REPORTS
IN
INFORMATICS

ISSN 0333-3590

Doubly Adaptive Quadrature Routines based on Newton-Cote Rules

Terje O. Espelid

REPORT NO 229 May 2002

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL `http://www.ii.uib.no/publikasjoner/texrap/ps/2002-229.ps`

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at `http://www.ii.uib.no/publikasjoner/texrap/`.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Doubly Adaptive Quadrature Routines based on Newton-Cote Rules

Terje O. Espelid

30th May 2002.

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.

1 Introduction.

Automatic algorithms are now used widely for the numerical calculation of integrals. Since the first such algorithm was given by McKeeman [18] in 1962, many new and sophisticated algorithms, both adaptive and non-adaptive, have been developed, among these [4, 6, 12, 19, 21].

Recently Gander and Gautschi [9, 10] 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 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. [19], Berntsen and Espelid [2] and Espelid [7]. One of the most successful adaptive quadrature algorithms so far, see [1] and Espelid and Sørevik [8], is QAG in QUADPACK, [19]. 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-Cote 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-Cote 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

$$I[f] \;=\; \int_a^b f(x)dx. \tag{1}$$

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

$$Q[f] = \sum_{i=-n}^{n} w_i f(x_i). \tag{2}$$

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 $\le d$ and fails to integrate exactly $f(x)=x^{d+1}$. An interpolatory quadrature rule based on these $2n+1$ distinct nodes has degree at least $2n$. A quadrature rule based on $2n+1$ distinct nodes of degree $d \ge 2n$ 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 J.N. Lyness, [15]. The following definition of a null rule is useful in this context.

Definition 1 *A rule*

$$N[f] = \sum_{i=-n}^{n} u_i f(x_i) \tag{3}$$

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

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

Note: Changing the direction of integration in (1) will leave $I[f]$ uneffected. Using symmetric rules and null rules imply that both $Q[f]$ and $N[f]$ are uneffected 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 uneffected of this change too.

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 $i = 1, 2, \ldots, n$

Note: Changing the direction of integration in (1) will leave $I[f]$ uneffected. Using symmetric rules and null rules imply that both $Q[f]$ and $N[f]$ are uneffected 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 uneffected of this change too.

Suppose that Q and N have degrees $d_Q \geq 0$ and d_N respectively. Then

$$Q_\lambda[f] = Q[f] + \lambda N[f] = \sum_{i=-n}^{n} (w_i + \lambda u_i) f(x_i)$$

is a quadrature rule of degree $d \geq \min(d_Q, d_N)$. Thus we see that a null rule can be written as the difference between two different quadrature rules of degrees ≥ 0, e. g.

$$\lambda N[f] = Q_\lambda[f] - Q[f],$$

and the scaling of a null rule is similar to replacing the rule of reference Q_λ. A null rule based on $2n+1$ nodes has furthermore degree of precision at most $2n-1$. This is obvious from the fact that if a null rule has degree of precision at least $2n$, then two different quadrature rules of precision at least $2n$ have to exist contradicting the uniqueness theorem.

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. The following formula is well known, see e. g. E. Isaacson and H.B. Keller [13],

$$f[z_0, z_1, \ldots, z_m] = \sum_{j=0}^{m} f(z_j) / \prod_{i=0, i\neq j}^{m} (z_j - z_i). \tag{4}$$

If f is sufficiently smooth, then

$$f[z_0, z_1, \ldots, z_m] = f^{(m)}(\xi_m)/m!, \tag{5}$$

for a value of $\xi_m \in [\min z_i, \max z_i]$. Now, (4) shows that a divided difference is a linear combination of function values and (5) shows that this linear combination of function values 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 given in (4) 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, define an inner product between two null rules, N_u and N_v, based on the same set of $2n+1$ points as follows

$$(N_u, N_v) = \sum_{i=-n}^{n} u_i v_i. \tag{6}$$

We obviously have, with this inner product, that a symmetric null rule and an anti-symmetric null rule are orthogonal null rules. Furthermore, we may define a 2-norm to a null rule as $||N_u||_2^2 = (N_u, N_u)$. It is now 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, all odd numbered null rules will retain it's symmetry after this orthogonalization process and this will similarly be true for the even numbered anti-symmetric null rules.

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 $N_j[f] = O(h^{2n+2-j})$ for $j = 1, 2, \ldots, 2n$.

