BIT Numerical Mathematics **43:** 319–337, 2003.
© 2003 *Kluwer Academic Publishers. Printed in The Netherlands*

DOUBLY ADAPTIVE QUADRATURE ROUTINES BASED ON NEWTON–COTES RULES *

TERJE O. ESPELID

Department of Informatics, University of Bergen, Postbox 7800, N-5020 Bergen, Norway. email: terje@ii.uib.no

Abstract.

In this paper we test two recently published Matlab codes, `adaptsim` and `adaptlob`, using both a Lyness–Kaganove test and a battery type of test. Furthermore we modify these two codes using sequences of null rules in the error estimator with the intention to increase the reliability for both codes. In addition two new Matlab codes applying a locally and a globally adaptive strategy respectively are developed. These two new codes turn out to have very good properties both with respect to reliability and efficiency. Both algorithms are using sequences of null rules in their local error estimators. These error estimators allow us both to test if we are in the region of asymptotic behavior and thus increase reliability and to take advantage of the degree of precision of the basic quadrature rule. The new codes compare favorably to the two recently published adaptive codes both when we use a Lyness–Kaganove testing technique and by using a battery test.

AMS subject classification (2000): 65D30, 65D32.

Key words: Adaptive quadrature, error estimation, software, Newton–Cotes rules.

1 Introduction.

Automatic algorithms are now used widely for the numerical calculation of integrals. Since the first such algorithm was given by McKeeman [19] in 1962, many new and sophisticated algorithms, both adaptive and non-adaptive, have been developed, among these [4, 6, 13, 20, 22].

Recently Gander and Gautschi [10, 11] published a paper describing two new adaptive quadrature codes, `adaptsim` and `adaptlob`. These two codes are written in Matlab and they are both compared to the two Matlab codes `quad` and `quad8` and in addition to several routines in a number of well known software libraries. Gander and Gautschi use a battery test of 23 test functions and conclude that the two new codes are much better than Matlab's currently available `quad` and `quad8` and that the two new codes in addition are better than the rest of the available quadrature software in 2/3 of the tested cases.

In automatic quadrature algorithms the estimate of the true error in the approximation of the integral governs the decision on whether to return the current approximation and terminate or to continue. Both the efficiency and the reliability therefore depend heavily on the error estimating procedure. In many

*Received June 2002. Revised November 2002. Communicated by Åke Björck.

adaptive algorithms the local error estimate is simply taken as the absolute value of the difference between two quadrature approximations, that is the absolute value of *one null rule*. Testing has shown, Berntsen [1] and Berntsen *et al.* [3], that routines applying such a simple local error estimate may be very unreliable. Unfortunately the two new codes by Gander and Gautschi are based on such a simple error estimate and are therefore potentially more unreliable than the tests run by Gander and Gautschi indicate.

More sophisticated local error estimating algorithms have been suggested by several authors de Boor [6], Piessens *et al.* [20], Berntsen and Espelid [2] and Espelid [7]. One of the most successful adaptive quadrature algorithms so far, see [1] and Espelid and Sørevik [9], is QAG in QUADPACK, [20]. This algorithm, using a Gauss–Kronrod rule as a basic rule, has a heuristic local error estimating algorithm developed especially for this type of rule. Data from experiments, showing the performance of the usual error estimating procedure, has been used to construct this local error estimating algorithm.

Ten years ago Berntsen and Espelid, [2], presented a new error estimator to be used in adaptive quadrature algorithms. This error estimator was designed using *a sequence of null rules* and can be applied in connection with many different basic quadrature rules. In [2] it was demonstrated that this error estimator functions well in adaptive algorithms using either Gauss–Legendre rules, Gauss–Kronrod rules, Clenshaw–Curtis rules or Lobatto rules as their basic quadrature rule. The main conclusion in the paper was that both Gauss–Legendre rules and Lobatto rules are good basic rules and will function well both with respect to reliability and efficiency in future codes using this new error estimator.

While the error estimator developed in [2] was based on a sequence of *symmetric null rules* of different polynomial degrees, Espelid in [7] suggested to use both *symmetric* and *anti-symmetric null rules* in a slightly modified error estimator. This modification makes it possible to construct error estimators in the original spirit but using fewer evaluation points in the basic integration rule.

We will, in this paper, test `adaptsim` and `adaptlob` using both a Lyness–Kaganove testing technique and a battery test. Furthermore, we will modify both `adaptsim` and `adaptlob` with an error estimator tailored to the two different rules the two codes use and demonstrate that it is possible to improve reliability and at the same time improve efficiency when the accuracy request is high.

Finally we will, inspired by the good results achieved by modifying `adaptsim`, develop two new codes, `coteda` and `coteglob`, both based on the five and nine points Newton–Cotes rules. We use these two rules in a doubly adaptive manner, which turns out to be superior to `adaptsim/adaptlob` both compared through the Lyness–Kaganove testing and the battery testing.

I have to admit that I am quite surprised by the good performance of the two new codes and I would never have tried to develop codes based on Newton–Cotes rules if it had not been for the paper by Gander and Gautschi.

2 A sequence of null rules.

We define the integral to be computed by

$$I[f] \;=\; \int_a^b f(x)\,dx. \tag{2.1}$$

We will in the following develop sequences of null rules designed to be used in connection with symmetric quadrature rules. We refer the reader interested in a more general presentation of these ideas to [2]. Given $2n+1$ distinct points x_i, $i=-n,\ldots,n$ in the interval $[a,b]$ and a quadrature rule based on these points

$$Q[f] = \sum_{i=-n}^{n} w_i f(x_i). \tag{2.2}$$

x_i and w_i, $i=-n,-n+1,\ldots,n$, are the rule's nodes and weights respectively. We assume $x_i < x_{i+1}$ for $i=-n,-n+1,\ldots,n-1$. By a simple translation this rule may be used on any of the local intervals produced by an adaptive algorithm. As mentioned we assume the rule to be symmetric, say $x_0 = (a+b)/2$ is the midpoint and then $x_{-i}-x_0 = -(x_i - x_0)$ for $i=1,2,\ldots,n$. Furthermore, $w_{-i} = w_i$ for $i=1,2,\ldots,n$.

A quadrature rule $Q[f]$ has degree d if it integrates exactly all polynomials of degree $\leq d$ and fails to integrate exactly $f(x)=x^{d+1}$. An interpolatory quadrature rule based on $2n+1$ distinct nodes has degree at least $2n$. A quadrature rule based on $2n+1$ distinct nodes of degree $d \geq 2n$ is unique and therefore has to be interpolatory. A quadrature rule based on $2n+1$ nodes has degree at most $4n+1$ (Gauss–Legendre).

