Local Versus Global Strategies for Adaptive Quadrature

MICHAEL A. MALCOLM and R. BRUCE SIMPSON

University of Waterloo

A study has been made of two alternative subinterval selection strategies for adaptive quadrature. The more commonly used is termed a local acceptance criterion (as in SQUANK), and the other is termed a global acceptance criterion (as in AIND). An efficient programming technique for the global alternative is given, and an analytic model for predicting the subinterval distributions selected by classes of adaptive algorithms is developed. To test the predictions of the model, the well-known algorithm SQUANK was reprogrammed to make it operate under the global criterion. Experiments with these routines were carried out comparing their actual performance with the predicted performance and with the performance of CADRE and AIND, and some conclusions are drawn.

Key Words and Phrases: adaptive quadrature, Simpson's rule, SQUANK, AIND, CADRE, numerical integration
CR Categories: 5.16

INTRODUCTION

In the past five years there have been a number of significant developments in adaptive quadrature methods for automatic numerical integration. Following a lull after McKeeman's original paper in 1963 [6], several algorithms have been published [1, 5, 7, 8] which differ in their local quadrature rules, strategies for subdividing the interval of integration, and criteria for termination. Some of these algorithms have special features for detecting roundoff error [5, 9], for detecting singular integrand behavior [1], or for returning an indication of the validity of results [1, 9]. Compared with one another empirically, these subroutines are found to require considerably different numbers of functions evaluations for the same integrands [7, 8, 9]. In general, there appears to have been little analysis of the performance of these algorithms with the notable exception of [11].

In this paper we attempt to understand some of these differences, focusing on the effects of the interval subdivision strategy and termination criterion.

1. INTERVAL SUBDIVISION STRATEGIES

In order to focus on the interval subdivision process, we shall regard an adaptive quadrature scheme as an algorithm for processing a sequence of subintervals $\{I_n\}$.

This work was supported in part by the National Research Council of Canada. A version of this paper was presented at Mathematical Software II, a conference held at Purdue University, West Lafayette, Indiana, May 29–31, 1974.
Authors' addresses: Department of Computer Science, University of Waterloo, Waterloo, Ont., Canada N2L 3G1.

The main components of an adaptive quadrature algorithm are:

(i) a local quadrature procedure for evaluating $Q(I_n) \approx \int_{I_n} f(x)\,dx$;
(ii) a method for calculating $E(I_n)$, an estimate for $|\, Q(I_n) - \int_{I_n} f(x)\,dx \,|$;
(iii) criteria for deciding which subinterval in the sequence $\{I_n\}$ to subdivide at each stage and for deciding when to terminate.

Using a user-specified absolute error tolerance, ϵ, the objective is to produce a number, *res*, for which $|\, res - \int_a^b f(x)\,dx \,| \leq \epsilon$.

We shall be particularly interested in adaptive Simpson's rules with components (i) and (ii) as used in SQUANK [5]. Using $S^{(m)}(I_n)$ to denote Simpson's rule applied m times to I_n, these can be expressed as

$$Q(I_n) = S^{(2)}(I_n) \quad \text{and} \quad E(I_n) = |\, S^{(1)}(I_n) - S^{(2)}(I_n) \,|/15.$$

In this section, we will concentrate on component (iii).

The intervals of the sequence I_n fall into three categories: I_n has already been subdivided and discarded ($n \in Dis$), or has been accepted by the algorithm ($n \in Acc$), or is pending further examination ($n \in Pen$). A program implementing an adaptive quadrature algorithm must store the data associated with $\{I_n \mid n \in Pen\}$ using some data structure and can accumulate the contributions to *res* from I_n for $n \in Acc$ as soon as the acceptance is recognized. A discussion of the relationships between subdivision strategies and data structures for storing the *Pen* set has been published by Rice [10].

A common interval acceptance criterion, using $l(I_n)$ to denote the length of I_n, is

$$E(I_n) \leq l(I_n)\epsilon/(b - a). \tag{1.1}$$

An interval I_n is accepted if and only if it satisfies (1.1) (e.g. CADRE [1] and SQUANK [5]). Algorithms using (1.1) generally use a stack to store *Pen* and start with $I_1 = [a, b]$, $1 \in Pen$; at each stage, I_n (the top of the stack) either satisfies (1.1) and is accepted, or is bisected and its right and left halves (in that order) are placed on the stack.

Criterion (1.1) has the following features:

(i) The decision is based entirely on information available from I_n.
(ii) If the error estimate $E(I_n)$ is in fact a bound for the error in $Q(I_n)$ for every $n \in Acc$ when the algorithm terminates, then the user's tolerance is guaranteed to be met (assuming exact arithmetic).

On the other hand, the local error criterion (1.1) decreases linearly with the interval length, and hence is most stringent as a tolerance, in regions where the adaptive process is working at subdivision the hardest.

Clearly feature (ii) is a crucial one; however, feature (i) may be relaxed. One alternative, then, is to retain as pending all the intervals in $\{I_n\}$ which have not been discarded, so that $\bigcup_{m \in Pen} I_m = \{x \mid a \leq x \leq b\}$. Then we can use

$$\sum_{m \in Pen} E(I_m) \leq \epsilon \tag{1.2}$$

as an acceptance criterion for the entire pending set. If at some stage of the process,

(1.2) is not satisfied, we can determine $M \in Pen$ such that

$$E(I_M) \geq E(I_m) \quad \text{for all } m \in Pen, \tag{1.3}$$

and bisect I_M, adding its left and right halves to the pending set and rechecking (1.2). We shall refer to criterion (1.2) as a global acceptance criterion; (1.2) and (1.3) form the strategy of the subroutine SQUAGE discussed in Section 3. The subroutine AIND [8, 9] uses this strategy except for a slight modification of (1.2) to account for an additional user-specified relative tolerance.

The intention of the global strategy (1.2) and (1.3) is to select subintervals so that the local errors are roughly equal in magnitude, rather than scaled by the length of the subintervals. Thus algorithms using global strategies should work no harder on subintervals where the integrand is difficult to integrate than on subintervals where it is easy. Hence a global criterion has the potential both for reducing the number of subintervals used and for generating a large list of data associated with the pending set. In this sense it appears to be a space-time tradeoff. In Section 2 we shall outline an analytic predictive technique for helping to quantify this tradeoff.

2. PREDICTION OF SUBINTERVAL DISTRIBUTIONS

In this section we develop a technique for predicting the subinterval distributions resulting from the strategies of Section 1. We then examine a simple example to illustrate the expected reduction in the number of subintervals resulting from the global acceptance criterion (1.2) instead of the local one (1.1). We conclude with a discussion of the difference in performance between these strategies when applied to functions with endpoint singularities.

