Reliable Adaptive Numerical Integration

Fred J. Hickernell, Martha Razo, and Sunny Yun

Abstract. Numerical algorithms are required when the solutions to mathematical problems cannot be expressed analytically. Adaptive algorithms expend the right amount of computational effort to meet the error tolerance—easy problems require less effort and hard problems require more effort. Although convenient for users, adaptive algorithms usually lack guarantees. Here, we discuss the shortcomings of the numerical univariate integration algorithms that are taught in calculus courses and numerical analysis courses. We then introduce a new adaptive trapezoidal rule algorithm that has a rigorous guarantee for a certain cone of non-spiky integrands. Moreover, we show that the computational cost of our new algorithm has the same asymptotic rate as the best possible algorithm.

1. INTRODUCTION. While some mathematical problems can be solved with pencil and paper, many others must be solved by numerical algorithms. Software such as Mathematica (Wolfram Research Inc. 2014), MATLAB (The MathWorks, 2014), the NAG library (The Numerical Algorithms Group, 2013), and R (R Development Core Team, 2014), solve a wide variety of mathematical and statistical problems. Many numerical algorithms in these software environments are *adaptive*, meaning that they automatically expend the right amount of computational effort needed to (hopefully) provide an approximate solution with an error no greater than the tolerance. MATLAB's `integral` for integration, `fminbnd` for optimization, and `ode45` for solving ordinary differential equations are examples of adaptive algorithms. The Chebfun MATLAB toolbox (Hale et al., 2014) is a suite of adaptive algorithms.

At the core of every adaptive algorithm is an estimate of the error of the numerical approximation based on the computations already performed. Unfortunately, these error estimates either require too much information from the user or are based on heuristics and not guaranteed. Our aim is to rectify this deficiency—starting with the problem of numerical integration, also known as quadrature.

Various analytic methods for evaluating integrals are taught in calculus courses, e.g., substitution, integration by parts, and partial fractions. Unfortunately, the antiderivatives of many elementary functions are *not* themselves elementary functions. One important example arising in statistical inference is the standard normal probability density, $\phi : x \mapsto \mathrm{e}^{-x^2/2}/\sqrt{2\pi}$, whose integral is the standard normal cumulative distribution, $\Phi : x \mapsto \int_{-\infty}^{x} \phi(t)\,\mathrm{d}t$. Since the Φ is not an elementary function, values of Φ must be provided via tables in statistics textbooks and special functions in calculators. By contrast, the derivatives of elementary functions are always elementary functions.

We want an algorithm `integral`, that is *guaranteed* to provide a numerical approximation to the true integral with an error no greater than a specified error tolerance, namely,

$$\left| \int_a^b f(x)\,\mathrm{d}x - \texttt{integral}(f,a,b,\varepsilon) \right| \le \varepsilon \qquad \forall f \in \mathcal{C},\ a,b \in \mathbb{R},\ a < b,\ \varepsilon > 0,$$

where $\mathcal{C}$ is some clearly defined set of integrands. The input f is a black-box algorithm that produces $f(x)$ for any given $x \in [a,b]$. The definition of $\mathcal{C}$ should entail general assumptions—but not detailed information—about f.

Here we focus on the trapezoidal rule, whose properties are reviewed in Section 2:

$$T_n(f) := \frac{b-a}{2n}[f(t_0) + 2f(t_1) + \cdots + 2f(t_{n-1}) + f(t_n)],$$
$$t_i = a + \frac{i(b-a)}{n},\ i = 0, \ldots, n,\ n \in \mathbb{N} := \{1, 2, \ldots\}. \quad (1)$$

We discuss *three* automatic numerical integration algorithms based on T_n. The first two are well-known, while the third is new. Each algorithm takes a different approach to the problem of choosing the number of trapezoids required to meet the error tolerance. Figure 1 displays four integrands with different characteristics. Table 1 indicates which algorithm is successful for which integrand(s).

- Students see the trapezoidal rule in their calculus courses, where they learn formula (1) and are provided an error bound, such as (4). Students might use the non-adaptive algorithm `ballint` presented in Section 3 to compute integrals numerically. This algorithm requires an upper bound on the total variation of f'. If the user chooses a modest upper bound, then `ballint` will work fine for integrands such as f_{easy}, defined in (5), but `ballint` will fail for integrands such as the others in Figure 1, whose first derivatives have large variations.

  We define the computational cost of an algorithm to be the number of function values required. Sect. 3 also discusses the computational cost of `ballint` and its optimality for balls of integrands.
- Courses in numerical analysis teach students to estimate the absolute error of $T_n(f)$ by $\left|T_n(f) - T_{n/2}(f)\right|/3$. This error estimate leads to the *adaptive* algorithm, `flawint`, given in Section 4. Because it is adaptive, `flawint` can handle large integrands, such as f_{big} defined in (9), as well as modestly sized integrands such as, f_{easy}. However, `flawint` is totally fooled by f_{fluky}, defined in (12), which resembles f_{big}. As pointed out by James Lyness (1983) and discussed in Section 6, the error estimate used by `flawint` has a serious flaw, which is shared by virtually all adaptive quadrature algorithms.
- Although Lyness presents a pessimistic view of adaptive quadrature, we have a better alternative. Section 7 describes a data-driven trapezoidal rule error bound for a precisely defined cone of non-spiky integrands. This provides the foundation for our new adaptive algorithm, `integral`, which succeeds for all integrands in Fig. 1 except the spiky one. As explained in Section 5, all algorithms must fail for sufficiently spiky integrands. Section 8 provides an upper bound on the computational cost of `integral`, and shows that this cost is asymptotically optimal among all possible algorithms for the same problem.

Section 9 discusses the ramifications of these developments for how we teach numerical integration. We also discuss how to develop new reliable adaptive numerical algorithms for other problems.

A different approach to reliable computation than the one we pursue here involves *interval arithmetic*, described by Moore et al. (2009) and Rump (2010) and implemented in INTLAB (Rump, 1999). This approach works with functions that take interval arguments and return interval outputs.

2. THE COMPOSITE TRAPEZOIDAL RULE AND ITS ERROR BOUND. The composite trapezoidal rule T_n approximates an integral by the sum of the areas of n trapezoids whose heights are function values (see (1) and Fig. 1(a)). The trapezoidal

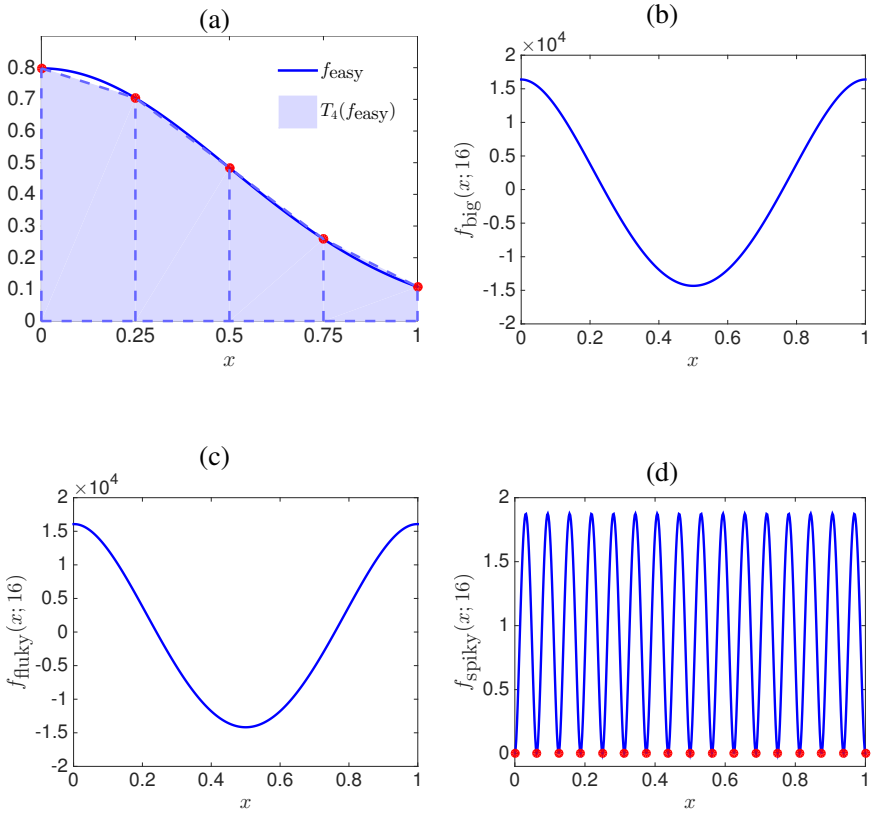

Figure 1. Four integrands illustrating the strengths and weaknesses of the three algorithms described in this article: (a) f_{easy} defined in (5), (b) $f_{\text{big}}(\cdot, 16)$ defined in (9), (c) $f_{\text{fluky}}(\cdot, 16)$ defined in (12), and (d) $f_{\text{spiky}}(\cdot, 16)$ defined in (11).