The term *null rule* was first used in 1965 by Lyness, [16]. The following definition of a null rule is useful in this context.

Definition 2.1. *A rule*

$$N[f] = \sum_{i=-n}^{n} u_i f(x_i) \tag{2.3}$$

is a null rule iff it has at least one nonzero weight and in addition

$$\sum_{i=-n}^{n} u_i = 0.$$

A null rule is furthermore said to have degree d if it integrates to zero all polynomials of degree $\leq d$ and fails to do so with $f(x)=x^{d+1}$. Assume that the rule's nodes are symmetric in the integration interval then a null rule is said to be symmetric if in addition $u_{-i}=u_i$ for $i=1,2,\ldots,n$. Similarly a null rule is said to be anti-symmetric if both $u_0=0$ and $u_{-i}=-u_i$ for $i=1,2,\ldots,n$.

Remark 2.1. Changing the direction of integration in (2.1) will leave $I[f]$ unaffected. Using symmetric rules and null rules imply that both $Q[f]$ and

$N[f]$ are unaffected too. An anti-symmetric null rule will give the same value but the opposite sign due to this change of integration direction. This implies that an error estimator which is based on the absolute values of symmetric and anti-symmetric null rules will be unaffected of this change too.

Suppose that the $2n+1$ nodes are fixed and that the unique interpolatory rule Q of degree d is chosen as the quadrature rule in the adaptive algorithm. A sequence of null rules $N_1, N_2, \ldots$ are now easily constructed based on these $2n+1$ nodes. Let $f[z_0, z_1, \ldots, z_m]$ be a divided difference for the function f based on the set of distinct points $\{z_0, z_1, \ldots, z_m\}$ which is a subset of the $2n+1$ nodes. It is well known that a divided difference is simply a linear combination of function values and that $f[z_0, z_1, \ldots, z_m]$ gives the value zero for all polynomials up to degree $m-1$ and the value one for $f(x) = x^m$. Therefore the divided difference $f[z_0, z_1, \ldots, z_m]$ is a null rule of degree $m-1$.

This implies that $f[x_{-m}, x_{-m+1}, \ldots, x_0, x_1, \ldots, x_m]$ is a null rule of degree $2m-1$, for $m = 1, 2, \ldots, n$. Furthermore it is easy to prove that these null rules are all symmetric. Similarly $f[x_{-m}, x_{-m+1}, \ldots, x_{-1}, x_1, \ldots, x_m]$ is a null rule of degree $2m-2$, for $m = 1, 2, \ldots, n$. These null rules are all anti-symmetric.

Now following [2], we define an inner product between two null rules, N_u and N_v, based on the same set of $2n+1$ points as follows

$$(N_u, N_v) = \sum_{i=-n}^{n} u_i v_i, \tag{2.4}$$

and a null rule's 2-norm as $||N_u||_2^2 = (N_u, N_u)$. We obviously have, with this inner product, that a symmetric null rule and an anti-symmetric null rule are orthogonal null rules.

It is straightforward to construct a sequence of null rules, $N_1, N_2, \ldots, N_{2n}$ of decreasing degrees $2n-1, 2n-2, \ldots, 1, 0$ that are all orthogonal. We only apply a Gram-Schmidt orthogonalization process separately on each of the two sequences starting with the null rules of highest degrees. Obviously, every odd numbered null rule will retain its symmetry after this orthogonalization process and this will similarly be true for every even numbered anti-symmetric null rule.

Furthermore we may assume that all these null rules have been normalized through the same 2-norm, a natural choice is $||N_j||_2^2 = ||Q||_2^2 = \sum_{i=-n}^{n} w_i^2$. Through this choice one can show ([2]) that when the null rules are applied to a smooth function f over an interval of length h then $N_j[f] = \mathrm{O}(h^{2n+2-j})$ for $j = 1, 2, \ldots, 2n$.

3 A local error estimating algorithm.

In this section we will develop a local error estimating algorithm which is a modification of the algorithm presented in [2]. Given a symmetric set of $2n+1$ distinct points and suppose furthermore that a sequence of $2n$ orthonormal null rules has been constructed as described in the previous section. For a given sub-interval of length h and function f we may compute the $2n$ inner products

$$E_j = |N_j[f]|, \quad j = 1, 2, \ldots, 2n.$$

Asymptotically we get

$$E_j = \mathrm{O}(h^{2n+2-j}), \quad j = 1, 2, \ldots, 2n. \tag{3.1}$$

Defining the local error as

$$E_0 = |Q[f] - I[f]|$$

gives the following asymptotic expression

$$E_0 = \mathrm{O}(h^{d+2}), \tag{3.2}$$

where $d \geq 2n$ is the rules degree of precision.

This implies that when h is sufficiently small and f is sufficiently smooth, then we can expect that

$$E_0 \ll E_1 \ll E_2 \ll \cdots \ll E_{2n}. \tag{3.3}$$

Furthermore we define the reduction factors

$$r_j = E_j / E_{j+1}, \quad j = 1, 2, \ldots, K,$$

for a value of $K < 2n$ and

$$r = \max_{j=1,2,\ldots,K} r_j.$$

Observe that $r = \mathrm{O}(h)$ asymptotically, and we would therefore expect $r < 1$ when h becomes sufficiently small. In view of (3.3) we see that a necessary test on whether h is small enough and f sufficiently smooth, is to check that $r < r_{\text{critical}}$ for a heuristic value of $r_{\text{critical}} < 1$. If this test is passed, we may apply an optimistic error estimate based on (3.1) and (3.2)

$$\hat{E} = c\, r^{\alpha} E_2. \tag{3.4}$$

Choosing E_2 in this error estimate instead of E_1 is an attempt to reduce possible phase factor effects (see [2]) on the error estimate. Since the order is satisfied then it is less likely that both E_2 and E_1 are influenced by phase factor effects at the same time. Observing that $r_0 = E_0/E_2$ is of order $\mathrm{O}(h^{d+2-2n})$ we may choose a value of α in the range $1 \leq \alpha \leq (d + 2 - 2n)$ depending on the degree of optimism we want to put into this algorithm.