Our discussion is based on the specific quadrature rules and estimates employed in SQUANK [5], and a modification of SQUANK (discussed in Section 3) to use the global acceptance criterion. However, it will be apparent that the technique has more general applicability. As mentioned in Section 1, SQUANK uses

$$Q(I_n) = S^{(2)}(I_n) \tag{2.1}$$

and

$$E(I_n) = | S^{(1)}(I_n) - S^{(2)}(I_n) |/15. \tag{2.2}$$

This estimate is based on the first term of the asymptotic expansion of the error in a power series in h [4]. In fact

$$E(I_n) = l(I_n)^5 | f^{iv}(c) |/(4^4 \cdot 180), \quad c \in I_n. \tag{2.3}$$

Moreover, $15E(I_n)$ is known to be a (sharp) bound on the error if $f^{iv}(x)$ has constant sign for $x \in I_n$ [12].

SQUANK employs the local interval acceptance criterion

$$E(I_n) \leq l(I_n)\epsilon/(b - a) \tag{2.4}$$

[(1.1) of Section 1]. Suppose that for a given integrand, $f(x)$, SQUANK selects intervals with endpoints $x_i : a = x_0 < x_1 < x_2 < \cdots < x_{L-1} < x_L = b$. Since each subinterval is acceptable under (2.4), we can conclude that for $i = 0, 1, \ldots, L - 1$,

$$(x_{i+1} - x_i)^5 | f^{iv}(c_i) | (4^4 \cdot 180) = \theta_i(x_{i+1} - x_i)\epsilon/(b - a), \tag{2.5}$$

where $x_i < c_i < x_{i+1}$ and $0 \leq \theta_i < 1$. Assuming that $f^{(iv)}(c_i) \neq 0$, this can be written

$$x_{i+1} = x_i + h_i\, 4(180)^{1/4}/|\, f^{iv}(c_i)\,|^{1/4}; \qquad x_0 = a, \tag{2.6}$$

for $h_i = (\theta_i \epsilon/(b-a))^{1/4}$, $i = 0, \ldots, L-1$. We can regard (2.6) as a rather crude single-step method for solving the initial-value problem

$$dx(s)/ds = 4(180)^{1/4}/|\, f^{iv}(x(s))\,|^{1/4}; \qquad x(0) = a, \tag{2.7}$$

using step sizes h_i. The variability of h_i due to the unknown fraction θ_i complicates the connection between (2.7) and (2.6). We shall assume for $i = 0, \ldots, L-1$,

(i) that $\frac{1}{16} \leq \theta_i \leq 1$ on the hypothesis that if $\theta_i < \frac{1}{16}$, a parent interval of $[x_i, x_{i+1}]$ would have been accepted;
(ii) that θ_i can be replaced by the mean of a uniform distribution over its range, i.e. $\frac{17}{32}$.

Hence, we replace h_i by $h = (\frac{17}{32})^{1/4}(\epsilon/(b-a))^{1/4}$.

If $x(s)$ is the solution of (2.7), then we might expect the approximation $x_i \approx x(ih)$, $i = 0, 1, \ldots, L$, to improve as ϵ approaches zero, subject to the validity of assumptions (i) and (ii) above. In particular, defining s^* to be the parameter value for which $x(s^*) = b$, we will use $Lpred(f; a, b, \epsilon) = \lfloor s^*/h \rfloor \approx \lfloor (32(b-a)/17)^{1/4} s^*/\epsilon^{1/4} \rfloor$ as a prediction for the number of subintervals selected by SQUANK. The predicted number of function evaluations required by SQUANK is given by $4\, Lpred(f; a, b, \epsilon) + 1$.

For example, consider $f(x) = (1 + c - x)^{-1}$ for positive parameter c and $a = 0$, $b = 1$. In this case (2.7) is

$$dx(s)/ds = 4(\tfrac{15}{2})^{1/4}(1 + c - x(s))^{5/4}; \qquad x(0) = 0,$$

with solution

$$(1 + c - x(s))^{-1/4} - (1 + c)^{-1/4} = (\tfrac{15}{2})^{1/4} s.$$

Substituting $x(s^*) = b = 1$ gives $s^* = (\frac{2}{15})^{1/4}(c^{-1/4} - (1 + c)^{-1/4})$, and

$$Lpred(f; 0, 1, \epsilon) = \lfloor (\tfrac{64}{2255})^{1/4}(c^{-1/4} - (1 + c)^{-1/4})\, \epsilon^{-1/4} \rfloor. \tag{2.8}$$

Experiments with this example are discussed in Section 4.

We now turn to predicting the subinterval distributions produced by the global acceptance criterion of Section 1. Consider the modification of SQUANK to be an algorithm using (2.1) and (2.2) and which accepts the entire pending set of subintervals if

$$\sum_{m \in Pen} E(I_m) \leq \epsilon \tag{2.9}$$

[using (1.2) and (1.3) of Section 1]. Suppose for a given integrand $f(x)$, this algorithm selects subintervals with endpoints $y_i: a = y_0 < y_1 < \cdots < y_{G-1} < y_G = b$. Then from (2.3) we can see that the error estimate for $[y_i, y_{i+1}]$ can be expressed as

$$(y_{i+1} - y_i)^5\, |\, f^{iv}(d_i)\,|/(4^5 \cdot 45), \quad y_i < d_i < y_{i+1},$$

and we can define $\theta_i \geq 0$ $(i = 0, \ldots, G-1)$ by

$$(y_{i+1} - y_i)^5\, |\, f^{iv}(d_i)\,|/(4^5 \cdot 45) = \theta_i \epsilon. \tag{2.10}$$

Using (2.9) we can conclude that

$$\sum_{i=0}^{G-1} \theta_i \leq 1. \tag{2.11}$$

As previously, if $f^{iv}(d_i) \neq 0$, we can rewrite (2.10) as

$$y_{i+1} = y_i + h_i\, 4(45)^{1/5}/|\, f^{iv}(d_i)\,|^{1/5}; \qquad y_0 = a, \tag{2.12}$$

where $h_i = (\theta_i \epsilon)^{1/5}$, $i = 0, 1, \ldots, G-1$. Equation (2.12) can be interpreted as a numerical method for solving

$$dy(t)/dt = 4(45)^{1/5}/|\, f^{iv}(y(t))\,|^{1/5}; \qquad y(0) = a, \tag{2.13}$$

using variable step sizes h_i. We shall again make the crude assumption that

(iii) θ_i can be replaced by some constant (average) value and from (2.11) we select $1/G$ for this constant.

Hence, we replace h_i by $h = (\epsilon/G)^{1/5}$. If $y(t)$ is the solution of (2.13), we might then expect the approximation $y_i \approx y(ih)$, $i = 0, 1, \ldots, G$, to improve as ϵ approaches zero and to be subject to assumption (iii) in particular; if we define t^* to be the parameter value for which $y(t^*) = b$, we can determine the predicted number of subintervals $Gpred(f; a, b, \epsilon)$ using $t^* = Gpred\, h = Gpred\, \epsilon^{1/5}/Gpred^{1/5}$. This gives

$$Gpred(f; a, b, \epsilon) = \lfloor t^{*5/4}/\epsilon^{1/4} \rfloor. \tag{2.14}$$

(Note that we have chosen the notation L and $Lpred$ to indicate local strategy; G and $Gpred$ indicate global strategy.)

