A Metalgorithm for Adaptive Quadrature

JOHN R. RICE

Purdue University, West Lafayette, Indiana

ABSTRACT. The few adaptive quadrature algorithms that have appeared are significantly superior to traditional numerical integration algorithms The concept of metalgorithm is introduced to provide a framework for the systematic study of the range of interesting adaptive quadrature algorithms A principal result is that there are from 1 to 10 million potentially interesting and distinct algorithms This is followed by a considerable development of metalgorithm analysis. In particular, theorems about the convergence properties of various classes of algorithms are established which theoretically show the experimentally observed superiority of these algorithms. Roughly, these theorems state. (a) for "well-behaved" integrands adaptive algorithms are just as efficient and effective as traditional algorithms of a "comparable" nature, (b) adaptive algorithms are equally effective for "badly behaved" integrands where traditional ones are ineffective The final part of the paper introduces the concept of a characteristic length and its role is illustrated in an analysis of three concrete realizations of the metalgorithm, including the algorithms CADRE and SQUANK

KEY WORDS AND PHRASES· quadrature, numerical integration, adaptive quadrature, quadrature convergence, numerical integration convergence, numerical integration programs, quadrature data structures, algorithm classes, algorithm analysis, algorithm components

CR CATEGORIES· 3.62, 5.16, 5.29

1. *Introduction*

The quadrature problem for a given function $f(x)$ is to estimate the value of

$$If = \int_0^1 f(x)\,dx.$$

Traditional *quadrature formulas* give estimates of the form

$$Q_N f = \sum_{i=1}^{N} w_i f(x_i)$$

for some suitably chosen weights w_i and abscissas x_i. *Adaptive quadrature* uses some algorithm to choose the abscissas and weights during the computation and thus is to dynamically adapt its estimate to the particular properties of the integrand $f(x)$.

A number of adaptive quadrature algorithms have appeared in the literature [2, 6, 8, 9] and there is some basis [3, 5] to believe that they are significantly superior to traditional quadrature formulas. As shown later, the few algorithms that have appeared only scratch the surface of the possibilities, and one purpose of this paper is to systematically study the range of interesting adaptive quadrature algorithms. The concept of *metalgorithm* is introduced for this purpose; the word means a framework or theory to study algorithms. Webster's relevant definition of the prefix "met" is "discipline designed to deal critically with the original one."

This work was supported in part by Grant GP 32960X from the National Science Foundation.

Author's address: Computer Sciences, Mathematical Sciences Building, Purdue University, West Lafayette, IN 47907.

A metalgorithm is presented in Section 2 and, in simple terms, it consists of a block diagram for a class of algorithms. The blocks or components may be chosen in one of several ways, and a small catalog is given of the various interesting components. This metalgorithm does not include all interesting adaptive quadrature algorithms, and yet we obtain the estimate that there are from 1 to 10 *million* algorithms that are potentially interesting and significantly different from one another.

This forbidding estimate leads naturally to the next section, 3, where *metalgorithm analysis* is discussed. The objective is to find tools and techniques to extract the better algorithms. Five somewhat independent approaches are identified and brief remarks are made on some of them.

One tool of metalgorithm analysis is called *component analysis*, and Section 4 of this paper presents such an analysis for the metalgorithm component: *interval collection management*. This is the most novel component of the metalgorithm, one that has not been identified or studied previously. Four basic data structures (ordered list, stack, queue, and "boxes") are described for the collection of intervals, and each of these leads to a distinct class of collection management procedures These four choices also lead to distinctive behaviors of the algorithms, and these are described in some detail. As one might expect, no one data structure or associated algorithm appears to be uniformly superior to the others. Thus one cannot reject any of these on a priori grounds but must make a choice of data structure by considering the algorithm components and the characteristics of the anticipated problem domain.

Sections 5–7 consider convergence aspects of adaptive quadrature algorithms. Usually it is almost trivial to prove that the adaptive algorithms converge, but this is not the question of real interest. The crucial question is how fast do they converge and, in particular, how effective are they for integrands $f(x)$ where traditional quadrature formulas are inefficient. Two basic results concerning interval partitioning algorithms are established in Section 5. They are applicable to the error analysis for adaptive algorithms other than quadrature (e.g. curve fitting or solving ordinary differential equations) and thus are presented somewhat abstractly and independent of the quadrature problem. These results are then applied to establish the rate of convergence theorems for four general classes of algorithms.

In intuitive terms, these results state two things. First, for "well-behaved" integrands, adaptive quadrature algorithms are just as effective and efficient as traditional quadrature formulas of a "comparable" nature. Second, adaptive quadrature algorithms are equally effective and efficient for a broad range of "badly behaved" integrands where traditional formulas are ineffective. This range of "badly behaved" integrands includes almost all integrands that one can visualize arising in real applications.

The final section, 8, considers specific algorithms from this metalgorithm and introduces the concept of *characteristic length*. This concept is used to distinguish between mathematical convergence (as usually established in convergence theorems) and algorithmic convergence (as usually required or desired in computer programs.) The role of the characteristic length is examined in some detail for three concrete algorithms including two well-known ones, SQUANK [8] and CADRE [1].

2. *Metalgorithm for Adaptive Quadrature*

The metalgorithm considered in this paper is represented graphically by the block diagram in Figure 1. An interval is usually more than just two endpoints and we use the word interval to include all the information associated with an interval, e.g. two endpoints, saved values of $f(x)$ for the interval, area and error bound estimates, status marker (error acceptable, discarded, being processed, etc.), and auxiliary information (trouble flags, neighbor information, etc.).

The interval collection is organized into some data structure and the purpose of the interval collection manager block is to create and maintain this structure. Intervals are

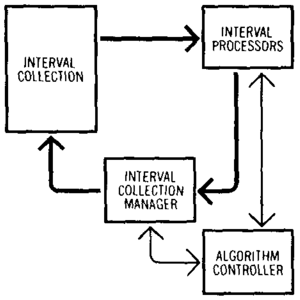


FIG 1 Block diagram of the metalgorithm for adaptive quadrature The heavy lines show the flow of intervals and the light lines show the flow of control and other information.

taken from the collection and the interval processor does computations to improve the area estimates and error bounds associated with this interval. The usual components for the interval processor produce two new intervals while processing an interval. The controller has two distinct components. One is the bound estimator which is involved in the decision to terminate the algorithms. The other is a collection (perhaps empty) of procedures to detect special types of behavior and to initiate special computations in such cases.

Rather than attempt to explain the functions of these components in more detail, a list of possible choices for them is given. It is assumed that the reader is familiar with numerical integration such as in [4] and thus rather cryptic terms are used in these lists.

Interval Processor Components:
1. Single rule applied to whole interval and to each half.
2. Independent rules of same order, agreement checked.
3. Different order rules, agreement checked.
4. Variable order rule with consistency checked.
5. Trapezoidal rule plus convexity assumption to obtain strict bounds.
6. Single rule applied to whole interval without error estimate check.

The interval processor produces error bound estimates for the individual intervals, but these must be combined in some way to obtain an overall error bound. This is the task of the

Bound Estimator Components:
1. Proportional distributions of error requirement over subintervals.
2. Other (sometimes ad hoc) distribution of error requirement.
3. Direct analysis of error bounds.

Special Behavior Components:
1. Roundoff level (unpredictable behavior).
2. Jump discontinuities.
3. x^α type singularities $(-1 < \alpha < 1)$.

A thoughtful discussion of interval processors, bound estimations, and special behavior analysis appears in [1], and thus we do not elaborate further on them here. The final component is interval management. Cryptic descriptions are also used for this list of components, and the reader who does not visualize the implications of these terms might look at Section 4, where they are expanded upon and where examples are given.

Interval Collection Management Components:
1. Stack—geometric orientation and intervals with small error estimates discarded.
2. Ordered list—intervals ordered according to the values of error estimates.
3. Queue.
4. Queue—intervals with small error estimates discarded.

5. Boxes—multilevel grouping based on error estimates.
6. Fixed sequence—no adaptation on interval collection management.

We now estimate the number of algorithms included in this metalgorithm using these components. We may take one choice from each of these four categories except for the special behavior components, where there are eight possible combinations. The number of algorithms included is then 6*3*8*6 = 864. This number does not take into consideration a variety of significant considerations:

A. Subchoices for interval processor components. Conservatively there are five subchoices within each choice listed above. However, we assume there are only three (e.g. for rules using three points per subinterval there are Simpson's, 3-point Gauss, 3-point Chebyshev-Gauss, 3-point Chebyshev, and open Newton-Cotes.)

B. Subchoices for interval management. There are at least two significant variations for each choice.

C. Subchoices for error bound estimation. There are at least three times as many as indicated above.

D. Other special behavior components. There are at least three more.

E. Significant variations on the next level of detail in the algorithm specification. These variations may make an order of magnitude difference in various attributes of the algorithm.

F. Polyalgorithms which dynamically (adaptively) make selections and changes in the components.

The first four considerations (A through D) increase the number of algorithms by a factor of at least 3*2*3*8 = 144, which gives 124,416 algorithms. The additional algorithms that arise from considerations E and F increase this number by a factor of 10, at least, and probably by a factor of 100. We thus arrive at the following conclusion: *There are 1 to 10 million adaptive quadrature algorithms which are potentially reasonable and interesting.*

This estimate is more likely to be low than high, for there are adaptive quadrature algorithms or components which are not included in this metalgorithm. A particularly pessimistic interpretation of this study is that there are over 1 million papers to be written that contain an adaptive quadrature algorithm with the following two properties: it is original; there are integrands for which this algorithm is more efficient than any of the other 999,999 algorithms.

3. *Metalgorithm Analysis*

It is clear that some tools and techniques must be developed to aid in the selection of the better algorithms from this metalgorithm. Five such techniques are listed below along with a brief discussion of the kind of results that one can hope to obtain. Even after great development takes place in all of these areas it seems that it still will be difficult to identify the truly superior algorithms Thus one can expect a steady stream of constantly improved algorithms to be constructed in the future.