In order to test whether we have reached the noise level or not we have also introduced a noise test, as in [20], following our local error estimating algorithm. Initially we define, for the whole interval $[a, b]$, the value $isabs = \sum_{i=-n}^{n} |w_i f(x_i)|$ and then the problem's noise level is defined through $noise = 50\, \epsilon\, isabs$, where ϵ is the machine epsilon. The local error estimating algorithm, defining a similar local noise level, then appears as follows:

The local error estimating algorithm A

Compute: $E_j = |N_j[f]|,\ j = 1, 2, \ldots, K+1;$
$r_j = E_j/E_{j+1},\ j = 1, 2, \ldots, K;$
$r = \max_{j=1,2,\ldots,K}\ r_j;$
Non-asymptotic: **if** $r > 1$ **then** $\hat{E} = C\ \max_{j=1,2,\ldots,K+1}\ E_j$
Weak asymptotic: **elseif** $r_{\text{critical}} \leq r$ **then** $\hat{E} = C\ r\ E_2$
Strong asymptotic: **else** $\hat{E} = C\ r_{\text{critical}}^{1-\alpha}\ r^{\alpha}\ E_2$
endif
The noise test : **if** $E_1 <$ *noise and* $E_2 <$ *noise* **then** $\hat{E} = 0$

Remark 3.1. a) we may get $r > 1$, even though we are in the asymptotic region, simply because the precision of the actual computer may influence the computations. If the correct values of E_1 and E_2 are very small, then they both may consist mainly of noise from the computations. Example: if f is constant in a subinterval then all null rules will give the value zero and the noise test will correctly put $\hat{E} = 0$.
b) The choice of constants are aiming at creating as smooth an error estimator as possible.

We have used The local error estimating algorithm A to modify the two codes developed by Gander and Gautschi in [10, 11]. Gander and Gautschi's two codes and the two modifications can be shortly described as follows:

- `adaptsim`: This code is developed by Gander and Gautschi. The code is based on a five point closed Newton–Cotes rule which can be viewed as an extrapolation of Simpson's rule. Furthermore the code applies bisection in a locally adaptive strategy making use of Matlab's recursive function option. All function evaluations, except for five extra function evaluations computed initially, contribute to the final estimate.

- `modsim`: This code is developed in this paper. It is based on the same quadrature rule as `adaptsim` but uses the local error estimating algorithm A tailored to this five point rule and intended to improve the code's reliability compared to `adaptsim`. Furthermore it uses the same adaptive strategy as `adaptsim`, but applies a nine point closed Newton–Cotes rule to get an initial estimate of the integral to be used in the error estimation of the relative error. Furthermore in the first step a division into four subintervals is used implying that all computed function values contribute to the final estimate.

- `adaptlob`: This code is developed by Gander and Gautschi. The code is based on a seven points Lobatto–Kronrod rule constructed by Gander and Gautschi in [10, 11]. The code uses a locally adaptive strategy with a division in six subintervals in each step. Thus all computed function values contribute to the final estimate (here too an exception occurs initially: six extra initial function evaluations.)

- `modlob`: This is a modification of the previous code with respect to two issues: First it uses the local error estimating algorithm A tailored to the seven points Lobatto–Kronrod rule in order to improve the codes reliability, and second the division in six subintervals is replaced by bisection in order to improve the codes adaptability. The last change implies that the code no longer makes use of all function evaluations in the final estimate.

In both `modsim` and `modlob` we use $C = 32$, $r_{\text{critical}} = 1/2$ and $K = 3$. We have $d = 5$ in `modsim` and $d = 9$ in `modlob` and we have chosen α one unit less than the maximum values in both cases due to the fact that we have locally adaptive algorithms. Furthermore in `modsim` and `modlob` we redefine the user specified tolerance to be on the *noise* level whenever the code finds the specified tolerance too small. Thus, using these two codes it is normally not possible to approximate an integral to machine precision contrary to what is possible in the two codes `adaptsim` and `adaptlob`.

4 Lyness–Kaganove testing of the four codes.

The test families used in our experiments are given in Table 4.1, and they are picked from [1, 18] and [21].

Table 4.1: Test families used in Lyness–Kaganove testing.

	Test families	Attributes
1.	$\int_0^1 (\lvert x-\lambda\rvert)^{\alpha_1}\,dx$	Singularity
2.	$\int_0^1 f_2(x)\,dx$ where $f_2(x,y) = \begin{cases} 0 & \text{if } x \le \lambda \\ \exp(\alpha_2 x) & \text{otherwise} \end{cases}$	Discontinuous
3.	$\int_0^1 \exp(-\alpha_3\lvert x-\lambda\rvert)\,dx$	C_0 function
4.	$\int_1^2 10^{\alpha_4}/((x-\lambda)^2 + 10^{2\alpha_4})\,dx$	One Peak
5.	$\int_1^2 \sum_{i=1}^4 10^{\alpha_5}/((x-\lambda_i)^2 + 10^{2\alpha_5})\,dx$	Four Peaks
6.	$\int_0^1 2B(x-\lambda)\cos(B(x-\lambda)^2)\,dx$ where $B = 10^{\alpha_6}/\max(\lambda^2,(1-\lambda)^2)$	Non-linear Oscillatory

In our experiments we have chosen the difficulty parameters α_i, $i = 1,...,6$, to be (numbered from family 1 to 6): $\underline{\alpha} = (-0.5, 0.5, 2.0, -4.0, -2.0, 2.0)$. The random parameters, λ(or λ_i, $i = 1,...,4$, for test family 5), are picked randomly from the region of integration using the Matlab function `random('unif',...)`. We have tested the codes for error tolerances tol $= 10^{-1}, 10^{-2}, ..., 10^{-12}$. (For

the Test family 1 we stop at 10^{-5}.) For these values of tol and for each test family we have asked all routines to compute the integrals for 1000 samples of random parameters, and in all cases the four routines `adaptsim`, `modsim`, `adaptlob` and `modlob` report that the returned values satisfies the error request. For the complete test results we refer the reader to the Appendix in Espelid [8] where we list six tables containing all results from these tests. In Figures 5.1 and 5.2 we plot the work and failures for all four codes and all six test families.

In order to summarize the Lyness–Kaganove testing we will characterize a code unreliable when applied to a specific family if we have more than 10 % failures for any of the error tolerances tested.

`adaptsim` appears to be unreliable in this sense for all six test families. In 51 out of 65 error tolerances counted over all six families this code has failures for more than 10 % of the cases and in many cases the number is greater than 30 %. In addition, for many of the families the average number of wrong digits is more than one for this code.