Returning to the example $f(x) = (1 + c - x)^{-1}$, $a = 0$, $b = 1$, we have for (2.13)

$$dy(t)/dt = 4(\tfrac{1}{8}^5)^{1/5}(1 + c - y(t)); \; y(0) = 0,$$

with solution

$$\log_e ((1 + c)/(1 + c - y(t)) = 4(\tfrac{1}{8}^5)^{1/5} t \quad \text{and} \quad t^* = (\tfrac{8}{1}^5)^{1/5} \log_e ((1 + c)/c)/4.$$

Using (2.14), we predict for the number of subintervals

$$Gpred(f; 0, 1, \epsilon) = \lfloor 2^{-7/4} 15^{-1/4} (\log_e ((1 + c)/c))^{5/4}/\epsilon^{1/4} \rfloor.$$

Comparing this with (2.8), we see that the local acceptance criterion yields a predicted number of subintervals which is $O(c^{-1/4})$ as c approaches zero, whereas the global one yields a predicted number of subintervals which is $O(|\log c|^{5/4})$.

These techniques predict a sharp difference between the results of using the local and global acceptance criteria on integrands with endpoint singularities. If we consider $f(x) = x^\alpha$, $a = 0$, $b = 1$, we find for the local acceptance criterion (2.4) that a finite number of subintervals is predicted only if $\alpha > 0$. That is, using $p_4(\alpha) = |\, \alpha(\alpha - 1)(\alpha - 2)(\alpha - 3)\,|$, $Lpred(f; 0, 1, \epsilon) = \lfloor (p_4(\alpha)/180\epsilon)^{1/4}/\alpha \rfloor$. However, the global strategy predicts a finite number of subintervals for $\alpha > -1$, i.e.

$$Gpred(f; 0, 1, \epsilon) = \lfloor (\tfrac{5}{4}(1 + \alpha))^{5/4} (p_4(\alpha)/45\epsilon)^{1/4} \rfloor.$$

Experiments confirming this difference are discussed in Section 4. The ability of adaptive schemes using the local acceptance criterion to handle $f(x) = x^\alpha$ for $\alpha > 0$ has also been observed by Rice [10], in which a convergence theorem for an adaptive trapezoidal rule is given.

3. PROGRAMMING THE GLOBAL STRATEGY

The global acceptance strategy (1.2) and (1.3) has been implemented in a program written in ANSI Standard Fortran. The technique used to store information associated with the pending set of intervals will be discussed in this section. The Fortran program is presented at the end of this section.

As discussed in Section 1, the global strategy requires that all intervals be kept in the pending set until the acceptance criterion

$$\sum_{m \in Pen} E(I_m) \leq \epsilon \tag{3.1}$$

is satisfied. At each stage of the process prior to satisfying condition (3.1), the program must determine an $M \in Pen$ such that

$$E(I_M) \geq E(I_m) \quad \text{for all } m \in Pen. \tag{3.2}$$

This could be accomplished by searching the entire list of subintervals in *Pen* at each stage. However, this would require $n - 1$ floating-point comparisons at each stage, where n is the number of subintervals currently in the list. If the list is kept sorted in decreasing order of the error estimates, then no comparisons are required for determining M, but more work is required for inserting new subintervals into the list. A bubble sort at each stage would require up to n floating-point comparisons for each new subinterval added to the list. (This is the technique used by AIND [8, 9].) If a binary search is used to determine where to insert the new subintervals, then only $\lceil \log_2 n \rceil$ comparisons are needed for each new subinterval, but up to n items in the list will need to be moved to different memory locations. Using N to denote the value of n when (3.1) is satisfied, we have a total of $\sum_{n=1}^{N-1} (n - 1) = N^2/2 - 3N/2 + 1$ comparisons required for the first of the above methods. The other techniques require similar [i.e. $O(N^2)$] amounts to work. [An estimate for N was given in (2.14).] For integrands for which N becomes large (say greater than 10^3), these comparisons (or memory fetches and stores) may require a major portion of the computation time.

Fortunately, a completely ordered list is not necessary for determining M in (3.2). It is sufficient to store the subintervals in a partially ordered binary tree, sometimes called a "heap." Each node of the binary tree (illustrated in Figure 1) contains the information for one of the subintervals. The nodes are *partially ordered* in the sense that the error estimate E for each node is no less than that for each of its subnodes: i.e. $E(I_{t_i}) \geq E(I_{t_{2i}})$ and $E(I_{t_i}) \geq E(I_{t_{2i+1}})$, $i = 1, 2, \ldots$. Hence, at each stage M is simply t_1, the root node (see Figure 1).

The interval subdivision process consists of removing and bisecting the root node subinterval and inserting the data for the two resulting subintervals into nodes of the tree. It is important that the new data be inserted into the tree so as to keep it balanced, i.e. the only vacant nodes occur at the lowest level. This is achieved by

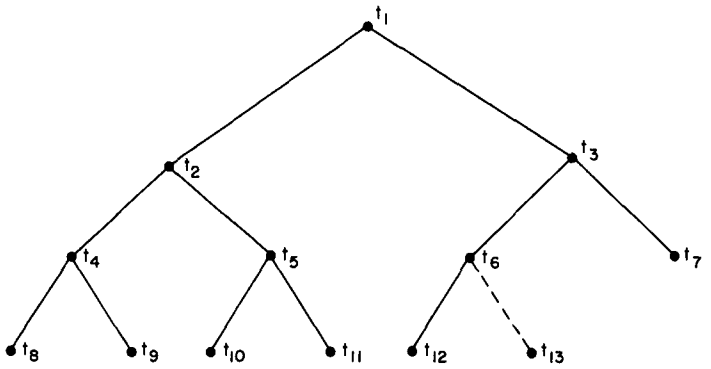

Fig. 1. A balanced binary tree for $n = 12$

inserting the newly created subinterval having the largest error estimate into the (just vacated) root node; the data for the other new subinterval is placed in the left-most vacant node in the lowest level of the tree. (In Figure 1 this would be t_{13}.) After each insertion, the new node must be repositioned in the tree (if necessary) to maintain partial ordering of the tree. Each new node may be moved up to $\lfloor \log_2 n \rfloor$ levels in the tree during this repositioning. The new root node may require up to $2\lfloor \log_2 n \rfloor$ comparisons as it moves down the tree, and the new node inserted at the lowest level in the tree may require at most $\lfloor \log_2 n \rfloor$ comparisons. This gives at most

$$\sum_{i=2}^{N-1} (2\lfloor \log_2 i \rfloor + \lfloor \log_2 (i+1) \rfloor) \leq 2 \log_2 (N-1)! + \log_2 (N!/2) \leq 3N \log_2 N$$

total comparisons. The use of a partially ordered binary tree has reduced the number of comparisons and memory references from $O(N^2)$ to $O(N \cdot \log N)$. Using this technique, the global strategy requires little computational overhead for very large values of N.

