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
16th October, 2018

ACKNOWLEDGMENT

Will be added once thesis is finished

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Definition
β	List of symbols will be added later

ABSTRACT

Abstract will be included once all parts are finished

CHAPTER 1

INTRODUCTION

In numerical analysis, numerical integration plays an important role. It uses a certain algorithm to calculate the numerical value of a definite integral. It is used instead of those general and mathematically sophisticated analytical methods because that sometimes the problem is lack of analytical solution, or the integrand is only known at certain points by sampling. In this thesis, we will focus on the univariate integration problem

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

We will provide two adaptive automatic algorithms, integral_t, and integral_s, that 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.

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

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

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

Our algorithms are adaptive. By adaptive, it means that the algorithms automatically expand 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 an error estimate of the numerical approximation based on the computations already performed. For those existing

adaptive algorithms, these error estimates either require too much information from the user or based on heuristics and not guaranteed. Details about those 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 lying in a ball-shaped function space. Traditional trapezoidal rule and Simpson's rule in calculus courses are two typical examples of fixed-cost algorithms. In order to make sure that the algorithms succeed, 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 Var $< \sigma$. The computational cost $n = C^{-1}(\varepsilon/\sigma)$. Those algorithms are automatic because it the computational effort required is depended on the error tolerance.

Those algorithms have drawbacks. Firstly, users may not obtain the 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. Secondly, if the algorithms work for f, it may not work for cf, where c is a constant and $c > 1$, because cf may fall out of the ball. Moreover, those algorithms have the computational cost depending on σ, which does not depend on function values. Thus, those 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. 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

The goal is to construct adaptive, automatic algorithms with guarantees of success. In Chapter 2, we introduce the definitions and assumptions used in the thesis. Chapter 3 will focus 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 4 to 6 for the trapezoidal rule and Simpson's rule.

CHAPTER 2

PROBLEM STATEMENT AND ASSUMPTIONS

The previous chapter introduced the univariate integration problem. 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 use $6n$ intervals is that firstly, Simpson's rule requires an even number of intervals. Secondly, we are going to use a third order finite difference to approximate the third derivative of f 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.

We also need to find error bounds to guarantee our methods. 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:

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

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

check 5832 add ref.

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 not necessarily equal spacing, and $a = x_0 \le x_1 < \cdots < x_n \le x_{n+1} = b$. We use $n+2$ points and ignore x_0 and x_{n+1} in (2.6) for convenience later. 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:

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

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, we set up the following assumptions 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:

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

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

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

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

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

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 will show how the algorithms confidently estimate the error and provide data-driven error bounds.

CHAPTER 3

DATA-DRIVEN ERROR BOUND

In this chapter, we will derive upper bounds with quadrature error 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 we will use finite differences to express derivative values in terms of function values.

3.1 Trapezoidal Rule

We used the backward finite difference to approximate f'. Let $h = u_{j+1} - u_j = (b-a)/n$ 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, (ref), 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 cannot be larger than two intervals. So

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

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

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

Therefore by combining the deduction above together, we obtain the error bound for our trapezoidal rule algorithm using only function values:

$$\begin{aligned}\overline{\mathrm{err}}(f,n) :=& C(n)\,\mathrm{Var}(f'), && \text{(by (2.5))}\\ \le& C(n)\mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^{n+1}))\widehat{V}(f',\{x_i\}_{i=0}^{n+1}), && \text{(by (2.9))}\\ =& C(n)\mathfrak{C}(\mathrm{size}(\{x_j\}_{i=0}^{n+1}))\widetilde{V}_1(f,n), && \text{(by (3.3))}\\ \le& \frac{(b-a)^2\mathfrak{C}(2(b-a)/n)\widetilde{V}_1(f,n)}{8n^2}. && \text{(by (3.2))}\end{aligned}$$

Thus, we have the following lemma.

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

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

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. Thus $\overline{\mathrm{err}}_{\mathrm{t}}(f,n)$ is decreasing w.r.t. n. Therefore, this error bound (3.4) of our algorithm using only function values provides guarantees that it will succeed. With a user provided tolerance ε, as long as n is big enough, the approximation error will eventually decrease below ε.

