

# Reliable Adaptive, Automatic Quadrature for Cones of Integrands

Fred J. Hickernell, Martha Razo, and Sunny Yun

September 2, 2013

## Abstract

Hi

## 1 Introduction

Although some mathematical problems can be solved using pencil and paper, many others can only be solved via numerical algorithms. These algorithms typically generate a sequence of approximations that converge to the true solution. The user wants to know at what point in the sequence the approximation differs from the true solution by no more than a specified tolerance. Existing stopping criteria are based either on inconvenient assumptions or on heuristics without guarantees. Here we describe a new stopping criterion for cones of integrands. This criterion leads to an adaptive trapezoidal algorithm for integration that is guaranteed to provide the correct answer. We also suggest changing the way numerical quadrature is taught and propose a new paradigm for constructing numerical algorithms.

### 1.1 Why Numerical Integration Is Needed?

There are various analytic methods for evaluating integrals, e.g., substitution, integration by parts, and partial fractions. Unfortunately, many elementary functions have integrals that *cannot* be expressed as elementary functions. One example is the standard normal probability density,  $\phi : x \mapsto e^{-x^2/2}/\sqrt{2\pi}$ , whose integral is the standard normal distribution,  $\Phi : x \mapsto \int_{-\infty}^x \phi(t) dt$ . Since the  $\Phi$  is not an elementary function, values of  $\Phi$  must be provided via tables in statistics textbooks and special functions in calculators. By contrast, the derivatives of elementary functions are always elementary functions.

### 1.2 The Basic Trapezoidal Rule

To compute integrals that are not amenable to analytic methods one must turn to numerical integration (quadrature) methods such as the trapezoidal rule,  $T_n$ ,

which is often introduced in calculus courses:

$$\int_0^1 f(x) dx \approx T_n(f) := \frac{1}{2n} [f(0) + 2f(1/n) + \cdots + 2f(1 - 1/n) + f(1)]. \quad (1)$$

The trapezoidal rule is the integral of the piecewise linear spline approximation to the integrand. The name of the rule comes from the fact that the integral of the linear spline is a sum of areas of trapezoids (see Figure 1). Under mild smoothness conditions on  $f$  one observes that  $T_n(f)$  approaches  $\int_0^1 f(x) dx$  as  $n$  increases.

For the user's convenience one would like to turn  $T_n$  into an *automatic* quadrature algorithm — one that automatically chooses  $n$  to ensure that the trapezoidal rule error is small enough. Given the user's tolerance for error,  $\varepsilon$ , an automatic trapezoidal algorithm should choose  $n$  such that

$$\text{err}(f, n) := \left| \int_0^1 f(x) dx - T_n(f) \right| \leq \varepsilon. \quad (2)$$

Although  $\text{err}(f, n)$  is rarely known exactly, there exist rigorous upper bounds on the error that are proportional to  $1/n^2$  and the roughness of the integrand, e.g., see Brass and Petras (2011, Sect. 7.2, (7.15)):

$$\text{err}(f, n) \leq \overline{\text{err}}(f, n) := \frac{\text{Var}(f')}{8n^2}, \quad n \in \mathbb{N} := \{1, 2, \dots\}. \quad (3)$$

Here  $\text{Var}(\cdot)$  represents the (total) variation of a function, which is defined as

$$\begin{aligned} &\text{Var}(f) \\ &:= \sup \left\{ \sum_{i=1}^n |f(x_i) - f(x_{i-1})| : n \in \mathbb{N}, 0 \leq x_0 < x_1 < \cdots < x_n \leq 1 \right\}. \end{aligned} \quad (4)$$

Intuitively,  $\text{Var}(f)$  is the total vertical distance up and down that one travels on a roller coaster whose track is the graph of  $f$ . The function  $f \mapsto \text{Var}(f')$  is a semi-norm. The trapezoidal rule gives the exact answer for linear integrands. Error bound (3) reflects this fact since if  $f(x) = ax + b$ , then  $f'(x) = a$ , and so  $\text{Var}(f') = 0$ .