The idea of a partially ordered binary tree was used in a well-known sorting algorithm due to Floyd [2]. (See Knuth [3] for a discussion of Floyd's tree sort algorithm.) It is a useful data structure for implementing other algorithms of numerical mathematics, e.g. Jacobi's method and Gauss-Southwell relaxation.

The binary tree is implemented in the Fortran subroutine SQUAGE (Simpson's quadrature used adaptively global error) using the integer array T. Nodes are inserted in the tree by the subroutine INSERT. New root nodes are moved down the tree (if necessary) by the subroutine BUBLDN; other new nodes are positioned in the tree by the subroutine BUBLUP. Each node in the tree contains the following information about its subinterval:

X1	left endpoint of the subinterval
L	length of the subinterval
F1, F2, F3, F4, F5, F6	function values at equally spaced abcissas (left to right) throughout the interval
ERREST	the (scaled) error estimate for the subinterval.

SQUAGE

```
      REAL FUNCTION  SQUAGE(A, B, EPS, ERR, NO, FUN)
      INTEGER  NO
      REAL  A, B, EPS, ERR, FUN
      EXTERNAL  FUN
C
C  SQUAGE -- SIMPSON*S QUADRATURE USED ADAPTIVELY WITH A BINARY
C           TREE OF ERROR ESTIMATES
C
C  SQUAGE - FIFTH-ORDER ESTIMATE OF INTEGRAL OF FUN(X) FOR
C           A.LT.X.LT.B
C  EPS    - REQUESTED ACCURACY (ABSOLUTE)
C  ERR    - ESTIMATE OF ACHIEVED ERROR
C  NO     - NUMBER OF FUNTION EVALUATIONS USED
C  FUN    - REAL FUNCTION OF ONE REAL VARIABLE TO BE INTEGRATED
C
C
C  DECLARE BINARY TREE.
C
      REAL  X1(1000),L(1000),F1(1000),F2(1000),F3(1000),
     2 F4(1000),F5(1000),ERREST(1000)
      INTEGER  T(1000)
C
C  X1 - LEFT END POINT OF INTERVAL
C  L  - LENGTH OF INTERVAL
C  F1-F5 - FUNCTION VALUES AT POINTS DELIMITING THE QUARTERSECTIONS OF
C          THE INTERVAL
C  ERREST - SCALED ERROR ESTIMATE FOR INTERVAL
C  T - USED FOR INDEXING THE OTHER ARRAYS
C
C  OTHER VARIABLES.
C
      REAL  L2, SEPS, SUM, ERR5
C
C  L2 - HALF THE LENGTH OF THE ROOT INTERVAL
C  SEPS - EPS SCALED BY 180. FOR ERROR TEST
C  SUM - USED FOR ACCUMULATING TOTAL ESTIMATE OF INTEGRAL
C  ERR5 - USED FOR ACCUMULATING SCALED FIFTH-ORDER CORRECTION TERM
C
      INTEGER  N, J, T1
C
C  N - THE CURRENT NUMBER OF INTERVALS
C
      REAL  XI1, FI1, FI3, FI5
C
C  XI1,...,FI5  ARE USED FOR PASSING PARAMETERS THAT ARE ALSO
C  IN COMMON
C
      COMMON /SQUA/ X1, L, F1, F2, F3, F4, F5, ERREST, ERR5, T, N
      SEPS = 180.*EPS
      N = 1
      T(1) = 1
      ERR5 = 0.
      CALL INSERT(T(1),A,B-A,FUN(A),FUN(0.5*(A+B)),FUN(B), FUN)
C
C FORCE AT LEAST ONE SUBDIVISION (9 FUNCTION EVALUATIONS)
C
      GO TO 101
 100  IF (DABS(ERR5).LE.SEPS) GO TO 200
C
C  SUBDIVIDE TOP NODE OF TREE
C
 101  T1 = T(1)
      ERR5 = ERR5 - ERREST(T1)
      L2 = 0.5*L(T1)
      N = N+1
```

SQUAGE (continued)

```
      T(N) = N
      XI1 = X1(T1)
      FI1 = F1(T1)
      FI3 = F2(T1)
      FI5 = F3(T1)
      CALL INSERT(T(N),XI1,L2,FI1,FI3,FI5,FUN)
      XI1 = X1(T1)+L2
      FI1 = F3(T1)
      FI3 = F4(T1)
      FI5 = F5(T1)
      CALL INSERT(T(1),XI1,L2,FI1,FI3,FI5,FUN)
      IF (DABS(ERREST(T1)).GE.DABS(ERREST(N))) GO TO 140
      T(1) = N
      T(N) = T1
 140  CALL BUBLUP
      CALL BUBLDN
      IF (N.LT.1000) GO TO 100
C
C NO CONVERGENCE, WRITE ERROR MESSAGE
C
      WRITE(6,150)
 150  FORMAT(26H MEMORY OVERFLOW IN SQUAGE)
C
C  THE ERROR TEST IS NOW SATISFIED, AND THE ESTIMATE CAN NOW BE SUMMED.
C
 200  SUM = 0.
      DO 300 J=1,N
         SUM = SUM + L(J)*(F1(J) + 4.*F2(J) + 2.*F3(J) + 4.*F4(J)
     2               + F5(J))
 300  CONTINUE
      SQUAGE = SUM/12. + ERR5/180.
      ERR = DABS(ERR5/180.)
      NO = 4*N + 1
      RETURN
      END




      SUBROUTINE  INSERT(TI, PX1, PL, PF1, PF3, PF5, FUN)
      INTEGER  TI
      REAL  PX1, PL, PF1, PF3, PF5, FUN
C
C  THIS SUBROUTINE SUBDIVIDES THE INTERVAL PASSED AS AN ARGUMENT TO
C  OBTAIN AN ERROR ESTIMATE, AND THEN INSERTS IT AS THE I-TH NODE IN
C  THE BINARY TREE.  TWO FUNCTION EVALUATIONS ARE PERFORMED DURING
C  EACH CALL OF THIS SUBROUTINE.
C
      REAL X1(1000),L(1000),F1(1000),F2(1000),F3(1000),
     2 F4(1000),F5(1000),ERREST(1000),ERR5
      INTEGER  T(1000), N
      COMMON /SQUA/ X1, L, F1, F2, F3, F4, F5, ERREST, ERR5, T, N
      X1(TI) = PX1
      L(TI) = PL
      F1(TI) = PF1
      F2(TI) = FUN(PX1 + 0.25*PL)
      F3(TI) = PF3
      F4(TI) = FUN(PX1 + 0.75*PL)
      F5(TI) = PF5
      ERREST(TI) = -PL*(F1(TI) - 4.*F2(TI) + 6.*F3(TI) - 4.*F4(TI)
     2             + F5(TI))
      ERR5 = ERR5 + ERREST(TI)
      RETURN
      END
```