3.2 Simpson's Rule

Similar to the trapezoidal rule, we used the third order backward finite difference to approximate f'''. Let $h = v_{j+1} - v_j = (b-a)/6n$ 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, (ref), for all $j = 1, 2, \cdots, n$, 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 $j = 1, 2, \cdots, n$. 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 cannot be larger than 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 we have the approximation $\widetilde{V}_1(f, n)$ to $\widehat{V}(f^{(p)}, \{x_i\}_{i=0}^{n+1})$ using only function values:

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

Therefore by combining the relative equations together, we obtain the error bound for our Simpson's rule algorithm using only function values:

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

Thus, we have the following lemma.

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:*

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

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. Thus $\overline{\mathrm{err}}_{\mathrm{s}}(f,n)$ is decreasing w.r.t. n. Therefore, this error bound (3.8) of our algorithm using only function values provides guarantees that it will succeed. With a user provided tolerance ε, as long as n is big enough, the approximation error will eventually decrease below ε.

The error bounds of our trapezoidal rule and Simpson's rule algorithms then can be treated as the stopping criterion. In the next Chapter, details about the algorithms will be introduced. The lower bounds and upper bounds of the computational cost will be discussed as well.

CHAPTER 4

ADAPTIVE, AUTOMATIC ALGORITHMS WITH GUARANTEES AND THEIR COMPUTATIONAL COST

In the previous Chapter, we have found the error bounds of our trapezoidal rule and Simpson's rule algorithms using input function values. In the Chapter, firstly we will provide details about the two algorithms. We build our algorithms as black-box algorithms. That is, as long as the user provides the input function and error tolerance, the algorithms will automatically give out an answer by deciding the number of functions value used, the location of the points, and the shape of the cone by themselves. In order to prevent the algorithms taking too long, we will also set a max number of iterations and a max number of input points. If the either of the max number is reached, the algorithms will quit and provide a best possible output. In order to save the computation time, we will use embedded input points 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 those 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 then 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, we will propose a method later in this section. This method tries to make the cone function space a more inclusive new one so that it will contain more functions than before. Firstly, we will find out a necessary condition for which $f \in \mathcal{C}^1$.

From (2.7), we know that $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ 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), we know that 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 $n \geq 2$. 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, $\mathfrak{C}(2(b-a)/n) \leq \mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$. Because the nodes used in the algorithm are nested, for $\widetilde{V}_1(f, n_k) = \widehat{V}(f', \{x_j\}_{j=0}^{n+1})$, $\widetilde{V}_1(f, n_k)$ 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 $\mathrm{Var}(f')$ will always be the one at the current kth loop. However, the smallest upper bound is not always the case. In the algorithm, the lower and upper bounds will be 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, we will try to make the cone function space more inclusive by halving the cut-off value $\mathfrak{h}$. The reason it 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 will be 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, we now provide 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 $k=0$, $n_k=1$,. Let $n_1 = \lceil 2(b-a)/\mathfrak{h} \rceil$. 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.

$$\eta_k \le \frac{8\varepsilon n_k^2}{(b-a)^2}, \mathfrak{C}(2(b-a)/n_k)\widetilde{V}_1(f,n_k) \le \frac{8\varepsilon n_k^2}{(b-a)^2}.$$

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.

The algorithm will always be true if the input function is in the cone and the number of nodes are large enough.

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

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

Proof. From Lemma 1 and the description below it, the algorithm will be guaranteed to success. □

4.1.2 Computational Cost of Trapezoidal Algorithm. Having created the multistage adaptive algorithm using trapezoidal rule, we now continue to find the computational cost of the algorithm. The computational cost of trapezoidal algorithm can be bounded by 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,* $\text{Var}(f')$.

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

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, $N(f,\varepsilon) \geq n_1 = (\lfloor (b-a)/\mathfrak{h} \rfloor + 1)$. Then the number of intervals increases until $\overline{\mathrm{err}}(f,n) \leq \varepsilon$. This implies the lower bound on $N(f,\varepsilon)$.

Let L be the value of l for which Algorithm 1 terminates. Since n_1 satisfies the upper bound, we may assume that $L \geq 2$. Let m be the multiplication integer found in Step 6, and let $m^* = \max(2,m)$. Note that $\zeta((m^*-1)n_{L-1})\,\mathrm{Var}(f') > \varepsilon$. For $m^* = 2$, this is true because $\zeta(n_{L-1})\,\mathrm{Var}(f') \geq \zeta(n_{L-1})\widetilde{V}_{n_{L-1}}(f) = \widetilde{\mathrm{err}}(f,n_{L-1}) > \varepsilon$. For $m^* = m > 2$ it is true because of the definition of m. Since ζ is a decreasing function, it follows that

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

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

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

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

For fixed $\alpha \in (0,1]$, we only need to consider that case where $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h}) \rfloor + 1$. This implies that $n^* - 1 > \lfloor 2(b-a)/(\alpha\mathfrak{h}) \rfloor \geq 2(b-a)/(\alpha\mathfrak{h})$ thus $\alpha\mathfrak{h} \geq 2(b-a)/(n^*-1)$.