The first tool is that of traditional numerical analysis:

(A) *Convergence and error analysis.* There are two basic ingredients here; one is a classification of functions and the other is a parameterization of algorithms. The typical result is then stated as: "If $f(x)$ belongs to the function class $\mathfrak{F}$ and if the algorithm is parameterized by the variable h(or N), then as $h \to 0$ (or as $N \to \infty$) we have $| Q_h f - If |$ $= \mathcal{O}(h^p)$ or $| Q_N f - If | = \mathcal{O}(N^{-p})$." In the concrete example of Simpson's Rule we have that $\mathfrak{F}$ is $C^4[0, 1]$, h is the partition length, and $\mathcal{O}(h^p)$ becomes $-f^{(4)}(\xi)h^4/180$.

There is no doubt that such results give one considerable insight into the behavior of algorithms which incorporate particular components. The primary weaknesses of these results are: (i) They are only asymptotic or, almost equivalently, they involve unknown constants. (ii) They do not apply to adaptive algorithms except by indirect and intuitive reasoning. (iii) The function classes involved are often difficult to relate to practical applications.

The second technique is often only a reformulation of the first and yet it provides a viewpoint that is more relevant and interesting:

(B) *Computational complexity.* The key idea is to examine the work (measured in units of arithmetic operations) required to calculate If to a certain accuracy. There are two reasonable ways to express this work. One is the number α of operations required to obtain d correct digits in If (to obtain an accuracy of $\epsilon = 10^{-d}$), and the other is the average number β of digits obtained per arithmetic operation while computing If correct to d digits. Results in this area also involve classes of functions and a typical result appears as: "If $f(x)$ belongs to the function class $\mathfrak{F}$, then we have $\alpha = \mathcal{O}(d^p)$ [or $\alpha = \mathcal{O}(\epsilon^{-q})$] and $\beta = \mathcal{O}(d^{1-p})$ [or $\beta = \mathcal{O}(\epsilon^q \log \epsilon)$]."

Note that quadrature formulas involve evaluation of the function $f(x)$ and thus the work of these evaluations must be incorporated into the results. Alternatively, one might choose to have the work measured in terms of function evaluations. If $f(x)$ requires a fixed number of arithmetic operations per evaluation, then the result above for Simpson's Rule may be restated as: $\alpha = \mathcal{O}(10^{-d/4})$ (work for d correct digits) or $\alpha = \mathcal{O}(\epsilon^{-1/4})$ (work for accuracy ϵ), and $\beta = \mathcal{O}(d*10^{d/4})$ (work per digit for d correct digits) or $\beta = \mathcal{O}(\epsilon^{-1/4} \log \epsilon)$ (work per digit for accuracy ϵ).

The next tool discussed is:

(C) *Component analysis* The possible choices of components in a metalgorithm are analyzed and various characteristics of these components are developed. This analysis is generally without regard to any interactions of the component with other parts of an algorithm. Many of the results on error analysis may be viewed as characteristics of components for the interval processor or bound estimator. Section 4 presents a systematic component analysis for interval collection management.

The next tool is almost totally undeveloped, and yet it is crucial to definitive comparisons and evaluations of the algorithms contained in the metalgorithm.

(D) *Population studies.* The sample results mentioned earlier about error analysis and computation complexity involved classes of functions in an essential way. Implicit in these studies is the hope that these abstract classifications are relevant to the population of functions that arise in application of the algorithms. There has been essentially no systematic study of the population of integrands that arise in applications. Such a study would probably require a large effort even to produce tentative results. There are indications that the traditional mathematical classifications of functions are of little relevance in "practical" curve fitting and thus there is reason to doubt the value of these classifications for the evaluation of quadrature algorithms. Yet it is meaningless to discuss the value of algorithms except with respect to a given population or class of integrands. It seems reasonable that even rough quantitative and qualitative characteristics of the integrand population will suffice to make definitive evaluations of algorithms. However, until such characteristics become known one must view comparative evaluations of algorithms with considerable suspicion

The final tool for metalgorithm analysis is·

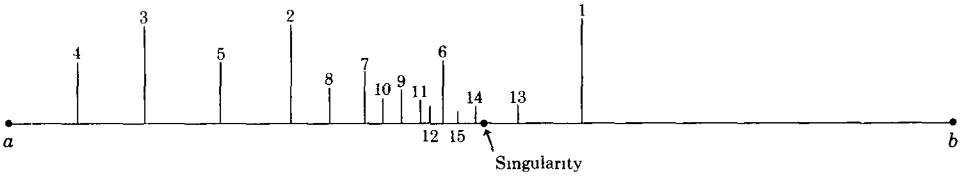
(E) *Experimental studies.* The idea is quite simple. One selects a representative sample from the integrand population and devises measures to evaluate algorithms. Then one takes several algorithms, performs the tests, and draws conclusions about their relative merits. An example of this approach is given in [5]. One must mistrust the samples until one learns something about the population involved but, nevertheless, the results are informative and much better than nothing. An intriguing related idea is that of the "performance profile" introduced in [7] and also used in [2]. A performance profile involves a one-parameter family of functions (e.g. x^α, $0 < \alpha < 1$; $\sin (Nx + .123)$, $1 < N < 500$; $(|x - \beta|)^{\frac{1}{2}}$, $0 < \beta < 1$) and one or more measures of performance. These measures are then plotted as a function of the parameters This approach is not only useful for comparing algorithms, but it also contributes to understanding the effects of population characteristics on algorithm performance.

4. *Analysis of Interval Collection Components*

In this section we consider four basic data structures (ordered list, stack, queue, and boxes) for the interval collection and the properties of associated procedures. This analysis is, in the end, inconclusive and yet it is a worthwhile endeavor. It is worthwhile because we do identify several properties (advantages and disadvantages) of each choice and it is inconclusive in that no one choice is clearly superior to the others. Furthermore, results established later in this paper show that the asymptotic convergence rates (computational efficiency) are the same in all cases (provided one is sensible in choosing the other components.)

We compare these data structures on the basis of the following questions about the quadrature algorithm. (A) How does it adapt itself to difficulties? e.g. singularities, general or endpoint, multiple areas of difficulty. (B) How much computational effort is required for managing the data structure? (C) How much memory does the data structure require? (D) How complex is the algorithm for managing the interval collection? (E) How much information does the algorithm provide about the global behavior of the integrand, especially before any irrevocable decisions must be made? (F) How systematic is the behavior of the quadrature error? Can one postpone the decision to terminate until late in the computations or must some a priori decisions be made?

Stack. We consider the stack first as it is the primary data structure in use in current adaptive quadrature algorithms. The stack is in a direct relationship with the geometry of the x-axis and the resulting algorithm processes the interval $[a, b]$ in a fixed direction (say left to right.) This feature tends to separate any distinct areas of difficulty in the integrand and thus we consider its behavior for a single difficult "spot" in the integration. If we assume that intervals are always bisected, then the presence of a singularity results in bisection points as indicated on the line below (for the first 15 points):



The corresponding stacks are (with the top at the left):

$$\begin{array}{rr}
[0, 1] & [\frac{5}{16}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] \\
[0, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{5}{16}, \frac{11}{32}][\frac{11}{32}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] \\
[0, \frac{1}{4}][\frac{1}{4}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{11}{32}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] \\
[0, \frac{1}{8}][\frac{1}{8}, \frac{1}{4}][\frac{1}{4}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{23}{64}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] \\
[\frac{1}{8}, \frac{1}{4}][\frac{1}{4}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] \\
[\frac{1}{4}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{3}{8}, \frac{7}{16}][\frac{7}{16}, \frac{1}{2}][\frac{1}{2}, 1] \\
[\frac{1}{4}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{3}{8}, \frac{13}{32}][\frac{13}{32}, \frac{7}{16}][\frac{7}{16}, \frac{1}{2}][\frac{1}{2}, 1] \\
[\frac{1}{4}, \frac{5}{16}][\frac{5}{16}, \frac{3}{8}][\frac{3}{8}, \frac{1}{2}][\frac{1}{2}, 1] & [\frac{3}{8}, \frac{25}{64}][\frac{25}{64}, \frac{13}{32}][\frac{13}{32}, \frac{7}{16}][\frac{7}{16}, \frac{1}{2}][\frac{1}{2}, 1]
\end{array}$$

The intervals coalesce near the singularity (as they should), accumulating on the left with an occasional bisection point on the right. Finally the intervals become small enough for the singularity to be jumped and the intervals then accumulate on the right, gradually moving away from the singularity and becoming longer. Some people have felt that this behavior pattern near a singularity is not ideal; the algorithm CADRE [2] modifies the stacking strategy in order to close in on the singularity from both sides in a more or less uniform manner. Overall, it appears that this data structure is satisfactory for adapting to singularities and other localized difficulties.

The stack is superior to the other data structures in terms of "overhead," i e. it uses the least memory, requires minimal computational effort, and is simple to program.

The primary weakness of this data structure is that it must make an irrevocable decision to discard some subintervals before there is an opportunity to examine the global behavior of the integrand. This weakness is especially pronounced if there is a singularity at or near the left endpoint. For example, if one desires to have a relative *accuracy* requirement, then one needs a reasonable estimate of the value of the integral before one is able to make a reasonable decision about discarding an interval. This is not possible with the stack data structure.

The stack data structure leaves a relatively large interval unexamined until the very end of the computation. Thus the quadrature error does not decrease steadily during the computation but rather has a sharp drop at the very end. If the error requirement is not met at this point there is no way to continue the computation to reduce the error, one must start over again.

Ordered list. The ordered list structure (based on the bound estimates) seems to be the most successful and efficient in adapting to various kinds of difficulties in the integrand. There is normally a very steady and regular decrease in the global error estimate and the algorithm gives each difficult "spot" an appropriate amount of attention.