Table 1. Success (✓) or Failure (✗) of Three Quadrature Algorithms for the Four Different Integrands Depicted in Figure 1.

Algorithm	f_{easy}	f_{big}	f_{fluky}	f_{spiky}
`ballint` (Section 3)	✓	✗	✗	✗
`flawint` (Section 4)	✓	✓	✗	✗
`integral` (Section 7)	✓	✓	✓	✗

rule is also the integral of the piecewise linear spline approximation to the integrand. Under mild smoothness conditions on f we know that $T_n(f) \to \int_a^b f(x)\,\mathrm{d}x$ as $n \to \infty$ (see (4) below).

We would like to turn T_n into an *automatic* quadrature algorithm that chooses n to ensure that the trapezoidal rule error is small enough. Given the user's tolerance for error, ε, an automatic trapezoidal algorithm should choose n such that

$$\mathrm{err}(f,n) := \left| \int_a^b f(x)\,\mathrm{d}x - T_n(f) \right| \le \varepsilon.$$

Upper bounds on the trapezoidal rule error assume that the integrand possesses some smoothness. For any function $f : [a, b] \to \mathbb{R}$, let $f'(x^-)$ and $f'(x^+)$ denote the left and right derivatives of f at x, respectively. Furthermore, let $f'(x) := f'(x^+)$ for $a \le x < b$ and $f'(b) := f(b^-)$. This makes f' well-defined even when it has jump discontinuities. All integrands considered in this article have derivatives with bounded variation, i.e., they lie in the linear space

$$\mathcal{V} := \{f : \mathrm{Var}(f') < \infty\}.$$

Here $\mathrm{Var}(\cdot)$ represents the (total) variation of a function:

$$\mathrm{Var}(f) := \sup\left\{\sum_{i=1}^{n} |f(x_i) - f(x_{i-1})| : \{x_i\}_{i=0}^{n} \text{ is a partition}, n \in \mathbb{N}\right\}. \tag{2}$$

A *partition*, $\{x_i\}_{i=0}^{n}$, is defined as an ordered sequence of points that includes the endpoints of the interval, $a =: x_0 \le x_1 \le \cdots \le x_{n-1} \le x_n := b$. Intuitively, $\mathrm{Var}(f)$ is the total vertical distance up and down that one travels on a roller coaster whose track is the graph of f. The function $f \mapsto \mathrm{Var}(f')$ is a semi-norm on the space $\mathcal{V}$. The restriction that $\mathrm{Var}(f')$ is finite implies that f' may have at most a countably infinite number of discontinuities.

An example of a function in $\mathcal{V}$ whose first derivative is discontinuous but has finite variation is the triangle-shaped function $\mathrm{peak}(\cdot; t, h)$, which is used later in our error analysis. For $a \le t < t + 2h \le b$ let

$$\mathrm{peak}(x; t, h) := \begin{cases} h - |x - t - h|, & t \le x < t + 2h \\ 0, & a \le x < t \text{ or } t + 2h \le x \le b, \end{cases} \tag{3a}$$

$$\mathrm{peak}'(x; t, h) = \begin{cases} 1, & t \le x < t + h, \\ -1, & t + h \le x < t + 2h, \\ 0, & a \le x < t \text{ or } t + 2h \le x \le b, \end{cases} \tag{3b}$$

$$\mathrm{Var}(\mathrm{peak}'(\cdot; t, h)) \le 4 \text{ with equality if } a < t < t + 2h < b, \tag{3c}$$

$$\int_a^b \mathrm{peak}(x; t, h)\,\mathrm{d}x = h^2. \tag{3d}$$

Figure 2 depicts two examples of functions in $\mathcal{V}$ with discontinuous first derivatives.

The true error of the trapezoidal rule, $\mathrm{err}(f, n)$, is rarely known in practice, but there exists a rigorous upper bound on the error of the form

$$\mathrm{err}(f, n) \le \frac{(b - a)^2\,\mathrm{Var}(f')}{8n^2} =: \overline{\mathrm{err}}(f, n), \qquad n \in \mathbb{N} \tag{4}$$

(see Brass and Petras (2011, Section 7.2, (7.15))). An error bound involving the stronger norm $\sup_{x\in[a,b]} |f''(x)|$ may be more common in the literature, but bound (4) is less restrictive on f with the same order of convergence as $n \to \infty$. The trapezoidal rule gives the exact answer for linear integrands. Error bound (4) reflects this fact since if $f(x) = \alpha x + \beta$, then $f'(x) = \alpha$, and so $\mathrm{Var}(f') = 0$.

To illustrate this error bound, consider the normal probability example mentioned above, but with a standard deviation of $1/2$. The integrand and the trapezoidal rule

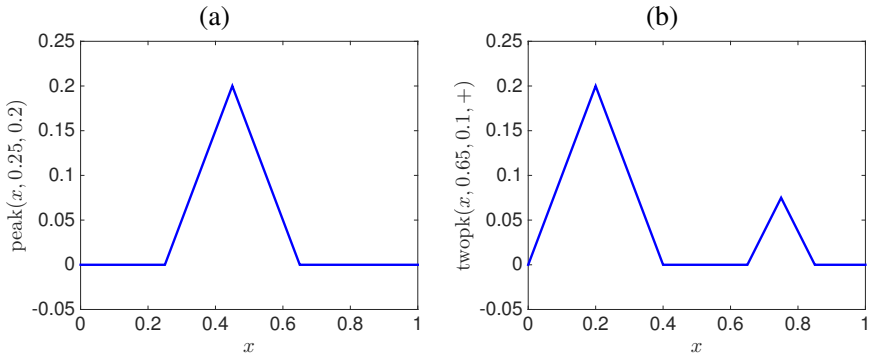

Figure 2. Two functions in $\mathcal{V}$ with discontinuous first derivatives: (a) a single peaked function, $\mathrm{peak}(\cdot, 0.25, 0.20)$ as defined in (3), and (b) a double-peaked function, $\mathrm{twopk}(\cdot, 0.65, 0.1, +)$ with $\mathfrak{h} = 0.2$ as defined in (16).

approximation over the interval $[0, 1]$ are depicted in Fig. 1(a) and defined as follows:

$$f_{\text{easy}}(x) = 2\phi(2x) = \sqrt{\frac{2}{\pi}}\mathrm{e}^{-2x^2}, \tag{5a}$$

$$\int_0^1 f_{\text{easy}}(x)\,\mathrm{d}x = 0.4772, \qquad T_4(f) = 0.4750, \tag{5b}$$

$$\mathrm{Var}(f'_{\text{easy}}) = 1.5038, \qquad \mathrm{err}(f_{\text{easy}}, 4) = 0.0022 \le 0.0117 = \overline{\mathrm{err}}(f_{\text{easy}}, 4). \tag{5c}$$

The value of the integral to four significant digits may be found by a variety of quadrature algorithms. As expected, the actual error is no greater than the error bound in (4).

3. AN AUTOMATIC ALGORITHM, `ballint`, FOR BALLS OF INTEGRANDS.

Error bound (4) can help us determine how large n must be to satisfy the error tolerance if an upper bound on $\mathrm{Var}(f')$ is available or can be correctly assumed. The following automatic algorithm fits this situation.

Algorithm `ballint` (Non-Adaptive, for Balls). Given an interval $[a, b]$, a ball radius $\sigma > 0$, an error tolerance, ε, and a routine for generating values of f, set

$$n = \left\lceil (b-a)\sqrt{\frac{\sigma}{8\varepsilon}} \right\rceil, \tag{6}$$

and return the trapezoidal rule $\texttt{ballint}(f, a, b, \varepsilon) = T_n(f, a, b)$ as the answer.

Theorem 1. *For all integrands f lying in the ball $\mathcal{B}_\sigma := \{g \in \mathcal{V} : \mathrm{Var}(g') \le \sigma\}$,* `ballint` *is successful, i.e.,*

$$\left| \int_a^b f(x)\,\mathrm{d}x - \texttt{ballint}(f, a, b, \varepsilon) \right| \le \varepsilon.$$

The computational cost of `ballint` *is $n + 1$ function values, where n is given by* (6).

The algorithm `ballint` is automatic because it expends as much effort as required by the error tolerance, ε and the radius of the ball, σ, to obtain the desired answer. It

is non-adaptive because the computational cost is the same for all integrands given ε and σ.