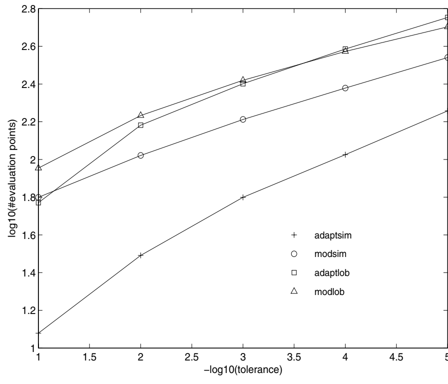
`adaptlob` appears to be much more reliable than `adaptsim`, but still only for Test family 2 (Discontinuous) this code appears to be reliable in the sense defined above. `adaptlob` has failures in more than 10% of the samples in 18 of the 65 error tolerances tested. For the test families 4, 5 and 6 failures above 10 % appear for low accuracies only, which is quite normal for a code with this simple type of error estimator. As commented by Gander and Gautschi both these codes are very unreliable on singular functions (Test family 1).

Both modifications appear to give reliable routines for all test families and are therefore an improvement compared to their counterparts with respect to reliability, as expected. Furthermore these two modifications appear to be generally more efficient than their counterparts especially for higher accuracies, say asking for more than five digits. This is due to the optimism we have put into The local error algorithm A when we detect strong asymptotic behavior.

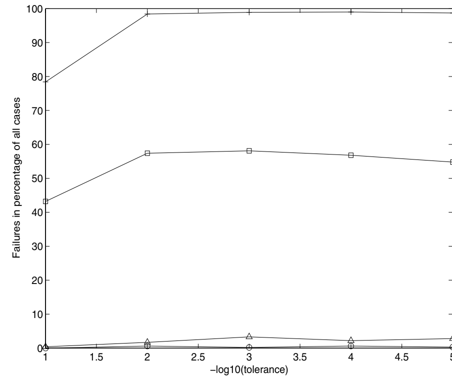
In such a comparison a code with a failure percent of more than 10% should not be considered more efficient no matter how few function evaluations it has used. In particular it is surprising how well `modsim` is doing when it comes to efficiency compared to `adaptlob` for five of the six tested families. This observation is the reason that I found it interesting to develop a doubly adaptive code based on two Newton–Cotes rules.

5 Two doubly adaptive algorithms.

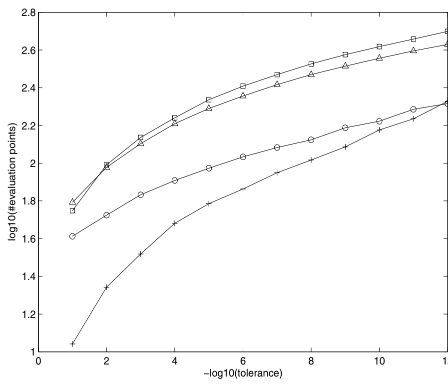
In this section we are going to develop two algorithms with different adaptive strategies. All four codes we have looked at so far make use of the recursive option in Matlab implying that all four codes are *locally adaptive*: the subintervals are processed from left to right until the integral over each subinterval satisfies the relative error requirement. This means that locally adaptive algorithms need an a priori approximation of the whole integral to be used in the local error estimator. This is due to the fact that as the computation proceeds in a locally adaptive algorithm this a priori estimate can not be updated until all subintervals are processed and the computation is finished.



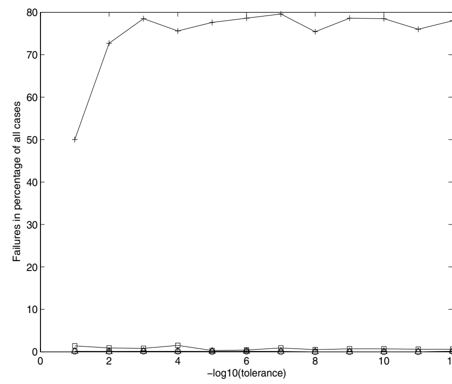
(a) Family 1: work



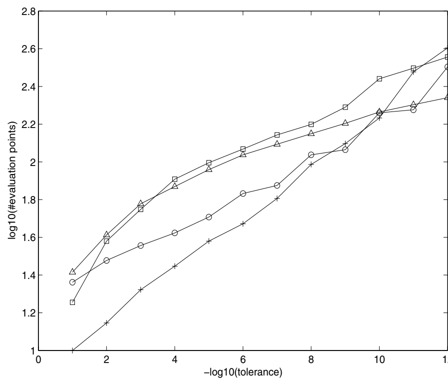
(b) Family 1: failures



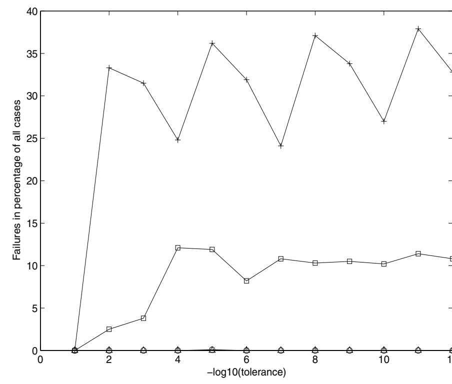
(c) Family 2: work



(d) Family 2: failures

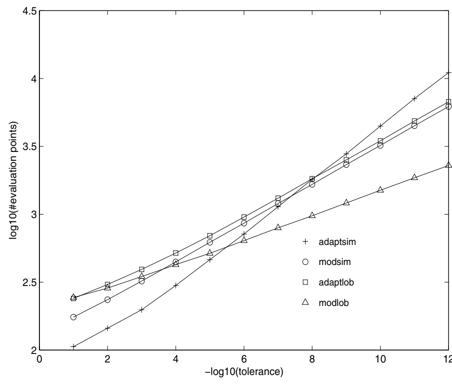


(e) Family 3: work

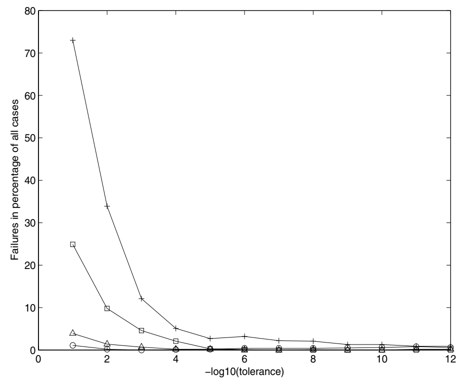


(f) Family 3: failures

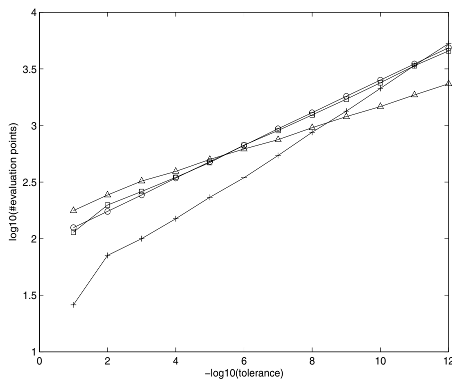
Figure 5.1: Test families 1, 2 and 3.