The ordered list should be maintained as a linked list; the following indicates two successive states of the list·

List index	*Interval*	*Bound*	*Link*	*Interval*	*Bound*	*Link*
1	$[0, \frac{1}{16}]$	06	0	$[0, \frac{1}{16}]$	.06	3
2	$[\frac{1}{2}, \frac{5}{8}]$	12	6	$[\frac{1}{2}, \frac{5}{8}]$	.12	6
3	$[\frac{3}{4}, 1]$	.40	4	$[\frac{3}{4}, \frac{7}{8}]$	04	0
4	$[\frac{1}{4}, \frac{1}{2}]$	.22	2	$[\frac{1}{4}, \frac{1}{2}]$	.22	7
5	$[\frac{5}{8}, \frac{3}{4}]$	07	1	$[\frac{5}{8}, \frac{3}{4}]$	.07	1
6	$[\frac{1}{16}, \frac{1}{8}]$	.08	5	$[\frac{1}{16}, \frac{1}{8}]$	.08	5
7				$[\frac{7}{8}, 1]$	.13	2
	ITOP = 3			ITOP = 4		

The variable ITOP is a pointer for the top of the list.

There is a real price paid for this superior performance. The overhead is high, especially in the computational effort to maintain the order in the list. If one maintains the order by merging the list with the two (or so) new intervals added at each time, then the total computational effort is of the order of N^2 where N is the number of intervals generated (or function evaluations). For large values of N, this effort exceeds that of all the other computations in the algorithm. If a bisection method is used then the N^2 order becomes $N \log N$, but even then this is asymptotically larger than the other calculations (there are an order of N of them.) It is not clear, however, if this is a significant factor for practical applications

The memory requirements of this data structure become nontrivial if considerable accuracy is requested. The list might have several hundred (or even thousand) intervals in it. It is possible and even reasonable to include some discard procedure in this data structure in order to reduce both the memory requirement and computational overhead.

The programs for implementing this data structure are significantly more complicated than the trivial ones for the stack.

While the interval list is ordered in a manner very relevant to the error and difficulty in the integration, there is no obvious correlation with the geometry of the problem. For example, if one wished to know the location of two or three singularities of $f(x)$ (assumed they are known to exist), it is a fair calculation to identify them from examining the interval list. This means, for example, that it would be difficult to identify certain characteristic systematic behaviors that occur in conjunction with certain types of singularities and difficulties.

On the other hand, the behavior of the quadrature error is usually very systematic, especially after a short transient period at the beginning of the algorithm. Figure 2 gives a plot of the bounds on the quadrature error generated by an adaptive trapezoidal rule

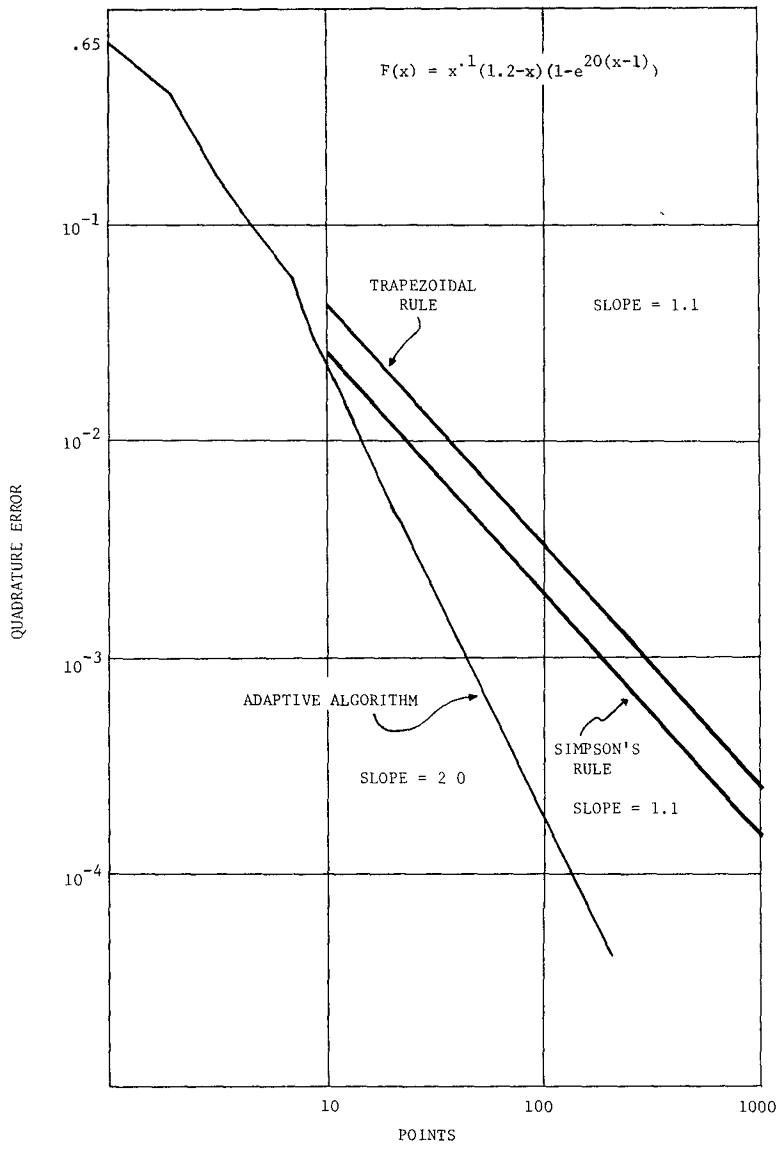


FIG. 2 The behavior of the error for three quadrature methods applied to $f(x) = x^{.1}(1.2 - x)(1 - e^{20(x-1)})$ The methods are (a) the composite trapezoidal rule, (b) the composite Simpson's rule, and (c) an adaptive trapezoidal rule using an ordered list data structure

for convex functions. The actual errors behave in the same manner as the bounds. The problem is

$$\int_0^1 x^{.1}(1.2 - x)(1 - e^{20(x-1)})\,dx,$$

which involves a difficult integrand. For comparison purposes the error for the ordinary trapezoidal and Simpson rules are also shown. The error behavior is very regular and, for example, Richardson's extrapolation is quite effective in obtaining improved accuracy.

This data structure is well suited for delaying the decision to terminate until late in the computations.

Queue. The queue data structure leads to an algorithm which makes repeated passes across the interval $[A, B]$. On each pass it subdivides each interval from the front of the queue and places the resulting smaller intervals at the end of the queue. The first few stages of the queue for $[A, B] = [0, 1]$ appear as

$$
\begin{array}{l}
[0, 1] \\
[0, \tfrac{1}{2}][\tfrac{1}{2}, 1] \\
[\tfrac{1}{2}, 1][0, \tfrac{1}{4}][\tfrac{1}{4}, \tfrac{1}{2}] \\
[0, \tfrac{1}{4}][\tfrac{1}{4}, \tfrac{1}{2}][\tfrac{1}{2}, \tfrac{3}{4}][\tfrac{3}{4}, 1] \\
[\tfrac{1}{4}, \tfrac{1}{2}][\tfrac{1}{2}, \tfrac{3}{4}][\tfrac{3}{4}, 1][0, \tfrac{1}{8}][\tfrac{1}{8}, \tfrac{1}{4}]
\end{array}
$$

A discard procedure is not essential to an algorithm with this data structure but it appears that efficiency is considerably enhanced if one is used.

This data structure has the same behavior as the stack as far as adaptiveness to singularities is concerned. Indeed, if we assume the same discard procedure is used, both these data structures lead to an identical set of intervals being generated—except they are generated in a different order.

The overhead in computational effort and program length is reasonably small. However, this data structure requires a large amount of memory when the queue becomes long. That, of course, is likely to happen when high accuracy is required. The size of the interval collection is likely to be even larger than for the ordered list structure since this approach normally generates more intervals.

The queue structure does not have the weakness of the stack with regards to making premature decisions about discarding intervals. This structure leads to algorithms which uniformly refine the level of examination of the integrand. Furthermore, the geometry of the x-axis is maintained in the structure and thus one would conjecture that this structure is very well suited if more sophisticated behavior adapting features are intended to be included in the algorithm.

Similarly, the queue structure allows one to delay the termination decision until late in the computations. There is a fairly regular decrease in the error as the algorithm progresses, and this may be used to aid in the termination decision. The discarding procedure (if any) may be made somewhat more conservative than with the stack data structure and yet allow one to obtain reasonable efficiency by terminating the algorithm before all intervals are discarded.

Boxes. As intervals are generated they are assigned to *boxes* (or *bins* or *buckets* or *groups*) according to the size of the error bound associated with the interval. Let *BOUND* be a generic variable for the error bound for any particular interval. Then, with six boxes used, the test for placing an interval is:

Box 1. $BOUND \geq BL(1)$	Box 4: $BL(3) > BOUND \geq BL(4)$
Box 2: $BL(1) > BOUND \geq BL(2)$	Box 5: $BL(4) > BOUND \geq BL(5)$
Box 3: $BL(2) > BOUND \geq BL(3)$	Box 6: $BL(5) > BOUND \geq 0$

The intent of using this data structure is to achieve some of the advantages of first processing the intervals with the largest *BOUND* values and to avoid the computational effort of maintaining a completely ordered list. It does make a reasonable compromise and is likely to be preferred to the ordered list structure from the computational point of view.

A weakness of this data structure is that it is "messier" than the others and hence more complex to program and more difficult to visualize. It also has the drawback of possibly generating a large collection of intervals whenever the accuracy requirement is high.