The computational cost of `ballint` is asymptotically optimal for the integration problem defined over $\mathcal{B}_\sigma$. This can be shown by constructing two fooling functions.

Theorem 2. *Let* `int` *be any (possibly adaptive) algorithm that succeeds for all integrands in* $\mathcal{B}_\sigma$*. For any error tolerance* $\varepsilon > 0$*,* `int` *must use at least* $-1 + (b-a)\sqrt{\sigma/(16\varepsilon)}$ *function values. As* $\sigma/\varepsilon \to \infty$ *the asymptotic rate of increase is the same as the computational cost of* `ballint`*.*

Proof. Let $\{x_i\}_{i=1}^{n-1}$ be the ordered nodes where $\texttt{int}(\cdot, a, b, \varepsilon)$ evaluates the zero integrand. Augment these points to form a partition of $[a, b]$, denoted $\{x_i\}_{i=0}^{n}$. The mesh size of this partition is defined as the maximum distance between adjacent points

$$\operatorname{size}(\{x_i\}_{i=0}^{n}) := \max_{i=1,\ldots,n} (x_i - x_{i-1}). \tag{7}$$

Let (x_-, x_+) be a pair of consecutive points in the partition with maximum separation, i.e., $x_+ - x_- = \operatorname{size}(\{x_i\}_{i=0}^{n}) \geq (b-a)/n$. By (3c) the integrands $f_\pm(x) := \pm\sigma \operatorname{peak}(x, x_-, (x_+ - x_-)/2)/4$ lie in $\mathcal{B}_\sigma$ and therefore must be integrated successfully by `int`. Moreover, since $f_\pm(x_1) = \cdots = f_\pm(x_{n-1}) = 0$, it follows that 0 and $f_\pm$ are indistinguishable to `int`, and thus $\texttt{int}(f_\pm, a, b, \varepsilon) = \texttt{int}(0, a, b, \varepsilon)$. Applying (3d) and the triangle inequality leads to a lower bound on n:

$$\begin{aligned}
\varepsilon &\geq \frac{1}{2}\left[\left|\int_a^b f_-(x)\,\mathrm{d}x - \texttt{int}(f_-, a, b, \varepsilon)\right| + \left|\int_a^b f_+(x)\,\mathrm{d}x - \texttt{int}(f_+, a, b, \varepsilon)\right|\right] \\
&= \frac{1}{2}\left[\left|\texttt{int}(f_-, a, b, \varepsilon) - \int_a^b f_-(x)\,\mathrm{d}x\right| + \left|\int_a^b f_+(x)\,\mathrm{d}x - \texttt{int}(f_+, a, b, \varepsilon)\right|\right] \\
&\geq \frac{1}{2}\left|\int_a^b f_+(x)\,\mathrm{d}x - \int_a^b f_-(x)\,\mathrm{d}x\right| = \int_a^b \frac{\sigma}{4} \operatorname{peak}(x, x_-, (x_+ - x_-)/2)\,\mathrm{d}x \\
&= \frac{\sigma(x_+ - x_-)^2}{16} \geq \frac{(b-a)^2\sigma}{16n^2}.
\end{aligned}$$

This inequality provides a lower bound on $n - 1$, which is the computational cost of this arbitrary, successful algorithm, `int`, as given in the statement of Theorem 2. The asymptotic rate of increase is $\mathcal{O}(\sqrt{\sigma/\varepsilon})$ as $\sigma/\varepsilon \to \infty$, the same as for (6). ∎

For f_{easy} in (5), one may choose $\sigma = 1.5038$, and then `ballint` uses $n = \lceil 0.4336/\sqrt{\varepsilon}\,\rceil$ trapezoids to get an answer within ε of the true answer. Picking any modest value of σ no smaller than 1.5038 would work also.

The numerical integration problems arising in calculus courses usually have integrands, f, for which $\operatorname{Var}(f')$ is easy to compute or bound. However, we are aiming for a quadrature algorithm that accepts f as a black-box whose formula might be quite complex. We do not want to require the user to provide an upper bound on $\operatorname{Var}(f')$.

4. AN ADAPTIVE, BUT FLAWED ALGORITHM, `flawint`**.** Adaptive quadrature algorithms don't need a value of σ, which `ballint` requires. Instead, adaptive quadrature algorithms bound or estimate their error using only function values and then determine the sample size accordingly. 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,

$$\widehat{\mathrm{err}}(f,n) := \frac{\left|T_n(f) - T_{n/2}(f)\right|}{3}, \qquad \frac{n}{2} \in \mathbb{N}. \tag{8}$$

This error estimate leads to the following adaptive quadrature algorithm, `flawint`. Each iteration doubles the previous number of trapezoids so that function values can be reused.

Algorithm `flawint` (Adaptive, for Cones). Given an error tolerance, ε, let $j = 1$ and $n_1 = 2$.

Step 1 Compute the error estimate $\widehat{\mathrm{err}}(f, n_j)$ according to (8).
Step 2 If $\widehat{\mathrm{err}}(f, n_j) \le \varepsilon$, 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.

Consider the integrand f_{big}, plotted above in Fig. 1(b), and defined below:

$$f_{\text{big}}(x;m) := 1 + \frac{15m^4}{2}\left[\frac{1}{30} - x^2(1-x)^2\right], \quad m \in \mathbb{N}, \tag{9a}$$

$$\int_0^1 f_{\text{big}}(x;m)\,\mathrm{d}x = 1, \quad T_n(f_{\text{big}}(x;m)) = 1 + \frac{m^4}{4n^4}, \tag{9b}$$

$$\mathrm{Var}(f'_{\text{big}}(x;m)) = \frac{10m^4}{\sqrt{3}}, \tag{9c}$$

$$\mathrm{err}(f_{\text{big}}(x;m),n) = \frac{m^4}{4n^4} \le \frac{5m^4}{4n^4} = \widehat{\mathrm{err}}(f_{\text{big}}(x;m),n). \tag{9d}$$

(As an aside, $\mathrm{err}(f_{\text{big}}(x;m),n) = \mathcal{O}(n^{-4})$ rather than only $\mathcal{O}(n^{-2})$ because f_{big} has sufficient periodic derivatives.) Algorithm `flawint` works well for this example because the error estimate is always larger than the true error no matter how large m is, but `flawint` does not need m or $\mathrm{Var}(f'_{\text{big}}(x;m))$ as an input. On the other hand, `ballint` with a fixed σ would fail for $f_{\text{big}}(\cdot;m)$ with m large enough.

The algorithm `flawint` must succeed for all integrands in the cone of functions where the true error is no greater than the error estimate:

$$\mathcal{C}_{\text{flaw}} := \{f \in \mathcal{V} : \mathrm{err}(f,n) \le \widehat{\mathrm{err}}(f,n)\ \forall n/2 \in \mathbb{N}\}.$$

However, one would desire a more intuitive understanding of what kinds of functions lie in $\mathcal{C}_{\text{flaw}}$.

Error estimate (8) may be explained by noting that Simpson's rule is actually the sum of the trapezoidal rule plus its error estimate:

$$\begin{aligned} S_n(f) &:= \frac{b-a}{3n}\left[f(t_0) + 4f(t_1) + 2f(t_2) + \cdots + 2f(t_{n-2}) + 4f(t_{n-1}) + f(t_n)\right] \\ &= \frac{4T_n(f) - T_{n/2}(f)}{3} = T_n(f) + \frac{T_n(f) - T_{n/2}(f)}{3}, \qquad \frac{n}{2} \in \mathbb{N}, \end{aligned}$$

where $t_i = a + i(b-a)/n$. The error bound for Simpson's rule then serves as an bound on the error of $\widehat{\mathrm{err}}(f,n)$ (Brass and Petras, 2011, Section 7.3, p. 231):

$$\begin{aligned}|\mathrm{err}(f,n) - \widehat{\mathrm{err}}(f,n)| &= \left| \left| \int_a^b f(x)\,\mathrm{d}x - T_n(f) \right| - \frac{|T_n(f) - T_{n/2}(f)|}{3} \right| \\ &\le \left| \int_a^b f(x)\,\mathrm{d}x - S_n(f) \right| \le \frac{(b-a)^4\,\mathrm{Var}(f''')}{36n^4}. \end{aligned} \tag{10}$$

Since $|\mathrm{err}(f,n) - \widehat{\mathrm{err}}(f,n)| = \mathcal{O}(n^{-4})$, while $\mathrm{err}(f,n) = \mathcal{O}(n^{-2})$, it follows that $\widehat{\mathrm{err}}(f,n)$ does an excellent job in approximating $\mathrm{err}(f,n)$ *for* n *large enough*, provided $\mathrm{Var}(f''')$ is not too large.

