ADAPTIVE QUADRATURE WITH A GENERAL ERROR CRITERION

BY

JIAZHEN LU

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Applied Mathematics
in the Graduate College of the
Illinois Institute of Technology

Approved ____________________________
Advisor

Chicago, Illinois
December 2017

ACKNOWLEDGMENT

This dissertation could not have been written without Dr. Fred Hickernell who not only served as my supervisor but also encouraged and challenged me throughout my academic program. I deeply appreciate his patience and help. Moreover, I would like to acknowledge with gratitude, the support and love of my family - my parents, my son and my husband.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

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. Here we introduce a new adaptive trapezoidal rule algorithm to satisfy a general error criterion, which includes both absolute and relative error tolerances. Then we derive the computational cost of the new algorithm and the complexity of this problem.

CHAPTER 1
INTRODUCTION

There are many definite integrals that cannot be evaluated by paper and pencil. That is why we need numerical algorithms. There is software for such problems and some algorithms are *adaptive*, meaning that they automatically expend the right amount of computational effort needed to provide an approximate solution with an error no greater than the error tolerance. 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.

There is a popular function, `integral`, in Matlab[7], which is a commonly used adaptive quadrature algorithm. It is based on adaptive Gaussian quadrature. The algorithm works recursively on subintervals of the original interval [a,b]. If the values obtained from the 7-point Gauss formula and the Kronrod extension for a total of 15 samples agree to within a specified tolerance, then an extrapolation taken from the two values is accepted as the value for the integral over the subinterval. If not, the algorithm is recursively applied to the two halves of the subinterval. Unfortunately, there is no theoretical proof that this error estimate is valid.

Similarly, there is *chebfun*, whose basis is Chebyshev polynomial interpolation. Chebfun has extensive capabilities for solving problems with linear and nonlinear differential and integral operators, and also includes continuous analogs of linear algebra notions like QR and singular value decomposition. However, there is still no proof of why it is guaranteed to provide the right answer.

Recently, a *reliable* adaptive algorithm `relIntegral` was developed and implemented for approximating

$$I(f) = \int_a^b f(x)\,\mathrm{d}x.$$

It satisfies an absolute error criterion with certainty for integrands lying in the well-chosen cone, $\mathcal{C}$:

$$|I(f) - \texttt{relIntegral}(f, a, b, \varepsilon_{\mathrm{a}})| \le \varepsilon_{\mathrm{a}} \quad \forall f \in \mathcal{C},\ a, b \in \mathbb{R},\ a < b,\ 0 < \varepsilon_{\mathrm{a}}. \tag{1.1}$$

Our goal is to extend this algorithm to satisfy a more general error criterion, i.e, to construct an approximate solution, which satisfies

$$\begin{multline}|I(f) - \texttt{newIntegral}(f, a, b, \varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{r}})| \le \max\left(\varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{r}}\,|I(f)|\,\right), \\ \forall f \in \mathcal{C},\ a, b \in \mathbb{R},\ a < b,\ 0 < \varepsilon_{\mathrm{a}},\ 0 \le \varepsilon_{\mathrm{r}} \le 1, \end{multline} \tag{1.2}$$

Our algorithm will be introduced in Chapter 2. Then we will provide the computational cost of this algorithm in Chapter 3. In Chapter 4, we will discuss the complexity of this problem. There will be some examples in the following chapter. We conclude with a summary and some problems for further research.

CHAPTER 2
ALGORITHM

2.1 Trapezoidal Rule and Its Error Bound

Trapezoidal rule T_n approximates an integral by the sum of the areas of n trapezoids whose heights are function values:

$$T_n(f) := \frac{b-a}{2n}[f(t_0) + 2f(t_1) + \cdots + 2f(t_{n-1}) + f(t_n)],$$
$$t_i = a + \frac{i(b-a)}{n}, \ i = 0, \ldots, n, \ n \in \mathbb{N} := \{1, 2, \ldots\}. \quad (2.1)$$

This 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 $n \to \infty$. We would like to turn T_n into an algorithm that chooses n to ensure that the trapezoidal rule error is small enough. 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 one-sided derivatives of f at x. 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 discountinuities. All integrands considered in this article have derivatives with bounded variation, i.e., they lie in the linear space

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

Let an *n-partition* of $[a, b]$, denoted $\boldsymbol{x}$, be defined as an $(n+1)$-vector whose elements are ordered and include the endpoints of the interval, $a =: x_0 \le x_1 \le \cdots \le x_{n-1} \le x_n := b$. For any such partition define

$$\widehat{V}(f', \boldsymbol{x}, \boldsymbol{z}) := \sum_{i=2}^{n-1} |z_i - z_{i-1}|, \qquad \text{where } z_i \text{ is between } f'(x^-)\text{and } f'(x^+). \quad (2.2)$$

Then the variation of f' may be written as

$$\operatorname{Var}(f') = \sup\{\widehat{V}(f', \boldsymbol{x}, \boldsymbol{z}) : z_i \text{ is between } f'(x^-) \text{and } f'(x^+), \\ \boldsymbol{x} \text{ is an } n\text{-partition and } \ n \in \mathbb{N}\}. \quad (2.3)$$

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

$$\operatorname{peak}(x; t, h) := \begin{cases} h - |x - t - h|, & t \le x < t + 2h \\ 0, & a \le x < t \text{ or } t + 2h \le x \le b, \end{cases} \quad (2.4a)$$

$$\operatorname{peak}'(x; t, h) = \begin{cases} 1, & t \le x < t + h, \\ -1, & t + h \le x < t + 2h, \\ 0, & a \le x < t \text{ or } t + 2h \le x \le b, \end{cases} \quad (2.4b)$$

$$\operatorname{Var}(\operatorname{peak}'(\cdot; t, h)) \le 4 \text{ with equality if } a < t < t + 2h < b, \quad (2.4c)$$

$$\int_a^b \operatorname{peak}(x; t, h) \, \mathrm{d}x = h^2. \quad (2.4d)$$

The true error of the trapezoidal rule,

$$\operatorname{err}(f, n) := \left| \int_a^b f(x) \, \mathrm{d}x - T_n(f) \right|, \quad (2.5)$$

is rarely known in practice, but there exists a rigorous upper bound on the error of the form

$$\operatorname{err}(f, n) \le \frac{(b - a)^2 \operatorname{Var}(f')}{8n^2} =: \overline{\operatorname{err}}(f, n) \qquad n \in \mathbb{N}. \quad (2.6)$$

An error bound involving the stronger norm $\sup_{x \in [a,b]} |f''(x)|$ may be more common in the literature. The trapezoidal rule gives the exact answer for linear integrands. Error bound (2.6) reflects this fact.

