$, we only need to consider that case where $n^*>\lfloor2(b-a)/(\alpha\mathfrak{h})\rfloor+1$. This implies that $n^*-1>\lfloor2(b-a)/(\alpha\mathfrak{h})\rfloor\ge2(b-a)/(\alpha\mathfrak{h})$ thus $\alpha\mathfrak{h}\ge2(b-a)/(n^*-1)$.

Also by the definition of n^*, ζ, and $\mathfrak{C}$ is non-decreasing:

$$\begin{aligned}
&\zeta(n^*-1)\operatorname{Var}(f''') > \varepsilon,\\
\Rightarrow 1 &< \left(\frac{\zeta(n^*-1)\operatorname{Var}(f''')}{\varepsilon}\right)^{1/4},\\
\Rightarrow n^*-1 &< n^*-1\left(\frac{\zeta(n^*-1)\operatorname{Var}(f''')}{\varepsilon}\right)^{1/4},\\
&= n^*-1\left(\frac{(b-a)^4\mathfrak{C}(2(b-a)/(n^*-1))\operatorname{Var}(f''')}{5832(n^*-1)^4\varepsilon}\right)^{1/4},\\
&\le (b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}.
\end{aligned}$$

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 BOUND OF COMPLEXITY

In the previous Chapter, the author discusses 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 [4, 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 $\mathrm{int}(\cdot, \varepsilon)$ as any algorithm that solves univariate integration problem. Two fooling functions are built for 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 Trapezoidal rule

The construction of fooling function for the trapezoidal rule starts from creating a triangle shaped function $\mathrm{peak}(x, t, \delta)$ 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:

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

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

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

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

Picture 5.1 shows the shape of the function.

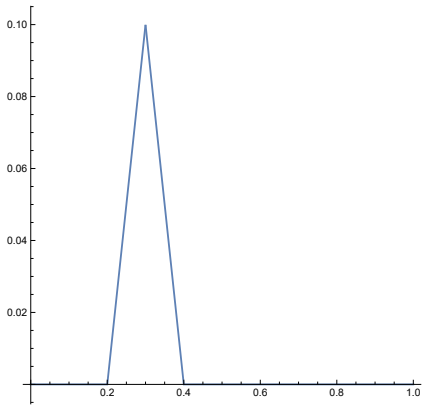

Figure 5.1. A peak function where $a = 0$, $b = 1$, $t = 0.2$, $\delta = 0.1$.

Other properties of the peak function are displayed below:

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

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

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

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

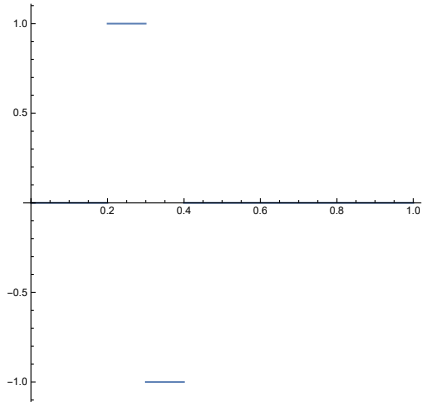

Figure 5.2. First derivative of peak function 5.1.

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

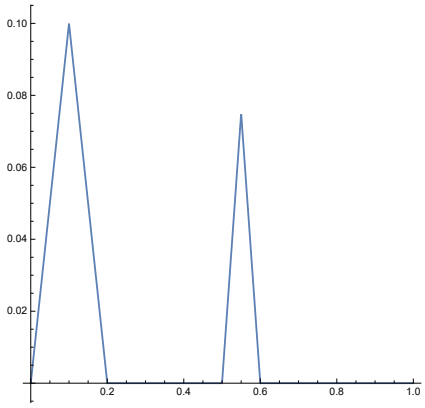

Figure 5.3. A fooling function that can fool the trapezoidal rule algorithms, where $a = 0$, $b = 1$, $t = 0.5$, $\delta = 0.05$ and $\mathfrak{h} = 0.1$.