SQUAGE (continued)

```
      SUBROUTINE  BUBLUP
C
C  THE ITEM IN NODE N OF THE TREE IS BUBBLED UP THE TREE TO
C  MAINTAIN PARTIAL ORDERING OF THE TREE.
C
      REAL  X1(1000),L(1000),F1(1000),F2(1000),F3(1000),
     2 F4(1000),F5(1000),ERREST(1000),ERR5
      INTEGER  T(1000), N
      COMMON /SQUA/ X1, L, F1, F2, F3, F4, F5, ERREST, ERR5, T, N
      INTEGER  I, J, ITEMP, TI, TJ
      I = N
 10   J = I/2
      IF (J.EQ.1) GO TO 100
      TI = T(I)
      TJ = T(J)
C
C COMPARE ITEM WITH PREDECESSOR NODE
C
      IF (DABS(ERREST(TI)).LE.DABS(ERREST(TJ))) GO TO 100
C
C IF NECESSARY, EXCHANGE WITH PREDECESSOR
C
      ITEMP = T(I)
      T(I) = T(J)
      T(J) = ITEMP
      I = J
      GO TO 10
 100  RETURN
      END

      SUBROUTINE  BUBLDN
C
C  THE ITEM IN THE ROOT NODE IS BUBBLED DOWN THE TREE TO
C  MAINTAIN PARTIAL ORDERING OF THE TREE.
C
      REAL  X1(1000),L(1000),F1(1000),F2(1000),F3(1000),
     2 F4(1000),F5(1000),ERREST(1000),ERR5
      INTEGER  T(1000), N, TI, TJ, TJ1
      COMMON /SQUA/ X1, L, F1, F2, F3, F4, F5, ERREST, ERR5, T, N
      INTEGER  I, J, ITEMP
      I = 1
 10   J = 2*I
      IF (J.GT.N) GO TO 100
      IF (J.EQ.N) GO TO 20
      TJ = T(J)
      TJ1 = T(J+1)
C
C DECIDE WHICH DECENDENT IS LARGEST
C
      IF (DABS(ERREST(TJ)).GE.DABS(ERREST(TJ1))) GO TO 20
      J = J+1
 20   TI = T(I)
      TJ = T(J)
C
C COMPARE ITEM WITH LARGEST DECENDENT
C
      IF (DABS(ERREST(TI)).GE.DABS(ERREST(TJ))) GO TO 100
C
C IF NECESSARY, EXCHANGE WITH DECENDENT
C
      ITEMP = T(I)
      T(I) = T(J)
      T(J) = ITEMP
      I = J
      GO TO 10
 100  RETURN
      END
```

As presently programmed, the array space reserved for the binary tree and subinterval information is 9000 memory cells. This is considerably more storage than that required by subroutines using local termination criteria. On some machines this could be a serious disadvantage of the global strategies. However, many computing systems are now equipped with vast amounts of either real or "virtual" memory. It is worth noting that the subroutine SQUAGE together with the function subprogram which evaluates the integrand would normally occupy only a few pages of virtual storage. Hence during the quadrature computation the working set is relatively small, thus satisfying the hypotheses of many virtual memory page swapping algorithms. On machines of this sort, the extra memory space required by the arrays in SQUAGE would not be a significant disadvantage. Higher order adaptive methods using global strategies, such as AIND, require much less memory space than SQUAGE for the storage of *Pen*.

We are not presenting SQUAGE as a serious competitor of the better quadrature subroutines in the literature. It is given here only as a programming example and was written to compute the experimental results given in Section 4. SQUAGE lacks many important attributes of serious quadrature subroutines, such as noise detection and a garbage collection mechanism for the case of binary-tree overflow.

4. EXPERIMENTAL COMPUTATIONS

In this section we give the results of three computational experiments which test the model given in Section 2. We also compare the performance of SQUANK and SQUAGE with two related subroutines, CADRE and AIND. These computations were all performed on the University of Waterloo's Honeywell 6050 computer using words of 36 bits with 28-bit fractions.

The first experiment involves the integrand $f(x) = (1 + c - x)^{-1}$ for a fixed error tolerance and various positive values of c. The second studies the same integrand for $c = .01$ while varying the absolute error tolerance ϵ. The third experiment uses the integrand $f(x) = x^{\alpha}$ as α varies. All integrations are over the interval $0 \leq x \leq 1$.

Both CADRE and AIND use higher order local quadrature procedures (on each subinterval) than SQUANK or SQUAGE. CADRE [1] uses a dynamic extrapolation process for its local quadrature procedure; when it senses that the extrapolation may not be working, it stops to make an adaptive subdivision of the interval. In this sense, it is based on a high (variable) order local quadrature rule. It uses an interval acceptance criterion which is intermediate between being local and global as we have defined them (see Appendix). The local quadrature rule for AIND is the 21-point Kronrod formula obtained by adding 11 points to the 10-point Gaussian formula [8]. Hence AIND uses a very high-order local quadrature rule. It is the only quadrature program known to the authors (other than SQUAGE) which uses a global interval acceptance criterion.

For the study of $f(x) = (1 + c - x)^{-1}$, an absolute error tolerance of $\epsilon = 10^{-6}$ was used. CADRE and AIND have both absolute and relative error tolerances appearing in their calling sequences, and they attempt to meet the least stringent of the two requirements. Hence, for purposes of comparison, we set the relative tolerances to zero for these tests. The predicted number of function evaluations for

SQUANK was developed in Section 2 and is

$$4\,Lpred(f; 0, 1, \epsilon) + 1 = 4\lfloor(\tfrac{64}{255})^{1/4}(c^{-1/4} - (1 + c)^{-1/4})\epsilon^{-1/4}\rfloor + 1.$$

For SQUAGE it is

$$4\,Gpred(f; 0, 1, \epsilon) + 1 = 4\lfloor 2^{-7/4}15^{-1/4}(\log_e(1 + c^{-1}))^{5/4}\epsilon^{-1/4}\rfloor + 1.$$

Plots of $-\log_{10} c$ versus $\log_{10} N$ are given in Figure 2 to show the comparison between predicted and actual numbers of function evaluations for SQUANK and SQUAGE. (Here N denotes the number of function evaluations.)