2.2 Guaranteed, Adaptive Trapezoidal Algorithm for Absolute Error Tolerance, `relIntegral`

Since $\operatorname{err}(f,n)$ can be bounded in terms of $\operatorname{Var}(f')$, our challenge becomes to how to estimate $\operatorname{Var}(f')$ using given data. Hickernell, Razo, and Yun give us an idea [2]. If an upper bound on $\operatorname{Var}(f')$ is available or can be correctly bounded, we can determine how large n must be to satisfy the error tolerance by the error bound (2.6).

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

$$\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) := \sum_{i=2}^{n-1} |\Delta_i - \Delta_{i-1}|, \qquad \text{where } \Delta_i \text{ is between } f'(x_i^-) \text{ and } f'(x_i^+). \tag{2.7}$$

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:

$$\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) \le \operatorname{Var}(f') \quad \forall f \in \mathcal{V},\ \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1},\ n \in \mathbb{N}. \tag{2.8}$$

The algorithm `relIntegral` 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 $\operatorname{Var}(f')$ by much:

$$\begin{aligned}\mathcal{C} := \{f \in \mathcal{V} : \operatorname{Var}(f') \le \mathfrak{C}(\operatorname{size}(\{x_i\}_{i=0}^n))\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) \text{ for all} \\ \text{choices of } n \in \mathbb{N},\ \{\Delta_i\}_{i=1}^{n-1}, \text{ and } \{x_i\}_{i=0}^n \text{ with } \operatorname{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}\}.\end{aligned} \tag{2.9}$$

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 $\operatorname{size}(\{x_i\}_{i=0}^n) \le \mathfrak{h}$, 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 always lies in $\mathcal{C}$:

$$\mathrm{twopk}(x; t, h, \pm) := \mathrm{peak}(x, 0, \mathfrak{h}) \pm \frac{3[\mathfrak{C}(h) - 1]}{4} \mathrm{peak}(x, t, h),$$
$$a + 3\mathfrak{h} \le t \le b - 3h, \ 0 \le h < \mathfrak{h}, \quad (2.10a)$$

$$\mathrm{Var}(\mathrm{twopk}'(x; t, h, \pm)) = 3 + \frac{3[\mathfrak{C}(h) - 1]}{4} \times 4 = 3\mathfrak{C}(h). \quad (2.10b)$$

From this definition it follows that

$$\begin{aligned}
&\mathfrak{C}(\mathrm{size}(\{x_i\}_{i=0}^n))\widehat{V}(\mathrm{twopk}'(x; t, h, \pm), \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1}) \\
&\quad \ge \begin{cases} \mathfrak{C}(0)\,\mathrm{Var}(\mathrm{twopk}'(x; t, h, \pm)), & 0 \le \mathrm{size}(\{x_i\}_{i=0}^n) < h, \\ \mathfrak{C}(h)\,\mathrm{Var}(\mathrm{peak}'(x; 0, \mathfrak{h})) = 3\mathfrak{C}(h), & h \le \mathrm{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}, \end{cases} \\
&\quad \ge \mathrm{Var}(\mathrm{twopk}'(x; t, h, \pm)), \qquad 0 \le \mathrm{size}(\{x_i\}_{i=0}^n) < \mathfrak{h}. \qquad (2.10c)
\end{aligned}$$

Although $\mathrm{twopk}(\cdot; t, h, \pm)$ may have a peak of arbitrarily small width half-width, h, the height of this peak is small enough so that $\mathrm{twopk}(\cdot; t, h, \pm)$ still lies in $\mathcal{C}$ by (2.10c).

We cannot use $\widehat{V}(f', \{x_i\}_{i=0}^n, \{\Delta_i\}_{i=1}^{n-1})$ to approximate $\mathrm{Var}(f')$ 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 $\mathrm{Var}(f')$, which is the total variation of the derivative of the linear spline approximation to f:

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

For all $f \in \mathcal{C}$ it follows that

$$\widetilde{V}_n(f) \le \operatorname{Var}(f') \le \mathfrak{C}(2(b-a)/n)\widetilde{V}_n(f) \tag{2.11}$$

for $n > 2(b-a)/\mathfrak{h}$.

Algorithm `relIntegral` 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 (2.11) 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 (2.6) to provide a data-based upper bound on the trapezoidal rule error:

$$\operatorname{err}(f, n_j) \le \overline{\operatorname{err}}(f, n_j) = \frac{(b-a)^2 \operatorname{Var}(f')}{8n_j^2} \le \frac{(b-a)^2\overline{V}_j}{8n_j^2} := \varepsilon_{n_j}. \tag{2.12}$$

This is the crux of their guaranteed adaptive trapezoidal rule, `relIntegral` [2], which is guaranteed to find an answer within the error tolerance for integrands in $\mathcal{C}$. For the absolute error criterion, `relIntegral` increases n until $\varepsilon_{n_j} \le \varepsilon_{\mathrm{a}}$.

2.3 Guaranteed, Adaptive Trapezoidal Algorithm for a Hybrid Error Tolerance `newIntegral`

For relative error, the situation is more complicated because there is $I(f)$ in the tolerance term. So we need to find a way to bound or estimate $I(f)$. Enlightened by the general error criterion in [3], we can extend `relIntegral` to meet more general error criterion.

Algorithm `newIntegral` Given an interval, $[a, b]$, an inflation function, $\mathfrak{C}$, a positive key mesh size, $\mathfrak{h}$, a positive error tolerance, ε_{a}, a relative error tolerance. ε_{r}, 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 $k = 1, \dots, j$. Otherwise, proceed.

Step 2 Compute $T_{n_j}, \varepsilon_{n_j} = \frac{(b-a)^2 \overline{V}_j}{8n_j^2}$, check $\Delta_{n_j} \ge \varepsilon_{n_j}$, where

$$\Delta_{n_j} = \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r|T_{n_j} + \varepsilon_{n_j}|) + \max(\varepsilon_a, \varepsilon_r|T_{n_j} - \varepsilon_{n_j}|)]. \tag{2.13}$$

If satisfies, then return

$$\hat{I}(f) = \frac{(T_{n_j} - \varepsilon_{n_j})\max(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{r}\left|T_{n_j} + \varepsilon_{n_j}\right|) + (T_{n_j} + \varepsilon_{n_j})\max(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{r}\left|T_{n_j} - \varepsilon_{n_j}\right|)}{\max(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{r}\left|T_{n_j} + \varepsilon_{n_j}\right|) + \max(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{r}\left|T_{n_j} - \varepsilon_{n_j}\right|)} \tag{2.14}$$

as the answer.

Step 3 Otherwise, increase the number of trapezoids to $n_{j+1} = 2n_j$, and go to Step 1.