3 A local error estimating algorithm.

In the following we will develop a local error estimating algorithm. The algorithm consists of several elements, each one with its own motivation. Let us start this section with summarizing our knowledge about the problems of error estimation in adaptive quadrature:

- *Asymptotic/non-asymptotic*: The classical way to estimate the local error is based on the knowledge that asymptotically, that is for f sufficiently smooth and the local interval h sufficiently small, we have $\mid I - Q \mid < |N[f]|$. As we know, using $|N[f]|$ as a local error estimate in an adaptive algorithm may give an unreliable routine and it is therefore important to test whether we, in a particular interval, have a situation where the asymptotic theory may be applied or not. We will design such a test using a sequence of local error estimates of decreasing asymptotic order in h. Thus we may test if the computed values correspond to the expected order of this sequence. In the asymptotic case we will like to use an optimistic error estimate based both on the knowledge of the null rule's and the basic rule's degree of precision.

- *Phase-factor*: As Lyness and Kaganove [16] have pointed out, phase factor effects may ruin $|N[f]|$ as an estimate of the true error: given a function $f_\lambda(x)$ which depends on a parameter λ. Then it is easy to demonstrate that $|N[f_\lambda]|$ may take the value zero for several values of the parameter λ while the true error stays away from zero for these values of λ. We will take such phase factors effects into account when designing the error estimator.

- *Rounding-noise*: Local error estimation focus on estimating the local *truncation error* as reliable and economical as possible. Rounding errors may of course influence both the estimate of the local integral and the evaluation of all null rules. We will, as in [19], define a certain noise level for the problem. All local error estimates below this noise level are defined to be zero, thus avoiding to subdivide such intervals further.

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

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

Observe that asymptotically

$$E_j = O(h^{2n+2-j}), \quad j = 1, 2, \ldots, 2n. \tag{7}$$

Define the local error

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

giving the asymptotic expression

$$E_0 = O(h^{d+2}), \tag{8}$$

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

$$E_0 \ll E_1 \ll E_2 \ll \cdots \ll E_{2n}. \tag{9}$$

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 $r = O(h)$ asymptotically, and we would therefore expect $r < 1$ when h becomes sufficiently small. In view of (9) we see that a necessary test on whether h is small enough and f sufficiently smooth, is to check that $r < r_{critical}$ for a heuristic value of $r_{critical} < 1$. If this test is passed, we may apply an optimistic error estimate based on (7) and (8)

$$\hat{E} = c\, r^{\alpha} E_2. \tag{10}$$

Choosing E_2 in this error estimate instead of E_1 is an attempt to reduce possible phase factor effects 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 factors at the same time. Observing that $r_0 = E_0/E_2$ is of order $O(h^{d+2-2n})$ 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. We will use the term *strong test on asymptotic behavior* on the test $r < r_{critical}$. The test $r_{critical} \leq r \leq 1$ we will denote *the weak test on asymptotic behavior* and when this test is passed we will suggest to damp the optimism, using an error estimate as follows

$$\hat{E} = c'\; r\; E_2. \tag{11}$$

In order to give a continuous error estimating function we choose

$$c' = c\, r_{critical}^{\alpha-1}.$$

The last test to consider is $1 < r$ which we denote *the test on non-asymptotic behavior.* In this case we cannot use (7) and (8) as a guide in the construction of the error estimate. In the non-asymptotic region we consider any ordering of the elements of this set as equally probable. Therefore we may, by simply bad luck, have $r \leq 1$ even though we are in the non-asymptotic region. However, the probability that this will occur is then $1/(K+1)!$. Since we in addition distinguish between weak and strong asymptotic behavior, we assume that we can usually maintain the reliability in spite of such failures. We suggest the following error estimate in the non-asymptotic case

$$\hat{E} = C \max_{j=1,2,\ldots,K+1} E_j. \tag{12}$$

The constant C should be chosen as a reasonable guarding constant to take care of the rare, but possible, situation that $E_0 > \max E_j$. Having chosen this constant equal to C, and in order to have as smooth an error estimating function as possible, we put

$$c' = c\, r_{critical}^{\alpha-1} = C,$$

giving

$$c = C\, r_{critical}^{1-\alpha}.$$

In order to test whether we have reached the noise level or not we have also introduced a noise test, 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. We summarize this section by giving a local error estimating algorithm including a noise test

The local error estimating algorithm A

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

Note: 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 $\hat{E} = 0$.