This data structure shares the disadvantage of the ordered list in that the geometric structure of the problem is not incorporated into this data structure in any natural way. One is able to delay the termination procedure with this data structure and the somewhat

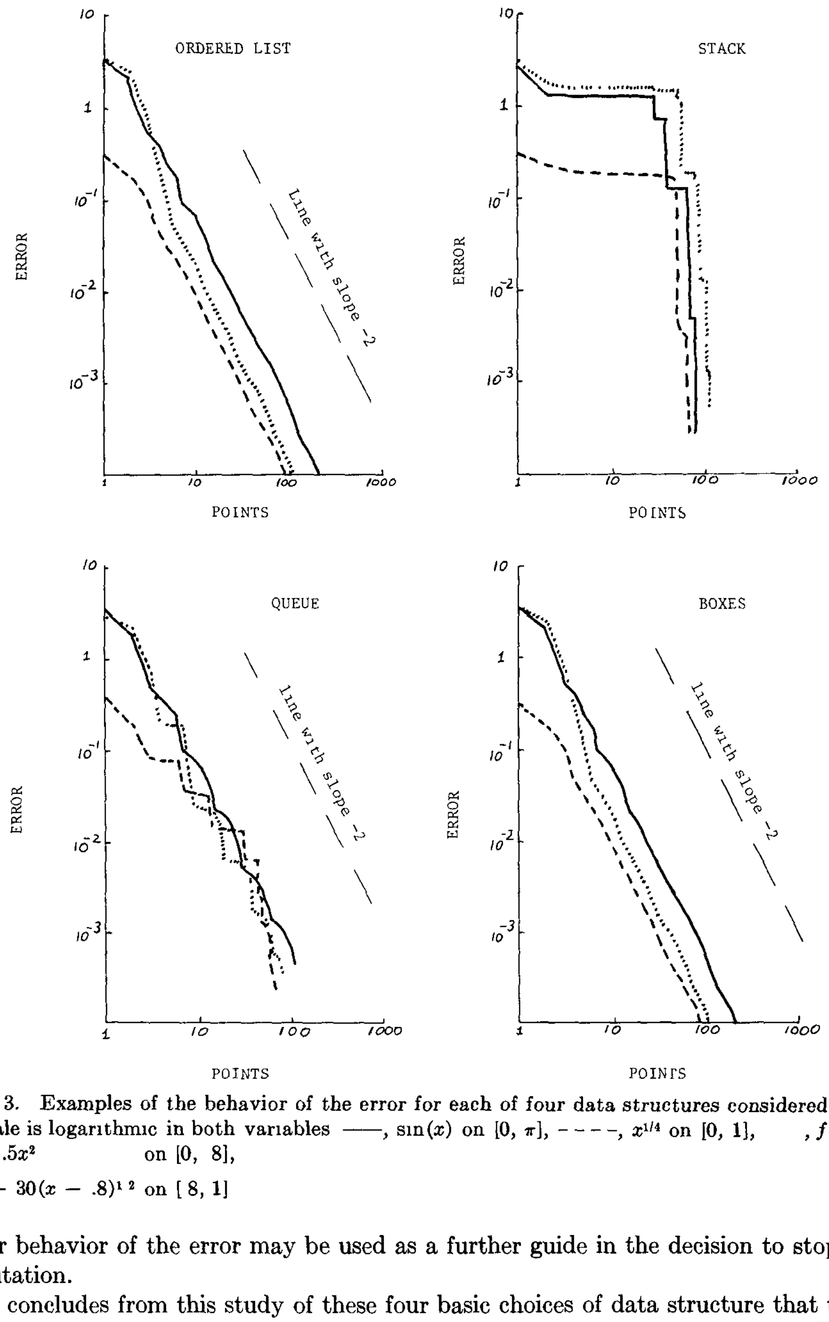


FIG. 3. Examples of the behavior of the error for each of four data structures considered here. The scale is logarithmic in both variables ——, sin(x) on [0, π], - - - -, $x^{1/4}$ on [0, 1], , $f(x) = \begin{cases} 7x - .5x^2 & \text{on } [0, 8], \\ 5\ 28 - 30(x - .8)^{1\ 2} & \text{on } [8, 1] \end{cases}$

regular behavior of the error may be used as a further guide in the decision to stop the computation.

One concludes from this study of these four basic choices of data structure that there are situations where each of them is the best one to use. It is indeed possible, as adaptive algorithms evolve, that hybrid data structures are the most suitable. For example, one could use a queue until its length reaches a limit and then treat each of the intervals in the queue as a new problem where a stack algorithm is used.

Figure 3 shows an example of the behavior of the error for each of these data structures. They were each embedded in an adaptive trapezoidal rule designed for integrating convex functions. One of the functions used in Figure 3 is entire, while another has a discontinuous derivative at an interior point of the interval of integration and the third has an infinite derivative at $x = 0$.

5. *Theorems on Interval Partitioning Processes*

There is little interest in proving that a quadrature algorithm converges, the interest lies in the rate of convergence and the domain of functions for which this rate is achieved. Experimental evidence has been accumulated which indicated that adaptive quadrature algorithms do have a much larger domain of integrands where rapid convergence is achieved. In sections 6–8 of this paper we provide theoretical verification of these experimental observations (at least for broad classes of algorithms) and delineate to some extent the domain of integrands for which rapid convergence takes place.

In this section we consider interval partitioning processes that arise in a variety of adaptive algorithms, including quadrature, curve fitting, and solving differential equations. These processes essentially involve an algorithm with the following general iterative steps: (1) consider a particular interval I and an associated error η; (2) subdivide I into two or more parts and compute new error estimates for each part; (3) put aside those intervals where the new error estimate is acceptable and return the others to the collection to be processed further.

We define two such algorithms here and establish various results about when they terminate. The first is given below and the motivation for the name "fixed" is to distinguish it from the second algorithm, where the name "proportional" naturally applies.

Fixed Partition Algorithm.

```
1. Initialization: We are given
   A. Numbers $\gamma, \beta < 1$, and $\epsilon > 0$.
   B. An empty set $M'$ and a set $M$ of intervals $I$ with associated numbers $\eta(I)$. $M$ contains a distinguished interval $I^*$.
   C. A process $P \cdot I \rightarrow (IL, IR)$ which divides an interval $I$ into left and right subintervals such that
        (i) If $I = I^*$ then $\eta(IL) = \eta(IR) = \beta_* \eta(I)$
                 and $I^* \leftarrow IL$ or $I^* \leftarrow IR$.
        (ii) If $I \neq I^*$ then $\eta(IL) = \eta(IR) = \gamma_* \eta(I)$
2. Operation:
   For $I \in M$ do
      $P : I \rightarrow (IL, IR)$
      If $(\eta(IL) < \epsilon)$ then $IL \in M'$ else $IL \in M$
      If $(\eta(IR) < \epsilon)$ then $IR \in M'$ else $IR \in M$
```

In the later applications a distinguished interval is one that contains a singularity, and the numbers $\eta(I)$ correspond to error bounds for some computation. Note that M contains at most one distinguished interval; it is generically denoted by I^*.

The operations of this algorithm may be illustrated by a tree as in Figure 4, which shows the partitioning of I^*. The nodes of the tree represent intervals and the number placed there is the factor which multiplies η as a result of the partition process.

The question is to determine the size of M' when the algorithm terminates. This depends on the values of γ, β, ϵ, the numbers $\eta(I)$, and the initial size of M. An estimate in terms of all of these quantities is developed in the course of the proof of Theorem 1, but we are primarily interested in the dependence on γ and ϵ.

Theorem 1. *Consider the Fixed Partition Algorithm with β, M, and $\eta(I)$ for $I \in M$ specified. Let $F(\gamma, \epsilon)$ be the size of M' when the algorithm terminates and we have $F(\gamma, \epsilon) = \mathcal{O}(\epsilon^{1/\log_2 \gamma})$.*

Proof. First consider those intervals in M which are not distinguished. If they are subdivided d times then the numbers associated with the resulting intervals are $\gamma^d \eta(I)$

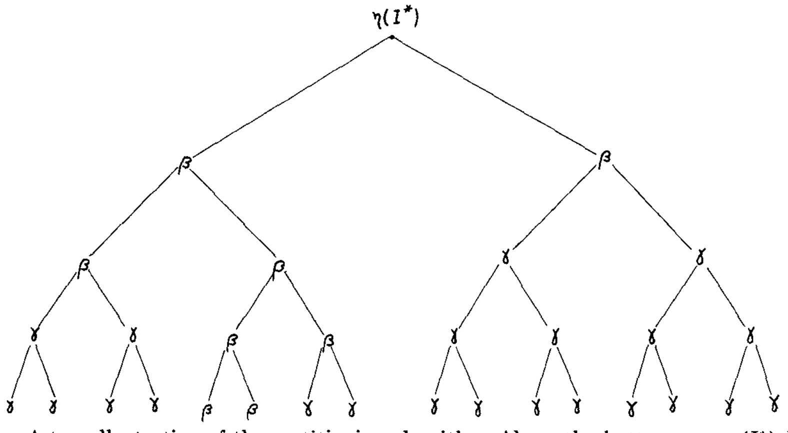


FIG. 4 A tree illustration of the partitioning algorithm. Along the bottom row $\eta(I^*)$ has been multiplied by a factor of the form $\gamma^k\beta^{4-k}$ for $k = 0, 1, 2, 3$

or less; hence they are placed in M' when $\gamma^d\eta(I) < \epsilon$ and the smallest such number d' is seen to satisfy

$$d' \geq \log_2 [\epsilon/\eta(I)]/\log_2 \gamma \geq d' - 1$$

and the number of intervals placed in M' is $2^{d'}$. Thus the number of intervals in M' from I satisfies

$$2^{d'} \leq [\epsilon/\eta(I)]^{1/\log_2\gamma}.$$

Clearly then the number of intervals placed in M' from the nondistinguished intervals of M is as stated in Theorem 1.

Now consider the distinguished interval I^* and set $\eta_0 = \eta(I^*)$. We see that the distinguished descendant of I^* (by d subdivisions) is placed in M' whenever $\beta^d \leq \epsilon/\eta_0$. Let d_0 be the smallest such d. We see that $d_0 \geq [\log_2 \epsilon/\eta_0]/\log_2 \beta \geq d_0 - 1$.