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. In Step 2, instead of checking ε_{n_j} is less than absolute error tolerance, we introduce a new item Δ_{n_j} to make the `newIntegral` to satisfy a hybrid error criterion.

Next, we want to show that algorithm `newIntegral` is successful.

Theorem 1. *T_{n_j} is the result of trapezoidal rule that satisfy the error bound $|I(f) - T_{n_j}| \le \varepsilon_{n_j}$, and the approximation $\hat{I}(f)$ is defined as (2.14). If the integrand is in the cone $\mathcal{C}$, and ε_{n_j} satisfies*

$$\varepsilon_{n_j} \le \Delta_{n_j} \tag{2.15}$$

then the algorithm `newIntegral` *is successful, i.e.,*

$$\left|I(f)-\hat{I}(f)\right| \le \max\left(\varepsilon_a, \varepsilon_r\,|I(f)|\,\right), \forall f \in \mathcal{C},\ 0<\varepsilon_a,\ 0\le\varepsilon_r\le 1.$$

Proof. Since $\left|I(f)-T_{n_j}(f)\right| \le \overline{\mathrm{err}}(f,n_j) \le \varepsilon_{n_j}$, we are certain that $I(f)\in[T_{n_j}-\varepsilon_{n_j}, T_{n_j}+\varepsilon_{n_j}]$. Let $\Delta_- = \frac{1}{2}\big(\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|)-\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)\big)$, then from (2.14) and triangle inequality, we have

$$\begin{aligned}
&\hat{I}(f)\big(\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)+\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|)\big) = \\
&\quad (T_{n_j}-\varepsilon_{n_j})\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)+(T_{n_j}+\varepsilon_{n_j})\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|) \\
&\Rightarrow 2\hat{I}(f)\Delta_{n_j} = 2T_{n_j}\Delta_{n_j}+\varepsilon_{n_j}\big(\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|)-\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)\big) \\
&\Rightarrow 2T_{n_j}+2\varepsilon_{n_j}\Delta_- = 2\hat{I}(f)\Delta_{n_j} \le 2T_{n_j}\Delta_{n_j}+\Delta_{n_j}\big(\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|)-\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right. \\
&\Rightarrow T_{n_j}+\varepsilon_{n_j}\frac{\Delta_-}{\Delta_{n_j}} = \hat{I}(f) \le T_{n_j}-\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)+\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|) \\
&\quad \frac{\Delta_-}{\Delta_{n_j}} = \varepsilon_{n_j}+\frac{-2\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)}{\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}+\varepsilon_{n_j}\right|)+\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|)} \\
&\Rightarrow T_{n_j}+\varepsilon_{n_j}-\frac{2\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|I(f)\right|)}{2} \le \hat{I}(f) \le T_{n_j}-\varepsilon_{n_j}+\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|T_{n_j}-\varepsilon_{n_j}\right|) \\
&\Rightarrow I(f)-\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|I(f)\right|) \le \hat{I(f)} \le I(f)+\max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|I(f)\right|) \\
&\quad \text{since } \alpha-\max(\varepsilon_a,\varepsilon_r|\alpha|) \text{ increase as } \alpha \text{ increases.} \\
&\Leftrightarrow \left|I(f)-\hat{I}(f)\right| \le \max(\varepsilon_\mathrm{a},\varepsilon_\mathrm{r}\left|I(f)\right|)
\end{aligned}$$

Then we prove algorithm `newIntegral` is successful. □

CHAPTER 3
COMPUTATIONAL COST

In addition to knowing when `newIntegral` is successful, we want to understand its computational cost.

Theorem 2. *Let $N(f,\varepsilon_a,\varepsilon_r)$ denote the final number of trapezoids that is required by* `newIntegral`(f,a,b,ε)*. Then this number is bounded below and above in terms of the true, yet unknown,* $\operatorname{Var}(f')$ *and* $I(f)$*.*

$$\max\left(\left\lfloor\frac{2(b-a)}{\mathfrak{h}}\right\rfloor+1, \left\lceil(b-a)\sqrt{\frac{\operatorname{Var}(f')(1-\varepsilon_r)}{8\max(\varepsilon_r|I|,\varepsilon_a)}}\right\rceil\right)\le N(f,\varepsilon_a,\varepsilon_r)$$
$$\le 2\min\left\{n\in\mathbb{N}: n\ge\left\lfloor\frac{2(b-a)}{\mathfrak{h}}\right\rfloor+1, \frac{n^2}{\mathfrak{C}(\frac{2(b-a)}{n})}\le\frac{(b-a)^2(1+\varepsilon_r)\operatorname{Var}(f')}{8\max(\varepsilon_r|I|,\varepsilon_a)}\right\} \quad (3.1)$$

The number of function values required by `newIntegral`$(f,a,b,\varepsilon_a,\varepsilon_r)$ *is* $N(f,\varepsilon_a,\varepsilon_r)+1$*.*

Proof. Let n be the value of n_j when the algorithm terminates. No matter what inputs f and $\varepsilon_a,\varepsilon_r$ 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 $\varepsilon_{n_j}\le\Delta_{n_j}$. We will have lower bound on $N(f,\varepsilon_a,\varepsilon_r)$ based on it.

Consider the possibility that $|T_n|\le\varepsilon_n$ and $\varepsilon_a<\varepsilon_r(\varepsilon_n-|T_n|)$. Then

$$\varepsilon_n\le\Delta_n=\frac{1}{2}[\varepsilon_r(|T_n|+\varepsilon_n)+\varepsilon_r(\varepsilon_n-|T_n|)]=\varepsilon_r\varepsilon_n<\varepsilon_n,$$

since $0\le\varepsilon_r<1$. This is a contradiction, so this situation is impossible. Thus we must have $|T_n| > \varepsilon_n$ or $\varepsilon_a \geq \varepsilon_r \left| |T_n| - \varepsilon_n \right|$,

$$\begin{aligned}
\Delta_n &= \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r(|T_n| + \varepsilon_n)) + \max(\varepsilon_a, \varepsilon_r(|T_n| - \varepsilon_n))] \\
&\leq \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r(|I(f)| + 2\varepsilon_n) + \max(\varepsilon_a, \varepsilon_r|I(f)|)] \\
&\leq \frac{1}{2}[2\max(\varepsilon_a, \varepsilon_r|I(f)|) + 2\varepsilon_r\varepsilon_n] \\
&= \max(\varepsilon_a, \varepsilon_r|I(f)|) + \varepsilon_r\varepsilon_n.
\end{aligned}$$

Moreover,

$$|T_n| - \varepsilon_n \leq |T_n| - |I(f) - T_n| \leq |I(f)| = |I(f) - T_n + T_n \leq |I(f) - T_n| \leq \varepsilon_n + |T_n|. \quad (3.2)$$

Therefore, we have

$$\begin{gathered}
\varepsilon_n \leq \Delta_n \leq \max(\varepsilon_a, \varepsilon_r|I(f)|) + \varepsilon_r\varepsilon_n \\
\frac{(b-a)^2 \operatorname{Var}(f')}{8n^2} \leq \frac{(b-a)^2 \overline{V}_j}{8n^2} = \varepsilon_{n_j} \leq \frac{\max(\varepsilon_a, \varepsilon_r|I|)}{1 - \varepsilon_r} \\
(b-a)\sqrt{\frac{\operatorname{Var}(f')(1-\varepsilon_r)}{8\max(\varepsilon_r|I(f)|, \varepsilon_a)}} \leq n = N(f, \varepsilon_r, \varepsilon_a)
\end{gathered}$$

which completes the proof of the lower bound on n.