To illustrate this error bound, consider the normal probability example mentioned above. The trapezoidal rule approximation using  $n = 4$  trapezoids, the actual error, and the error bound are as follows:

$$f(x) = 2\phi(2x) = \sqrt{\frac{2}{\pi}} e^{-2x^2}, \quad \int_0^1 f(x) dx = 0.4772, \quad T_4(f) = 0.4750, \quad (5a)$$

$$\text{Var}(f') = 1.5038, \quad \text{err}(f, 4) = 0.0022 \leq 0.0117 = \overline{\text{err}}(f, 4). \quad (5b)$$

(The value of the integral to four significant digits may be found by a variety of quadrature algorithms.) Figure 1 depicts the integrand and the trapezoidal rule

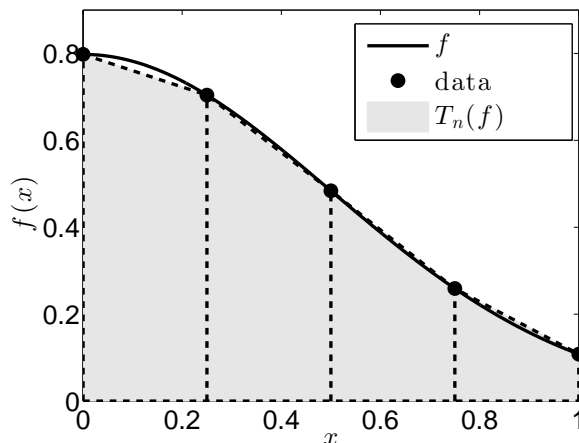


Figure 1: The integrand in (5) and the trapezoidal rule approximation for the integral.

approximation. As expected, the actual error is bounded above by the error bound in (3).

Error bound (3) can help us determine how large  $n$  must be to meet the error tolerance if an upper bound on  $\text{Var}(f')$  is available. Section 2 provides the details of an automatic algorithm for such cases. Unfortunately, in practice nothing may be known a priori about  $\text{Var}(f')$ . Adaptive algorithms do not require an upper bound on  $\text{Var}(f')$  but estimate the error based on the function values sampled and then apply a stopping criterion to determine how many function values are required. Section 3 describes the stopping criterion commonly taught in numerical analysis courses, but which has significant flaws, as pointed out by James Lyness (1983). An improved stopping criterion for integrands lying in cones is described in Section 4. Much of this section is taken from Clancy et al. (2013), which presents a general framework for guaranteed automatic algorithms and also describes the new adaptive trapezoidal algorithm presented here. In this article we discuss in greater detail the new adaptive trapezoidal algorithm. In Section ?? we discuss the ramifications of these developments for how we teach numerical integration in calculus courses and later on. We also discuss the how we should go about developing new numerical algorithms.

## 2 An Automatic Algorithm for Balls of Integrands

If  $f$  has a simple form, like the example in (5), then one may be able to derive an upper bound on  $\text{Var}(f')$ . In this case the following automatic algorithm based on error bound (3) may be used.

**Algorithm 1** (Automatic, for Balls). Given a  $\sigma > 0$  and an error tolerance,  $\varepsilon$ , set

$$n = \left\lceil \sqrt{\sigma/(8\varepsilon)} \right\rceil, \quad (6)$$

and return the trapezoidal rule approximation  $T_n(f)$  as the answer.

**Theorem 1.** *If  $\text{Var}(f') \leq \sigma$ , then Algorithm 1 is successful, i.e.,*

$$\left| \int_0^1 f(x) dx - T_n(f) \right| \leq \varepsilon.$$

Algorithm 1 is guaranteed to work for all integrands lying in the ball  $\mathcal{B}_\sigma = \{f : \text{Var}(f') \leq \sigma\}$ . This algorithm is automatic because it expends as much effort as required by the error tolerance,  $\varepsilon$  and the radius of the ball,  $\sigma$ , to obtain the desired answer. It is not an adaptive algorithm because the computational cost does not depend on the particulars of  $f$ .

For  $f$  in (5), one may choose  $\sigma = \text{Var}(f') = 1.5038$ , and then Algorithm 1 uses  $n = \lceil 0.4336/\sqrt{\varepsilon} \rceil$  trapezoids. However, not all integrands have first derivatives whose variations are so easy to bound.