Observe now that each time I^* is subdivided it produces another I^* and an interval I_k with associated number $\beta^k\eta_0$, and this occurs for $k = 1, 2, \cdots, d_0 - 1$. The descendants of each of these intervals are placed in M' after d_k subdivisions where

$$\beta^k\gamma^{d_k} \leq \epsilon/\eta_0 \leq \beta^k\gamma^{d_k-1}.$$

Thus the total number of intervals descendant from I^* that are placed in M' is bounded by

$$2 + \sum_{k=1}^{d_0-1} 2^{d_k} \leq 2 + \sum_{k=1}^{d_0-1} 2^{[(\log_2 \epsilon/\beta^k\eta_0)/\log_2\gamma]}$$

$$\leq 2 + \sum_{k=1}^{d_0-1} [\epsilon/\beta^k\eta_0]^{1/\log_2\gamma} \leq 2 + (\epsilon/\eta_0)^{1/\log_2\gamma}\sum_{k=0}^{\infty} (\beta^{-1/\log_2\gamma})^k.$$

The geometric series is convergent since $\gamma, \beta < 1$ and may be summed to obtain the bound $2 + (\epsilon/\eta_0)^{1/\log_2\gamma}[1 - \beta^{-1/\log_2\gamma}] = \mathcal{O}(\epsilon^{1/\log_2\gamma})$. This concludes the proof.

We now state some slight generalizations of this result as corollaries.

COROLLARY 1. *If the Fixed Partition Algorithm is modified so that each interval is divided into m parts or less and if there are k distinguished intervals, then the conclusion of Theorem* 1 *becomes* $F(\gamma, \epsilon) = \mathcal{O}(\epsilon^{1/\log_m\gamma})$.

COROLLARY 2. *Consider a real valued function g defined on intervals with the property that* $I_1 \subseteq I_2$ *implies* $g(I_1) \leq g(I_2)$. *Suppose that in the interval division process P the factors* γ *and* β *are replaced by* $\gamma_*g(IR)$, $\gamma_*g(IL)$, $\beta_*g(IR)$, *and* $\beta_*g(IL)$ *as appropriate. Then the conclusion of Theorem* 1 *remains valid.*

Let π_m be a partition of $[a, b]$ obtained by subdividing intervals into m equal parts.

Corollary 3. *Consider the Fixed Partition Algorithm before it terminates and let K be the number of intervals I in π_m with $\eta(I) > \eta_0$. Then we have that there is a constant C (independent of π_m, ϵ, and η_0) so that $K < C(\eta_0^{1/\log_m \gamma})$.*

The Fixed Partition Algorithm is motivated by the thought that one wants to make the numbers $\eta(I)$ each less than ϵ. In quadrature one wants rather that the sum of the numbers (which are interpreted as error estimates) to be less than ϵ. Thus we may visualize ϵ as a global quadrature error requirement which is to be distributed among the intervals as the subdivision takes place. This leads us to introduce the second interval partitioning algorithm:

Proportional Partition Algorithm. Let $|I|$ denote the length of the interval I and modify the tests in the operation of the algorithm to

$$If\ (\eta(IL) < \epsilon\,|\,IL\,|)$$

$$If\ (\eta(IR) < \epsilon\,|\,IR\,|)$$

It is seen that this modification leads to the sum of the $\eta(I)$ for $I \in M'$ to be ϵ times the length of the original intervals in M. Note that one is not forced to use a proportional partition algorithm in order to achieve a requirement on the sum of the $\eta(I)$, it simply makes it more convenient in some applications.

The result for the proportional partition algorithm is quite similar to that of Theorem 1. We have

Theorem 2. *Assume γ, $\beta < \frac{1}{2}$ and consider Proportional Partition Algorithm with β, M, and $\eta(I)$ for $I \in M$ specified. Let $F(\gamma, \epsilon)$ be the size of M' when the algorithm terminates and we have $F(\gamma, \epsilon) = \mathcal{O}(\epsilon^{1/\log_2 2\gamma})$.*

Proof. We follow the lines of the proof of Theorem 1 and abbreviate some of the argument. Consider a nondistinguished interval I_0; the condition for its descendants to be placed in M' is $\gamma^d \eta(I) < \epsilon\,|\,I\,| = \epsilon\,|\,I_0\,|\,2^{-d}$, and the rest of the argument is the same with γ replaced by 2γ.

For the initial distinguished interval I_0^* (of length $|\,I_0^*\,|$), we find d_0 determined as before except that β is replaced by 2β. The bound on the total number of intervals placed in M' from I_0^* may then be shown to be bounded by

$$2 + (\epsilon/\eta_0)^{1/\log_2 2\gamma} \sum_{k=0}^{\infty} ((2\beta)^{-1/\log_2 2\gamma})^k,$$

and the conclusion follows directly from this since the geometric series converges because we have γ, $\beta < \frac{1}{2}$.

Corollaries of Theorem 2 are valid analogous to those of Theorem 1. They are summarized as follows:

Corollary 1. *With the hypothesis of Theorem 2 and the Proportional Partition Algorithm replacing the Fixed Partition Algorithm, we have the conclusion of Corollaries 1, 2, and 3 of Theorem 1 valid provided $\log_2 \gamma$ is replaced by $\log_2 2\gamma$.*

The seemingly minor replacement of γ by 2γ in these results will have a significant consequence in the domain of effectiveness of adaptive quadrature algorithms.

6. *Convergence for Stack and Queue Algorithms*

In this section we return to the quadrature problem and present basic definitions and hypotheses concerning the integrand, the interval subdivision procedure, the error bounds (both true and calculated), and the algorithms for adaptive quadrature. Recall that we are to estimate $\int_0^1 f(x)\,dx$ and we have

Assumption 1. *Assume $f(x)$ has singularities $S = \{s_i \mid i = 1, 2, \cdots, R < \infty\}$ and set $w(x) = \prod_{i=1}^{R} (x - s_i)$. (i) If $x_0 \notin S$ then $f^{(p)}(x)$ is continuous in a neighborhood of x_0 with $p \geq 2$. (ii) There are constants K and α so that $|\,f^{(p)}(x)\,| \leq K\,|\,w(x)\,|^{\alpha - p}$.*

This assumption states that $f(x)$ has p continuous derivatives except for a finite number R of algebraic singularities.

For concreteness and simplicity of notation we assume that *the interval subdivision procedure always divides an interval into equal parts*. We denote by π_2 any partitioning of $[0, 1]$ that may be obtained by interval halving.

The quadrature algorithms compute an approximate area and an estimate of the error in this approximate area for each particular subinterval considered. The approximate area plays no role in this analysis, so we merely point out that it would probably be obtained from one of the classical quadrature rules. The error plays a central role in the analysis and one needs to distinguish between three quantities: the actual error of the approximate area, the estimate of this error used by the algorithm, and the bound on this estimate used in the analysis below. Note that all intervals in π_2 are of the form $[x, x + 2^{-k}]$, so we may use the pair (x, k) to specify any such interval. *The estimate of the error generated by the algorithm for the interval* $[x, x + 2^{-k}]$ *is denoted by* ERROR(x, k) (or, sometimes, simply by ERROR(x)). Certain assumptions are made below about bounds on ERROR(x, k). We occasionally say things like "ERROR(x, k) is divided by 2" when we actually mean to say that "the bound on ERROR(x, k) is divided by 2." The bound on the error is of course the only known quantity in the analysis and we believe that this practice does not introduce any ambiguity.

Every algorithm must have a way to relate the local error estimates to a global one as discussed in [1]. One may visualize that the algorithm has specified an error ϵ, and the question is how to distribute the error over the subintegrals generated. We consider two possibilities as indicated in Section 5. The first is called *fixed error distribution*, where the global error is simply the sum of the local errors. The second is called *proportional error distribution*, where the global error is the sum of the local errors weighted by the reciprocal lengths of the intervals involved. Surprisingly, there is a significant difference in the results obtained below for these two possibilities. We now formally state the assumption about the bounds on the error estimates.

ASSUMPTION 2. *Let s denote a point of singularity of $f(x)$ and set*

$$F_p(x, k) = \max_{t \in [x, x+2^{-k}]} |f^{(p)}(t)|.$$

There are constants p, K, and α (the same as in Assumption 1) so that:

(*i*) *$ERROR(x, k) \leq KF_p(x, k)2^{-k(p+1)}$ if $[x, x + 2^{-k}]$ contains no singularity.*
(*ii*) *$ERROR(x, k) \leq K2^{-k(1+\alpha)}$ if $s \in [x, x + 2^{-k}]$.*

We have now described those algorithm components (relevant to a convergence analysis) except the management of the interval collection. In this section we assume that each interval is in one of two boxes according to whether or not ERROR$(x, k) < \epsilon$ (fixed error distribution), ERROR$(x, k) < \epsilon 2^{-k}$ (proportional error distribution) One may think of these boxes as containing "active" and "discarded" intervals and the algorithm merely chooses (by any manner whatsoever) an active interval, processes it, and returns the resulting intervals to the appropriate boxes. We refer to this as a *2-box algorithm.*

The analysis presented below actually investigates the number of interval processings required to empty the active box or, alternatively, to reduce the global error to a certain level. Convergence results for quadrature are not naturally stated in these terms, and we wish to bound the quadrature error in terms of the number N of evaluations of the integrand $f(x)$. In order to do this we have

ASSUMPTION 3. *The integrand $f(x)$ is evaluated at most q times in the processing and subdivision of a single interval.*

This approach to measuring the work or complexity of the quadrature problem omits all of the overhead and auxiliary computations of the algorithm. In at least one instance this overhead may constitute the bulk of the actual computations required.

We now define the four types of algorithms considered in this analysis for adaptive quadrature.

Definition 1. We call an adaptive quadrature algorithm an ordered list, stack, queue, or boxes algorithm if:

(i) The indicated data structure is used for the interval collection.

(ii) The interval processor bounds satisfy Assumption 2, and either the fixed or proportional error distribution is used.

(iii) Assumption 3 is satisfied.

(iv) The interval processor divides the length of an interval by at most m, where m is a fixed constant.

This definition does not entirely specify an algorithm, but it serves to collect the principal attributes together.