Next, we prove the upper bound on the computational cost. Let $\tilde{n} = n/2$, where n is the value of n_j for which Algorithm `newIntegral` terminates. Since n_1 satisfies the upper bound, we may assume that $\tilde{n}$ exists. Note that $\varepsilon_{\tilde{n}} = \frac{(b-a)^2 \overline{V}_{\tilde{n}}}{8\tilde{n}^2} > \Delta_n$. Since

$$\begin{aligned}
\varepsilon_{\tilde{n}} > \Delta_{\tilde{n}} &= \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r(|T_{\tilde{n}}| + \varepsilon_{\tilde{n}})) + \max(\varepsilon_a, \varepsilon_r(|T_{\tilde{n}}| - \varepsilon_{\tilde{n}})] \\
&\geq \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r(|T_{\tilde{n}} + \varepsilon_{\tilde{n}}|) + \max(\varepsilon_a, \varepsilon_r(|T_{\tilde{n}}| - \varepsilon_{\tilde{n}}))] \\
&\geq \frac{1}{2}[\max(\varepsilon_a, \varepsilon_r|I(f)|) + \max(\varepsilon_a, \varepsilon_r(|I(f)| - 2\varepsilon_{\tilde{n}}))] \text{ by (3.2)} \\
&\geq \frac{1}{2}[2\max(\varepsilon_a, \varepsilon_r|I(f)|) - 2\varepsilon_r\varepsilon_{\tilde{n}}] \\
&= \max(\varepsilon_a, \varepsilon_r|I(f)|) - \varepsilon_r\varepsilon_{\tilde{n}}
\end{aligned}$$

Therefore, we have

$$
\begin{aligned}
&\varepsilon_{\tilde{n}} > \frac{\max(\varepsilon_a, \varepsilon_r|I(f)|)}{1+\varepsilon_r} \text{ and } \varepsilon_{\tilde{n}} = \frac{(b-a)^2\overline{V}_{\tilde{n}}}{8\tilde{n}^2},\\
&\text{so } \tilde{n}^2 < \frac{(b-a)^2\overline{V}_{\tilde{n}}(1+\varepsilon_r)}{8\max(\varepsilon_a, \varepsilon_r|I(f)|)} \le \min_{k=1,\dots,j} \mathfrak{C}\left(\frac{2(b-a)}{n_k}\right)\operatorname{Var}(f') \le \mathfrak{C}\left(\frac{2(b-a)}{n}\right)\operatorname{Var}(f')\\
&\Rightarrow \frac{\tilde{n}^2}{\mathfrak{C}\left(\frac{2(b-a)}{\tilde{n}}\right)} < \frac{(b-a)^2(1+\varepsilon_r)\operatorname{Var}(f')}{8\max(\varepsilon_r|I(f)|, \varepsilon_a)}
\end{aligned}
\tag{3.3}
$$

Recall $n = 2\tilde{n}, N(f, \varepsilon_a, \varepsilon_r) = n = 2\tilde{n}$. Pluging it into (3.3) completes the proof of the upper bound on the computational cost. $\square$

CHAPTER 4
COMPLEXITY

Not only is it important to understand the maximum possible computational cost of `newIntegral`, but it is also desirable to know whether this cost is optimal among all possible algorithms utilizing function values.

Theorem 3. *Let* `int` *be any (possibly adaptive) algorithm that succeeds for all integrands in $\mathcal{C}$, and only uses function values. For any error tolerance $\varepsilon_a \geq 0, \varepsilon_a\varepsilon_r > 0$ and any arbitrary value of* $\mathrm{Var}(f')$*, there will be some $f \in \mathcal{C}$ for which* `int` *must use at least*

$$\min\left\{(b-a)\sqrt{\frac{\mathrm{var}(f')}{4\max(\varepsilon_a, \varepsilon_r|I|)}}, \frac{(b-a-3\mathfrak{h})g\left(\frac{\mathrm{var}(f')}{4\max(\varepsilon_a,\varepsilon_r|\mu|)}\right)-3}{2}\right\} \tag{4.1}$$

function values and $g(x) = \min\left\{y \in [\frac{1}{\delta\mathfrak{h}}, \infty) : \frac{\mathfrak{C}(1/y)y^2}{\mathfrak{C}(1/y)-1} \geq x\right\}$.

Proof. Defind the double peaked function which always lies in $\mathcal{C}$:

$$\mathrm{twopk}(x;t,h,\pm) := \mathrm{peak}(x,0,\mathfrak{h}) \pm \frac{3[\mathfrak{C}(h)-1]}{4}\mathrm{peak}(x,t,h),$$
$$a+3\mathfrak{h} \leq t \leq b-3h,\ 0 \leq h < \mathfrak{h}, \tag{4.2}$$

$$\mathrm{Var}(\mathrm{twopk}'(x;t,h,\pm)) = 3 + \frac{3[\mathfrak{C}(h)-1]}{4} \times 4 = 3\mathfrak{C}(h). \tag{4.3}$$

$$\int_a^b \mathrm{twopk}(x;t,h,\pm)\,\mathrm{d}x = \frac{\mathfrak{h}^2}{2} \pm \frac{3[\mathfrak{C}(h)-1]h^2}{8} \tag{4.4}$$

For any β and any positive α, suppose that $\mathtt{int}(\cdot, a, b, \varepsilon_a, \varepsilon_r)$ evaluates the triangle peak shaped integrand $\beta+\alpha\,\mathrm{peak}(\cdot;0,\mathfrak{h})$ at n nodes before returning an answer. Let $\{x_i\}_{i=1}^m$ be the $m \leq n$ ordered nodes used by $\mathtt{int}(\beta+\alpha\,\mathrm{peak}(\cdot;a,\mathfrak{h}), a, b, \varepsilon_a, \varepsilon_r)$ that fall in the interval (x_0, x_{m+1}), where $x_0 := a+3\mathfrak{h}, x_{m+1} := b-h$, and $h :=$

