TITLE

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
4th October, 2018

ACKNOWLEDGMENT

Will be added once thesis is finished

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

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 be focusing 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 automatic algorithms that adaptively determine the computational cost required to obtain a result of the integral. The required inputs are both an error tolerance, ε, and the integrand f. The output will be guaranteed to not differ from the true answer by more than ε.

Unfortunately, most commonly used guaranteed algorithms are not adaptive. They cannot adjust their effort based on the property of the input function. On the other hand, msot existing adaptive algorithms are not guaranteed to provide answers satisfying the error tolerance. Out goal here is to construct adaptive, automatic algorithms that are guaranteed to satisfy the error tolerance.

1.1 Fixed-Cost Algorithms on Balls

1. Non-adaptive

1.2 Adaptive Algorithms with No Guarantees

2. adaptive no guarantees

1.3 Motivation and Outline

The goal is to construct adaptive, automatic algorithms with guarantees of success. In Chapter 2, I will introduce the definitions and assumptions used in the thesis. Chapter 3 will be focusing on finding the upper bound of the approximation errors. The guaranteed algorithms based on trapezoidal rule and Simpson's rule will be presented in Chapter 4 with rigorous theoretical proof of success/successfulness. Chapter 5 will study the lower and upper bound of computational cost. Chapter 6 will study the lower bound of complexity. In Chapter 7, the results from numerical experiences will be discussed.

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

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

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

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

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. The traditional non-adaptive trapezoidal rule and Simpson's rule have upper bounds on approximation error in terms of the total variation of a particular derivative of integrand:

$$\begin{aligned}\mathrm{err}(f,n) \le \overline{\mathrm{err}}(f,n) :=& C(n)\,\mathrm{Var}(f^{(p)}), \\ C(n) =& \begin{cases} \frac{(b-a)^2}{8n^2}, p=1, & \text{trapezoidal rule[1, (7.15)]}, \\ \frac{(b-a)^4}{5832n^4}, p=3, & \text{Simpson's rule.}\end{cases}\end{aligned} \tag{2.5}$$

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 $a = x_0 \leq x_1 < \cdots < x_n \leq x_{n+1} = b$. 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:

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

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:

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

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

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

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

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

Since (2.6) is true for all partition $\{x_i\}_{i=0}^{n+1}$, it is true for (3.1) with this particular partition $\{x_j\}_{i=0}^{n+1}$. 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:

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

Therefore by combining the deduction above together, 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

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

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

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

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:

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

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

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

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. In order to decide whether one input function 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 $\operatorname{Var}(f')$, i.e. $\widehat{V}(f', \{x_j\}_{j=0}^{n+1}) \le \operatorname{Var}(f')$. 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 $\mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))$, i.e. $\operatorname{Var}(f') \le \mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$. The size of the partition is unknown, due to the definition of the cut-off value in (2.9) and the fact that $\mathfrak{C}(\cdot)$ is non-decreasing, we can have $\mathfrak{C}(\mathfrak{h}) \le \mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))$. As the number of nodes n increases, $\widehat{V}(f', \{x_j\}_{j=0}^{n+1})$ increases. But the inflation factor $\mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))$ decreases since it is a non-decreasing function and the size is smaller. Therefore, the largest lower bound on $\operatorname{Var}(f')$ 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:

$$\widehat{V}(f', \{x_k\}_{k=0}^{n+1}) \leq \mathfrak{C}(\mathfrak{h})\widehat{V}(f', \{x_j\}_{j=0}^{n+1}), \qquad \text{for all } j \leq k. \tag{4.1}$$

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. With 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 $\mathfrak{h} \leq (b-a)$. Set $n_0 = 1$, $k = 0$, $\eta_0 = \infty$. Let $n_1 = \lceil 2(b-a)/\mathfrak{h}\rceil$. 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 $n_{k+1} = n_k \times \lceil 2(b-a)/(\mathfrak{h}n_k)\rceil$.

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 $\widetilde{V}_1(f, n_k) \leq \eta_k$, then go to Step 5. Otherwise, set $\mathfrak{h} = \mathfrak{h}/2$.

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

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

$$\begin{aligned} &\text{For } j' = \min\mathfrak{J}, \text{ let } \eta_{j'} = \mathfrak{C}(2(b-a)/n_{j'})\widetilde{V}_1(f, n_{j'}), \\ &\text{Compute } \eta_j = \min\{\eta_{j-1}, \mathfrak{C}(2(b-a)/n_j)\widetilde{V}_1(f, n_j)\}, \text{ for } j = j'+1, \cdots, k. \end{aligned}$$

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.

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

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.,*

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

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

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

The number of function values required by the algorithm is $N(f,\varepsilon)+1$.

Give η another name and definition.