A stack algorithm is not operational without a discard procedure, and we have

Definition 2. A discard procedure is said to be interval dependent if it depends only upon the interval and associated quantities and is independent of how the interval was generated. A discard procedure is said to be bound dependent if the interval $[x, x + 2^{-k}]$ is discarded from the interval collection whenever

$$d\,(\mathrm{ERROR}(x, k)) < \epsilon \quad \text{or} \quad d\,(\mathrm{ERROR}(x, k)) < \epsilon 2^{-k},$$

where ϵ is a constant and $d(t)$ is a monotone increasing real valued function of t.

Most, but not all, discard procedures in current use are both interval dependent and bound dependent. A stack algorithm is thus a special case of a two-box algorithm where the stack is used to specify the order for selecting intervals from the active box. We have then as a direct consequence of Theorems 1 and 2 and their corollaries:

THEOREM 3. *Consider a stack algorithm with a bound dependent discard procedure and let $f(x)$ satisfy Assumption 1. Then with $\alpha > -1$ and a fixed error distribution or with $\alpha > 0$ and a proportional error distribution we have as $N \rightarrow \infty$, $|If - Q_N f| \leq \mathcal{O}(1/N^p)$.*

PROOF. We note that the interval partition algorithms (fixed or proportional) are initiated with $M = [0, 1]$ and the distinguished intervals are those which contain a singularity. One singularity may produce two distinguished intervals if it is an endpoint of a subdivision, but, in any case, there is a fixed maximum number of distinguished intervals.

The numbers associated with the intervals are governed by Corollary 2 when

$$g[x, x + 2^{-k}] \quad \text{is} \quad F_p(x, k)$$

as defined in Assumption 2. The values of β and γ are $\gamma = 2^{-(p+1)}$ and $\beta = 2^{-(\alpha+1)}$. Thus for the fixed error distribution we find from Theorem 1 that the number D of discarded intervals is the order of $\epsilon^{-1/(p+1)}$. The number of intervals generated is at most $2D$ and thus the number N of function evaluations satisfies, for some constant C,

$$N \leq 2Cq\epsilon^{-1/(p+1)}.$$

On the other hand, the total integration error satisfies

$$|If - Q_N f| \leq D\epsilon = \mathcal{O}(\epsilon^{1-1/(p+1)}) = \mathcal{O}(1/N^p),$$

which establishes the first part of the conclusion. Note that $\beta < 1$ merely requires $\alpha > -1$.

For the proportional error distribution we have from Theorem 2 that D is the order of $\epsilon^{-1/p}$ and hence N satisfies $N \leq 2Cq\epsilon^{-1/p}$. The total integration error is just ϵ so we have $|If - Q_N f| \leq \epsilon = \mathcal{O}(1/N^p)$. The condition $\beta < \frac{1}{2}$ requires that $\alpha > 0$ and the proof is complete.

There can be a close relationship between a stack algorithm and a queue algorithm with discards.

LEMMA 1. *Consider a stack algorithm and a queue algorithm which are identical except for the data structure used for the interval collection. Assume that the queue algorithm does* *not terminate until all intervals are discarded. If the discard procedure is interval dependent, then the two algorithms generate the same set of intervals.*

PROOF. We establish this lemma by induction on the sets S_k of intervals generated of length 2^{-k}. Recall that the original interval is of length 1. It is clear that S_0 is the same for both algorithms.

The induction hypothesis is that the sets S_k are identical for algorithms and we are to show that this implies that the sets S_{k+1} are identical. We observe that all the members of S_{k+1} come from the processing of intervals in S_k and the assumptions imply for each algorithm that every interval in S_k is processed. Thus if $[x, x + 2^{-k}]$ is in S_k, the interval processor generates $[x, x + 2^{-(k+1)}]$ and $[x + 2^{-(k+1)}, x + 2^{-k}]$. Each of these two intervals must either be discarded or placed in S_{k+1}. The discard procedure is interval dependent so the decision made is independent of the algorithm. This establishes that the sets S_{k+1} are identical, which completes the induction step and the proof.

COROLLARY. *If the assumption in Lemma 1 on the termination of the queue algorithm is removed, then the queue algorithm generates a set of intervals which is a subset of those generated by the stack algorithm.*

We now have as a direct consequence of Theorem 3 and Lemma 1:

THEOREM 4. *Consider a queue algorithm with a discard procedure which is bound and interval dependent and let $f(x)$ satisfy Assumption 1. Then with $\alpha > -1$ and a fixed error distribution or with $\alpha > 0$ and a proportional error distribution we have as $N \to \infty$, $|If - Q_nf| \leq \Theta(1/N^p)$.*

7. *Convergence for Ordered List and Boxes Algorithms*

We begin with

THEOREM 5. *Consider an ordered list algorithm and let $f(x)$ satisfy Assumption 1. Then with $\alpha > -1$ and a fixed error distribution or with $\alpha > 0$ and a proportional error distribution we have, as $N \to \infty$, $|If - Q_nf| \leq \Theta(1/N^p)$.*

PROOF. These algorithms have no boxes naturally associated with them, but they may be viewed as defining an infinite sequence of boxes. Let $\epsilon > 0$ be given and consider the case of fixed error distribution. The first time that the head of the list has ERROR$(x) \leq \epsilon$ we have from Theorem 1 that the length of the list (which corresponds to M') is the order of $1/\epsilon^{(p+1)}$ (since $\log_2\gamma = -p - 1$). As in the proof of Theorem 3 we then have the conclusion established for this case.

Note that when we use the proportional error distribution, this proportioned error is used to order the list. In this case the first time the head of the list has ERROR$(x, k) < \epsilon \cdot 2^{-k}$ we have from Theorem 2 that the length of the list is the order of $\epsilon^{-1/p}$. We use the reasoning of the proof of Theorem 3 to complete the proof.

The boxes data structure is designed to behave similarly to the ordered list and thus one would expect to have similar results. However, one must prevent some of the boxes from being too large; that is the purpose of the next assumption.

ASSUMPTION 4. *Consider the box limits $BL(J)$, $J = 1, 2, \cdots$ for a boxes data structure. There is a constant t so that $0 < t \leq BL(J + 1)/BL(J) < 1$ for all J. Further, $\lim_{J\to\infty} BL(J) = 0$.*

We now have

THEOREM 6. *Consider a boxes algorithm that satisfies Assumption 4 and let $f(x)$ satisfy Assumption 1. Then with $\alpha > -1$ and a fixed error distribution or with $\alpha > 0$ and a proportional error distribution we have as $N \to \infty$, $|If - Q_Nf| = \Theta(1/N^p)$.*

PROOF. Given $\epsilon > 0$, let J be the index so that $BL(J + 1) \leq \epsilon < BL(J)$. Assumption 4 implies that such an index exists and further that $BL(J + 1) \geq t\epsilon$. We now apply Theorems 1 and 2 to the quantity $t\epsilon$. In the case of a fixed error distribution we find that N is the order of $(t\epsilon)^{-1/(p+1)}$.

The estimate $|If - Q_Nf| \leq NBL(J + 1) \leq N\epsilon$ holds and the proof is completed as before in this case. The case of proportional error distribution is similar and the proof is completed as before.

8. *Characteristic Lengths of Functions and Concrete Realizations of the Metalgorithm*

The perceptive reader will realize that algorithms cannot exist which correctly integrate all $f(x) \in C^p[0, 1]$ (or similar classes of functions), and thus a flaw must exist somewhere in the theorems of the preceding sections. The difficulty is that no algorithm can satisfy Assumption 2 about the error bounds for all such functions. This makes the theorems technically vacuous, though still of interest. Vacuous theorems may distress the reader, but recall that this is common in numerical analysis and the usual convergence results for, say, Simpson's Rule are also vacuous in the algorithmic sense. Results about Simpson's Rule involve a quantity (the fourth derivative of $f(x)$) which cannot in general be obtained by any algorithm and thus they imply nothing about the convergence of an actual computation using Simpson's Rule applied to an arbitrary member of $C^4[0, 1]$. That is to say that every sequence of numbers printed by a computer program is the sequence of Simpson's Rule estimates of the integral of some member of $C^4[0, 1]$.

The difficulty lies in the difference between convergence of a sequence and of an algorithm. For a sequence s_i to converge to S^* one must merely know, given $\epsilon > 0$, that at some index j we have $| s_i - S^* | < \epsilon$ for all $i \geq j$. We need not have any information about j other than that it exists. An algorithm must terminate and hence for an algorithm to converge to S^* one must know, given $\epsilon > 0$, how to terminate the algorithm so as to assure obtaining an element s_i with $| s_i - S^* | < \epsilon$.

The typical quadrature routine has a tolerance, say EPS, as input and terminates the computation whenever some computed bound falls below EPS. The naive user who needs an estimate to within ϵ would automatically accept the result produced but this is, of course, an invalid procedure for almost all routines known to this author.

The first objective of this section is to introduce a framework within which the naive user's natural approach is correct, viz. that one may terminate the computation whenever the error bound is less than a given value EPS. This can be accomplished only by modifying the algorithms or restricting thier domain of applicability (or both). We, of course, wish to do as little modification as possible and the approach used is to require one additional fact (number) about the integrand and to add one additional decision to the algorithm. We call this number a *characteristic length* $\lambda(f)$ of the integrand $f(x)$. This approach is illustrated generally by the flowchart in Figure 5. The terminology in Figure 5 is purposely nonspecific as the definition and use of $\lambda(f)$ varies from algorithm to algorithm.

The characteristic length need not be an actual length, but it is in the concrete realizations discussed later. In more traditional numerical quadrature methods it might be a bound on the magnitude of some appropriate high derivative. In the algorithms discussed in this paper the condition involving $\lambda(f)$ is simply: "Is the length of the interval being processed less than $\lambda(f)$?" Thus $\lambda(f)$ is simply such that the error bounds used are valid for *all* intervals of length less that $\lambda(f)$. This length is not necessarily related to an h-value so that $\mathcal{O}(h^k)$ terms are actually negligible in some Taylor's series expansion. This fact is illustrated by the three concrete realizations discussed later. It is important to note that $\lambda(f)$ is a quantity that *cannot* be determined by an algorithm which solely involves the analysis of values of $f(x)$. One can, of course, in many instances determine $\lambda(f)$ by other means.