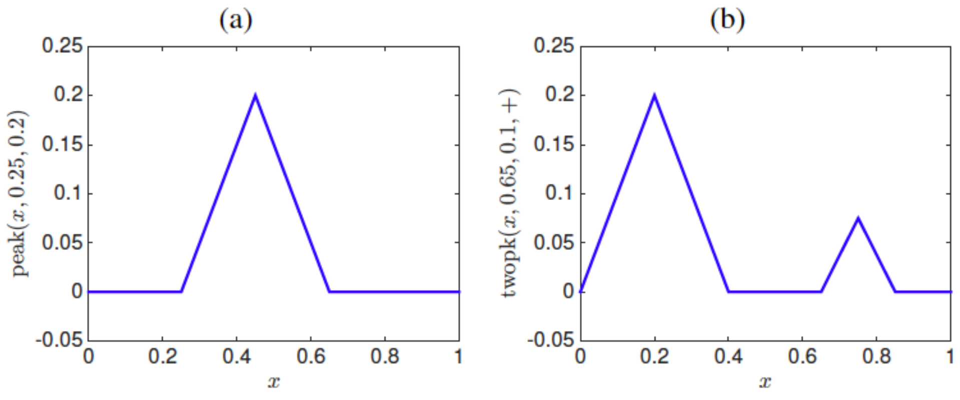

Figure 4.1. (a)single peak function as defined in (2.4), (b)double-peak function as defined in (4.2)

$\min\left(\delta\mathfrak{h}, [b-a-3\mathfrak{h}]/(2n+3)\right), 0 < \delta < 1$. There must be at least one of these x_i with $i = 1, ..., m$ for which

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

Choose such x_i, and call it t. The choice of t and h ensure that $\texttt{int}(\cdot, a, b, \varepsilon_a, \varepsilon_r)$ cannot distinguish between $\beta + \alpha\,\mathrm{twopk}(\cdot; t, h, \pm)$ and $\beta + \alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})$ which are defined as . Thus,

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

Note that

$I(\beta + \alpha\,\mathrm{twopk}(\cdot; t, h, +)) > I(\beta + \alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})) > I(\beta + \alpha\,\mathrm{twopk}(\cdot; t, h, -))$,

and

$$\overline{V}_j(\beta + \alpha\,\mathrm{twopk}(\cdot; t, h, +)) = \overline{V}_j(\beta + \alpha\,\mathrm{peak}(\cdot; 0, \mathfrak{h})) = \overline{V}_j(\beta + \alpha\,\mathrm{twopk}(\cdot; t, h, -)).$$

Then due to triangle inequality we have:

$$
\begin{aligned}
&\max\big(\varepsilon_a, \varepsilon_r\,|I(\beta+\alpha\,\mathrm{peak}(\cdot;0,\mathfrak{h}))|\big)\\
&\geq \frac{1}{2}\big[\max\big(\varepsilon_a, \varepsilon_r\,|I(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,-))|\big)+\max\big(\varepsilon_a, \varepsilon_r\,|I(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,+))|\big)\big]\\
&\geq \frac{1}{2}\bigg[\bigg|\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,-)dx-\mathtt{int}(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,-),a,b,\varepsilon_a,\varepsilon_r)\bigg|\\
&+\bigg|\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,+)dx-\mathtt{int}(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,+),a,b,\varepsilon_a,\varepsilon_r)\bigg|\bigg]\\
&\geq \frac{1}{2}\bigg[\bigg|\mathtt{int}(\beta+\alpha\,\mathrm{peak}(\cdot;0,\mathfrak{h}),a,b,\varepsilon_a,\varepsilon_r)-\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,-)dx\bigg|\\
&+\bigg|\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,+)dx-\mathtt{int}(\beta+\alpha\,\mathrm{peak}(\cdot;0,\mathfrak{h}),a,b,\varepsilon_a,\varepsilon_r)\bigg|\bigg]\\
&\geq \frac{1}{2}\bigg|\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,+)dx-\int_a^b \beta+\alpha\,\mathrm{twopk}(x;t,h,-)dx\bigg|\\
&=\int_a^b \frac{3\alpha[\mathfrak{C}(h)-1]}{4}\,\mathrm{peak}(x;t,h)dx,
\end{aligned}
$$

by (4.3), it equals:

$$
\begin{aligned}
\frac{3\alpha[\mathfrak{C}(h)-1]h^2}{4}&=\frac{3\alpha[\mathfrak{C}(h)-1]h^2}{4}\cdot\frac{\mathrm{Var}(\alpha\,\mathrm{twopk}'(\cdot;t,h.+))}{3\alpha\mathfrak{C}(h)}\\
&=\big[\frac{\mathfrak{C}(h)-1}{\mathfrak{C}(h)}\big]\frac{h^2\,\mathrm{Var}(\alpha\,\mathrm{twopk}'(\cdot;t,h.+))}{4}
\end{aligned}
$$

Then it follows

$$
\frac{\mathfrak{C}(h)}{(\mathfrak{C}(h)-1)h^2}\geq\frac{\mathrm{Var}(\alpha\,\mathrm{twopk}'(\cdot;t,h.+))}{4\max(\varepsilon_a,\varepsilon_r|I(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,+))|)} \tag{4.5}
$$

Let $\mu_+ = I(\beta+\alpha\,\mathrm{twopk}(\cdot;t,h,+))$ and $\sigma=\mathrm{Var}(\alpha\,\mathrm{twopk}'(\cdot;t,h.+))$, so (4.5) becomes

$\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}$. Then let $g(x) = \min\left\{y \in [\frac{1}{\delta\mathfrak{h}}, \infty) : \frac{\mathfrak{C}(1/y)y^2}{\mathfrak{C}(1/y)-1} \geq x\right\}$,we have

$$\begin{aligned}
&\frac{1}{h} \geq g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) \geq \frac{1}{\delta\mathfrak{h}} \\
\Rightarrow & h \leq \frac{1}{g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right)} \leq \delta\mathfrak{h} \\
\Rightarrow & \min\left(\delta\mathfrak{h}, \frac{b-a-3\mathfrak{h}}{2n+3}\right) \leq \frac{1}{g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right)} \leq \delta\mathfrak{h} \\
\Rightarrow & \max\left(\frac{1}{\delta\mathfrak{h}}, \frac{2n+3}{b-a-3\mathfrak{h}}\right) \geq g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) \geq \frac{1}{\delta\mathfrak{h}} \\
\Rightarrow & \frac{2n+3}{b-a-3\mathfrak{h}} \geq g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) \\
& \text{or } g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) = \delta\mathfrak{h} \\
\Rightarrow & 2n+3 \geq (b-a-3\mathfrak{h})g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) \\
& \text{or } \frac{\mathfrak{C}(\mathfrak{h})}{\mathfrak{h}^2(\mathfrak{C}(\mathfrak{h})-1)} = \frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}
\end{aligned}$$