We have used The local error estimating algorithm A to modify the two codes developed by Gander and Gautschi in [9, 10]. 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-Cote 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-Cote 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 [9, 10]. 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.

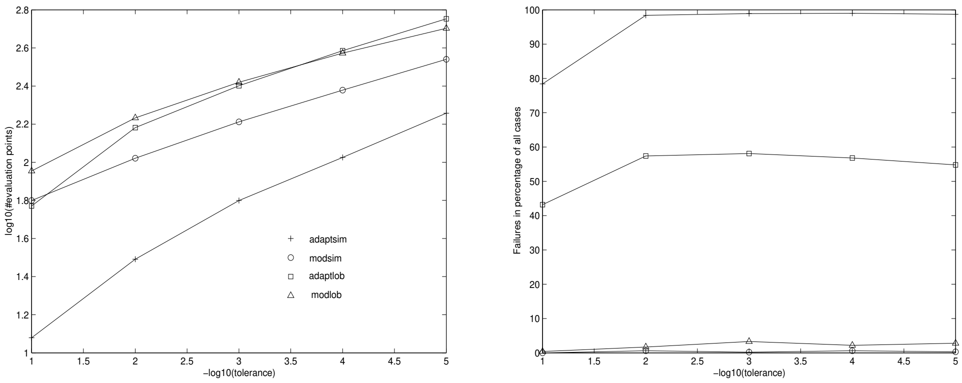
In both `modsim` and `modlob` we use $C = 32$, $r_{critical} = 1/2$ and $K = 3$. We have $d = 5$ in `modsim` and $d = 9$ in `modlob` and we have chosen α one unit less than the maximum values in both cases due to the fact that we have locally adaptive algorithms. Furthermore in `modsim` and `modlob` 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 below, and they are picked from [1] ,[17] and [20].

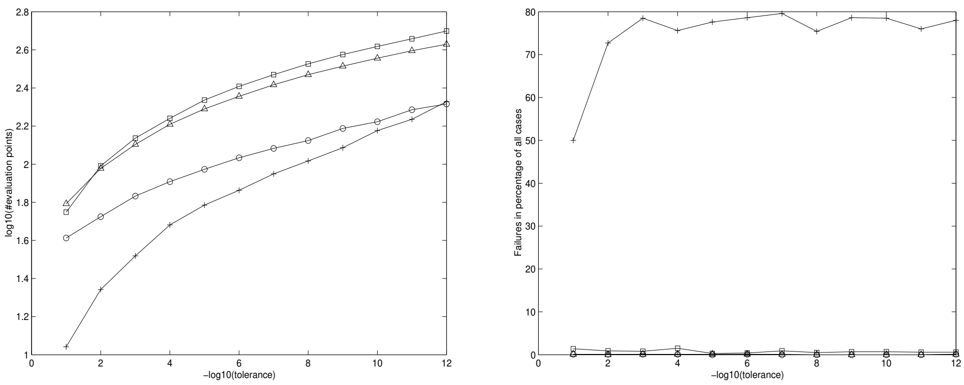
	Test families	Attributes
1.	$\int_0^1 (\lvert x-\lambda\rvert)^{\alpha_1} dx$	Singularity
2.	$\int_0^1 f_2(x)dx$ where $f_2(x,y) = \begin{cases} 0 & \text{if } x \le \lambda \\ exp(\alpha_2 x) & \text{otherwise} \end{cases}$	Discontinuous
3.	$\int_0^1 exp(-\alpha_3 \lvert x-\lambda\rvert)dx$	C_0 function
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 $\alpha_i, i = 1, ..., 6$, 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 $\lambda_i, i = 1, ..., 4$, 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 to the Appendix where we list six tables containing all the results from these tests.



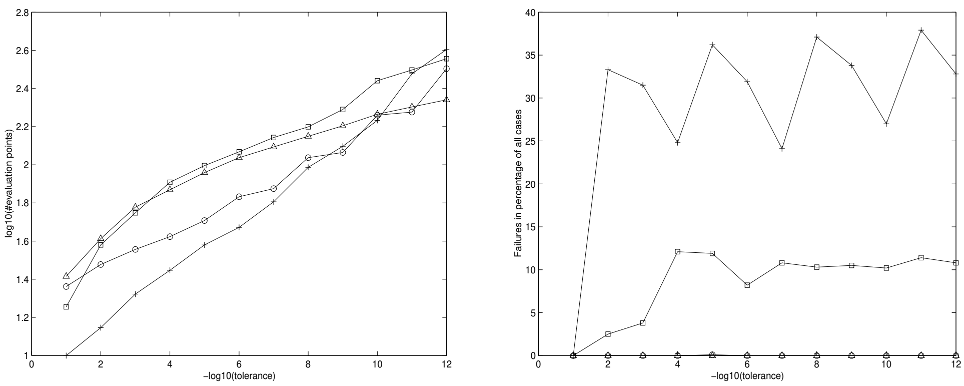
(a) work (b) failures

Figure 1: Test family 1 (Singularity).