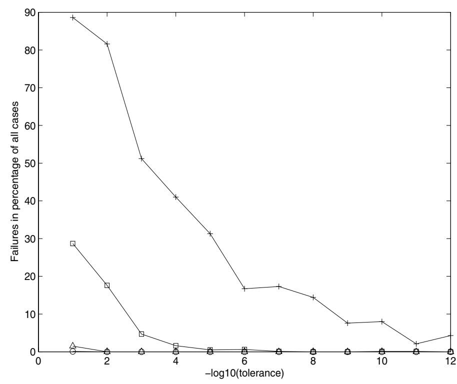
(a) Family 4: work



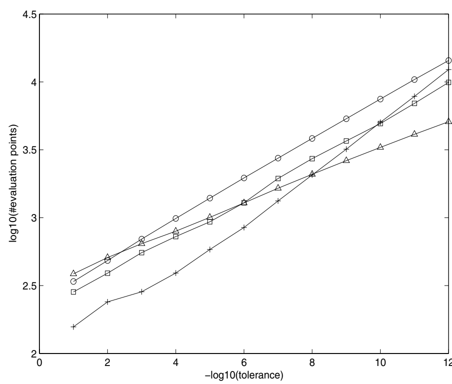
(b) Family 4: failures



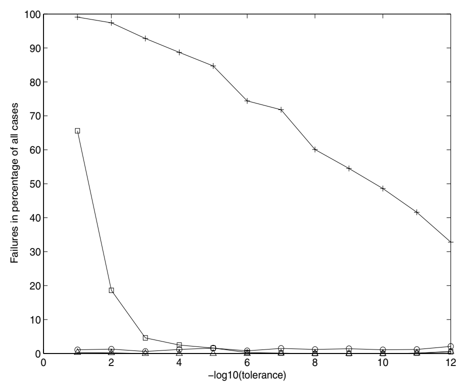
(c) Family 5: work



(d) Family 5: failures



(e) Family 6: work



(f) Family 6: failures

Figure 5.2: Test family 4, 5 and 6.

This is in contrast to a *globally adaptive* strategy where one in each step of the computation chooses as the next interval to be processed the interval with the greatest error estimate. This interval is then subdivided and both parts (using bisection) are processed and stored in a data structure before the next interval is picked. Thus one may update the current global approximation all the time, which implies that this approximation is as good as it possibly can be. Furthermore, in a global strategy one may set an upper bound on the number of function values to be used and then leave it to the algorithm to find the best approximation.

A disadvantage with the global strategy is the need to keep all intervals explicitly in a data structure in order to find the interval with the largest error estimate. It is quite common to use a heap to organize the intervals, however in this Matlab code we have decided to use a simple table in order to make use of Matlab's `max` function when searching for the next interval to be processed. The local strategy, making use of the recursive option, is considerably simpler and saves time using only a stack to store intervals that have to wait before they can be processed further.

These two *adaptive* strategies are in contrast to a so-called *non-adaptive* strategy where the original interval is never subdivided: given a sequence of quadrature rules $Q_1, Q_2, \ldots, Q_L$ for some integer $L > 1$. These rules are based on an increasing number of nodes and are possibly of increasing polynomial degree. The non-adaptive algorithm applies the rules one at the time starting with the cheapest rule, estimates the current error and then decide whether to stop or to continue with the next rule in the sequence.

One may combine an adaptive and a non-adaptive strategy as follows: (1) Pick an interval to be processed in an adaptive strategy and, (2) start a non-adaptive handling of this interval and, (3) do not bisect the interval until all L quadrature rules have been applied to this interval. The idea of such a doubly adaptive strategy was first presented by Cools and Haegemans in [5]. They also allowed the algorithm to decide how many of the available L rules to use before subdivision should take place.

We will develop two different doubly adaptive algorithms in the following. The two codes' properties can be described as follows:

- `coteda`: this is a new code making use of both the five point closed Newton–Cotes rule and the nine point closed Newton–Cotes rule in a locally doubly adaptive fashion based on bisection. The code is allowed to stop either because the five point estimate is considered good enough or because the nine point estimate is considered good enough. The nine point estimate requires four new points in addition to the five point estimate, but these are the same points needed in two applications of the five point rule after bisecting the interval.

- `coteglob`: This code is basically similar to `coteda` except that it applies a globally doubly adaptive strategy instead of the recursive strategy. This implies a need for explicit handling of data structures in order to retrieve

information about intervals that we want to process further at a later stage. The code is here mainly included in order to illustrate a major difference between local and global strategies.

The following two Newton–Cotes rules, here presented for the interval $[-1,1]$, are thus used by both codes

$$Q_A[f] = \{7[f(-1)+f(1)] + 32[f(-1/2)+f(1/2)] + 12f(0)\}/45,$$

and the nine point rule

$$Q_B[f] = \{989[f(-1)+f(1)] + 5888[f(-3/4)+f(3/4)] - 928[f(-1/2) + f(1/2)] + 10496[f(-1/4)+f(1/4)] - 4540f(0)\}/14175.$$

These two rules have degree of precision five and nine respectively. The nine point rule has some negative weights, however $||Q_B||_2 \approx 1.25$ compared to $||Q_A||_2 \approx 1.0634$ so there is little difference in the rules' 2-norms.

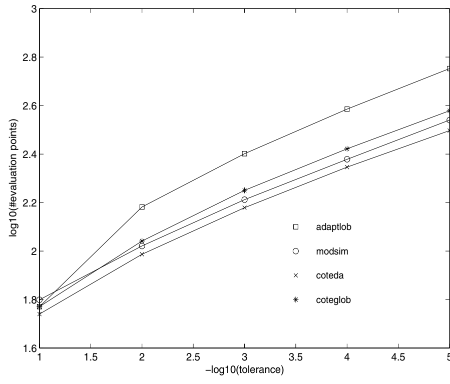
Furthermore, observe that having applied Q_B on an interval and deciding that it is necessary to bisect this interval then it is possible to apply Q_A to each half without any extra function evaluations. On the other hand, having applied Q_A to an interval and deciding that we have to process this interval further, then after computing four new function values we may apply Q_B on this interval and maybe stop or decide that further subdivision is necessary.