Since $\mathfrak{h} \geq \frac{b-a}{n}$, then

$$\begin{aligned}
n &\geq \frac{(b-a-3\mathfrak{h})g\left(\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}\right) - 3}{2} \\
\text{or } n &\geq (b-a)\sqrt{\frac{(\mathfrak{C}(\mathfrak{h})-1)\sigma}{4\mathfrak{C}(\mathfrak{h})\max(\varepsilon_a,\varepsilon_r|\mu_+|)}} \geq (b-a)\sqrt{\frac{\sigma}{4\max(\varepsilon_a,\varepsilon_r|\mu_+|)}}
\end{aligned}$$

Since α, β are arbitrary number, the value of σ, μ are arbitrary as well. ‘ □

CHAPTER 5
NUMERICAL TEST

L. E. Shampine give some examples integration for test numerical algorithms in [1]. Our new algorithm `newIntegral` has been implemented into GAIL [6] and tested with two functions in [1]. Then compare the results with function integral in MATLAB.

The first test function is a fuction family has an oscillatory integrand

$$\int_0^1 1 + \cos(a\pi x)dx$$

which we solve for 50 values of a randomly generated from $\frac{1}{3}$ to $83 + \frac{1}{3}$ following uniform distribution. The absolute error ε_{a} is set up from 10^{-1}, up to 10^{-9}, and ε_{r} is hold at 5×10^{-5}.

The result are shown in Table 5, where the MATLAB means the integral algorithm in MATLAB and `newIntegral` means our algorithm.

Table 5.1. Success rate of the oscillatory integrand

ε_{a}	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{8}	10^{-9}
`integral`	50	50	50	50	50	50	50	50	50
`newIntegral`	50	50	50	50	50	50	50	50	50

From the test result, we can see our `newIntegral` works as well as the Integral in MATLAB, which is based on Gaussian quadrature. We expect it will have better performance than our `newIntegral` since it is a higher order method. It turned out up to 10^{-9} tolerance, they both succeed and no failures.

Next, we try another family of problems involving a parameter,

$$\int_0^1 |x-\alpha| + |x-(\alpha+h)|\,dx,$$

where α is uniform random number from h to $1-h$. This famly of functions have a big flat part. We want to see these two quadrature can catch it. In this test the absolute error ε_a is held fixed at 10^{-6}, and relative error ε_r is set to 5×10^{-6}.

Table 5.2. Tests Result of Function with a Flat Line

	$h=0.1$		$h=0.01$	
	succeess rate	failure rate	succeess rate	failure rate
`Integral`	95%	5%	92%	8%
`newIntegral`	100%	0%	100%	0%

From Table 5.2, the failure rate of the `Integral` in MATLAB is 5 % . On the other hand, our algorithm has 0 % failure rate. Since h is about 8% of the interval, missing it is not a small error to the final answer. Moreover, when h gets smaller, there is more chance `Integral` miss the flat part. The failure rate increases to 8% when h is 0.01. While our algorithm is still guaranteed to provide a numerical approximation to the true integral with an error no greater than tolerance.

CHAPTER 6
CONCLUSION

Numerical computation is an important part of mathrmastics because it provides us answers when analytic methods cannot. Trapezoidal rule works for evaluationg integrals that do exist, but whose values cannot be expressed in terms of analytic functions. In addition to that, 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.

In [2], they solve the problem "How large must n be to ensure that error, $\mathrm{err}(f, n)$ is no greater than the tolerance, ε_a?" Instead of telling 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. Now we extend this algorithm to satisfy a more general error criteria.

As demonstrated in Chapter 2, it is now possible to set relative error criteria or hybrid error criteria. Rather than pretend that all realistic problems have upper bound on $\mathrm{Var}(f')$, we can solve it by our adaptive quadrature algorithms.

There are several problems we want to work in the future. Now the algorithm is global adaptive, which means all the nodes are chosen equally spaced. It will be more efficient if local adaptive. More nodes on the spiky part and less on the plat line. Moreover, extend this idea to more more powerful algorithms. Instead of Trapezoidal rule, we can use higher order quadrature methods for better guaranteed algorithms.