In Figure 2 the predicted growth rates of N with c appear to be borne out; however, the actual numbers of function evaluations tend to be greater than predicted by a factor of about 1.3 for SQUANK, and 1.1 for SQUAGE. This probably represents a minor failing of assumptions (i) and (ii) of Section 2.[1] In Figure 3 we show the actual numbers of function evaluations used as they vary with c for all four subroutines mentioned. In Table I, we tabulate the actual errors made. The performances of CADRE and AIND for that integrand are also listed in Table I for comparison.

Turning now to the second experiment, we fix the integrand at $f(x) = (1.01 - x)^{-1}$ and look at the effects of varying ϵ. Table II lists the numbers of function evaluations predicted and used by SQUAGE and SQUANK. Again, the predictions for SQUANK and SQUAGE are low by factors of about 1.3 and 1.1, respectively. However, the actual numbers used seem to grow like $\epsilon^{-1/4}$, as pre-

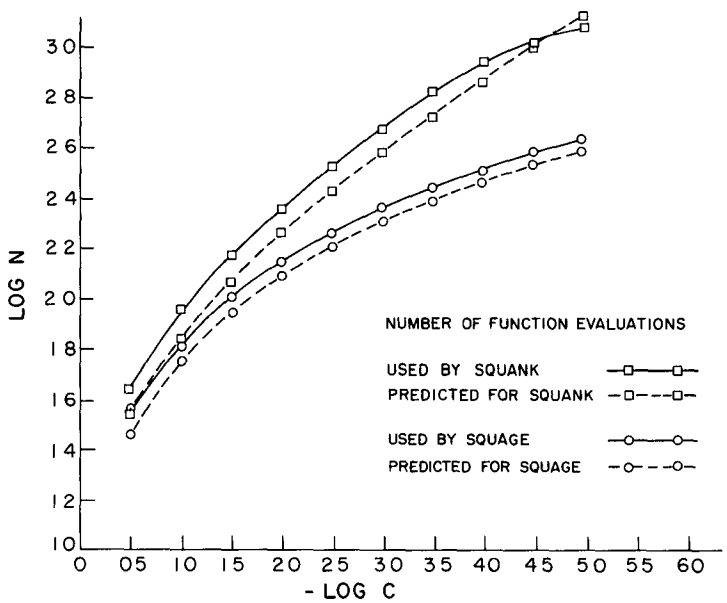

Fig. 2. The number of function evaluations used by and predicted for SQUANK and SQUAGE applied to $1./(1. + c - x)$

[1] J. Lyness has analyzed these assumptions and has suggested more appropriate ones.

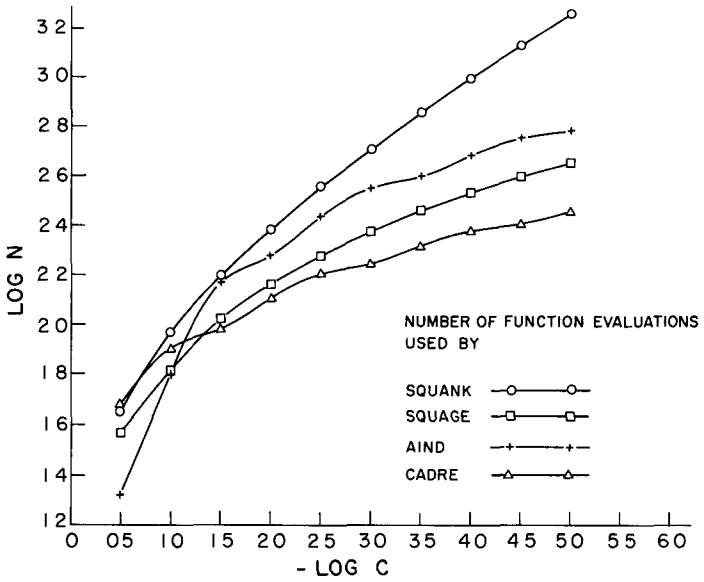

Fig. 3. The number of function evaluations used by four subroutines applied to $1./(1 + c - x)$

dicted. The performances of CADRE and AIND are also listed in Table II. The much slower growth rate of N with ϵ for CADRE and AIND demonstrates an advantage of using high-order local quadrature procedures.

It should be noted from Tables I and II that SQUAGE shows a tendency to meet the prescribed error tolerance more closely (from below) than SQUANK.

Our third experiment involves integrands $f(x) = x^\alpha$ for various values of α.

Table I. A Comparison of Performance on $\int_0^1 (1 + c - x)^{-1}\, dx$ for $\epsilon = 10^{-6}$

Col. A: error. Col. B: number of function evaluations used.

$LOG_{10}(c)$	SQUANK		SQUAGE		CADRE		AIND	
	A	B	A	B	A	B	A	B
- .5	-1.5 E-8	45	-1.5 E-8	37	4.2 E-9	48	4.8 E-9	21
-1.0	-3.0 E-8	93	-6.0 E-8	65	7.8 E-9	80	8.8 E-8	63
-1.5	0	157	0	105	9.9 E-9	96	1.8 E-8	147
-2.0	6.0 E-8	241	0	145	1.8 E-8	128	4.7 E-7	189
-2.5	-6.0 E-8	361	-1.2 E-7	189	1.7 E-8	160	9.0 E-8	273
-3.0	0	513	-6.0 E-8	241	2.1 E-8	176	-9.9 E-9	357
-3.5	-1.2 E-7	733	-2.4 E-7	289	3.0 E-8	208	3.9 E-7	399
-4.0	0	957	-3.6 E-7	341	3.0 E-8	240	9.8 E-8	483
-4.5	0	1153	2.4 E-7	405	3.3 E-8	256	3.2 E-8	567
-5.0	0	1381	3.6 E-7	457	4.2 E-8	288	3.4 E-7	609

Table II. A Comparison of Performance on $\int_0^1 (1.01 - x)^{-1}\,dx$

Col. A: $\log_{10}$ | error | . Col. B: number of function evaluations used.
Col. C: number of function evaluations predicted.

$\log_{10}\epsilon$	SQUANK			SQUAGE			CADRE		AIND	
	A	B	C	A	B	C	A	B	A	B
-3	- 4.1	37	34	- 3.5	29	22	- 4.9	56	- 4.3	147
-4	- 5.7	77	61	- 4.7	49	40	- 5.9	80	- 4.3	147
-5	- 6.8	137	108	- 6.4	85	72	- 7.1	120	- 6.3	189
-6	- 8.3	241	193	- 7.8	145	129	- 7.7	128	- 6.3	189
-7	- 9.8	425	344	- 9.3	253	229	- 8.7	176	- 9.1	231
-8	-11.3	757	612	-10.7	453	408	-10.0	240	- 9.1	231
-9	-12.8	1325	1089	-12.3	805	726	-10.9	256	- 9.1	231
-10	-14.3	2373	1938	-13.8	1429	1292	-12.1	384	-12.6	273
-11	-15.8	4249	3446	-15.3	2537	2298	-12.9	480	-12.6	273
-12	-16.5	7585	6128	-15.8	4001	4087	-13.5	512	-12.6	273

The model in Section 2 predicts that the subinterval selection process using the local acceptance criterion (1.1) does not terminate if $\alpha < 0$. Although SQUANK uses (1.1), it has a limited storage space for the stack used to store the subintervals in *Pen*. Thus, after a maximum number of bisections (when the stack is full), the current subinterval is accepted without reference to (1.1). In the version of SQUANK given in [5], the maximum number of bisections (maximum stack length) is set at 30. However, for this experiment, the maximum was reset to 500 to avoid the effect of default acceptance of subintervals not accounted for in the model of Section 2. (For these integrands, the default acceptance of subintervals is benign in the sense that the error tolerance is always met using the unchanged version of SQUANK.)