Using two rules is the simplest possible doubly adaptive strategy and we observe that in our case this is achieved with little extra cost compared to not introducing Q_B in the algorithm: we need a local error estimator to be designed for the nine point rule. Having nine nodes makes it possible to construct eight orthonormal null rules and then combine these rules in pairs, as in [2], to reduce phase factor effects:

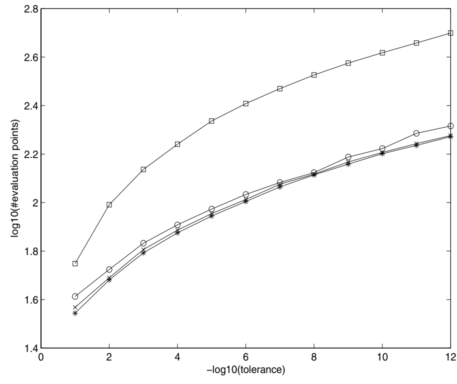
The local error estimating algorithm B

Compute:	$e_j = N_j[f],\ j = 1,2,\ldots,2K+2;$
	$E_j = \sqrt{e_{2j-1}^2 + e_{2j}^2},\ j = 1,2,\ldots,K+1;$
	$r_j = E_j/E_{j+1},\ j = 1,2,\ldots,K;$
	$r = \max_{j=1,2,\ldots,K}\ r_j;$
Non-asymptotic:	**if** $r > 1$ **then** $\hat{E} = C\ \max_{j=1,2,\ldots,K+1}\ E_j$
Weak asymptotic:	**elseif** $r_{\text{critical}} \le r$ **then** $\hat{E} = C\ r\ E_1$
Strong asymptotic:	**else** $\hat{E} = C\ r_{\text{critical}}^{1-\alpha}\ r^{\alpha}\ E_1$
	endif
The noise test:	**if** $E_1 < noise$ *and* $E_2 < noise$ **then** $\hat{E} = 0$

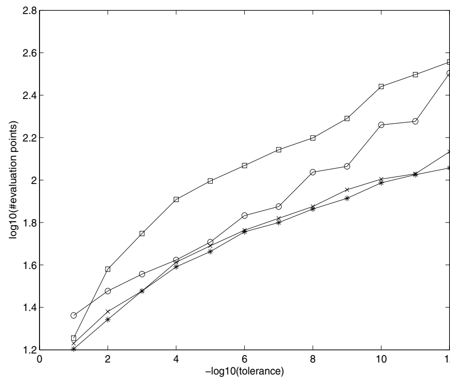
Note that $r = \mathrm{O}(h^2)$ if f is smooth and the interval is small enough. Both codes use this error estimator with $K = 3$, $r_{\text{critical}} = 1/4$. The constants C and α are set differently in `coteda` and `coteglob`. α is set one unit larger in the global code than in the local code. The basic algorithm for this doubly adaptive global code has the following structure:



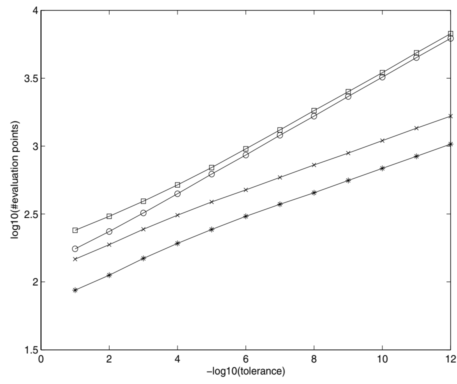
(a) Test family 1: Singularity.



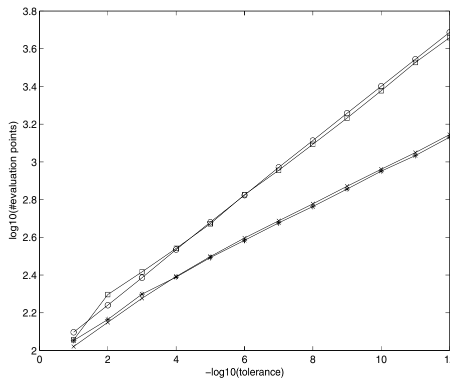
(b) Test family 2: Discontinuous.



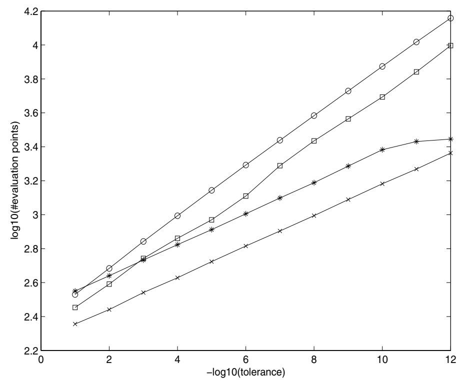
(c) Test family 3: C_0 function.



(d) Test family 4: One peak.



(e) Test family 5: Four peaks.



(f) Test family 6: Non-linear oscillatory.

Figure 5.3: Test families 1,2,. . . ,6: work.

A Globally Doubly Adaptive Quadrature Algorithm

Initialize:	Initialize the interval collection and put $M = 1$;
	Use Q_B to produce $\hat{Q}_1$, $\hat{E}_1$; Put $\hat{Q} = \hat{Q}_1$; $\hat{E} = \hat{E}_1$;
Control:	**while** $\hat{E} > \text{tol} * \lvert\hat{Q}\rvert$ **do**
	begin
	Pick an interval from the collection; say interval H_k;
Process this interval:	**if** rule Q_B has been applied **then**
Apply rule Q_A twice:	Compute $\hat{Q}_k^{(1)}$, $\hat{E}_k^{(1)}$, $\hat{Q}_k^{(2)}$, $\hat{E}_k^{(2)}$; Put $m = 2$;
	else
Apply rule Q_B once:	Compute $\hat{Q}_k^{(1)}$ and $\hat{E}_k^{(1)}$; Put $m = 1$;
	end
Update:	$\hat{Q} = \hat{Q} + \sum_{i=1}^{m} \hat{Q}_k^{(i)} - \hat{Q}_k$;
	$\hat{E} = \hat{E} + \sum_{i=1}^{m} \hat{E}_k^{(i)} - \hat{E}_k$;
	Let these m intervals replace interval H_k in
	the collection and put $M = M + m - 1$;
	end

As remarked earlier four new function values must be computed prior to applying rule Q_B (nine in the initialization step), while no new function evaluations is needed when rule Q_A is applied. Furthermore we have suppressed the test on too small intervals, designed in the spirit of Gander and Gautschi, the counting of function evaluations and a continuously updating of the *noise* level (through updating *isabs*) which all are included in the globally adaptive code.