Unfortunately, (10) provides insufficient justification for `flawint`. We need conditions under which $\mathrm{err}(f,n) \le \widehat{\mathrm{err}}(f,n)$, as in the definition of $\mathcal{C}_{\text{flaw}}$, not just $\mathrm{err}(f,n) \approx \widehat{\mathrm{err}}(f,n)$. More troubling, to use (10) would require an upper bound on $\mathrm{Var}(f''')$. If one is available, then we should use an automatic, non-adaptive algorithm like `ballint`, but based on Simpson's rule. This would provide a higher order convergence rate than that of `flawint`.

5. SPIKY INTEGRANDS. Any quadrature algorithm may fail to give the correct answer if f has significant spikes between the sites where it is sampled. Figure 1 (d) depicts the following spiky integrand with m spikes on $[0,1]$:

$$f_{\text{spiky}}(x;m) = 30[\{mx\}(1-\{mx\})]^2, \qquad m \in \mathbb{N}, \quad \{x\} := x \bmod 1, \tag{11a}$$

$$\int_0^1 f_{\text{spiky}}(x;m)\,\mathrm{d}x = 1, \qquad \mathrm{Var}(f'_{\text{spiky}}(\cdot;m)) = \frac{40m^2}{\sqrt{3}}, \tag{11b}$$

$$T_n(f_{\text{spiky}}(\cdot;m)) = 0, \qquad \text{for } \frac{m}{n} \in \mathbb{N}, \tag{11c}$$

$$\mathrm{err}(f_{\text{spiky}}(\cdot;m),n) = 1 \le \frac{5m^2}{\sqrt{3}n^2} = \overline{\mathrm{err}}(f_{\text{spiky}}(\cdot;m),n) \qquad \text{for } \frac{m}{n} \in \mathbb{N}, \tag{11d}$$

$$\mathrm{err}(f_{\text{spiky}}(\cdot;m),n) = 1 > 0 = \widehat{\mathrm{err}}(f_{\text{spiky}}(\cdot;m),n) \qquad \text{for } \frac{m}{n} \in \mathbb{N}. \tag{11e}$$

Suppose that $\varepsilon < 1$, and `ballint` chooses a particular $n = \sqrt{\sigma/(8\varepsilon)}$. Then for all integrands $f_{\text{spiky}}(\cdot;m)$ with m an integer multiple of n, it follows from (11d) that $\mathrm{err}(f_{\text{spiky}}(\cdot;m),n) = 1 > \varepsilon$, so `ballint` fails. However, Theorem 1 remains valid because $f_{\text{spiky}}(\cdot;m)$ falls outside $\mathcal{B}_\sigma$:

$$\mathrm{Var}(f'_{\text{spiky}}(\cdot;m)) = \frac{40m^2}{\sqrt{3}} = \frac{5m^2\sigma}{\sqrt{3}n^2\varepsilon} > \sigma.$$

The choice of σ reflects how spiky an integrand `ballint` is willing to tolerate.

Algorithm `flawint` also fails for $f_{\text{spiky}}(\cdot;m)$ with $m/n \in \mathbb{N}$ because the error estimate $\widehat{\mathrm{err}}(f_{\text{spiky}}(\cdot;m),n)$ is badly wrong. Note that replacing error estimate $\widehat{\mathrm{err}}(f,n)$ in Step 2 of `flawint` by a more conservative error estimate of the form $A\widehat{\mathrm{err}}(f,n)$ with $A > 1$ does not help.

The challenge of spiky integrands applies to *any* quadrature algorithm that depends on function values, including our proposed `integral` below. In contrast to `ballint`,

our new algorithm is adaptive and succeeds for a cone of integrands, $\mathcal{C}$. In contrast to `flawint` our new algorithm has rigorous characterization of how spiky an integrand can be and still be inside $\mathcal{C}$.

6. FLUKY INTEGRANDS. Adaptive algorithm `flawint` has an even worse flaw than its inability to handle spiky integrands. This was pointed out over thirty years ago by James Lyness in his SIAM Review article, *When Not to Use an Automatic Quadrature Routine*. Lyness (1983, p. 69) claimed:

> 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 call this a *fluky* integrand.

Figure 1(c) shows a fluky integrand that fools the error estimate in (8):

$$f_{\text{fluky}}(x;m) = f_{\text{big}}(x;m) + \frac{15m^2}{2}\left[-\frac{1}{6} + x(1-x)\right], \quad m \in \mathbb{N}, \tag{12a}$$

$$\int_0^1 f_{\text{fluky}}(x;m)\,\mathrm{d}x = 1, \quad T_n(f_{\text{fluky}}(x;m)) = 1 + \frac{m^2(m^2-5n^2)}{4n^4}, \tag{12b}$$

$$\operatorname{err}(f_{\text{fluky}}(\cdot;m),n) = \frac{m^2\,|m^2-5n^2|}{4n^2}, \tag{12c}$$

$$\widehat{\operatorname{err}}(f_{\text{fluky}}(\cdot;m),n) = \frac{15m^2\,|m^2-n^2|}{4n^4}. \tag{12d}$$

$$\operatorname{err}(f_{\text{fluky}}(\cdot;n),n) = 1 > 0 = \widehat{\operatorname{err}}(f_{\text{fluky}}(\cdot;n),n). \tag{12e}$$

The integrand f_{fluky} appears quite "reasonable"—similar in shape to f_{big}. We label integrands like this one "fluky" because their construction is rather delicate, and they totally fool the error bound while having a small number of local optima.

While we may more readily understand that sufficiently spiky integrands fall outside $\mathcal{C}_{\text{flaw}}$, the cone of integrands for which `flawint` succeeds, it is now also apparent that non-spiky integrands, such as f_{fluky}, also fall outside this cone. Even inflating the error estimate by a constant does not help. The next section presents an adaptive algorithm that only fails for spiky functions.

Nearly all existing adaptive quadrature algorithms can be fooled by both spiky and fluky integrands. Figure 3 displays two integrands that fool MATLAB's premier quadrature algorithm, `integral` (The MathWorks, 2014), which is based on an adaptive composite Gauss-Konrod scheme devised by Larry Shampine (2008). The difference between two Gauss rules with different orders of accuracy—but using the same nodes–is used to estimate the error. Fig. 3(a) depicts a spiky function whose integral is 1 but for which MATLAB gives a value of 0. Figure 3(b) depicts a fluky function whose integral is 0.278827 but for which MATLAB gives a value of $0.278\mathbf{799}$. In both cases the absolute and relative error tolerances are set to 10^{-13}, but clearly are not met. There is no theory describing the conditions under which MATLAB's `integral` must succeed.

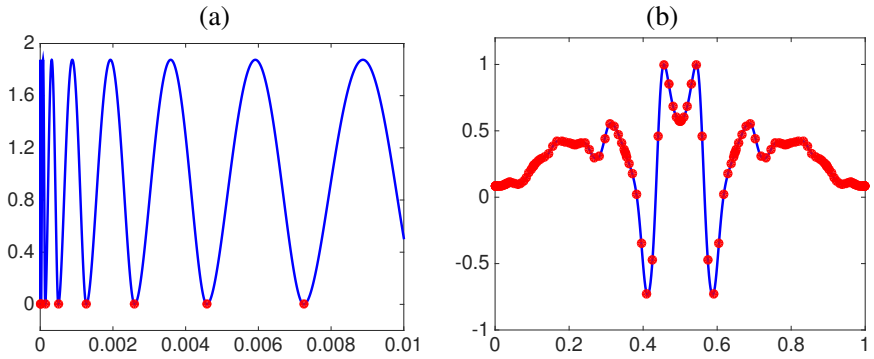

Figure 3. Two integrands defined on $[0, 1]$ and designed to fool MATLAB's `integral` and the data sampled by `integral`: (a) a spiky integrand, and (b) a fluky integrand.

7. A GUARANTEED, ADAPTIVE TRAPEZOIDAL ALGORITHM `integral`.

Non-adaptive `ballint` uses no values of f to determine how many trapezoids are needed because an upper bound on $\mathrm{Var}(f')$ is assumed. Adaptive `flawint` uses values of f to determine how many trapezoids are needed, but in a way that does not detect a large $\mathrm{Var}(f')$ for fluky integrands. In this section we construct an adaptive algorithm that reliably bounds $\mathrm{Var}(f')$ for a certain cone of integrands, $\mathcal{C}$.