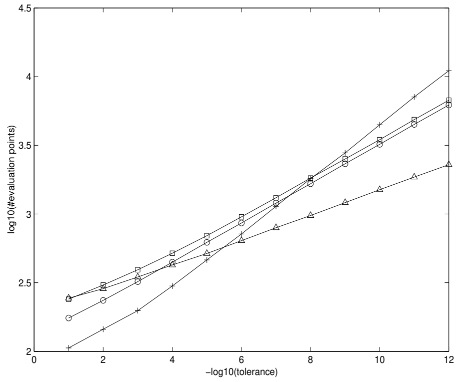
(a) work (b) failures

Figure 2: Test family 2 (Discontinuous).

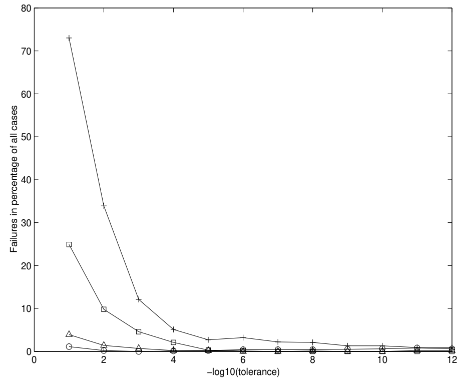


(a) work (b) failures

Figure 3: Test family 3 (C_0 function).

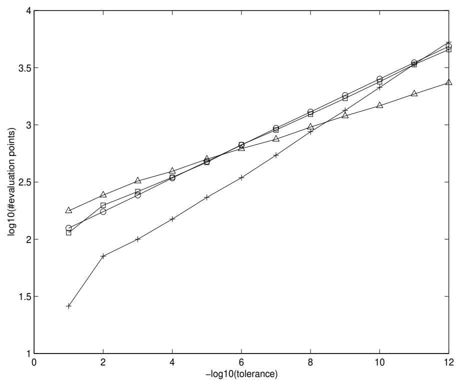


(a) work

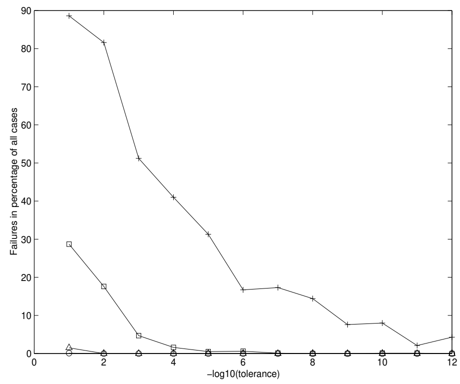


(b) failures

Figure 4: Test family 4 (One peak).

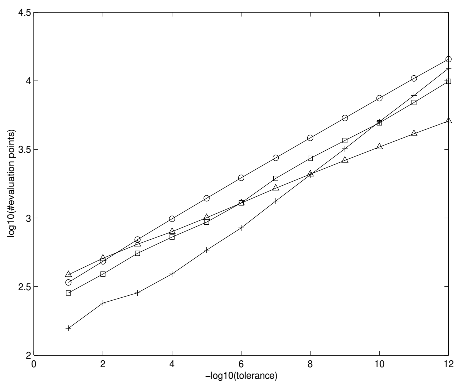


(a) work

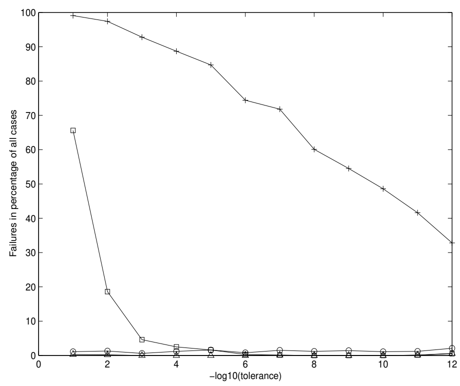


(b) failures

Figure 5: Test family 5 (Four peaks).