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 leads to the lower bound of complexity of the trapezoidal algorithm. The key to the proof is to find one appropriate δ. This δ makes the smaller peak resides within one interval of data sites. Thus the small peak cannot be detected by the algorithm. Therefore, the algorithm cannot distinguish between peak$'(x; t, \delta)$

and twopk$(x;t,\delta,\pm)$. Assume that the algorithm 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 trapezoidal rule can be represented as:

Theorem 5. *Let* 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* $\mathrm{Var}(f')$*, there will be some* $f \in \mathcal{C}^1$ *for which* int$(\cdot,\varepsilon)$ *must use at least*

$$-\frac{3}{2} + \frac{b-a-3\mathfrak{h}}{8}\sqrt{\left[\frac{[\mathfrak{C}(0)-1]\,\mathrm{Var}(f')}{\varepsilon}\right]} \tag{5.3}$$

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

Proof. For any positive α, suppose that int$(\cdot,\varepsilon)$ evaluates integrand αpeak$'(\cdot;t,\delta)$ at n nodes before returning to an answer. Let $\{x_j\}_{j=1}^{m}$) be the $m < n$ ordered nodes used by `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 $\text{int}(\cdot,\varepsilon)$ cannot distinguish between $\alpha\text{peak}(\cdot;t,\delta)$ and $\alpha\text{twopk}(\cdot;t,\delta,\pm)$. Thus

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

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 $\text{int}(\cdot,\varepsilon)$ is successful for all of the functions.

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

Substituting δ in terms of n:

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

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

Finally, comparing the upper bound on the computational cost of integral in (4.4) with the lower bound on the computational cost of the best algorithm in (5), both of them increase as $\mathcal{O}((\mathrm{Var}(f')/\varepsilon))^{1/2}$ as $(\mathrm{Var}(f')/\varepsilon)^{1/2} \to \infty$. Thus integral is optimal. □

5.2 Simpson's rule

To build the fooling functions for Simpson's rule, the B-Spline expansion in the previous section continuous 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.

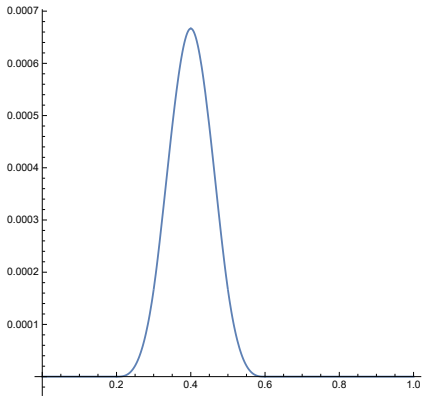

Figure 5.4. A bump function, where $a = 0$, $b = 1$, $t = 0.2$, $\delta = 0.1$ and $\mathfrak{h} = 0.1$.

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 third derivative with finite variation of the third derivatives. Note that the following double-bump function that is always

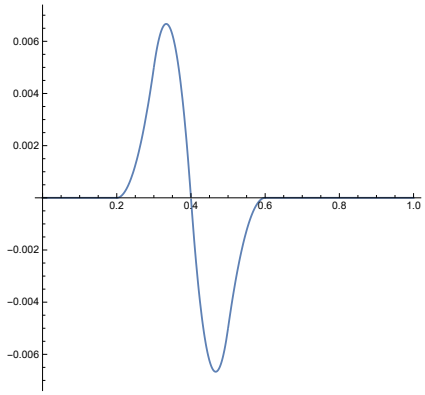

Figure 5.5. First derivative of bump function 5.4.

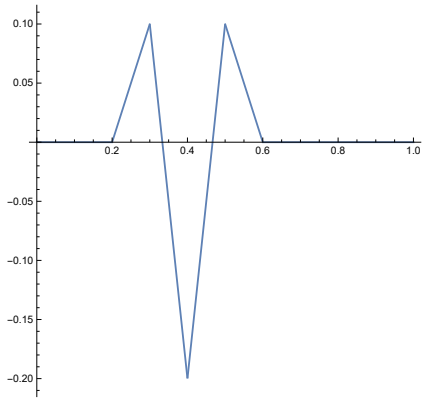

Figure 5.6. Second derivative of bump function 5.4.