APPENDIX A

MATLAB CODE

```
function [q,out_param] = integralNoPenalty1_g(varargin)
%INTEGRAL_G 1-D guaranteed function integration using trapezoidal rule
%
%   q = INTEGRAL_G(f) computes q, the definite integral of function f
%   on the interval [a,b] by trapezoidal rule with
%   in a guaranteed absolute error of 1e-6. Default starting number of
%   sample points taken is 100 and default cost budget is 1e7. Input f is
%   function handle. The function y = f(x) should accept a vector argument
%   x and return a vector result y, the integrand evaluated at each elemen
%   of x.
%
%   q = INTEGRAL_G(f,a,b,abstol) computes q, the definite
%   integral of function f on the finite interval [a,b] by trapezoidal rul
%   with the ordered input parameters, and guaranteed absolute error tolera
%   abstol.
%
%   q = INTEGRAL_G(f,'a',a,'b',b,'abstol',abstol)
%   computes q, the definite integral of function f on the finite interval
%   [a,b] by trapezoidal rule within a guaranteed absolute error tolerance
%   abstol.
%   All four field-value pairs are optional and can be supplied.
%
%   q = INTEGRAL_G(f,in_param) computes q, the definite integral of
%   function f by trapezoidal rule within a guaranteed absolute error
%   in_param.abstol. If a field is not specified, the default value is
%   used.
%
%   [q, out_param] = INTEGRAL_G(f,...) returns the approximated
%   integration q and output structure out_param.
%
%   Input Arguments
%
%
%     f --- input function
%
%     in_param.a --- left end of the integral, default value is 0
%
%     in_param.b --- right end of the integral, default value is 1
%
%     in_param.abstol --- guaranteed absolute error tolerance, default valu
%     is 1e-6
%
%     in_param.reltol --- relative error tolerance,default value is 5e-3
%
%   Opitional Input Arguments (Recommended not to change very often)
%
%
%     in_param.nlo --- lowest initial number of function values used, defa
%     value is 10
%
%     in_param.nhi --- highest initial number of function values used,
%     default value is 1000
%
%     in_param.nmax --- cost budget (maximum number of function values),
%     default value is 1e7
%
```

```
%    in_param.maxiter --- max number of iterations, default value is 1000
%
%  Output Arguments
%
%    q --- approximated integral
%
%    out_param.f --- input function
%
%    out_param.a --- low end of the integral
%
%    out_param.b --- high end of the integral
%
%    out_param.abstol --- guaranteed absolute error tolerance
%
%    out_param.reltol --- relative error tolerance
%
%    out_param.nlo --- lowest initial number of function values
%
%    out_param.nhi --- highest initial number of function values
%
%    out_param.nmax --- cost budget (maximum number of function values)
%
%    out_param.maxiter --- max number of iterations
%
%    out_param.ninit --- initial number of points we use, computed by nlo
%    and nhi
%
%    out_param.exceedbudget --- it is true if the algorithm tries to use
%        more points than cost budget, false otherwise.
%
%       out_param.tauchange --- it is true if the cone constant has been
%       changed, false otherwise. See [1] for details. If true, you may wish
%       change the input in_param.ninit to a larger number.
%
%       out_param.iter --- number of iterations
%
%       out_param.npoints --- number of points we need to
%       reach the guaranteed error tolerance.
%
%       out_param.errest --- approximation error defined as the differences
%       between the true value and the approximated value of the integral.
%
%       out_param.nstar --- final value of the parameter defining the cone o
%       functions for which this algorithm is guaranteed; nstar = ninit-2
%       initially and is increased as necessary
%

% check parameter satisfy conditions or not
[f,out_param, flip] = integral_g_param(varargin{:});

%% main alg
out_param.tau=ceil((out_param.ninit-1)*2-1); % computes the minimum number
out_param.exceedbudget=false;    % if the number of points used in the calc
out_param.conechange=false;   % if the cone constant has been changed
ntrap=out_param.ninit-1; % number of trapezoids
intervallen=out_param.b-out_param.a; % length of integration interval (>=0
Varfp=zeros(10,1); %initialize vector of approximations to Var(f')
Varfpup=[inf; zeros(10,1)]; %initialize vector of upper bounds to Var(f')
ii=1; %index

if intervallen>0
    steplen=intervallen/ntrap; % length of subinterval
    steplenvec=zeros(10,1); %vector to record subinterval lengths
    hcut=2*intervallen/(ntrap-1); %minimum interval size
    inflatelim=1.5;
    xpts=(out_param.a:steplen:out_param.b)'; % generate ninit uniformly sp
    fpts=f(xpts);    % get function values at xpts
    sumf=(fpts(1)+fpts(out_param.ninit))/2+sum(fpts(2:ntrap)); % computes
    while true
        steplenvec(ii)=steplen;

        %Compute approximation to Var(f')
        Varfp(ii)=sum(abs(diff(fpts,2)))/steplen; %approx Var(f')
        Varfpup(ii+1)=min(Varfpup(ii),Varfp(ii)*inflatelim*hcut/(hcut-2*ste
            %update upper bound on Var(f')

        %Check necessary condition for integrand to lie in cone
        if Varfp(ii) > Varfpup(ii+1) %f lies outside cone
            %Decrease hcut
            tempa=1-inflatelim*Varfp(1:ii)/Varfp(ii);
            whpos=find(tempa>0,1,'first');
            hcut=min(steplenvec(whpos:ii)./tempa(whpos:ii));
            out_param.conechange=true; %flag the changed tau
            warning('GAIL:integralNoPenalty_g:spiky','This␣integrand␣is␣spi
            %Update Varfpup
            Varfpup(whpos+1:ii+1)=min(Varfp(whpos:ii)*inflatelim*hcut/(hcut
        end

        %Check error
        errest=Varfpup(ii+1)*(steplen.^2)/8;
        t=sumf*steplen; %compute the integral
        tminus=t-errest;
        tplus=t+errest;%compute extreme value
        tol=(4*errest^2)/(max(out_param.abstol,out_param.reltol*abs(tplus))

        if tol <= 1 %tolerance is satisfied
            q=(tminus*max(out_param.abstol,out_param.reltol*abs(tplus))+tpl
                (max(out_param.abstol,out_param.reltol*abs(tplus))+max(out_
            %keyboard
            break %exit while loop
        else %need to increase number of trapezoids
            %proposed inflation factor to increase ntrap by
            inflation=2;
        end
        if ntrap*inflation+1 > out_param.nmax
                %cost budget does not allow intended increase in ntrap
            out_param.exceedbudget=true; %tried to exceed budget
            warning('GAIL:integralNoPenalty_g:exceedbudget','integralNoPen
            inflation=floor((out_param.nmax-1)/ntrap);
                %max possible increase allowed by cost budget
            if inflation == 1 %cannot increase ntrap at all
```

```
                q=sumf*steplen; %compute the integral
                break %exit while loop
            end
        end

        %Increase number of sample points
        steplen=steplen/inflation;
        ntrapnew=ntrap*inflation;
        xnew=bsxfun(@plus,(1:inflation-1)'*steplen,xpts(1:ntrap)'); %addit
        ynew=f(xnew); %additional f(x) values
        sumf=sumf+sum(ynew(:)); %updated weighted sum of function values
        xpts = [reshape([xpts(1:ntrap)'; xnew],ntrapnew,1); xpts(ntrap+1)]