We have tested these two new codes on the six families given in Table 4.1. When it comes to reliability the two new codes hold the same high quality on these tests as do `modsim` and `modlob` so we do not include plots on the failures for the two new codes. The interested reader is referred to the Appendix in [8] where this information is available. In Figure 5.3 we plot the work for the six different families.

`coteda` is the most efficient code on the test families 1 and 6, while `coteglob` is the most efficient code on the four other test families, however there are small differences between `coteglob` and `coteda` for test families 2, 3 and 5. Finally, `coteglob` gives a very reliable impression with very few failures. In addition this code gives a number of warnings about too small intervals when the accuracy request is high.

Family 4 demonstrates the advantage of the globally adaptive approach nicely. An initial estimate of such a peak function tends to underestimate the correct value, maybe with several orders of magnitude. The effect of this in a local code will be that the effective absolute error tolerance becomes much smaller than intended implying increased computational cost to meet such a requirement. A global code avoids this problem by updating the estimate regularly as the computation proceeds. This observation implies that comparisons of different local codes are difficult since they are all very sensitive to getting the magnitude of the initial estimate correct. A severe underestimate may therefore improve

reliability and ruin the efficiency no matter how good the error estimator and quadrature rule are, conversely an overestimate has the opposite effect.

Finally one should add that codes based on rules with few nodes, all codes in this paper uses less than ten nodes, may have trouble in handling some problems with a stronger difficulty than considered in this paper. To illustrate this one may e. g. try to increase the difficulty parameter for Test family 6 from two to three and discover that none of the codes in this paper are able to handle such a difficult oscillation problem well, especially for low accuracy requests. Tests done in [2] demonstrate that quadrature software based on basic rules using 21 nodes is able to handle this particular difficulty level for Test family 6 quite well. Thus, quadrature software packages should offer codes where choosing the number of nodes in the basic quadrature rule is a user option. Few nodes implies a very adaptive code, however for oscillating problems adaptivity is normally less important.

6 Results from the battery test

We have also tested all six codes[1] discussed in this paper on the 23 test problems used by Gander and Gautschi in their battery test. Gander and Gautschi [10, 11] have picked a total of 23 different test problems from two different sources: the 21 first comes from Kahaner [15] and the last two are picked from [12], see Table 6.1.

We have tested all six codes discussed in this paper on twelve different error tolerances tol $=10^{-1}, 10^{-2}, \ldots, 10^{-12}$ and the results can be found in the Appendix in [8]. In order to summarize the results of this battery test we have constructed the Table 6.2. Here we give, for each of the 23 problems, the following information:

- A blank position: for all twelve accuracies we have success.
- An integer gives the number of cases out of the twelve tested accuracies where we have a failure. In parenthesis we also give the number of cases where the error is more than one digit.
- A star means that this code uses the fewest number of function evaluations in at least six successful cases out of the twelve tested accuracies. The minimum number is picked among those of the six codes with a satisfied accuracy request. A star may appear in combination with a blank or an integer.

Observe that `coteda` and `coteglob` are the most efficient codes (indicated by the star) in 14 out the 23 problems tested. Furthermore there is little observed difference between these two codes for many of the tested problems. Only Problem 9 seems to have no winning code, however if we remove `adaptsim` from this

[1] The six Matlab codes are all available on the Web: `adaptsim` and `adaptlob` on W. Gander's homepage http://www.inf.ethz.ch/personal/gander/ while `modsim`, `modlob`, `coteda` and `coteglob` can be found via T. O. Espelid's homepage: http://www.ii.uib.no/~terje/.

Table 6.1: Test problems in this battery test.

	Test problems
1.	$\int_0^1 \exp(x)\,dx$.
2.	$\int_0^1 f(x)\,dx$, where $f = 1$ if $x > 0.3$ else $f = 0$.
3.	$\int_0^1 \sqrt{x}\,dx$.
4.	$\int_{-1}^1 (\frac{23}{25}\cosh(x) - \cos(x))\,dx$.
5.	$\int_{-1}^1 1/(x^4 + x^2 + 0.9)\,dx$.
6.	$\int_0^1 \sqrt{x^3}\,dx$.
7.	$\int_0^1 1/\sqrt{x}\,dx$.
8.	$\int_0^1 1/(1 + x^4)\,dx$.
9.	$\int_0^1 2/(2 + \sin(10\pi x))\,dx$.
10.	$\int_0^1 1/(1 + x)\,dx$.
11.	$\int_0^1 1/(1 + \exp(x))\,dx$.
12.	$\int_0^1 x/(\exp(x) - 1)\,dx$.
13.	$\int_{0.1}^1 \sin(100\pi x)/(\pi x)\,dx$.
14.	$\int_0^{10} \sqrt{5}\exp(-50\pi x^2)\,dx$.
15.	$\int_0^{10} 25\exp(-25x)\,dx$.
16.	$\int_0^{10} 50/(\pi(2500x^2 + 1))\,dx$.
17.	$\int_0^1 50(\sin(50\pi x)/(50\pi x))^2\,dx$.
18.	$\int_0^\pi \cos(\cos(x) + 3\sin(x) + 2\cos(2x) + 3\cos(3x))\,dx$.
19.	$\int_0^1 f(x)\,dx$, if $x > 10^{-15}$ then $f = \log(x)$ else $f = 0$.
20.	$\int_{-1}^1 1/(1.005 + x^2)\,dx$.
21.	$\int_0^1 \sum_{i=1}^3 1/\cosh(20^i(x - 2i/10))\,dx$.
22.	$\int_0^1 4\pi^2 x\sin(20\pi x)\cos(2\pi x)\,dx$.
23.	$\int_0^1 1/(1 + (230x - 30)^2)\,dx$.

competition due to lack of reliability then `coteda` becomes the best code in this case too.

`adaptsim` appears clearly as the most unreliable code of these six codes based on these $12 \times 23 = 276$ tests. `adaptsim` does not meet the requested error tolerance in 117 of these cases. Furthermore in 40 of these 117 cases the error is greater than one digit. This confirms the impression from the Lyness–Kaganove testing that this code is very unreliable.

`adaptlob` appears on the other hand to be very reliable based on this battery test with 11 failures in these 276 cases. 6 of these 11 failures are severe with more than one digit wrong. However, 4 out of these 6 severe failures appear on Problem 21. Thus the battery test and the Lyness–Kaganove test give a different impression of this code when it comes to reliability.

Table 6.2: Summarizing the results for the battery test.