### 3 An Adaptive Trapezoidal Algorithm with a Flawed Stopping Criterion

Adaptive quadrature algorithms do not require the user to specify  $\sigma$ . Instead, they bound or estimate the error using only function values and then determine the sample size accordingly. MATLAB (2013), Mathematica (2012), NAG (2012), and R (2012) all contain automatic quadrature algorithms. Unfortunately, their stopping criteria are unreliable, as pointed out by James Lyness (1983). To illustrate the flawed logic that leads to unreliable error estimation in these sophisticated algorithms, we focus on the simpler case of an adaptive trapezoidal algorithm.

#### 3.1 A Popular Error Estimate

Texts such as Burden and Faires (2010, p. 223–224), Cheney and Kincaid (2013, p. 233), and Sauer (2012, p. 270), advise readers to estimate the error of  $T_n(f)$  by comparing it to  $T_{n/2}(f)$ , specifically,

$$\widehat{\text{err}}(f, n) = \frac{|T_n(f) - T_{n/2}(f)|}{3}, \quad \frac{n}{2} \in \mathbb{N}. \quad (7)$$

This error estimate leads to the following adaptive, automatic quadrature algorithm. For each iteration the number of trapezoids is doubled so that the data from the previous iteration can be reused.

**Algorithm 2.** Given an error tolerance,  $\varepsilon$ , let  $j = 1$  and  $n_1 = 2$ .

**Step 1** Compute the error estimate  $\widehat{\text{err}}(f, n_j)$  according to (7).

**Step 2** If  $\widehat{\text{err}}(f, n_j) \leq \varepsilon$ , then return the trapezoidal rule approximation  $T_{n_j}(f)$  as the answer.

**Step 3** Otherwise let  $n_{j+1} = 2n_j$ , increase  $j$  by one, and go to Step 1.

Error estimate (7) may be justified by noting that Simpson's rule may be written as

$$\begin{aligned} S_n(f) &:= \frac{4T_n(f) - T_{n/2}(f)}{3} = T_n(f) + \frac{T_n(f) - T_{n/2}(f)}{3} = \\ &= \frac{1}{3n} [f(0) + 4f(1/n) + 2f(2/n) + 4f(3/n) + \cdots + 4f(1 - 1/n) + f(1)]. \end{aligned}$$

The error bound for Simpson's rule then serves as an bound on the error of  $\widehat{\text{err}}(f, n)$  (Brass and Petras, 2011, Sect. 7.3, p. 231):

$$\begin{aligned} |\text{err}(f, n) - \widehat{\text{err}}(f, n)| &\leq \left| \int_0^1 f(x) dx - T_n(f) - \frac{T_n(f) - T_{n/2}(f)}{3} \right| \quad (8) \\ &= \left| \int_0^1 f(x) dx - S_n(f) \right| \leq \frac{\text{Var}(f''')}{36n^4}, \quad \frac{n}{2} \in \mathbb{N}. \end{aligned}$$

Comparing the orders of convergence in (3) and (8), it follows that  $\widehat{\text{err}}(f, n)$  does an excellent job in approximating  $\text{err}(f, n)$  for  $n$  large enough, assuming finite  $\text{Var}(f''')$ .

Unfortunately, without knowing an upper bound on  $\text{Var}(f''')$ , one cannot know how large  $n$  must be for  $\widehat{\text{err}}(f, n)$  to be reliable. If one really does have an upper bound on  $\text{Var}(f''')$ , then it makes more sense to apply Simpson's rule and obtain a higher order convergence rate than that of Algorithm 2. Since there is no really rigorous and compelling argument to use Algorithm 2, we do not formulate any theorem guaranteeing it.

One may identify integrands,  $f$ , that fool the error estimate completely, e.g.,

$$\int_0^1 f(x) dx = 1, \quad T_n(f) = T_{n/2}(f) = -1, \quad \text{err}(f, n) = 2 \neq 0 = \widehat{\text{err}}(f, n). \quad (9)$$

Equation (8) implies that  $\text{Var}(f''') \geq 36n^4 |\text{err}(f, n) - \widehat{\text{err}}(f, n)|$ . So, unless  $n$  is small, any integrand  $f$  satisfying (9) must have large  $\text{Var}(f''')$ . Figure 2 shows two such integrands for  $n = 16$ . These two integrands illustrate two different ways in which the error estimate  $\widehat{\text{err}}(\cdot)$  fails, and hence, Algorithm 2 also fails. One way is ultimately unavoidable but should be quantifiable. The other way is inexcusable, but is prevalent in widely used automatic numerical quadrature.