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

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

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

4.2 Simpson's rule

Similar to the trapezoidal algorithm, firstly we will provide the necessary condition for the input function lying in the cone. From (2.7), we know that $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$ 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 $n \ge 6$. From the definition of the cone (2.9), we know that the variation of the first derivative of the input function can be bounded by the approximation $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$ times an appropriate inflation factor $\mathfrak{C}(\mathrm{size}(\{x_j\}_{j=0}^{n+1}))$, 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 $n \ge 6$. 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:

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

We will use the same mechanism to make the cone function space more inclusive in the Simpson's rule case.

4.2.1 Simpson's Rule Algorithm. Now we provide 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 $\mathfrak{h} \le (b-a)/6$. Set $k=0$, $n_k=0$. Let $n_1 = \lceil (b-a)/\mathfrak{h} \rceil$. 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 $\widetilde{V}_3(f,n_k) \le \eta_k$, then go to Step 5. Otherwise, set $\mathfrak{h} = \mathfrak{h}/2$.

a) Let $\mathfrak{J} = \{j = 1, \cdots, k : n_j \ge (b-a)/\mathfrak{h}\}$. 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.

$$\eta_k \le \frac{5832\varepsilon n_k^4}{(b-a)^4}, \mathfrak{C}((b-a)/n_k)\widetilde{V}_3(f,n_k) \le \frac{5832\varepsilon n_k^4}{(b-a)^4}.$$

a) If this is 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.

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

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

Proof. From Lemma 2 and the description below it, the algorithm will be guaranteed to success. □

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,* $\text{Var}(f''')$.

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

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, $N(f,\varepsilon) \geq n_1 = (\lfloor (b-a)/\mathfrak{h} \rfloor + 1)$. Then the number of intervals increases until $\overline{\mathrm{err}}(f,n) \leq \varepsilon$. This implies the lower bound on $N(f,\varepsilon)$.

Let L be the value of l for which Algorithm 2 terminates. Since n_1 satisfies the upper bound, we may assume that $L \geq 2$. Let m be the multiplication integer found in Step 6, and let $m^* = \max(2,m)$. Note that $\zeta((m^*-1)n_{L-1}) \operatorname{Var}(f''') > \varepsilon$. For $m^* = 2$, this is true because $\zeta(n_{L-1}) \operatorname{Var}(f''') \geq \zeta(n_{L-1}) \widetilde{V}_{n_{L-1}}(f) = \widetilde{\mathrm{err}}(f, n_{L-1}) > \varepsilon$. For $m^* = m > 2$ it is true because of the definition of m. Since ζ is a decreasing function, it follows that

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

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

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

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

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

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

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, we 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, we will discuss the complexity of the problem.

CHAPTER 5

LOWER BOUND OF COMPLEXITY

In the previous Chapter, we have discussed the algorithms and their computational cost. In this Chapter, we will discuss the lower bound of complexity. 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 (ref p11). That is, to build a fooling function to show that no matter how good the algorithms are, it will 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, the algorithm is always fooled by the fooling function. 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. We will start from defining $\text{int}(\cdot, \varepsilon)$ as any algorithm that solves univariate integration problem. Then we will build two fooling functions for trapezoidal rule and Simpson's rule respectively and discuss how to find the lowest number of nodes required to solve the problem.

5.1 Trapezoidal rule

To construct the fooling function for the trapezoidal rule, we will start from creating a triangle shaped function $\text{peak}(x, t, h)$ on (a, b) that starts at point t, with the width of $2h$ and zero value elsewhere than $t, t + 2h$. Considering the B-Spline to the first order:

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

Figure 5.1. A peak function.

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

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), $t_{j+1} - t_j = h$, and $t = t_j$. We can define our peak function as

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

Picture 5.1 shows the shape of the function.

Figure 5.2. A fooling function that can fool the trapezoidal rule algorithm.