Using $p_4(\alpha) = |\,\alpha(\alpha - 1)(\alpha - 2)(\alpha - 3)\,|$ of Section 2, the predicted numbers of function evaluations as given in Section 2 are, for SQUANK,

$$4\,Lpred(f; 0, 1, \alpha) + 1 = 4\lfloor (p_4(\alpha)/180\epsilon)^{1/4}/\alpha \rfloor + 1 \quad \text{if } \alpha \geq 0, \qquad (4.1)$$

and for SQUAGE,

$$4\,Gpred(f; 0, 1, \epsilon) = 4\lfloor (\tfrac{5}{4}(1 + \alpha))^{5/4}(p_4(\alpha)/45\epsilon)^{1/4} \rfloor + 1 \quad \text{if } \alpha > -1. \qquad (4.2)$$

Predictions (4.1) and (4.2) were tested using error tolerance $\epsilon = 10^{-6}$. The results for SQUANK are given in Figure 4; those for SQUAGE are given in Figure 5.

In addition to these three computations, a general comparison between SQUANK and SQUAGE was made by testing them using a quadrature testing program written by P. Keast of the University of Toronto. Keast's program uses 131 test functions divided into seven groups, each group exhibiting a specific behavior (e.g. oscillation, endpoint singularity, etc.). The quadrature subroutine is tested on each of these functions at 10 different absolute tolerance levels: $\epsilon = 10^{-n}$, $n = 1(1)10$. The observations made from the results of Keast's tests on SQUANK and SQUAGE are similar for all groups except groups 5 and 6. With the exception of groups 5 and 6, SQUAGE always took fewer function evaluations, typically by a factor of

2 or 3 at the smaller values of ϵ. In general SQUAGE also tended to come closer to the prescribed tolerance. Group 5 is composed of functions with singularities, like $f(x) = |x - c|^{\alpha}$ for c in the range of integration, and $-1 < \alpha < 0$. For this group, SQUANK's performance is markedly inferior to SQUAGE's, as our previous discussion would lead us to expect. Group 6 is comprised of various step functions. For these SQUANK generally took about 20 to 40 function evaluations and gave up, apparently interpreting the discontinuities as noise in the function evaluations. Since SQUAGE does not attempt to recognize roundoff error, or other noise, it continued on and met the tolerance for about half of the functions in group 6. Of course for this group SQUAGE generally used substantially more function evaluations.

It is perhaps worth noting that group 5 represents an important class of functions for which SQUAGE is successful but SQUANK fails, using vastly more function evaluations. Group 6 is not as important since neither of the subroutines is designed for integrands having discontinuities. Using group 6 for testing these particular quadrature subroutines is, as Lyness has remarked, a little like testing a truck by trying to drive it over a river.

There were no examples in which both subroutines met the error tolerance with SQUANK requiring fewer function evaluations. However, simple examples can be constructed in which both are successful, but SQUANK requires fewer subintervals; so no simple statement concerning relative numbers of function evaluations seems provable.

Keast's tests for SQUANK took .575 hours of processor time, whereas the tests for SQUAGE took .205 hours. Thus the number of function evaluations appears to be a reasonable measure of complexity when comparing the two subroutines.

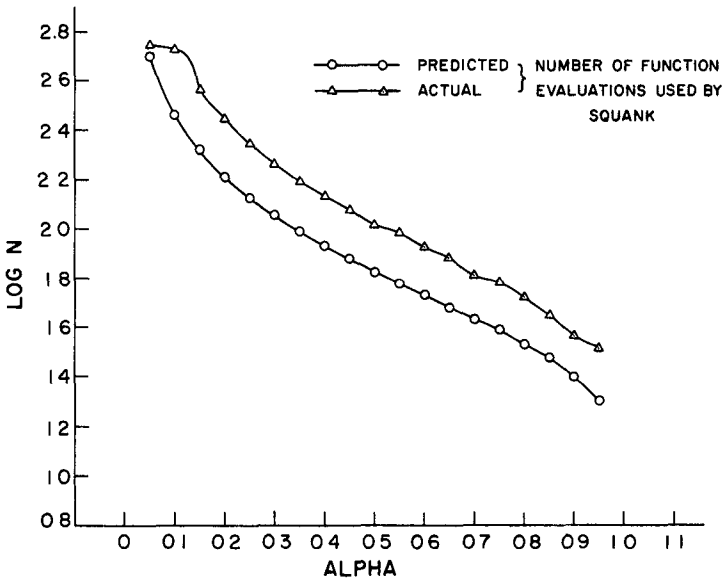

Fig. 4. The number of function evaluations predicted and used by SQUANK applied to x^{α}

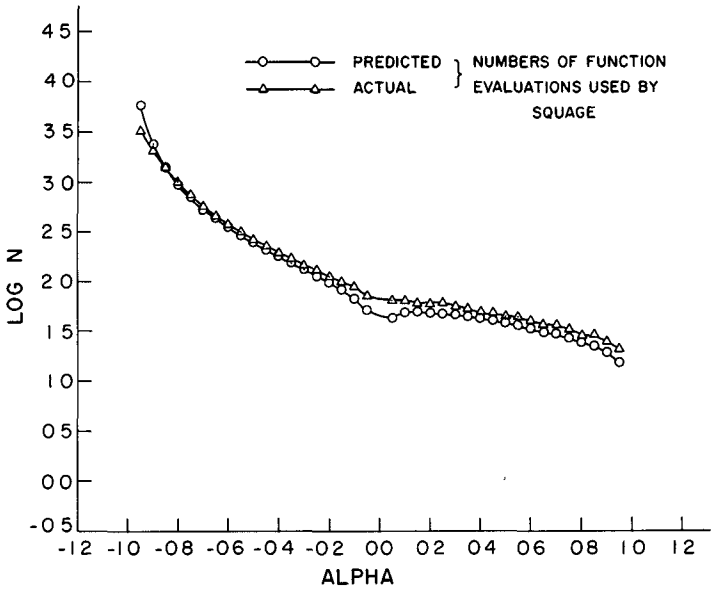

Fig. 5. The number of function evaluations predicted and used by SQUAGE applied to x^{α}

5. CONCLUSIONS

The results of Section 4 suggest that the analytical model of Section 2 works well enough to be a useful tool for studying adaptive quadrature algorithms. This model indicates that the number of subintervals selected may be regarded as the product of two factors: one that summarizes the interaction between the algorithm and the integrand, and one that depends only on the absolute error tolerance.