Problem	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
1					*	*
2	12				*	*
3	11					*
4	3(2)				*	*
5						*
6						*
7	12		3		*	
8						*
9	4					
10					*	*
11					*	*
12					*	*
13	5(2)		1(1)		*	
14	8(3)				*	
15	11(4)				*	
16	10(5)				*	
17	10(9)	1	2(1)		3(2)/*	1
18						*
19	11					*
20						*
21	8(4)	3(2)	5(4)	9(8)	4(3)	5(4)/*
22	12(11)				*	
23					*	
Sum	117(40)	4(2)	11(6)	9(8)	7(5)	6(4)

The other four codes have all less than ten failures on this battery test. These failures appear on the problems 17 and 21. Problem 21 is a very difficult three peak problem where the width of the strongest peak is the major difficulty. We observe between three and nine failures for the four codes on this problem, most of them of the severe kind.

On the other hand, Problem 21, which all codes have trouble handling, becomes much easier for all codes if the problem is split in two intervals with division point 0.6 (the center of the strongest peak).

7 Conclusions

`adaptsim` turns out to be a very unreliable code both in the Lyness–Kaganove test and in the battery test. `adaptlob` on the other hands gives a very reliable impression in the battery test, while the Lyness–Kaganove test gives a different picture.

I will characterize both modifications as successful in the following sense: they both appear to be very reliable codes. Naturally we get an increased cost for

low accuracies, but when we have a high accuracy request both modifications demonstrate generally better economy than their counterparts `adaptsim` and `adaptlob`. However, compared to `coteda` and `coteglob` the two modifications are not able to compete with respect to efficiency neither for low nor for high accuracy requests.

Both `coteda` and `coteglob` demonstrate very good efficiency and reliability both in the Lyness–Kaganove test and in the battery test. Furthermore both codes have a very good error tolerance responsiveness: that is being sensitive to changes in the error tolerance. Finally both codes demonstrate that they are generally far better than both `adaptsim` and `adaptlob` when asking for high accuracy.

Five of the tested codes are based on Matlab's recursive function option and thus locally adaptive. `coteglob` is the only code in this test which is globally adaptive and needs an explicit data structure in order to handle the global strategy. If we do not include `adaptsim` and `coteglob` in the comparison (lack of reliability/explicit data structure) then `coteda` becomes the best code for all 23 problems in the battery test and at the same time the best code for all six test families in the Lyness–Kaganove test. Being aware of the fact that both `adaptsim` and `adaptlob` now, in slightly modified versions, have replaced `quad` and `quad8` in Matlab's quadrature software I would consider `coteda` a strong competitor to both these two new Matlab codes and to codes in other software libraries.

REFERENCES

1. J. Berntsen, *A test of some well known quadrature routines*, Reports in Informatics 20, Dept. of Informatics, Univ. of Bergen, Norway, 1986.
2. J. Berntsen and T. O. Espelid, *Error estimation in automatic quadrature routines*, ACM Trans. Math. Software, 17 (1991), pp. 233–252.
3. J. Berntsen, T. O. Espelid, and A. Genz, *A test of ADMINT*, Reports in Informatics 31, Dept. of Informatics, Univ. of Bergen, Norway, 1988.
4. J. Berntsen, T. O. Espelid, and A. Genz, *An adaptive algorithm for the approximate calculation of multiple integrals*, ACM Trans. Math. Software, 17 (1991), pp. 437–451.
5. R. Cools and A. Haegemans, *Cubpack: Progress report*, in Numerical Integration, Recent Developments, Software and Applications, T. O. Espelid and A. Genz, eds., NATO ASI Series C: Math. and Phys. Sciences Vol. 357, Kluwer Academic Publishers, Dordrecht, 1992, pp. 305–315.
6. C. de Boor, *On writing an automatic integration algorithm*, in Mathematical Software, J. R. Rice, ed., Academic Press, 1971.
7. T. O. Espelid, *DQAINT: An algorithm for adaptive quadrature over a collection of finite intervals*, in Numerical Integration, Recent Developments, Software and Applications, NATO ASI Series C: Mathematical and Physical Sciences - Vol. 357, Kluwer Academic Publishers, Dordrecht, 1992, pp. 341–342.
8. T. O. Espelid, *Doubly Adaptive Quadrature Routines based on Newton–Cotes rules*, Reports in Informatics 229, Dept. of Informatics, Univ. of Bergen, Norway, 2002.

9. T. O. Espelid and T. Sørevik, *A discussion of a new error estimate for adaptive quadrature*, BIT, 29 (1989), pp. 283–294.
10. G. Gander and W. Gautschi, *Adaptive quadrature—revisited*, Report 306, Dept. Informatik., ETH, Zurich, 1998.
11. G. Gander and W. Gautschi, *Adaptive quadrature—revisited*, BIT, 40 (2000), pp. 84–101.
12. S. Garriba, L. Quartapelle, and G. Reina, *Algorithm 36 - SNIFF: Efficient self-tuning algorithm for numerical integration*, Computing, 20 (1978), pp. 363–375.
13. A. Genz and A. Malik, *An adaptive algorithm for numerical integration over an N-dimensional rectangular region*, J. Comput. Appl. Math., 6 (1980), pp. 295–302.
14. E. Isaacson and H. Keller, *Analysis of Numerical Methods*, North Oxford Academic, 1966.
15. D. Kahaner, *Comparison of numerical quadrature formulas*, in Mathematical Software, J. Rice, ed., Academic Press, New York, 1971, pp. 229–259.
16. J. Lyness, *Symmetric integration rules for hypercubes III. Construction of integration rules using null rules*, Math. Comp., 19 (1965), pp. 625–637.
17. J. Lyness and J. Kaganove, *Comments on the nature of automatic quadrature routines*, ACM Trans. Math. Software, 2 (1976), pp. 65–81.
18. J. Lyness and J. Kaganove, *A technique for comparing automatic quadrature routines*, Comput. J., 20 (1977), pp. 170–177.
19. W. M. McKeeman, *Algorithm 145. Adaptive numerical integration by Simpson's rule*, Comm. ACM, 5 (1962), p. 604.
20. R. Piessens, E. de Doncker-Kapenga, C. Überhuber, and D. Kahaner, *QUADPACK, A Subroutine Package for Automatic Integration*, Series in Computational Math., 1., Springer-Verlag, Berlin, 1983.
21. T. Sørevik, *Reliable and Efficient Algorithms for Adaptive Quadrature*, Thesis for the degree Doctor Scientiarum, Department of Informatics, University of Bergen, Norway, 1988.
22. P. van Dooren and L. de Ridder, *An adaptive algorithm for numerical integration over an N-dimensional cube*, J. Comput. Appl. Math., 2 (1976), pp. 207–217.