The failure of Algorithm 2 cannot be solved by using a more conservative stopping criterion of the form  $C\widehat{\text{err}}(f, n_j) \leq \varepsilon$  in Step 2 where  $C > 1$ . Integrands satisfying (9) have an estimated error of  $C\widehat{\text{err}}(f, n) = 0$  regardless of how large  $C$  is.

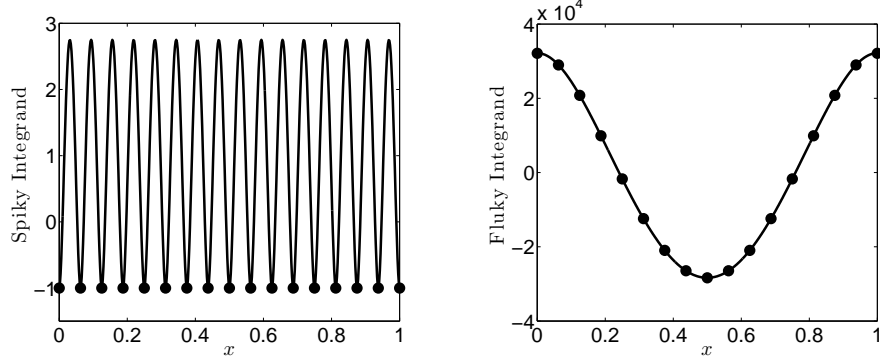


Figure 2: Examples of a spiky integrand (left) and a fluky integrand (right), which satisfy the failure conditions (9) for  $n = 16$ .

### 3.2 Spiky Integrands

Figure 2 (left) depicts the following spiky integrand, which satisfies (9):

$$f_{\text{spiky}}(x; n) = -1 + 60[\{nx\}(1 - \{nx\})]^2, \quad \frac{n}{2} \in \mathbb{N}, \quad \{x\} := x \bmod 1. \quad (10)$$

The spiky integrand is constructed to be constant at the data points with spikes in between. It has  $n + 1$  local minima of  $-1$ , and  $n$  local maxima of  $11/4$ . The variation of the first and third derivatives of  $f_{\text{spiky}}(\cdot; n)$  increase with  $n$  as follows:

$$\text{Var}(f'_{\text{spiky}}(\cdot; n)) = \frac{80n^2}{\sqrt{3}}, \quad \text{Var}(f'''_{\text{spiky}}(\cdot; n)) = 1440n^4.$$

The problem of spiky integrands is not unique to Algorithm 2. *Any* quadrature algorithm that depends only on function values to estimate error will be fooled by sufficiently spiky integrands. Suppose that the quadrature algorithm samples the integrand at a sequence of points,  $\xi_1, \xi_2, \dots$ , and suppose that  $f(\xi_i) = -1$  for all  $\xi_i$ . Here the algorithm may even be adaptive, meaning that  $\xi_{i+1}$  may depend on  $(\xi_1, f(\xi_1)), \dots, (\xi_i, f(\xi_i))$ . Eventually the algorithm stops. If the algorithm is to be accurate for the constant integrand,  $x \mapsto -1$ , then the estimate of the integral to be  $-1$  (or very close). However, one can construct a non-constant function that shares the same data —  $f(\xi_i) = -1$  for all  $\xi_i$  — while having any value of the integral one wants. Such functions completely fool the algorithm.

Although Algorithm 1 for balls of integrands can be fooled by spiky integrands, Theorem 1 remains valid because those spiky integrands lie outside the ball  $\mathcal{B}_\sigma$ . By increasing  $\sigma$  one may accommodate spikier integrands. Algorithm 3 in Section 4 below can also be fooled by spiky integrands, however, its guarantee in Theorem 2 remains valid because those spiky integrands lie outside the cone,  $\mathcal{C}_\tau$ . By increasing  $\tau$  one may accommodate spikier integrands. Moreover, the cone parameter itself corresponds to an inverse length scale corresponding to the amount of spikiness that one wishes to tolerate.