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.

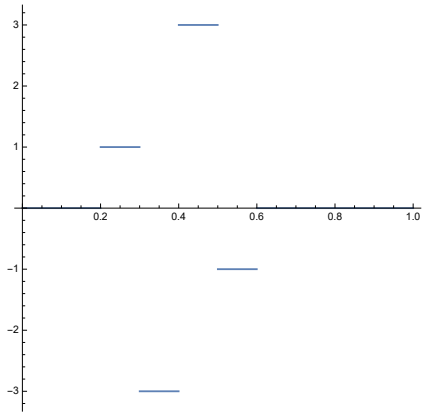

Figure 5.7. Third derivative of bump function 5.4.

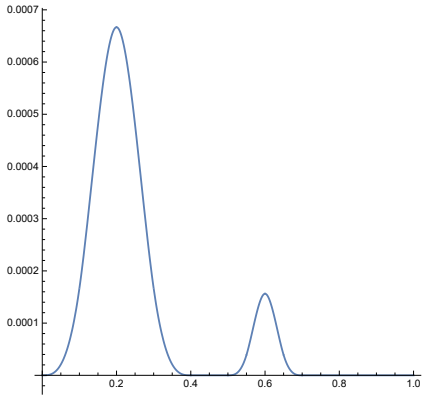

Figure 5.8. A fooling function that can fool the Simpson's rule algorithms, where $a = 0$, $b = 1$, $t = 0.5$, $\delta = 0.05$ and $\mathfrak{h} = 0.1$.

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 $\mathcal{C}$, and only uses function values. For any error tolerance $\varepsilon > 0$ and any arbitrary value of* $\mathrm{Var}(f''')$*, there will be some $f \in \mathcal{C}$ 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 Algorithm 2. The adaptive Simpson's Algorithm 2 has optimal order for integration of functions in $\mathcal{C}^3$.*

Proof. For any positive α, suppose that $\texttt{int}(\cdot, \varepsilon)$ 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 $\mathrm{int}(\cdot, \varepsilon)$ 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 $\mathrm{int}(\cdot, \varepsilon)$ cannot distinguish between $\alpha\mathrm{bump}(\cdot; t, \delta)$ and $\alpha\mathrm{twobp}(\cdot; t, \delta, \pm)$. Thus

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

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

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

Substituting δ in terms of n:

$$\begin{aligned}
4n+5 = \frac{b-a-5\mathfrak{h}}{\delta} &\geq (b-a-5\mathfrak{h})\left[\frac{[\mathfrak{C}(\delta)-1]\,\mathrm{Var}(\alpha\mathrm{bump}'''(\cdot;a,\mathfrak{h})))}{16\varepsilon}\right]^{1/4}, \\
&\geq -\frac{5}{4} + \frac{b-a-5\mathfrak{h}}{8}\left[\frac{[\mathfrak{C}(0)-1]\,\mathrm{Var}(\alpha\mathrm{bump}'''(\cdot;a,\mathfrak{h}))}{\varepsilon}\right]^{1/4}.
\end{aligned}$$

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

Finally, comparing the upper bound on the computational cost of `integral` in (4.4) with the lower bound on the computational cost of the best algorithm in (6), both of them increase as $\mathcal{O}((\mathrm{Var}(f''')/\varepsilon))^{1/4}$ as $(\mathrm{Var}(f''')/\varepsilon)^{1/4} \to \infty$. Thus `integral` is optimal. □

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

In the next chapter, the numerical results from Algorithm 1 and Algorithm 2 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 $t \sim \mathcal{U}[0, 1-4\delta]$, $\log_{10}(h) \sim \mathcal{U}[-4, -1]$. 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 randomm quarter width expands from 0.0001 to 0.1. As an experiment, we chose 10000 random test functions and applied Algorithm 1 and Algorithm 2 with an error tolerance of $\varepsilon = 10^{-8}$. The initial values of $\mathfrak{h}$ are set as 0.1, 0.01, and 0.001. The algorithm is considered successful for a particular f if the exact and approximate integrals have a difference no greater than ε. The success and failure rates are given in Table 6.