The required modifications in the algorithm definition are as follows.

Definition 1A. We call an adaptive quadrature algorithm an ordered list (stack) (queue) (boxes) algorithm if the conditions of Definition 1 are satisfied and

(v) Quadrature error estimates are considered valid only when a valid test involving a characteristic length $\lambda(f)$ is satisfied.

(vi) The algorithm is terminated when the total error bound is less than a prescribed amount EPS.

(vii) $0 < t \leq BL(J + 1)/BL(J) < 1$ for a boxes algorithm.

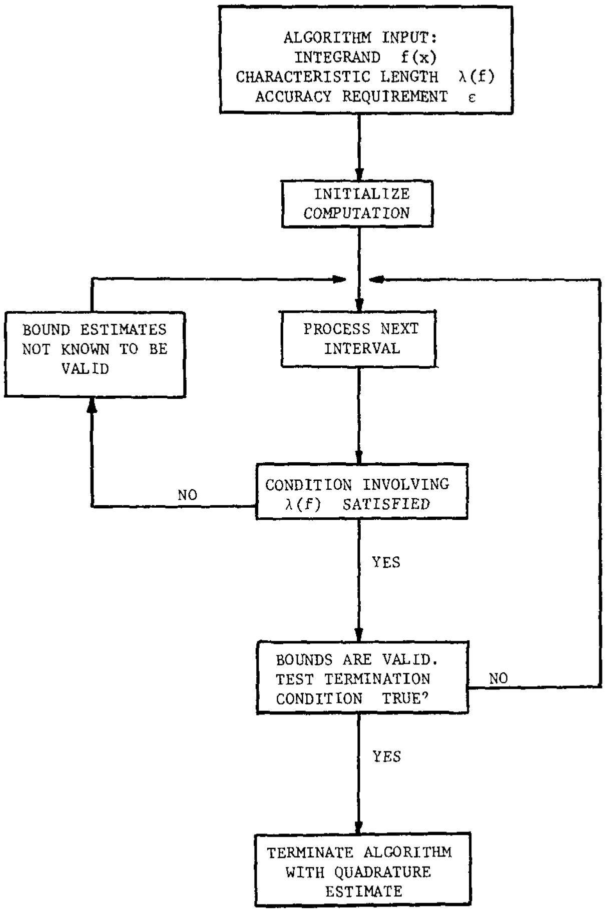

ALGORITHM INPUT:
INTEGRAND f(x)
CHARACTERISTIC LENGTH λ(f)
ACCURACY REQUIREMENT ε
INITIALIZE COMPUTATION
BOUND ESTIMATES NOT KNOWN TO BE VALID
PROCESS NEXT INTERVAL
CONDITION INVOLVING λ(f) SATISFIED
NO
YES
BOUNDS ARE VALID. TEST TERMINATION CONDITION TRUE?
NO
YES
TERMINATE ALGORITHM WITH QUADRATURE ESTIMATE

FIG. 5 Illustration of the general approach to reconcile the inconsistency between asymptotic theorem and algorithm behavior

(viii) The discard procedure is bound dependent for a stack algorithm.

(ix) The discard procedure is bound and interval dependent for a queue algorithm.

The resulting modification in the general convergence theorem is then:

THEOREM 7. *Consider an algorithm as defined in Definition 1A and let $f(x)$ satisfy Assumption 1. Let $\lambda(f)$ be a characteristic length valid for the algorithm considered and let the algorithm terminate when the total error bound is less than a prescribed value $EPS > 0$. Then with $\alpha > -1$ and a fixed error distribution or with $\alpha > 0$ and a proportional error distribution we have as $N \to \infty$ or, equivalently, as $EPS \to 0$, $|If - Q_N f| \leq EPS \leq \mathcal{O}(1/N^p)$.*

The exact assumptions in this theorem have been relegated to the statements of the assumptions and definitions because of the many cases involved. The key question at this point is: How much more restrictive are these hypotheses than those of Theorems 3–6? The restriction on the algorithms are superficial in that another component is added which apparently does not change the basic nature of the algorithm. It simply insures that the algorithm does not terminate prematurely.

The restriction on the domain of functions may be more than superficial. In the first concrete realization discussed below the effect is to limit the domain to functions with a finite number of jump discontinuities, cusps, and inflection points. In this case even though $f(x) = x^t \sin(1/t)$ satisfies Assumption 1 for $t > 2$, it is excluded from this theorem because $\lambda(f) = 0$ is *not* valid for the algorithm since the intervals would never have length less than or equal to $\lambda(f)$. On the other hand there is no effect whatsoever for the second concrete realization considered. There is a valid $\lambda(f)$ for every function which satisfies Assumption 1. The third algorithm is a quite complex (and realistic) example where it is difficult to isolate the effect. It would appear that there is a valid $\lambda(f)$ for every function which satisfies Assumption 1, but that remains to be verified.

8.1. First Concrete Realization: A Simple Adaptive Trapezoidal Rule. This algorithm has been proposed for educational rather than practical use as described in [10]. The trapezoidal rule is used and error bounds are determined as illustrated geometrically in Figure 6. It is seen that there are four distinct situations depending on whether $f(x)$ is convex, is concave, has an inflection point, or has a cusp in the interval being processed. One may determine the situation by examining the behavior of the sign of the angles between adjacent line segments. This sign is constant over a convex or concave curve segment and makes a simple change near an inflection point. At a cusp it changes sign twice in succession (i.e. only one angle has a sign opposite from its neighbors.) This detection process is valid whenever there are at least two points between any cusp, inflection point, or end of the interval. Thus if the lengths of the intervals are restricted to be less than one-fifth the separation of such points, one is assured of detecting them and obtaining a valid bound on the size of the function in order to form the fourth side of the quadrilateral.

It is a straightforward exercise in trigonometry to derive formulas for the areas involved in these bounds and to verify that Assumption 2 is satisfied for $p = 2$. Let the characteristic length $\lambda(f)$ be one-fifth of the minimum separation between inflection points, cusps, and the endpoints of the intervals and let $d(f)$ be the maximum magnitude of $f(x)$ on $[0, 1]$. The condition to be tested for the validity of an error bound is: "*Are the lengths of the current interval and its two neighbors less than $\lambda(f)$?*" The value $d(f)$ must be incorporated as input to the algorithm when cusps are present. These specifications give the first two components (interval processor and bound estimator.) A concrete realization of the metalgorithm is then obtained by selecting no special behavior component and any one of the interval collection management components as specified in Definition 1A. For any such selection we have

Corollary 1. *Let $f(x)$ satisfy Assumption 1 and have $\lambda(f) > 0$. Then the conclusion of Theorem 7 is applicable to the adaptive trapezoidal rule described in* [10].

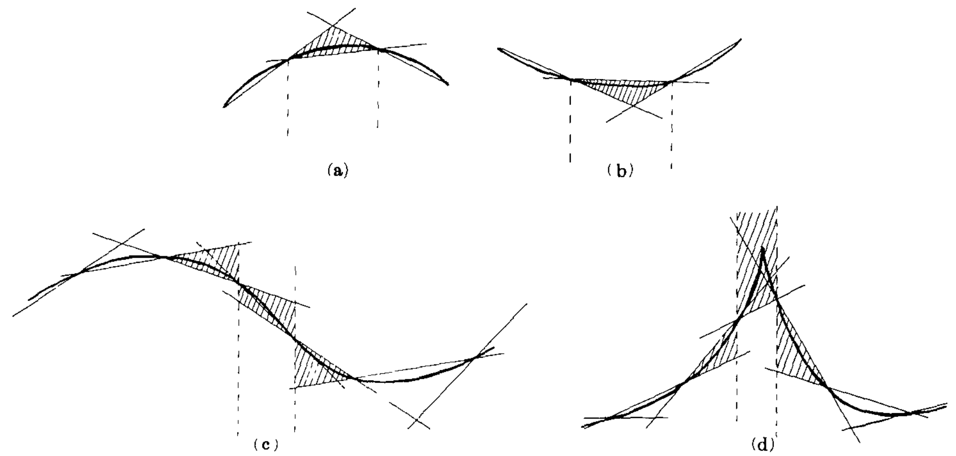


Fig. 6. The four situations for bounding the error: (a) convex, (b) concave, (c) inflection point, (d) cusp In each one the shaded region (triangle or quadrilateral) provides a bound on the error in the trapezoidal rule estimate of the area under the curve.

8.2 Second Concrete Realization: SQUANK. This algorithm is a specific Fortran program [8] for adaptive Simpson's Rule quadrature. The components are:

Interval processor: Simpson's Rule applied to a whole interval and each half.

Bound estimator: The interval $[x_1, x_5]$ is bisected by x_3 and EST, EST1, and EST2 are the Simpson's Rule estimates for the intervals $[x_1, x_5]$, $[x_1, x_3]$, and $[x_3, x_5]$, respectively. The error bound estimate is $\text{ERROR}(x_1) = |\text{EST} - \text{EST1} - \text{EST2}|/15$.

Special behavior: There is an elaborate mechanism to detect the presence of roundoff in the values of $f(x)$ which allows one to request an accuracy with EPS $= 0$. This algorithm then produces a quadrature estimate consistent with the roundoff level in the evaluation of $f(x)$. This special behavior was one of the main objectives in the development of SQUANK.

Interval collection management: Stack.

We ignore the special behavior component as it is outside the scope of the present analysis and we, of course, suppress various machine- and Fortran-dependent considerations.

The algorithm as presented has no characteristic length input and therefore must tacitly assume some value for $\lambda(f)$. Assume $f(x) \in C^4[A, B]$ and one sees that, as $h = x_5 - x_1 \to 0$,

$$\text{true quadrature error on } [x_1, x_5] \sim \text{ERROR}(x_1).$$

Thus for any $f(x)$ there is a $\lambda(f)$ such that $x_5 - x_1 < \lambda(f)$ implies that the ERROR(x) estimate is valid. The tacit assumption made in SQUANK is that $\lambda(f) \geq (B - A)/2$ where $[A, B]$ is the interval of integration.