Given any partition, $\{x_i\}_{i=0}^n$, define an approximation to $\mathrm{Var}(f')$ as follows:

$$\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) := \sum_{i=2}^{n-1} |\Delta_i - \Delta_{i-1}|\,,$$

$$\text{where } \Delta_i \text{ is between } f'(x_i^-) \text{ and } f'(x_i^+). \quad (13)$$

Note that $\widehat{V}$ does not involve the values of f' at $x_0 = a$ or $x_n = b$. Also note that by definition the approximation is actually a lower bound:

$$\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) \le \mathrm{Var}(f') \quad \forall f \in \mathcal{V},\ \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1},\ n \in \mathbb{N}. \quad (14)$$

Our new algorithm will be guaranteed to work for the cone of integrands for which $\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1})$ does not underestimate $\mathrm{Var}(f')$ by much:

$$\mathcal{C} := \{f \in \mathcal{V} : \mathrm{Var}(f') \le \mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^n))\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) \text{ for all}$$
$$\text{choices of } n \in \mathbb{N},\ \{\Delta_i\}_{i=1}^{n-1}, \text{ and } \{x_i\}_{i=0}^n \text{ with } \mathrm{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}\}. \quad (15)$$

The cut-off value $\mathfrak{h} \in (0, b-a]$ and 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-decreasing. One possibility is $\mathfrak{C}(h) := \mathfrak{C}(0)\mathfrak{h}/(\mathfrak{h} - h)$.

The cone $\mathcal{C}$ is defined to rule out sufficiently spiky functions because f' is not allowed to change much over a small distance if $f \in \mathcal{C}$. If a function looks like a line on a sufficiently fine mesh, i.e., if $f'(x_i^-) = f'(x_i^+) = \beta$ for $i = 1, \ldots, n-1$ for some real β and some partition $\{x_i\}_{i=0}^n$ with $\mathrm{size}(\{x_i\}_{i=0}^n) \le \mathfrak{h}$, then f must be the linear function $f(x) = f(a) + \beta(x - a)$. While the triangular peak function $\mathrm{peak}(\cdot; t, h)$ lies inside $\mathcal{C}$ for $h \ge \mathfrak{h}$ and $t \in [a + \mathfrak{h}, b - 3\mathfrak{h}]$, it lies outside $\mathcal{C}$ for $h < \mathfrak{h}/2$.

The definition of $\mathcal{C}$ does not rule out all functions with narrow spikes. The following double peaked function—depicted in Fig. 2(b)—always lies in $\mathcal{C}$:

$$\operatorname{twopk}(x;t,h,\pm) := \operatorname{peak}(x,0,\mathfrak{h}) \pm \frac{3[\mathfrak{C}(h)-1]}{4}\operatorname{peak}(x,t,h), \\ a+3\mathfrak{h} \le t \le b-3h,\ 0 \le h < \mathfrak{h}, \quad \text{(16a)}$$

$$\operatorname{Var}(\operatorname{twopk}'(x;t,h,\pm)) = 3 + \frac{3[\mathfrak{C}(h)-1]}{4} \times 4 = 3\mathfrak{C}(h). \quad \text{(16b)}$$

From this definition it follows that

$$\begin{aligned}
&\mathfrak{C}(\operatorname{size}(\{x_i\}_{i=0}^n))\widehat{V}(\operatorname{twopk}'(x;t,h,\pm),\{x_i\}_{i=0}^n,\{\Delta_i\}_{i=1}^{n-1}) \\
&\quad \ge \begin{cases} \mathfrak{C}(0)\operatorname{Var}(\operatorname{twopk}'(x;t,h,\pm)), & 0 \le \operatorname{size}(\{x_i\}_{i=0}^n) < h, \\ \mathfrak{C}(h)\operatorname{Var}(\operatorname{peak}'(x;0,\mathfrak{h})) = 3\mathfrak{C}(h), & h \le \operatorname{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}, \end{cases} \\
&\quad \ge \operatorname{Var}(\operatorname{twopk}'(x;t,h,\pm)), \qquad 0 \le \operatorname{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}. \quad \text{(16c)}
\end{aligned}$$

Although $\operatorname{twopk}(\cdot;t,h,\pm)$ may have a peak of arbitrarily small width half-width, h, the height of this peak is small enough so that $\operatorname{twopk}(\cdot;t,h,\pm)$ still lies in $\mathcal{C}$ by (16c).

We cannot use $\widehat{V}(f',\{x_i\}_{i=0}^n,\{\Delta_i\}_{i=1}^{n-1})$ to approximate $\operatorname{Var}(f')$ because it depends on values of f', not values of f. However, $\widehat{V}(f',\{x_i\}_{i=0}^n,\{\Delta_i\}_{i=1}^{n-1})$ is closely related to the following approximation to $\operatorname{Var}(f')$, which is the total variation of the derivative of the linear spline approximation to f:

$$\begin{aligned}
\widetilde{V}_n(f) &:= \sum_{i=1}^{n-1}\left|\frac{f(t_{i+1})-f(t_i)}{t_{i+1}-t_i} - \frac{f(t_i)-f(t_{i-1})}{t_i-t_{i-1}}\right| \\
&= \frac{n}{b-a}\sum_{i=1}^{n-1}\left|f(t_{i+1})-2f(t_i)+f(t_{i-1})\right|, \\
&\qquad\qquad t_i = a + \frac{i(b-a)}{n},\ i=0,\ldots,n,\ n\in\mathbb{N}.
\end{aligned}$$

Lemma 1. *For all $f \in \mathcal{C}$ it follows that $\widetilde{V}_n(f) \le \operatorname{Var}(f') \le \mathfrak{C}(2(b-a)/n)\widetilde{V}_n(f)$ for $n > 2(b-a)/\mathfrak{h}$.*

Proof. For all $i = 0,\ldots,n-1$ it follows that

$$f(t_{i+1}) - f(t_i) = \int_{t_i}^{t_{i+1}} f'(x)\,\mathrm{d}x = (t_{i+1}-t_i)\Delta_i \quad \text{(17)}$$

for some Δ_i between $f'(x_i^-)$ and $f'(x_i^+)$ and for some $x_i \in [t_i,t_{i+1}]$. This follows from a mean value argument. Since it is impossible for $f'(x)$ to lie strictly below or strictly above $[f(t_{i+1})-f(t_i)]/(t_{i+1}-t_i)$, there must exist $\xi_\pm \in [t_i,t_{i+1}]$ with $f'(\xi_-) \le [f(t_{i+1})-f(t_i)]/(t_{i+1}-t_i) \le f'(\xi_+)$. A bisection algorithm then converges to an x_i with the desired property. Augmenting $\{x_i\}_{i=1}^n$ with a and b to become

a partition, $\{x_i\}_{i=0}^{n+1}$, it follows from (17) that

$$\widetilde{V}_n(f) = \sum_{i=1}^{n-1} |\Delta_i - \Delta_{i-1}| = \widehat{V}(f', \{x_i\}_{i=0}^{n+1}, \{\Delta_i\}_{i=1}^{n})$$

$$\begin{cases} \le \operatorname{Var}(f') & \text{by (14),} \\ \ge \dfrac{\operatorname{Var}(f')}{\mathfrak{C}(\operatorname{size}(\{x_i\}_{i=0}^{n+1}))} \ge \dfrac{\operatorname{Var}(f')}{\mathfrak{C}(2(b-a)/n)} & \text{by (15),} \end{cases}$$

where $x_i - x_{i-1} \le t_{i+1} - t_{i-1} \le 2(b-a)/n$ for $i = 1, \ldots, n$. ∎

Algorithm `integral` below computes $\widetilde{V}_{n_j}(f)$ for an increasing sequence of integers $n_1, n_2, \ldots$ with $n_1 > 2(b-a)/\mathfrak{h}$. Because the nodes used in the algorithm are nested, it follows that $\widetilde{V}_{n_j}(f)$, $j \in \mathbb{N}$ is a non-decreasing lower bound on $\operatorname{Var}(f')$. By Lemma 1 we also have an upper bound on $\operatorname{Var}(f')$ given by

$$\overline{V}_j := \min_{k=1,\ldots,j} \mathfrak{C}\left(\frac{2(b-a)}{n_k}\right) \widetilde{V}_{n_k}(f), \qquad j \in \mathbb{N}.$$

Thus, a necessary condition for $f \in \mathcal{C}$ is that $\widetilde{V}_{n_j}(f) \le \overline{V}_j$ for $j \in \mathbb{N}$.

The upper bound on $\operatorname{Var}(f')$ can be combined with the error bound in (4) to provide a data-based upper bound on the trapezoidal rule error:

$$\operatorname{err}(f, n_j) \le \overline{\operatorname{err}}(f, n_j) = \frac{(b-a)^2 \operatorname{Var}(f')}{8n_j^2} \le \frac{(b-a)^2 \overline{V}_j}{8n_j^2}. \tag{18}$$

This is the crux of our guaranteed adaptive trapezoidal rule, `integral`, which is guaranteed to find an answer within the error tolerance for integrands in $\mathcal{C}$.