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

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

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

The peak function above is continuous to the first derivative and with finite variation of the first derivative of the function. With this peak function $\text{peak}'(x;t,h)$, we can now build the following double-peak function:

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

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

Picture 5.2 shows the shape of the double-peak function. Note that $\text{twopk}(x;t,h,\pm)$

are always in $\mathcal{C}^1$: From this definition it follows that

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

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

With the fooling function, we can find the lower bound of complexity of the trapezoidal algorithm. The key to the proof is that, one appropriate h needs to be found such that the smaller peak will reside within one interval of data sites, then cannot be detected by the algorithm. Then, the algorithm cannot distinguish between $\text{peak}'(x;t,h)$ and $\text{twopk}(x;t,h,\pm)$. Assuming that the algorithm $\text{int}(\cdot,\varepsilon)$ uses n equally spaced nodes to successfully return and answer. Moreover, assuming that $\{x_j\}_{j=1}^{m}$ is a subinterval of $\{x_j\}_{j=0}^{n}$ where the smaller peak resides. In order to find the proper h, notice that average width of intervals is $(x_{m+1}-x_0)(m+1)$ and $x_{m+1}-x_0 = b-a-3\mathfrak{h}-h$. Choose one interval such that the width is less than $2h$, the width of small bump. In order to do that, we can choose interval wider than average and $2h$ narrower than average.

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

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

Theorem 5. *Let* $\mathrm{int}(\cdot,\varepsilon)$ *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* $\mathrm{int}(\cdot,\varepsilon)$ *must use at least*

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

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 $\mathrm{int}(\cdot,\varepsilon)$ evaluates integrand $\alpha\mathrm{peak}'(\cdot;t,h)$ 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 $x_0 := a + 2\mathfrak{h}$, $x_{m+1} := b - h$ and $h := (b-a-3\mathfrak{h})/(2n+3)$. There must e at least one of these x_j with $i = 0, \cdots, m$ for which

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

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

$$\mathrm{int}(\alpha\mathrm{twopk}(\cdot;t,h,\pm),a,b,\varepsilon) = \mathrm{int}(\alpha\mathrm{peak}(\cdot;t,h),a,b,\varepsilon)$$

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

$$\varepsilon \geq \frac{1}{2}\,[\,\left|\int_a^b \alpha\text{twopk}(x;t,h,-)dx - \text{int}(\alpha\text{twopk}(\cdot;t,h,-),a,b,\varepsilon)\right| + \left|\int_a^b \alpha\text{twopk}(x;t,h,+)dx - \text{int}(\alpha\text{twopk}(\cdot;t,h,+),a,b,\varepsilon)\right|\,]$$

$$\geq \frac{1}{2}\,[\,\left|\text{int}(\alpha\text{peak}(\cdot;t,h,-),a,b,\varepsilon) - \int_a^b \alpha\text{twopk}(x;t,h,-)dx\right| + \left|\int_a^b \alpha\text{twopk}(x;t,h,+)dx - \text{int}(\alpha\text{peak}(\cdot;t,h,+),a,b,\varepsilon)\right|\,]$$

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

Substituting h in terms of n:

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

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 (**??**), both of them increase as $\mathcal{O}((\text{Var}(f')/\varepsilon))^{1/2}$ as $(\text{Var}(f')/\varepsilon)^{1/2} \to \infty$. Thus integral is optimal. □

5.2 Simpson's rule

To build the fooling functions for Simpson's rule, we will continue our B-Spline expansion in the previous section in order to find the functions the such that f, f', f'' are continuous and $\mathrm{Var}(f''') < \infty$.??? Change f

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

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

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

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

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

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), $t_{j+1} - t_j = h$, and $t = t_j$. We can define our peak function as

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

$$\text{bump}'''(x;t,h) := \begin{cases} 1, & t \le x < t+h, \\ -3, & t+h \le x < t+2h, \\ 3, & t+2h \le x < t+3h, , \\ -1, & t+3h \le x < t+4h, \\ 0, & \text{otherwise}, \end{cases} \tag{5.5b}$$

$$\text{Var}(\text{bump}'''(\cdot;t,h)) \le 16 \text{ with equality if } a < t < t+4h < b, \tag{5.5c}$$

$$\int_a^b \text{peak}(x;t,h)dx = h^4. \tag{5.5d}$$

Picture 5.3 shows the shape of the function. The bump function above is continuous to the third derivative. We can now build the following double-bump function that is always in $\mathcal{C}^3$:

$$\text{twobp}(x;t,h,\pm) := \text{bump}(x;a,\mathfrak{h}) \pm \frac{15[\mathfrak{C}(h)-1]}{16}\text{bump}(x;t,h)$$
$$a+5\mathfrak{h} \le h \le b-5h, 0 \le h < \mathfrak{h}. \tag{5.6a}$$

$$\text{Var}(\text{twobp}'''(x;t,h,\pm)) = 15 + 16\frac{15[\mathfrak{C}(h)-1]}{16} = 15\mathfrak{C}(h). \tag{5.6b}$$

Figure 5.3. A bump function.