### 3.3 Fluky Integrands

Thirty years ago James Lyness wrote a SIAM Review article, *When Not to Use an Automatic Quadrature Routine*. While recognizing the problem of spiky integrands, he claims that there is even a more serious problem with automatic quadrature methods (Lyness, 1983, p. 69):

While prepared to take the risk of being misled by chance alignment of zeros in the integrand function, or by narrow peaks which are “missed,” the user may wish to be reassured that for “reasonable” integrand functions which do not have these characteristics all will be well. It is the purpose of the rest of this section to demonstrate by example that he cannot be reassured on this point. In fact the routine is likely to be unreliable in a significant proportion of the problems it faces (say 1 to 5%) and there is no way of predicting in a straightforward way in which of any set of apparently reasonable problems this will happen.

Lyness went on to describe how to construct a “reasonable” integrand that fools an automatic quadrature algorithm. We would call this a *fluky* integrand.

Figure 2 (right) plots a fluky integrand for Algorithm 2:

$$f_{\text{fluky}}(x; n) = \frac{2 - 5n^2 + n^4}{2} + 15n^2x(1-x)\{1 - n^2x(1-x)\}, \quad n \in \mathbb{N}. \quad (11)$$

The integrand  $f_{\text{fluky}}(\cdot; n)$  satisfies (9) for even  $n$ . In contrast to the spiky integrand, this fluky integrand has only one local minimum of size  $\mathcal{O}(n^4)$  located at  $1/2$  and two local maxima of size  $\mathcal{O}(n^4)$  located at  $\mathcal{O}(n^{-2})$  and  $1 - \mathcal{O}(n^{-2})$ . The variation of the first and third derivatives of  $f_{\text{fluky}}(\cdot; n)$  increase with  $n$ , specifically,

$$\text{Var}(f'_{\text{fluky}}(\cdot; n)) = \frac{10n}{3} \left[ 9n + 2\sqrt{3(-2 + n^2)^3} \right], \quad \text{Var}(f'''_{\text{fluky}}(\cdot; n)) = 360n^4.$$

Disregarding its large magnitude  $f_{\text{fluky}}$  appears to quite “reasonable”. We describe integrands like this one as fluky because their construction is rather delicate, and they satisfy conditions (9) with a small number of local optima. In this case  $f_{\text{fluky}}$  was chosen to be a fourth degree polynomial — to ensure a small number of local optima — with symmetry about  $1/2$ . The remaining coefficients were then determined uniquely by the constraints in (9).

Algorithm 2 is not the only adaptive quadrature algorithm that fails to integrate fluky functions. Figure ???

The failure of the error estimate in (7) for  $f_{\text{fluky}}$  seems avoidable. Although the values of the trapezoidal rule are moderate, the sampled function values,  $(f_{\text{fluky}}(i/n; n))_{i=1}^n$  vary widely, which implies that  $\text{Var}(f_{\text{fluky}})$  must be large. In the next section we transform this intuition into a better method for bounding the error of the trapezoidal rule.

## 4 A Guaranteed, Adaptive, Automatic Trapezoidal Algorithm

Algorithm 1 does not use any values of  $f$  to determine how many trapezoids are needed to approximate the integral of  $f$  because an upper bound on  $\text{Var}(f')$  is assumed. Algorithm 2 uses values of  $f$  to determine how many trapezoids are needed, but the way Algorithm 2 uses these values does not always allow it to detect an occurrence of large  $\text{Var}(f')$ . In this section we present a way for reliably estimating  $f'$ , which then can be used to construct a reliable stopping criterion for an adaptive trapezoidal algorithm.

Estimating  $\text{Var}(f')$  reliably and directly requires additional smoothness assumptions on  $f$ , but if we assume more smoothness, then we ought to use a different rule, like Simpson's rule. On the other hand, knowing that  $\text{Var}(f')$  is finite does allow us to reliably estimate, a weaker semi-norm of  $f$ , such as  $\text{Var}(f)$ . As observed in Section 3.3 above, the fluky algorithms that fool Algorithm 2 not only have large  $\text{Var}(f')$  but also large  $\text{Var}(f)$ . The approach taken in this section is to explore what can be accomplished by estimating a weaker semi-norm.