Algorithm `integral` (Adaptive, for Cones of Integrands, $\mathcal{C}$). Given an interval, $[a, b]$, an inflation function, $\mathfrak{C}$, a positive key mesh size, $\mathfrak{h}$, a positive error tolerance, ε, and a routine for generating values of the integrand, f, set $j = 1$, $n_1 = \lfloor 2(b-a)/\mathfrak{h} \rfloor + 1$, and $\overline{V}_0 = \infty$.

Step 1 Compute $\widetilde{V}_{n_j}(f)$ and $\overline{V}_j = \min\left(\overline{V}_{j-1}, \mathfrak{C}\left(\dfrac{2(b-a)}{n_j}\right) \widetilde{V}_{n_j}(f)\right)$. If it happens that $\widetilde{V}_{n_j}(f) > \overline{V}_j$, then re-define $\mathfrak{h}$ and $\mathfrak{C}$ so that $\widetilde{V}_{n_k}(f) \le \overline{V}_k$ for $k = 1, \ldots, j$. Otherwise, proceed.

Step 2 If $(b-a)^2 \overline{V}_j \le 8n_j^2 \varepsilon$, then return $T_{n_j}(f)$ as the answer.

Step 3 Otherwise, increase the number of trapezoids to $n_{j+1} = \max(2, m) n_j$, where

$$m = \min\{r \in \mathbb{N} : \eta(r n_j) \widetilde{V}_{n_j}(f) \le \varepsilon\},$$

$$\text{with} \quad \eta(n) := \frac{(b-a)^2 \mathfrak{C}(2(b-a)/n)}{8n^2}, \tag{19}$$

increase j by one, and go to Step 1.

Theorem 3. *Algorithm* `integral` *is successful, i.e.,*

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

Although, in practice we may be unable to be sure that our particular integrand is in the cone $\mathcal{C}$, Step 1 does check a necessary condition. We expect the default values of $\mathfrak{C}$ and $\mathfrak{h}$ to be chosen such that this check fails only rarely in practice.

If $\mathfrak{C}$ takes the form suggested in the explanation following the definition of $\mathcal{C}$ in (15), and $\mathfrak{C}(0)$ is fixed, then the only tuning parameter left to fix is $\mathfrak{h}$, which corresponds to the horizontal scale of the integrand. A small value of $\mathfrak{h}$ would be required for `integral` to handle f_{spiky} in Fig. 1(d). In contrast, the tuning parameter for `ballint`, namely σ, may depend on both the vertical scale and horizontal scale of the integrands of interest. A large value of σ would be required for `ballint` to handle either f_{big} in Fig. 1(b) or f_{spiky} in Fig. 1(d). Thus, we would claim that it is harder to provide a reasonable default value for σ in `ballint` than for $\mathfrak{h}$ in `integral`.

8. COMPUTATIONAL COST OF `integral`. Besides knowing when `integral` is successful, we want to understand its computational cost, which corresponds to the number of trapezoids required plus one. Theorem 4 provides an upper bound on the computational cost of `integral`. Theorem 5 provides a lower bound on the computational cost of any successful algorithm for integrands lying in $\mathcal{C}$. Because these two bounds are asymptotically equivalent, we know that `integral` is efficient.

Theorem 4. *Let $N(f,\varepsilon)$ denote the final number of trapezoids that is required by* `integral`(f,a,b,ε). *Then this number is bounded below and above in terms of the true, yet unknown,* $\operatorname{Var}(f')$.

$$\begin{aligned}
&\max\left(\left\lfloor \frac{2(b-a)}{\mathfrak{h}} \right\rfloor + 1, \left\lceil (b-a)\sqrt{\frac{\operatorname{Var}(f')}{8\varepsilon}} \right\rceil\right) \le N(f,\varepsilon) \\
&\qquad \le 2\min\left\{ n \in \mathbb{N} : n \ge \left\lfloor \frac{2(b-a)}{\mathfrak{h}} \right\rfloor + 1,\ \eta(n)\operatorname{Var}(f') \le \varepsilon \right\} \\
&\qquad \le 2 \min_{0<\alpha\le 1} \max\left(\left\lfloor \frac{2(b-a)}{\alpha\mathfrak{h}} \right\rfloor + 1, \left\lceil (b-a)\sqrt{\frac{\mathfrak{C}(\alpha\mathfrak{h})\operatorname{Var}(f')}{8\varepsilon}} \right\rceil\right). \quad (20)
\end{aligned}$$

The number of function values required by `integral`(f,a,b,ε) *is* $N(f,\varepsilon)+1$.

Proof. No matter what inputs f and ε are provided, the number of trapezoids must be at least $n_1 = \lfloor 2(b-a)/\mathfrak{h} \rfloor + 1$. Then the number of trapezoids is increased until $(b-a)^2\overline{V}_j \le 8n_j^2\varepsilon$, which by (18) implies that $\overline{\operatorname{err}}(f,n) \le \varepsilon$. This implies the lower bound on $N(f,\varepsilon)$

Let J be the value of j for which Algorithm `integral` terminates. Since n_1 satisfies the upper bound, we may assume that $J \ge 2$. Let m be the integer found in Step 3, and let $m^* = \max(2,m)$. Note that $\eta((m^*-1)n_{J-1})\operatorname{Var}(f') > \varepsilon$. For $m^* = 2$ this follows because

$$\eta(n_{J-1})\operatorname{Var}(f') \ge \frac{(b-a)^2\mathfrak{C}(2(b-a)/n_{J-1})\widetilde{V}_{n_{J-1}}(f)}{8n_{j-1}^2}$$

$$\geq \frac{(b-a)^2 \overline{V}_{J-1}(f)}{8n_{j-1}^2} > \varepsilon.$$

For $m^* = m > 2$ this follows by the definition of m in Step 3. Since η is a decreasing function, this implies that

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

Thus, $n_J = m^* n_{J-1} < [m^*/(m^*-1)]n^* \leq 2n^*$, which corresponds to the first part of the upper bound in (20).

To establish the second part of the upper bound, we show that

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

For fixed $\alpha \in (0,1]$, we need only consider the case where $n^* > \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor + 1$. This implies that $n^* - 1 \geq \lfloor 2(b-a)/(\alpha\mathfrak{h})\rfloor + 1 > 2(b-a)/(\alpha\mathfrak{h})$. The definition of n^* and η and the non-decreasing nature of $\mathfrak{C}$ then imply that

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

which completes the proof of the upper bound on n^*. ∎

Clancy et al. (2014) derived an adaptive trapezoidal rule algorithm for integration on $[0,1]$ that is similar to our `integral`. Clancy et al.'s algorithm is guaranteed for integrands in the cone $\widehat{\mathcal{C}} := \{f \in \mathcal{V} : \mathrm{Var}(f') \leq \tau \int_0^1 |f'(x) - f(1) + f(0)|\,\mathrm{d}x\}$. This cone contains f for which $\mathrm{Var}(f')$ is bounded above by τ times a *weaker* semi-norm of f. Here, $1/\tau$ is similar to our $\mathfrak{h}$ and represents a length scale roughly corresponding to the narrowest spike that the algorithm can handle successfully. The disadvantage of Clancy et al.'s algorithm is that the computational cost is bounded above by $\sqrt{\tau\,\mathrm{Var}(f')/(4\varepsilon)} + \tau + 4$. There is a multiplicative factor of $\sqrt{\tau}$ in this cost that becomes large as one tries to accommodate increasingly spiky integrands. Similarly, `ballint` has a multiplicative factor of $\sqrt{\sigma}$. In contrast, in our `integral` the effect of decreasing $\mathfrak{h}$ to accommodate spikier integrands increases the minimum number of nodes needed, but is minor for large $\mathrm{Var}(f')/\varepsilon$ because $\mathfrak{C}(\alpha\mathfrak{h}) \downarrow \mathfrak{C}(0)$ as $\mathfrak{h} \to 0$.

Not only is it important to understand the maximum possible computational cost of `integral`, but it is also desirable to know whether this cost is optimal among all possible algorithms utilizing function values. The optimality of `integral` may be demonstrated by an argument similar to the proof of Theorem 2.

Theorem 5. *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*

$$-\frac{3}{2} + (b-a-3\mathfrak{h})\sqrt{\frac{[\mathfrak{C}(0)-1]\,\mathrm{Var}(f')}{16\varepsilon}} \tag{21}$$

function values. As $\mathrm{Var}(f')/\varepsilon \to \infty$ *the asymptotic rate of increase is the same as the computational cost of* `integral`*, provided* $\mathfrak{C}(0) > 1$.