        fpts = [reshape([fpts(1:ntrap)'; ynew],ntrapnew,1); fpts(ntrap+1)]
        ntrap=ntrapnew; %new number of trapezoids
        ii=ii+1; %increment the counter

    end
else
    q = 0;
    errest = 0;
end
if flip
    q = -1*q;
end
out_param.npoints=ntrap+1;  % number of points finally used
out_param.errest=tol;      % error of integral
out_param.VarfpCI=[Varfp(ii) Varfpup(ii+1)];
```

```
function [f, out_param, flip] = integral_g_param(varargin)
% parse the input to the integralNoPenalty_g function

% Default parameter values
default.abstol  = 1e-6;
default.reltol = 5e-3;
default.nmax  = 1e7;
default.nlo = 10;
default.nhi = 1000;
default.a = 0;
default.b = 1;
% if a<b, flip = 0; if a>b, flip = 1;
flip = false;

if isempty(varargin)
    help integral_g
    warning('GAIL:integralNoPenalty_g:nofunction','Function␣f␣must␣be␣spec
    f = @(x) x.^2;
    out_param.f=f;
else
  if gail.isfcn(varargin{1})
    f = varargin{1};
    out_param.f = f;
  else
    warning('GAIL:integralNoPenalty_g:notfunction','Function␣f␣must␣be␣a␣fu
    f = @(x) x.^2;
    out_param.f = f;
  end
```

```
end;

validvarargin=numel(varargin)>1;
if validvarargin
    in2=varargin{2};
    validvarargin=(isnumeric(in2) || isstruct(in2) ...
        || ischar(in2));
end

if ~validvarargin
    %if only one input f, use all the default parameters
    out_param.a = default.a;
    out_param.b = default.b;
    out_param.abstol = default.abstol;
    out_param.reltol = default.reltol;
    out_param.nlo = default.nlo;
    out_param.nhi = default.nhi;
    out_param.nmax = default.nmax;
else
    p = inputParser;
    addRequired(p,'f',@gail.isfcn);
    if isnumeric(in2)%if there are multiple inputs with
        %only numeric, they should be put in order.
        addOptional(p,'a',default.a,@isnumeric);
        addOptional(p,'b',default.b,@isnumeric);
        addOptional(p,'abstol',default.abstol,@isnumeric);
        addOptional(p,'reltol',default.reltol,@isnumeric);
        addOptional(p,'nlo',default.nlo,@isnumeric);
```

```
            addOptional(p,'nhi',default.nhi,@isnumeric);
            addOptional(p,'nmax',default.nmax,@isnumeric);
        else
            if isstruct(in2) %parse input structure
                p.StructExpand = true;
                p.KeepUnmatched = true;
            end
            addParamValue(p,'a',default.a,@isnumeric);
            addParamValue(p,'b',default.b,@isnumeric);
            addParamValue(p,'abstol',default.abstol,@isnumeric);
            addOptional(p,'reltol',default.reltol,@isnumeric);
            addParamValue(p,'nlo',default.nlo,@isnumeric);
            addParamValue(p,'nhi',default.nhi,@isnumeric);
            addParamValue(p,'nmax',default.nmax,@isnumeric);
        end
        parse(p,f,varargin{2:end})
        out_param = p.Results;
    end;

    if (out_param.a == inf||out_param.a == -inf||isnan(out_param.a)==1)
        warning('GAIL:integralNoPenalty_g:anoinfinity',['a can not be infinity
        out_param.a = default.a;
    end;
    if (out_param.b == inf||out_param.b == -inf||isnan(out_param.b)==1)
        warning('GAIL:integralNoPenalty_g:bnoinfinity',['b can not be infinity
        out_param.b = default.b;
    end;
    if (out_param.b < out_param.a)
```

```
        tmp = out_param.b;
        out_param.b = out_param.a;
        out_param.a = tmp;
        flip=1;
    end

    % let error tolerance greater than 0
    if (out_param.abstol <= 0 )
        warning('GAIL:integralNoPenalty_g:abstolnonpos',['Error tolerance shoul
                 ' Using default error tolerance ' num2str(default.abstol)])
        out_param.abstol = default.abstol;
    end
    % let relative error between 0 and 1
    if (out_param.reltol < 0 )
        warning('GAIL:integralNoPenalty_g:abstolnonpos',['Relative error tolera
                 ' Using default error tolerance ' num2str(default.reltol)])
        out_param.reltol = default.reltol;
    end
    if (out_param.reltol >= 1 )
        warning('GAIL:integralNoPenalty_g:abstolnonpos',['Relative error tolera
                 ' Using default error tolerance ' num2str(default.reltol)])
        out_param.reltol = default.reltol;
    end



    % let initial number of points be a positive integer
    if (~gail.isposint(out_param.nlo))
        if isposge3(out_param.nlo)
```

```
            warning('GAIL:integralNoPenalty_g:lowinitnotint',['Lowest␣initial␣n
                '␣Using␣', num2str(ceil(out_param.nlo))])
            out_param.nlo = ceil(out_param.nlo);
        else
            warning('GAIL:integralNoPenalty_g:lowinitlt3',['Lowest␣initial␣num
                '␣Using␣default␣number␣of␣points␣' int2str(default.nlo)])
            out_param.nlo = default.nlo;
        end
    end
    if (~gail.isposint(out_param.nhi))
        if isposge3(out_param.nhi)
            warning('GAIL:integralNoPenalty_g:highinitnotint',['Highest␣initial
                '␣Using␣', num2str(ceil(out_param.nhi))])
            out_param.nhi = ceil(out_param.nhi);
        else
            warning('GAIL:integralNoPenalty_g:highinitlt3',['Highest␣initial␣nu
                '␣Using␣default␣number␣of␣points␣' int2str(default.nhi)])
            out_param.nhi = default.nhi;
        end
    end
    if (out_param.nlo > out_param.nhi)
        if isposge3(out_param.nhi)
            warning('GAIL:integralNoPenalty_g:nlobtnhi',['Highest␣initial␣numb
                '␣Using␣', num2str(ceil(out_param.nhi)), '␣as␣nlo'])
            out_param.nlo = ceil(out_param.nhi);
        else
            warning('GAIL:integralNoPenalty_g:highinitlt3',['Highest␣initial␣nu
                '␣Using␣default␣number␣of␣points␣' int2str(default.nhi)])
            out_param.nhi = default.nhi;
        end
    end

    out_param.ninit = max(ceil(out_param.nhi*(out_param.nlo/out_param.nhi)^(1/(
    if (~gail.isposint(out_param.nmax))
        if ispositive(out_param.nmax)
            warning('GAIL:integralNoPenalty_g:budgetnotint',['Cost␣budget␣shoul
                '␣Using␣cost␣budget␣', num2str(ceil(out_param.nmax))])
            out_param.nmax = ceil(out_param.nmax);
        else
            warning('GAIL:integralNoPenalty_g:budgetisneg',['Cost␣budget␣should
                '␣Using␣default␣cost␣budget␣' int2str(default.nmax)])
            out_param.nmax = default.nmax;
        end;
    end
```

BIBLIOGRAPHY

[1] Lawrence F. Shampine. Vectorized adaptive quadrature in Matlab. *Journal of comptational and applied mathematics.* 211(2008)131-140, 2006.

[2] Fred J.Hickernell, Martha Razo, Sunny Yun. Reliable Adaptive Numerical Itegration, 2015, Preprint

[3] Fred J. Hickernell, Lluís Antoni Jiménez Rugama, and Da Li. Adaptive Quasi-Monte Carlo Methods for Cubature. aeXiv:1702.01491v1, 2017

[4] Saulo P. Oliveira, Alexandre L. Madureira and Frederic Valentin. Weighted Quadrature Rules for Finite Element Methods. *Journal of comptational and applied mathematics.* May 2009, Pages 93-101

[5] 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*, Pages 21-45

[6] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Da Li, Jagadeeswaran Rathinavel, Lluis Antoni Jimenez Rugama, Xin Tong, Kan Zhang, Yizhi Zhang, and Xuan Zhou. GAIL: Guaranteed Automatic Integration Library (Version 2.2), MATLAB Software, 2017.

[7] MATLAB version 8.4. Natick, Massachusetts: The MathWorks Inc., 2014.