For both SQUANK and SQUAGE this second factor is $\epsilon^{-1/4}$, and it is clearly derived from the use of Simpson's rule as a local quadrature procedure. In this sense then we can call both SQUAGE and SQUANK fourth-order algorithms and note that the interval acceptance criterion has no influence on the order.

The connection between the order of the local quadrature rule and the global performance of an adaptive quadrature method has also been observed by Rice [11] in the context of the rate of convergence to zero of the quadrature error. These results seem to be consistent with his, when the error estimate used is actually an error bound for each subinterval.

While it may not influence the order of an adaptive quadrature algorithm, the subinterval selection strategy appears to influence its performance in at least three ways: (i) by affecting the number of function evaluations required for an integral; (ii) by changing the domain of integrands which can be handled by the algorithm; (iii) by affecting the closeness with which the user's tolerance is achieved. Each of these ways has been observed both in the experimental computations and the general testing using Keast's program. The first is clearly evident in the reduced number of function evaluations used by SQUAGE as compared with SQUANK when applied to $f(x) = (1 + c - x)^{-1}$ and in the 2.05/5.75 ratio of running times for the general tests on these two routines. The second influence is demonstrated by

the difference in success between SQUAGE and SQUANK when applied to $(1 - x)^\alpha$ for $-1 < \alpha < 0$. This effect has also been observed by Rice and given a more rigorous treatment in [10] and [11]. The third influence of the subinterval selection process mentioned above can be seen in Tables I and II and was also evident in the general testing.

The advantages of global strategies over local strategies were seen to be in some sense a space-time tradeoff. The reduced numbers of required function evaluations and the apparent increase in domain of applicability using the global strategy comes at the expense of additional memory requirements. However, the memory requirements of SQUAGE are not unreasonable for many applications, and when a higher order local quadrature rule is used, such as in AIND, the additional memory requirements are very small.

An adaptation of Floyd's tree sort algorithm has proved to be an effective way of programming the global strategy when a large number of subintervals are anticipated. Using this technique, the additional overhead involved in the global strategy is negligible.

APPENDIX. SUBINTERVAL ACCEPTANCE IN CADRE

In this appendix we attempt to give a brief description of the interval subdivision process in CADRE [1].[2] In this program the local quadrature procedure, $Q(I_n)$, is a variable order one (Romberg integration) with the capability of handling several standard forms of nonsmooth integrands. This procedure provides an error estimate, $E(I_n)$, which is compared to a local error tolerance described below. CADRE has two alternatives for achieving the specified error tolerances: subdivision of the current subinterval, and variable order extrapolation applied to the current subinterval. The basic strategy appears to be that extrapolation is pursued on the current subinterval as long as it meets a set of criteria for being successful. Hence there are a variety of criteria which will cause interval subdivision to take place, only some of which involve comparing the local error estimate to the local error tolerance.

When an interval is to be subdivided, it is halved and the subintervals are distinguished as right and left subintervals of their parent. At the time of subdivision, the local error tolerance is made current by halving it as well. Then the right half is investigated. If it is acceptable under the new local tolerance, its contribution to the result is made and the local tolerance is doubled. If it is not acceptable, its data are stacked and the local tolerance retained as is. Then the left half subinterval's acceptability is tested. Hence, the local tolerance after completing one subdivision is reduced only if the right half subinterval was not acceptable and consequently was stacked.

Clearly, this design is too complicated to be modeled by the simple model used in this paper. The order of the quadrature rule, $Q(I_n)$, is determined adaptively so that the form of $E(I_n)$ is not predetermined. Moreover, the interval acceptance decision is a multicriterion one, depending on the algorithm's perception of integrand behavior as well as a comparison of error estimates to local error tolerance.

---

[2] The authors are indebted to C. deBoor for helpful discussions of this topic.

It is, however, interesting in the context of our comparison of local acceptance criteria with global ones. The local error tolerance is $f2^{-I}E$, where E is the user-specified tolerance, f is $\frac{1}{2}$, 1, or 2 depending on the stage of current left and right subinterval investigations, and I is the current stack height. Since the stack height is not simply the number of subdivisions of the original range of integration, 2^{-I} is not simply related to the current subinterval length. Hence I retains some memory of the subdivision history that cannot be known from the data for the current subinterval. On the other hand, the local error tolerance (i.e. I) does not require global knowledge of the current subdivisions of the range of integration. This permits CADRE to proceed to accumulate the contributions to the numerical result before all the acceptable subintervals have been determined. It demonstrates that interval acceptance criteria need not be totally local as in SQUANK or totally global as in SQUAGE and doubtless a variety of such intermediate stages are possible.

REFERENCES

1. DEBOOR, C. CADRE: an algorithm for numerical quadrature. In *Mathematical Software*, J. Rice, Ed. Academic Press, New York, 1971, pp. 417–449.
2. FLOYD, R., W. Treesort 3, Algorithm 245. *Comm. ACM 7*, 12 (Dec. 1964), 701.
3. KNUTH, D. E. Sorting and searching. In *The Art of Computer Programming, Vol. 3*. Addison Wesley, Reading, Mass.
4. LYNESS, J. N. Notes on the adaptive Simpson quadrature routine. *J. ACM 16*, 3 (July 1969), 483–495.
5. LYNESS, J. N. SQUANK (Simpson quadrature used adaptively—noise killed), Algorithm 379. *Comm. ACM 13*, 4 (April 1970), 260–263.
6. MCKEEMAN, W. M. Adaptive numerical integration by Simpson's rule, Algorithm 145. *Comm. ACM 5*, 12 (Dec. 1962), 604.
7. PATTERSON, T. N. L. Algorithm for automatic numerical integration over a finite interval, Algorithm 468. *Comm. ACM 16*, 11 (Nov. 1973), 694–699.
8. PIESSENS, R. An algorithm for automatic integration. *Angewandte Informatik* (Sept. 1973), 399–401.
9. PIESSENS, R. A quadrature routine with round-off error guard. Rep. TW 17, Appl. Math. and Programming Div., Catholic University of Leuven, Belgium, March 1974.
10. RICE, J. An educational adaptive quadrature algorithm. ACM SIGNUM Newsletter *8*, 2 (April 1973), 27–41.
11. RICE, J. A metalgorithm for adaptive quadrature. *J. ACM 22*, 1 (Jan. 1975), 61–82.
12. ROWLAND, J. H., AND VAROL, Y. L. Exit criteria for Simpson's compound rule. *Math. Computation 26*, 119 (July 1972), 699–704.

Received April 1974