Proof. For any positive α, suppose that $\texttt{int}(\cdot, a, b, \varepsilon)$ evaluates the triangle peak shaped integrand $\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})$ at n nodes before returning an answer. Let $\{x_i\}_{i=1}^m$ be the $m \le n$ ordered nodes used by $\texttt{int}(\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h}), a, b, \varepsilon)$ that fall in the interval (x_0, x_{m+1}), where $x_0 := a + 3\mathfrak{h}$, $x_{m+1} := b - h$, and $h := [b - a - 3\mathfrak{h}]/(2n+3)$. There must be at least one of these x_i with $i = 0, \ldots, m$ for which

$$\frac{x_{i+1} - x_i}{2} \ge \frac{x_{m+1} - x_0}{2(m+1)} \ge \frac{x_{m+1} - x_0}{2n+2} = \frac{b-a-3\mathfrak{h}-h}{2n+2} = \frac{b-a-3\mathfrak{h}}{2n+3} = h.$$

Choose one such x_i, and call it t. The choice of t and h ensure that $\texttt{int}(\cdot, a, b, \varepsilon)$ cannot distinguish between $\alpha\,\mathrm{twopk}(\cdot; t, h, \pm)$ and $\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})$. Thus,

$$\texttt{int}(\alpha\,\mathrm{twopk}(\cdot; t, h, \pm), a, b, \varepsilon) = \texttt{int}(\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h}), a, b, \varepsilon).$$

Moreover, as discussed in the previous section, the functions $\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})$ and $\alpha\,\mathrm{twopk}(\cdot; t, h, \pm)$ all belong to the cone $\mathcal{C}$. This means that `int` is successful for all of these functions. By the definitions of peak in (3) and twopk in (16), it follows that

$$\begin{aligned}
\varepsilon &\ge \frac{1}{2}\left[\left|\int_a^b \alpha\,\mathrm{twopk}(x; t, h, -)\,\mathrm{d}x - \texttt{int}(\alpha\,\mathrm{twopk}(\cdot; t, h, -), a, b, \varepsilon)\right|\right. \\
&\qquad \left. + \left|\int_a^b \alpha\,\mathrm{twopk}(x; t, h, +)\,\mathrm{d}x - \texttt{int}(\alpha\,\mathrm{twopk}(\cdot; t, h, +), a, b, \varepsilon)\right|\right] \\
&\ge \frac{1}{2}\left[\left|\texttt{int}(\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h}), a, b, \varepsilon) - \int_a^b \alpha\,\mathrm{twopk}(x; t, h, -)\,\mathrm{d}x\right|\right. \\
&\qquad \left. + \left|\int_a^b \alpha\,\mathrm{twopk}(x; t, h, +)\,\mathrm{d}x - \texttt{int}(\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h}), a, b, \varepsilon)\right|\right] \\
&\ge \frac{1}{2}\left|\int_a^b \alpha\,\mathrm{twopk}(x; t, h, +)\,\mathrm{d}x - \int_a^b \alpha\,\mathrm{twopk}(x; t, h, -)\,\mathrm{d}x\right| \\
&= \int_a^b \frac{3\alpha[\mathfrak{C}(h)-1]}{4}\,\mathrm{peak}(x; t, h)\,\mathrm{d}x = \frac{3\alpha[\mathfrak{C}(h)-1]h^2}{4} \\
&= \frac{[\mathfrak{C}(h)-1]h^2\,\mathrm{Var}(\alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h}))}{4}.
\end{aligned}$$

Substituting for h in terms of n gives a lower bound on n:

$$2n+3 = \frac{b-a-3\mathfrak{h}}{h} \ge (b-a-3\mathfrak{h})\sqrt{\frac{[\mathfrak{C}(h)-1]\,\mathrm{Var}(\alpha\,\mathrm{peak}'(\cdot; 0, \mathfrak{h}))}{4\varepsilon}}$$

$$\geq (b-a-3\mathfrak{h})\sqrt{\frac{[\mathfrak{C}(0)-1]\,\mathrm{Var}(\alpha\,\mathrm{peak}'(\cdot;0,\mathfrak{h}))}{4\varepsilon}}.$$

Since α is an arbitrary positive number, the value of $\mathrm{Var}(\alpha\,\mathrm{peak}'(\cdot;0,\mathfrak{h}))$ is arbitrary as well.

Finally, compare the upper bound on the computational cost of `integral` in (20) with the lower bound on the computational cost of the best algorithm in (21). Both increase as $\mathcal{O}(\sqrt{\mathrm{Var}(f')/\varepsilon})$ as $\mathrm{Var}(f')/\varepsilon \to \infty$, provided $\mathfrak{C}(0) > 1$. ■

9. DISCUSSION. Numerical computation is an important part of mathematics because it provides us answers when analytic methods cannot. The work presented here challenges both mathematics educators and numerical analysis researchers.

We have found that university students have difficulty combining analytic and numerical methods holistically to solve realistic mathematical problems (Berkaliev et al., 2014). Students tend to work in just one mode. Or, they think that the lack of an analytic formula for the answer means that no answer exists. Our teaching should be designed to correct this. Calculus is a good place to introduce students to the trapezoidal rule for evaluating integrals that do exist, but whose values cannot be expressed in terms of analytic functions.

However, the trapezoidal rule as taught in a calculus course leaves open the question of how large n needs to be, unless the problem is simple enough to be handled by `ballint` (Section 3). Rather than pretend that all realistic problems have feasible upper bounds on $\mathrm{Var}(f')$, we should explain to students that this question is answered by adaptive algorithms, and then point students to a later course or the literature.

When students get to a numerical analysis course that discusses error estimates for quadrature, we would urge instructors *not* to teach $\widehat{\mathrm{err}}(f,n)$ in (8), as is commonly done now. As discussed here in Section 4 and 6 and three decades ago by Lyness (1983), $\widehat{\mathrm{err}}(f,n)$ is flawed. We now have a better alternative, $\widetilde{\mathrm{err}}(f,n)$ in (18), which should be mentioned—if not taught—in beginning numerical analysis courses.

The treatment of numerical integration in this article highlights the gap that exists in the theory and practice of numerical computation for a wide range of problems. There are good basic methods, such as the trapezoidal rule used here. Error bounds of the form

$$\mathrm{err}(f,n) := \|S(f) - A_n(f)\| \leq C(n)\,\|f\|$$

already exist, where S denotes the solution operator (integration in our case), A_n denotes the numerical method (T_n in our case), $\|f\|$ denotes some semi-norm of the input element f ($\mathrm{Var}(f')$ in our case), C is some known function, and n corresponds to the number of steps or nodes. The question begging for an answer is, "How large must n be to ensure that error, $\mathrm{err}(f,n)$ is no greater than the tolerance, ε?"

When $S(f)$ corresponds to evaluation of elementary or special functions, such as $\cos(f)$ or $\mathrm{erf}(f)$ for $f \in \mathbb{R}$, we do not tell our calculators and computer languages how to choose n, the number of terms in a polynomial approximation. That number is chosen invisibly and large enough so that the error is negligible compared to machine accuracy. Theory guarantees that these algorithms work.

But for more complicated problems, such as integration, function approximation, function optimization, and solutions of differential equations, there are not many practical automatic and adaptive algorithms with theoretical guarantees of success. Brent (2013) describes guaranteed algorithms for finding one zero of a function and for finding minima of unimodal functions that date from the early 1970s. He even considers

the challenges of finite precision arithmetic. Wasilkowski and Gao (1992), Plaskota and Wasilkowski (2005), Plaskota et al. (2008), Plaskota and Wasilkowski (2009), and Plaskota et al. (2013) have shown that adaptive algorithms can be successful for integration and function approximation problems where the functions have singularities. Novak (1996) has discussed what a priori knowledge about a mathematical problem may allow adaptive algorithms to be superior to non-adaptive ones.

Guaranteed adaptive multivariate integration algorithms have been derived using Monte Carlo (Hickernell et al., 2014) and quasi-Monte Carlo methods (Hickernell and Jiménez Rugama, 2014; Jiménez Rugama and Hickernell, 2014). Guaranteed adaptive algorithms for univariate function approximation (Clancy et al., 2014) and optimization of multimodal univariate functions (Tong, 2014) have been derived using linear splines. These recent algorithms rely on identifying data-based upper bounds on $\|f\|$ that are valid for f inside certain cones.