Let  $A_n$  denote the linear spline approximation using  $n + 1$  equally spaced points, i.e.,

$$A_n(f)(x) = \begin{cases} f(0)(1 - nx) + f(1/n)nx, & 0 \leq x < 1/n, \\ f(1/n)(2 - nx) + f(2/n)(nx - 1), & 1/n \leq x < 2/n, \\ \vdots & \vdots \\ f(1 - 1/n)(n - nx) + f(1)(nx - n + 1), & 1 - 1/n \leq x \leq 1. \end{cases} \quad (12)$$

The trapezoidal rule can be written as  $T_n(f) = \int_0^1 A_n(f)(x) dx$ . Using this definition of linear spline we focus on the semi-norm defined by  $f \mapsto \text{Var}(f - A_1(f))$ . Like  $f \mapsto \text{Var}(f')$  this weaker semi-norm also vanishes for functions that are linear in  $x$ . If  $\text{Var}(f')$  is finite, then it follows that  $\text{Var}(f) = \|f'\|_1 := \int_0^1 |f'(x)| dx$ , so the weaker semi-norm may be written as  $f \mapsto \|f' - f(1) + f(0)\|_1$ . It is shown in Clancy et al. (2013, Section 5.1) that

$$2 \|f' - f(1) + f(0)\|_1 \leq \text{Var}(f') \quad \forall f \text{ with } \text{Var}(f') < \infty. \quad (13)$$

The data-driven quantity

$$\|A_n(f)' - f(1) + f(0)\|_1 = \sum_{i=1}^n \left| f\left(\frac{i}{n}\right) - f\left(\frac{i-1}{n}\right) - \frac{f(1) - f(0)}{n} \right| \quad (14)$$

approximates  $\|f' - f(1) + f(0)\|_1$  well provided that  $\text{Var}(f')$  is not too large. Again in Clancy et al. (2013, Section 5.1) it is shown that

$$0 \leq \|f' - f(1) + f(0)\|_1 - \|A_n(f)' - f(1) + f(0)\|_1 \leq \frac{\text{Var}(f')}{2n}. \quad (15)$$



At this point it seems that we have reached a dead end because we have an error bound in terms of  $\text{Var}(f')$ , just like the trapezoidal rule error bound (3). However, we make a further assumption that  $f$  lies in the cone defined as

$$\mathcal{C}_\tau := \{f : \text{Var}(f') \leq \tau \|f' - f(1) + f(0)\|_1\}. \quad (16)$$

This cone consists of functions whose stronger semi-norms are not too much greater than their weaker semi-norms. The set  $\mathcal{C}_\tau$  is called a cone because  $f \in \mathcal{C}_\tau$  implies that  $cf \in \mathcal{C}_\tau$  for all  $c \in \mathbb{R}$ . Substituting this upper bound on  $\text{Var}(f')$  into (15) and re-arranging the terms implies a data-driven upper bound on  $\text{Var}(f')$ :

$$\text{Var}(f') \leq \tau \|f' - f(1) + f(0)\|_1 \leq \frac{\tau \|A_n(f)' - f(1) + f(0)\|_1}{1 - \tau/(2n)} \quad \forall f \in \mathcal{C}_\tau, \quad (17)$$

provided that  $n > \tau/2$ . There is also a companion bound in the other direction based on (13) and (15):

$$\|A_n(f)' - f(1) + f(0)\|_1 \leq \|f' - f(1) + f(0)\|_1 \leq \frac{\text{Var}(f')}{2}. \quad (18)$$

Plugging (17) into (3) yields the following guaranteed error bound for the trapezoidal rule:

$$\text{err}(f, n) \leq \widetilde{\text{err}}(f, n) := \frac{\tau \|A_n(f)' - f(1) + f(0)\|_1}{4n(2n - \tau)} \quad \forall f \in \mathcal{C}_\tau. \quad (19)$$

This new error bound can then be used to construct the following adaptive, automatic quadrature algorithm, which is a simplification of Clancy et al. (2013, Algorithm 4):

**Algorithm 3** (Adaptive, Automatic, for Cones). Given  $\tau > 0$ , and an error tolerance,  $\varepsilon$ , let  $j = 1$  and  $n_1 = \lceil (\tau + 1)/2 \rceil$ .

**Step 1** Compute the error estimate  $\widetilde{\text{err}}(f, n_j)$  according to (19).

**Step 2** If  $\widetilde{\text{err}}(f, n_j) \leq \varepsilon$ , then return the trapezoidal rule approximation  $T_{n_j}(f)$  as the answer.