Note that SQUANK is designed to take advantage of additional smoothness in $f(x)$. If we assume that $f(x) \in C^6[A, B]$ then, as $h \to 0$, we have:

$$\text{true quadrature error on } [x_1, x_5] \sim h^2 * \text{ERROR}(x_1).$$

This implies that the error estimate quickly becomes very conservative provided $f^{(6)}(x)$ is of reasonable behavior.

SQUANK is not designed to handle singularities, but experiments show it to be effective if $f(x)$ has a singularity like $|x - s|^\alpha$ for $\alpha > 0$. An examination of the program shows that once an interval is less than $(B - A)*2^{-30}$ in length then it is simply discarded with no change to the quadrature or error estimate. Thus when the integral (and error) over this short interval is negligible, SQUANK produces a valid quadrature estimate. SQUANK uses a proportional error distribution except for this special handling of small intervals. This discard procedure also strictly limits the size of the interval stack.

One cannot establish a convergence result for SQUANK beyond the obvious one where $f(x) \in C^4[A, B]$ and $\lambda(f) \geq (B - A)/2$. The domain of effectiveness of SQUANK can be enlarged by the following two modifications:

(i) Explicitly include the characteristic length $\lambda(f)$ in the algorithm. This would, for example, allow SQUANK to be effective for highly oscillatory functions where it now fails.

(ii) Include a cruder alternative error bound estimate which is of the order of h for an interval of length h. Such estimates may be made using the current interval processor which are valid for singularities like $|x - s|^\alpha$ for $\alpha > 0$.

If SQUANK is modified in these two ways then one can establish a general convergence theorem along the lines of Theorem 7.

8.3. Third Concrete Realization: CADRE This algorithm is also a specific Fortran program [2] whose name comes from "cautious adaptive Romberg extrapolation." This complex algorithm has proved to be both effective and efficient in a wide variety of applications. We consider a simplified version here. The algorithm components are:

Interval processor: Romberg extrapolation of trapezoidal sums in a T-table.

Bound estimator: A complex decision procedure which may accept estimates from various rows and columns of the T-table. More details are given below.

Special behavior: The bound estimator includes provisions to detect two types of singularities: jump discontinuities and $|x - s|^\alpha$, $\alpha > 0$ singularities. A special behavior analysis for roundoff effects is also present in rudimentary form.

Interval collection management: A modified stack. When an interval is subdivided into two parts, the order in which the parts are replaced on the stack alternates between left-right and right-left. This modification has no effect on the earlier proofs about stack algorithms, but it seems to give better efficiency in detecting interior singularities as they are bracketed.

The Romberg T-table has entry $T(I, J)$ in row I and column J. $T(I, 1)$ is the Ith trapezoidal sum and it involves $2^{(I-1)} + 1$ equispaced points and the other columns are the extrapolated values. The decision makes extensive use of certain ratios of differences of table values and they are stored in the upper triangular portion of the T array as follows·

$$T(J, I) = [T(I, J) - T(I - 1, J)]/[T(I + 1, J) - T(I, J)], \quad I > J.$$

The size of the T-table varies dynamically from a minimum of order 3 to a maximum of order 10, and the rightmost column is simply $T(I, J) - T(I - 1, J)$ rather than the ratio given above. There are three distinct cases in the decision procedure ($[A, B]$ is the current interval):

1. (Normal smooth function) The error in the $T(L, J + I)$ estimate has

$$\text{ERROR}(x) = |[T(L, J) - T(L - 1, J)]/(4^J - 1)|(B - A).$$

This error estimate is accepted only if the following two conditions are satisfied:

$$|T(J, L1)/4^J - 1| \leq .1 \quad \text{and} \quad |T(1, L1) - 4| \leq .15,$$

where $L1$ is the current size of the T-table.

2. (Jump discontinuity) The error in $T(L, 1)$ estimate has

$$\text{ERROR}(x) = (B - A)|T(L, 1) - T(L - 1, 1)|.$$

This error estimate is accepted only if $|T(1, L)| - 2 \leq .01$.

3. ($|x - s|^\alpha$ singularity) Let γ_i denote the ith term in the sequence obtained by merging $\alpha + 1, \alpha + 2, \alpha + 3, \ldots$ and $2, 4, 6, \ldots$. The estimate $T(LJ + 1)$ has

$$\text{ERROR}(x) = |[T(L, J) - T(L - 1, J)]/(2^{\gamma_J} - 1)|,$$

where the T-table is now a *modified Romberg* table defined by

$$T(L, J + 1) = T(L, J) + [T(L, J) - T(L - 1, J)]/(2^{\gamma_J} - 1).$$

This error estimate is accepted only if all six of the following conditions are satisfied:

$\lvert T(1, L) - T(1, L - 1)]/T(1, L - 1)\rvert \leq .1$	Ratios in first column agree
$T(1, L) \geq 1.1$	T-tables ratio not ridiculous
$1.1 \leq T(L - 2, L) \leq 4.5$	Ratio used to estimate α is in the middle range
$\lvert [T(L - 2, L) - T(L - 3, L)]/T(1, L)\rvert \leq .15$	Two best guesses for α agree
$T(J, L) \geq 0.85{*}2^{\gamma_J}$; $T(J, L - 1) \geq 0.85{*}2^{\gamma_J}$	Jth column coverages like h^{γ_J}

The estimate for α is derived from the relation $2^{\alpha+1} = T(L - 2, L)$. It is emphasized that this is a simplified description of the actual procedure in CADRE, and we ignore the exact logic used to select the values of L and J. Normally L is chosen as small as possible and J as large as possible consistent with the condition stated above. CADRE generally uses a proportional error distribution but, as in SQUANK, a modification is made for extremely small intervals which makes CADRE behave (in some instances at least) as if it used a fixed error distribution.

CADRE is a variable order method and thus it appears impossible to assign a particular value of p in Assumption 2. Clearly $p = 4$ is included because of the trapezoidal rule and the constants on the table size force CADRE to compute the Simpson Rule estimate which corresponds to $p = 4$. Suppose that $f(x) \in C^6[A, B]$. There appears to be no guarantee that a sixth-order quadrature estimate is made; CADRE might well select the Simpson's Rule estimate. Thus even though CADRE has the provisions to handle singularities, we are unable to directly apply the earlier theorems because of this variable order. While this variable order does present problems for an analysis, it may be the reason that CADRE is comparatively robust. That is, it performs well for a larger range of integrand types and accuracies. Variable order methods are known to have superior robustness for solving differential equations.

The characteristic length for CADRE is simple to state: $\lambda(f)$ *is that length so that* $B - A \leq \lambda(f)$ *implies that the above bound estimates are valid whenever the associated conditions are satisfied.* This does not, however, provide a great deal of insight into how one estimates $\lambda(f)$ for any particular function. CADRE tacitly assumes that $\lambda(f) \geq B - A$. One can obtain an estimate of $\lambda(f)$ by a standard Taylor's expansion analysis. A characteristic length derived in this way is sufficient, but not necessary, for CADRE to perform correctly. This approach would give a very small $\lambda(f)$ for a highly oscillatory $f(x)$, just as one obtains for SQUANK. However, these two algorithms perform quite differently for such functions. SQUANK generally fails but the cautious decision mechanism of CADRE normally detects the oscillatory behavior and allows CADRE to perform correctly. Incorrect results may be produced by a certain "resonance" [2] between the regular point selection of CADRE and the oscillations of $f(x)$.

We call the modified CADRE algorithm the result of introducing the characteristic length $\lambda(f)$ as input to CADRE and including a test of $\lambda(j)$ in the decision procedure for the bound estimates. We have then

COROLLARY 2. *Let* $f(x)$ *satisfy Assumptions* 1 *with* $p \geq 4$ *and have* $\lambda(f) > 0$. *Then the conclusion of Theorem* 7 *is applicable to the modified version of CADRE described here.*

An important open question is whether one obtains an order of convergence of (N^{-p}) if $f(x)$ satisfies Assumption 1 with $5 \leq p \leq 10$ (and $\lambda(f) > 0$).

ACKNOWLEDGMENTS. I wish to thank Carl de Boor and the referees for suggestions that considerably clarified the proofs in this paper.

REFERENCES

1 DE BOOR, C On writing an automatic integration algorithm. In *Mathematical Software*, J. R. Rice, Ed , Academic Press, New York, 1971, pp 201-209

2 DE BOOR, C CADRE. An algorithm for numerical quadrature In *Mathematical Software*, J. R. Rice, Ed , Academic Press, New York, 1971, Chap 7, pp 417-449

3 CASALETTO, J , PICKET, M , AND RICE, J A comparison of some numerical integration programs CSD TR 37, Purdue U , Lafayette, Ind , June 1969, also *SIGNUM Newsletter 4* (1969), 30-40

4 DAVIS, P , AND RABINOWITZ, P *Numerical Integration* Blaisdell, Waltham, Mass , 1967

5. KAHANER, D. K Comparison of numerical quadrature formulas In *Mathematical Software*, J R Rice, Ed , Academic Press, New York, 1971, pp 229-259

6. KUBIK, R Havie integrator, Algorithm 257 *Comm ACM 8*, 6 (June 1965), 381

7 LYNESS, J N Notes on the adaptive Simpson quadrature routine *J. ACM 16*, 3 (July 1969), 483-495

8 LYNESS, J N SQUANK (Simpson quadrature used adaptively-noise killed), Algorithm 379. *Comm ACM 13*, 4 (April 1970), 260-263

9 MCKEENMAN, W M. Adaptive numerical integration by Simpson's rule, Algorithm 145 *Comm. ACM 5*, 12 (Dec. 1962), 604.

10 RICE, J R An educational adaptive quadrature algorithm. CSD TR 90, Purdue U., Lafayette, Ind., March 1973, also *SIGNUM Newsletter 8* (1973), 27-41

RECEIVED APRIL 1973; REVISED MAY 1974