Figure 5.4. A fooling function that can fool the Simpson's rule algorithm.

Picture 5.4 shows the shape of the double-peak function. From this definition it follows that

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

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

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

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

With the fooling function, we can find the lower bound of complexity of the Simpson's algorithm.

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*

$$-\frac{5}{4} + \frac{b - a - 5\mathfrak{h}}{8} \left[\frac{[\mathfrak{C}(0) - 1]\,\mathrm{Var}(f''')}{\varepsilon} \right]^{1/4} \tag{5.7}$$

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 $\alpha\mathrm{bump}'''(\cdot; t, h)$ 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 $x_0 := a + 3\mathfrak{h}$, $x_{m+1} := b - h$ (why h but not $\mathfrak{h}$ or $5h$?) and $h := (b - a - 5\mathfrak{h})/(4n + 5)$. There must e at least one of these x_j with $i = 0, \cdots, m$ for which

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

Choose one such x_j and call it t. The choice of t and h ensures that $\texttt{int}(\cdot, a, b, \varepsilon)$ cannot distinguish between $\alpha\text{bump}(\cdot; t, h)$ and $\alpha\text{twobp}(\cdot; t, h, \pm)$. Thus

$$\texttt{int}(\alpha\text{twobp}(\cdot; t, h, \pm), a, b, \varepsilon) = \texttt{int}(\alpha\text{bump}(\cdot; t, h), a, b, \varepsilon)$$

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

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

Substituting h in terms of n:

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

Since α is an arbitrary positive number, the value of $\operatorname{Var}(\alpha\texttt{bump}'''(\cdot; a, \mathfrak{h}))$ 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 (**??**), both of them increase as $\mathcal{O}((\operatorname{Var}(f''')/\varepsilon))^{1/4}$ as $(\operatorname{Var}(f''')/\varepsilon)^{1/4} \to \infty$. Thus `integral` is optimal. □

CHAPTER 6

NUMERICAL EXPERIMENTS

We use the fooling function (5.6a) to build our test function. Consider the family of bump test functions on interval $(0, 1)$ defined by

$$f(x, t, h) = a\text{twobp}(x; t, h, +) \tag{6.1}$$

with $t \sim \mathcal{U}[0.5, 1]$, $h \sim \mathcal{U}[0, 1]$, and $a = 1/(h^4)$ chosen to make $\int_0^1 f(x)\,\mathrm{d}x = 1$.

			Success	Success	Failure
	τ	Prob($f \in \mathcal{C}_\tau$)	No Warning	Warning	No Warning
	10	$0\% \to 25\%$	25%	$< 1\%$	75%
Algorithm **??**	100	$23\% \to 58\%$	56%	2%	42%
	1000	$57\% \to 88\%$	68%	20%	12%
`quad`			8%		92%
`integral`			19%		81%
`chebfun`			29%		71%

Table 6.1. The probability of the test function lying in the cone for the original and eventual values of τ and the empirical success rate of Algorithm **??** plus the success rates of other common quadrature algorithms.

Some commonly available numerical algorithms in MATLAB are `quad` and `integral` [**?**] and the MATLAB Chebfun toolbox [**?**]. We applied these three routines to the random family of test functions. Their success and failure rates are also recorded in Table 6.1.

6.1 Simpson's Rule I need to use a good example to run the tests. Then I will need to compare the results for both algorithms with other algorithms. After that, I

need to compare trap and sim to get the conclusion such as sim has faster convergence rate than trap. This will require an sample function good to both trap and sim.

CHAPTER 7

CONCLUSION

A

7.1 Summary

A

APPENDIX A

TABLE OF TRANSITION COEFFICIENTS FOR THE DESIGN OF LINEAR-PHASE FIR FILTERS

Your Appendix will go here !

APPENDIX B

NAME OF YOUR SECOND APPENDIX

Your second appendix text....

APPENDIX C

NAME OF YOUR THIRD APPENDIX

Your third appendix text....

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.