**Step 3** Otherwise let  $n_{j+1} = 2n_j$ , increase  $j$  by one, and go to Step 1.

Error bound (19) is used to prove that Algorithm ?? is successful for integrands in  $\mathcal{C}_\tau$ . Let  $N(f, \varepsilon, \tau)$  denote the eventual number of trapezoids required by this algorithm. Because the algorithm is adaptive, the number of trapezoids does depend on the particulars of  $f$ . The two bounds (17) and (18) may be used to derive both lower and upper bounds on  $N(f, \varepsilon, \tau)$ :

**Theorem 2.** (Clancy et al., 2013, Theorem 7) *Let  $N(f, \varepsilon, \tau)$  denote the final  $n_j$  in Algorithm ??. Then this algorithm is successful for all functions in  $\mathcal{C}_\tau$ ,*

i.e.,  $\left| \int_0^1 f(x) dx - T_{N(f, \varepsilon, \tau)}(f) \right| \leq \varepsilon$ . Moreover,  $N(f, \varepsilon, \tau)$  is bounded below and above as follows:

$$\begin{aligned} & \max \left( \left\lceil \frac{\tau + 1}{2} \right\rceil, \left\lceil \sqrt{\frac{\text{Var}(f')}{8\varepsilon}} \right\rceil \right) \\ & \leq \max \left( \left\lceil \frac{\tau + 1}{2} \right\rceil, \left\lceil \sqrt{\frac{\tau \|f' - f(1) + f(0)\|_1}{8\varepsilon}} \right\rceil \right) \\ & \leq N(f, \varepsilon, \tau) \\ & \leq \sqrt{\frac{\tau \|f' - f(1) + f(0)\|_1}{2\varepsilon}} + \tau + 3 \leq \sqrt{\frac{\tau \text{Var}(f')}{4\varepsilon}} + \tau + 3. \quad (20) \end{aligned}$$

Note that the cost of the algorithm depends on  $\text{Var}(f')$ , but without  $\text{Var}(f')$  needing to be known in advance. For an integrand like (5) where  $\text{Var}(f')$  is known, the number of trapezoids required by Algorithm ?? is no greater than  $\sqrt{\tau \text{Var}(f')/(4\varepsilon)} + \tau + 3$ , whereas for Algorithm 1 with  $\sigma = \text{Var}(f')$  the number of trapezoids is no more than  $\lceil \sqrt{\text{Var}(f')/(8\varepsilon)} \rceil$ . The extra cost for Algorithm ?? comes in the form of the factor  $\sqrt{2\tau}$  and the additional additive term of  $\tau + 1$ . On the other hand, if  $\text{Var}(f')$  is not known in advance, then the disadvantage of Algorithm 1 is that if  $\text{Var}(f')$  is not known in advance, and  $\sigma$  is chosen conservatively, then Algorithm ?? could cost arbitrarily more than  $\sqrt{\text{Var}(f')/(8\varepsilon)} + 1$  in the limit of ,

## 5 Acknowledgements

The authors are grateful for discussions with a number of colleagues. This research is supported in part by grant NSF-DMS-1115392.

## References

- Abramowitz, M. and I. A. Stegun (eds.) 1964. *Handbook of mathematical functions with formulas, graphs and mathematical tables*, U. S. Government Printing Office, Washington, DC.
- Brass, H. and K. Petras. 2011. *Quadrature theory: the theory of numerical integration on a compact interval*, First, American Mathematical Society, Rhode Island.
- Burden, R. L. and J. D. Faires. 2010. *Numerical analysis*, Ninth, Cengage Brooks/Cole, Belmont, CA.
- Cheney, W. and D. Kincaid. 2013. *Numerical mathematics and computing*, Seventh, Brooks/Cole, Boston.
- Clancy, N., Y. Ding, C. Hamilton, F. J. Hickernell, and Y. Zhang. 2013. *The complexity of guaranteed automatic algorithms: Cones, not balls*. submitted for publication, arXiv.org:1303.2412 [math.NA].
- Lyness, J. N. 1983. *When not to use an automatic quadrature routine*, SIAM Rev. **25**, 63–87.
- R Development Core Team. 2012. *The R Project for Statistical Computing*.

- Sauer, T. 2012. *Numerical analysis*, Pearson.
- The MathWorks, Inc. 2013. *MATLAB 8.1*, Natick, MA.
- The Numerical Algorithms Group. 2012. *The NAG library*, Mark 23, Oxford.
- Wolfram Research Inc. 2012. *Mathematica 9*, Champaign, IL.