Some commonly available numerical algorithms in MATLAB are `integral` [3] and the MATLAB Chebfun toolbox [2]. 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.

The test results in Table 6 shows that both Algorithm 1 and Algorithm 2 returns higher success rates as the initial cut-off value $\mathfrak{h}$ being smaller. This makes sense because higher cut-off value provides finer mesh. The chance of a bump function to have a width less than the input interval is smaller. Thus the algorithms cannot be fooled as easily as the mesh is coarser. The Simpson's rule algorithm outperformed the trapezoidal rule algorithm when the two use the same initial cut-off value. It also makes sense since the test function changes more rapidly to the first derivative than

	$\mathfrak{h}$	Success	Failure
	0.1	5%	95%
Algorithm 1	0.01	26%	74%
	0.001	48%	52%
	0.1	33%	67%
Algorithm 2	0.01	60%	40%
	0.001	77%	23%
`integral`		15 %	85%
`chebfun`		29%	71%

Table 6.1. The success rate of Algorithm 1 and Algorithm 2 plus the success rates of other common quadrature algorithms.

the third derivative. The family of $f(x, t, \delta)$ has higher chance to fall out of $\mathcal{V}^1$ than $\mathcal{V}^3$. Both Algorithm 1 and Algorithm 2 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.

CHAPTER 7

CONCLUSION

In this thesis, we study the algorithms for univariate integration problems. Both algorithms using 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 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 input function. 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 trapezoidal rule and Simpson's rule respectively, where the input functions lie in a cone shaped function subspace of $\mathcal{V}^1$ and $\mathcal{V}^3$. For trapezoidal rule, $\mathcal{V}^1 := \{f \in C^{(1)}[a,b] : \mathrm{Var}(f') < \infty\}$ and the function space is

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

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

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

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

$$\overline{\mathrm{err}}_{\mathrm{s}}(f,n) \le \frac{(b-a)^4\mathfrak{C}((b-a)/n)\widetilde{V}_3(f,n)}{5832n^4}.$$

Detailed processes of the two new algorithms are provided. The two algorithm are proved to be successful if the input function lies into the appropriate cone shaped function space. The computational cost of trapezoidal rule 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 Simpson's rule has lower and upper bound as

$$\begin{aligned}
&\max\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor, \left\lceil(b-a)\left(\frac{\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}\right\rceil\right) \le N(f,\varepsilon) \\
&\le 2\min\left\{n\in\mathbb{N}: n\ge\left(\left\lfloor\frac{(b-a)}{\mathfrak{h}}\right\rfloor\right), \zeta(n)\operatorname{Var}(f''')\le\varepsilon\right\} \\
&\le 2\min_{0<\alpha\le 1}\max\left(\left(\left\lfloor\frac{(b-a)}{\alpha\mathfrak{h}}\right\rfloor\right), (b-a)\left(\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f''')}{5832\varepsilon}\right)^{1/4}\right),
\end{aligned}$$

where $\zeta(n) = (b-a)^4\mathfrak{C}((b-a)/n)/(5832n^4)$. 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 trapezoidal rule and Simpson's rule are studied by constructing fooling functions. For trapezoidal rule, the complexity has a lower bound of

$$-\frac{3}{2}+\frac{b-a-3\mathfrak{h}}{8}\sqrt{\left\lfloor\frac{[\mathfrak{C}(0)-1]\operatorname{Var}(f')}{\varepsilon}\right\rfloor}.$$

For 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 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 Simpson's rule algorithm has better success rate than trapezoidal rule algorithm. Both algorithms has better success rate than MATLAB's `integral` and Chebfun.

7.2 Future Work

Our new algorithms are designed to solve the univariate integration problems. The algorithms use 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, where additional points are only added to the necessary locations according to the shape of the input function. Making the algorithms locally adaptive may significantly save the computational cost. The convergence rates of trapezoidal rule and Simpson's rule are not fast enough. New algorithms can be created using methods with faster convergence rate.

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

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

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