Proof. No matter what inputs f and ε are provided, $N(f,\varepsilon) \ge n_1 = (\lfloor (b-a)/\mathfrak{h}\rfloor + 1)$. Then the number of intervals increases until $\overline{\mathrm{err}}(f,n) \le \varepsilon$. 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 integer found in Step 3, and let $m^* = \max(2, m)$. Note that $\eta((m^*-1)n_{L-1}) \operatorname{Var}(f') > \varepsilon$. For $m^* = 2$, this is true because $\eta(n_{L-1}) \operatorname{Var}(f') \geq \eta(n_{L-1}) \widetilde{V}_{n_{L-1}}(f) = \widetilde{\operatorname{err}}(f, n_{L-1}) > \varepsilon$. For $m^* = m > 2$ it is true because of the definition of m. Since η is a decreasing function, it follows that

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

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

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

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^*, η, and $\mathfrak{C}$ is non-decreasing:

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

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 $\operatorname{Var}(f''')$, i.e. $\widehat{V}(f''', \{x_j\}_{j=0}^{n+1}) \leq \operatorname{Var}(f''')$. 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 $\mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))$, i.e. $\operatorname{Var}(f''') \leq \mathfrak{C}(\operatorname{size}(\{x_j\}_{j=0}^{n+1}))\widehat{V}(f''', \{x_j\}_{j=0}^{n+1})$.. 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:

$$\widehat{V}(f''', \{x_k\}_{k=0}^{n+1}) \leq \mathfrak{C}(\mathfrak{h})\widehat{V}(f''', \{x_j\}_{j=0}^{n+1}), \quad \text{for all } j \leq k. \tag{4.3}$$

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 $n_0 = 1$, $k = 0$, $\eta_0 = \infty$. 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 $\mathrm{Var}(f''')$, n_j, for $j \in \mathfrak{J}$ 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.

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

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

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

The number of function values required by the algorithm is $6N(f,\varepsilon)+1$.

Give η another name and definition.

Proof. No matter what inputs f and ε are provided, $N(f,\varepsilon) \ge n_1 = (\lfloor(b-a)/\mathfrak{h}\rfloor+1)$. Then the number of intervals increases until $\overline{\text{err}}(f,n) \le \varepsilon$. 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 $L \ge 2$. Let m be the integer found in Step 3, and let $m^* = \max(2,m)$. Note that $\eta((m^*-1)n_{L-1})\,\text{Var}(f''') > \varepsilon$. For $m^*=2$, this is true because $\eta(n_{L-1})\,\text{Var}(f''') \ge \eta(n_{L-1})\widetilde{V}_{n_{L-1}}(f) = \widetilde{\text{err}}(f,n_{L-1}) > \varepsilon$. For $m^* = m > 2$ it is true because of the definition of m. Since η is a decreasing function, it follows that

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

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

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^*, η, and $\mathfrak{C}$ is non-decreasing:

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

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 how well performed our algorithm is. 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 method of proving the lower bound of complexity is by building 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. 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 and discuss how to find the lowest number of nodes required to solve the problem.

5.1 Trapezoidal rule

Firstly, we construct the fooling function. We choose the triangle shaped function $f_0 : x \mapsto 1/2 - |1/2 - x|$. Then

$$\text{Var}(f_0') = 2$$

For any $n \in \mathbb{N}_0$, suppose that the one has the data $L_i(f) = f(\xi_i)$, $i = 1, \dots, n$ for arbitrary ξ_i, where $0 = \xi_0 \le \xi_1 < \cdots < \xi_n \le \xi_{n+1} = 1$. There must be some $j = 0, \dots, n$ such that $\xi_{j+1} - \xi_j \ge 1/(n+1)$. The function f_1 is defined as a triangle function on the interval $[\xi_j, \xi_{j+1}]$:

$$f_1(x) := \begin{cases} \dfrac{\xi_{j+1} - \xi_j - |\xi_{j+1} + \xi_j - 2x|}{8} & \xi_j \le x \le \xi_{j+1}, \\ 0 & \text{otherwise.} \end{cases}$$

This is a piecewise linear function whose derivative changes from 0 to 1/4 to $-1/4$ to 0 provided $0 < \xi_j < \xi_{j+1} < 1$, and so $\text{Var}(f_1') \le 1$. Moreover,

$$\text{INT}(f) = \int_0^1 f_1(x)\,\mathrm{d}x = \frac{(\xi_{j+1} - \xi_j)^2}{16} \ge \frac{1}{16(n+1)^2}$$

With the fooling function, we can find the lower bound of complexity of the trapezoidal algorithm.