There are two reasons why it makes sense to focus on cones of f. Problems that are homogeneous ($S(cf) = cS(f)$ for all $c \in \mathbb{R}$) typically are solved by homogeneous numerical methods, A_n. This makes the true error positively homogeneous ($\operatorname{err}(cf, n) = |c| \operatorname{err}(f, n)$ for all $c \in \mathbb{R}$). Good error bounds, $\widetilde{\operatorname{err}}(f, n)$, also tend to be positively homogeneous, which means that the set of functions for which the error bound is successful, $\{f : \operatorname{err}(f, n) \le \widetilde{\operatorname{err}}(f, n)\}$, must be a cone.

A second reason is that cones need not be convex. It is known that adaptive algorithms have no significant advantage over non-adaptive algorithms for linear problems defined under rather general conditions (Traub et al., 1988, Chap. 4, Corollary 5.2.1). One of these conditions is that the set of input functions be convex. The ball $\mathcal{B}_\sigma$ defined in Theorem 2 is convex, and so adaptive algorithms cannot significantly improve upon the non-adaptive algorithm `ballint`. However, the cone $\mathcal{C}$ defined in (15) is not convex. While $\pm \operatorname{twopk}(\cdot, t, h, \pm) \in \mathcal{C}$, as verified in (16), the convex combination

$$\frac{1}{2} \operatorname{twopk}(\cdot, t, h, +) + \frac{1}{2}[- \operatorname{twopk}(\cdot, t, h, -)] = \frac{3[\mathfrak{C}(h) - 1)]}{4} \operatorname{peak}(\cdot, t, h)$$

lies outside $\mathcal{C}$ for $h < \mathfrak{h}/2$. Since $\mathcal{C}$ is not convex, it is possible for adaption to provide an advantage over non-adaption.

Our challenge to the numerical analysis community is to take the ideas illustrated here and extend them to other problems and more powerful algorithms. The trapezoidal rule is admittedly inefficient if the integrand has a higher degree of smoothness. The arguments used here should have analogs for higher order quadrature methods. Adaptive algorithms for other mathematical problems exist, but they lack theoretical justification. Again, using the ideas presented here we hope that these algorithms can either be justified or replaced by better, guaranteed algorithms.

The scientific computing community has been discussing how to make our numerical computations reproducible (Bailey and Borwein, 2013; Bangerth and Heister, 2014 January; Buckheit and Donoho, 1995; LeVeque, 2013 April; Stodden et al., 2014), i.e., ensuring that one person's computations can be replicated by others. This includes making software readily available, transparent, and easy to use.

We think that the emphasis should be broadened to include *reliable* numerical computations. Reproducing someone else's answer has less value if the answer is wrong. This means that numerical software should also be thoroughly tested, efficiently coded, and come with the theoretical guarantees of success that are so often missing. Choi (2014) has discussed these ideas and an attempt to teach them to computational mathematics graduate students. These students have applied what they have learned to the development of the Guaranteed Automatic Integration Library (GAIL) (Choi et al.,

2013–2014). GAIL is an attempt to develop truly reliable numerical software, and it already includes some of the algorithms mentioned above. A future version should include the new, adaptive algorithm `integral` described here.

10. ACKNOWLEDGEMENTS. The authors are grateful for discussions with a number of colleagues and collaborators. This research was supported in part by grant NSF-DMS-1115392.

References

Bailey, D. H. and J. M. Borwein. 2013. *Set the default to "open": Reproducible science in the computer age*. http://www.huffingtonpost.com/david-h-bailey/set-the-default-to-open-r_b_2635850.html.

Bangerth, W. and T. Heister. 2014 January. *Quo vadis, scientific software*, SIAM News.

Berkaliev, Z., S. Devi, G. E. Fasshauer, F. J. Hickernell, O. Kartal, X. Li, P. McCray, S. Whitney, and J. S. Zawojewski. 2014. *Initiating a programmatic assessment report*, PRIMUS **24**, 403–420.

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

Brent, R. P. 2013. *Algorithms for minimization without derivatives*, Dover Publications, Inc., Mineola, NY. republication of the 1973 edition by Prentice-Hall, Inc.

Buckheit, J. B. and D. L. Donoho. 1995. *Wavelab and reproducible research*, Technical Report 474, Department of Statistics, Standford University.

Burden, R. L. and J. D. Faires. 2010. *Numerical analysis*, Ninth Edition, Cengage Brooks/Cole, Belmont, CA.

Cheney, W. and D. Kincaid. 2013. *Numerical mathematics and computing*, Seventh Edition, Brooks/Cole, Boston.

Choi, S.-C. T. 2014. *MINRES-QLP pack and reliable reproducible research via staunch scientific software*, J. Open Research Software **2**, no. 1, 1–7.

Choi, S.-C. T., Y. Ding, F. J. Hickernell, L. Jiang, Ll. A. Jiménez Rugama, X. Tong, Y. Zhang, and X. Zhou. 2013–2014. *GAIL: Guaranteed Automatic Integration Library (versions 1.0–2.0)*.

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

Hale, N., L. N. Trefethen, and T. A. Driscoll. 2014. *Chebfun version 5*.

Hickernell, F. J., L. Jiang, Y. Liu, and A. B. Owen. 2014. *Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling*, Monte Carlo and quasi-Monte Carlo methods 2012, pp. 105–128.

Hickernell, F. J. and Ll. A. Jiménez Rugama. 2014. *Reliable adaptive cubature using digital sequences*. submitted for publication, arXiv:1410.8615 [math.NA].

Jiménez Rugama, Ll. A. and F. J. Hickernell. 2014. *Adaptive multidimensional integration based on rank-1 lattices*. submitted for publication, arXiv:1411.1966.

LeVeque, R. J. 2013 April. *Top ten reasons to not share your code (and why you should anyway)*, SIAM News. http://www.siam.org/news/news.php?id=2064.

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

Moore, R. E., R. B. Kearfott, and M. J. Cloud. 2009. *Introduction to interval analysis*, Cambridge University Press, Cambridge.

Novak, E. 1996. *On the power of adaption*, J. Complexity **12**, 199–237.

Plaskota, L. and G. W. Wasilkowski. 2005. *Adaption allows efficient integration of functions with unknown singularities*, Numer. Math. **102**, 123–144.

_____. 2009. *The power of adaptive algorithms for functions with singularities*, J. Fixed Point Theory and Applications **6**, 227–248.

Plaskota, L., G. W. Wasilkowski, and Y. Zhao. 2008. *The power of adaption for approximating functions with singularities*, Math. Comput. **77**, 2309–2338.

_____. 2013. *An adaptive algorithm for weighted approximation of singular functions over* $\mathbb{R}$, SIAM J. Numer. Anal. **51**, 1470–1493.

R Development Core Team. 2014. *The R Project for Statistical Computing*.

Rump, S. M. 1999. *INTLAB - INTerval LABoratory*, Developments in Reliable Computing, pp. 77–104. `http://www.ti3.tuhh.de/rump/`.

_____. 2010. *Verification methods: Rigorous results using floating-point arithmetic*, Acta Numer. **19**, 287–449.

Sauer, T. 2012. *Numerical analysis*, Pearson.

Shampine, L. F. 2008. *Vectorized adaptive quadrature in* MATLAB, J. Comput. Appl. Math. **211**, 131–140.

Stodden, V., F. Leisch, and R. D. Peng. 2014. *Implementing reproducible research*, CRC Press, Boca Raton, FL.

The MathWorks, Inc. 2014. *MATLAB 8.4*, Natick, MA.

The Numerical Algorithms Group. 2013. *The NAG library*, Mark 23, Oxford.

Tong, X. 2014. *A guaranteed, adaptive, automatic algorithm for univariate function minimization*, Master's Thesis.

Traub, J. F., G. W. Wasilkowski, and H. Woźniakowski. 1988. *Information-based complexity*, Academic Press, Boston.

Wasilkowski, G. W. and F. Gao. 1992. *On the power of adaptive information for functions with singularities*, Math. Comp. **58**, 285–304.

Wolfram Research Inc. 2014. *Mathematica 10*, Champaign, IL.

FRED J. HICKERNELL received his PhD in mathematics from the Massachusetts Institute of Technology. He held faculty positions in Hong Kong and the United States. His research straddles computational mathematics and statistics.
Department of Applied Mathematics, Illinois Institute of Technology, 10 W. 32th St., Chicago, IL 60616
hickernell@iit.edu

MARTHA RAZO is an undergraduate student at the Illinois Institute of Technology. She has a passion for both mathematical research and teaching.
Department of Applied Mathematics, Illinois Institute of Technology, 10 W. 32th St., Chicago, IL 60616
mrazo@hawk.iit.edu

SUNNY YUN is an undergraduate student at the University of Illinonis at Urbana-Champaign.
???
???