Theorem 5. *The complexity of the integration problem over the cone of functions $\mathcal{C}^1$ defined in* (2.9) *is bounded below as*

$$\left\lceil \sqrt{\frac{((b-a)/\mathfrak{h}) - 2)\,\text{Var}(f')}{32\mathfrak{h}\varepsilon}} \right\rceil - 1. \tag{5.1}$$

As $\operatorname{Var}(f')/\varepsilon \to \infty$ *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. need to change □

5.2 Simpson's rule

Firstly we need to build fooling function such that it can make the algorithm fail. This function should be continuous to the third derivative and lie in the cone. We start by considering the following bump function:

$$\operatorname{bump}(x;t,h) := \begin{cases} (x-t)^3/6, & t \le x < t+h, \\ [-3(x-t)^3 + 12h(x-t)^2 - 12h^2(x-t) + 4h^3]/6, & t+h \le x < t+2h, \\ [3(x-t)^3 - 24h(x-t)^2 + 60h^2(x-t) - 44h^3]/6, & t+2h \le x < t+3h, \\ (t+4h-x)^3/6, & t+3h \le x < t+4h, \\ 0, & \text{otherwise,} \end{cases} \tag{5.2a}$$

$$\operatorname{bump}'''(x;t,h) := \begin{cases} 1, & t \le x < t+h, \\ -3, & t+h \le x < t+2h, \\ 3, & t+2h \le x < t+3h, , \\ -1, & t+3h \le x < t+4h, \\ 0, & \text{otherwise,} \end{cases} \tag{5.2b}$$

$$\operatorname{Var}(\operatorname{bump}'''(\cdot;t,h)) \le 16 \text{ with equality if } a < t < t+4h < b, \tag{5.2c}$$

$$\int_a^b \operatorname{peak}(x;t,h)dx = h^4. \tag{5.2d}$$

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

$$\operatorname{twobp}(x;t,h,\pm) := \operatorname{bump}(x;a,\mathfrak{h}) \pm \frac{15[\mathfrak{C}(h)-1]}{16}\operatorname{bump}(x;t,h)$$
$$a + 5\mathfrak{h} \le h \le b - 5h, 0 \le h < \mathfrak{h}. \tag{5.3a}$$

$$\operatorname{Var}(\operatorname{twobp}'''(x;t,h,\pm)) = 15 + 16\frac{15[\mathfrak{C}(h)-1]}{16} = 15\mathfrak{C}(h). \tag{5.3b}$$

From this definition it follows that

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

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

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* $\text{Var}(f''')$*, there will be some $f \in \mathcal{C}$ for which int must use at least*

$$-\frac{5}{4} + \frac{b-a-5\mathfrak{h}}{8}\left[\frac{[\mathfrak{C}(0)-1]\,\text{Var}(f''')}{\varepsilon}\right]^{1/4} \tag{5.4}$$

function values. As $\text{Var}(f''')/\varepsilon \to \infty$ *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 $\alpha\text{bump}'''(\cdot;t,h)$ 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 $\texttt{int}$ is successful for all of the functions.

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

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

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

Substituting h in terms of n:

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

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

CHAPTER 6

NUMERICAL EXPERIMENTS

6.1 Traezoidale rule

Consider the family of bump test functions defined by

$$f(x) = \begin{cases} b[4a^2 + (x-z)^2 + (x-z-a)|x-z-a| \\ \qquad -(x-z+a)|x-z+a|], & z-2a \le x \le z+2a, \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

with $\log_{10}(a) \sim \mathcal{U}[-4,-1]$, $z \sim \mathcal{U}[2a, 1-2a]$, and $b = 1/(4a^3)$ chosen to make $\int_0^1 f(x)\,\mathrm{d}x = 1$. It follows that $\|f' - f(1) + f(0)\|_1 = 1/a$ and $\mathrm{Var}(f') = 2/a^2$. The probability that $f \in \mathcal{C}_\tau$ is $\min\left(1, \max(0, (\log_{10}(\tau/2) - 1)/3)\right)$.

As an experiment, we chose 10000 random test functions and applied Algorithm **??** with an error tolerance of $\varepsilon = 10^{-8}$ and initial τ values of $10, 100, 1000$. The algorithm is considered successful for a particular f if the exact and approximate integrals agree to within ε. The success and failure rates are given in Table 6.1. Our algorithm imposes a cost budget of $N_{\max} = 10^7$. If the proposed n_{i+1} in Stages 2 or 3 exceeds $N_{\max}$, then our algorithm returns a warning and falls back to the largest possible n_{i+1} not exceeding $N_{\max}$ for which $n_{i+1} - 1$ is a multiple of $n_i - 1$. The probability that f initially lies in $\mathcal{C}_\tau$ is the smaller number in the third column of Table 6.1, while the larger number is the empirical probability that f eventually lies in $\mathcal{C}_\tau$ after possible increases in τ made by Stage 2 of Algorithm **??**. For this experiment Algorithm **??** was successful for all f that finally lie inside $\mathcal{C}_\tau$ and for which no attempt was made to exceed the cost budget.

			Success	Success	Failure
	τ	$\mathrm{Prob}(f \in \mathcal{C}_\tau)$	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. They do not give warnings of possible failure.

6.2 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.