(a) work



(b) failures

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

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 the modified codes appear to be generally more efficient than their counterparts, especially for higher accuracies, say asking for more than five correct 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 Newton-Cote 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 to 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.

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-Cote rule and the nine point closed Newton-Cote 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-Cote 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

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

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 ($L = 2$) is the simplest possible doubly adaptive strategy and we observe that in our case this is achieved with basically no extra cost compared to not introducing Q_B in the algorithm. Of course introducing Q_B implies some cost: we need a local error estimator to be designed for the nine point rule. Having nine points makes it possible to construct eight orthonormal null rules and then to combine these rules into pairs to reduce phase factor effects. This means that nine inner products have to be computed each time we want to apply Q_B:

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** $r_{critical} \le r$ **then** $\hat{E} = C\ r\ E_1$
Strong asymptotic:	**else** $\hat{E} = C\ r_{critical}^{1-\alpha}\ r^{\alpha}\ E_1$
	endif
The noise test:	**if** $E_1 < noise\ and\ E_2 < noise$ **then** $\hat{E} = 0$

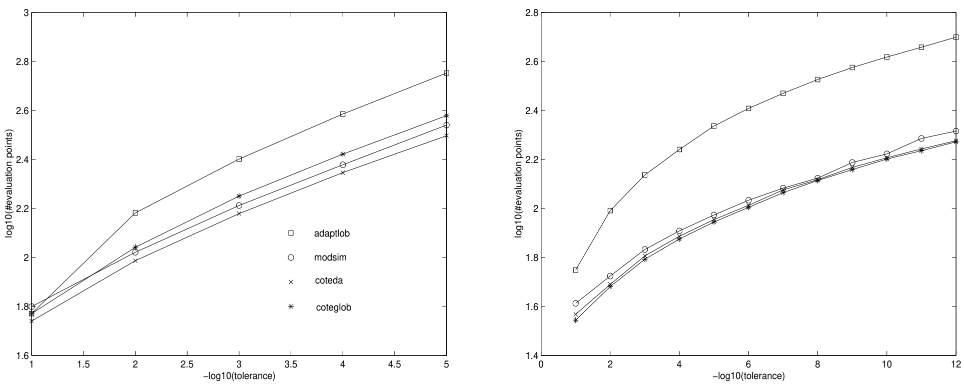
Note that $r = O(h^2)$ if f is smooth and the interval is small enough. Both codes use this error estimator with $K = 3$ and $r_{critical} = 1/4$. The constants C and α are chosen differently in `coteda` and `coteglob`: α is set one unit larger in the global code than in the local code. We present the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

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** $\hat{E} > tol * \lvert\hat{Q}\rvert$ **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

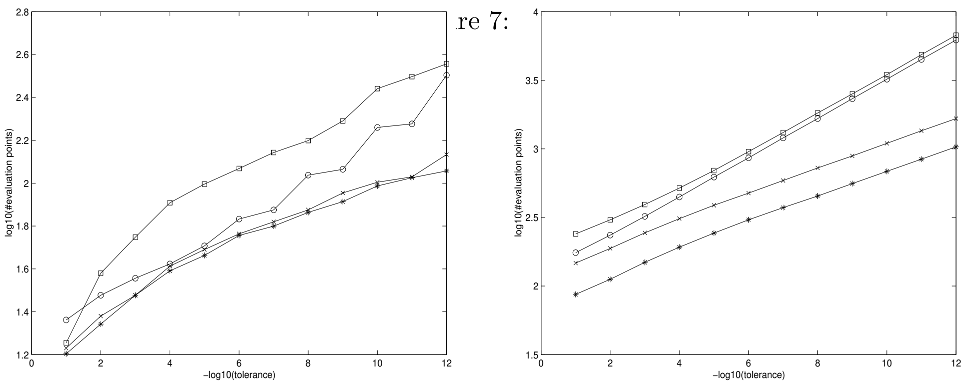
As remarked earlier four new function values must be computed prior to applying rule Q_B (nine in the initialization step), while no new function evaluations is needed when rule Q_A is applied. Furthermore we have suppressed the test on too small intervals, designed in the spirit of Gander and Gautschi, the counting of function evaluations and a continuous 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 Section 4. 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. We refer to the Appendix where all information from these tests is available. We plot here the work only for the six different families.



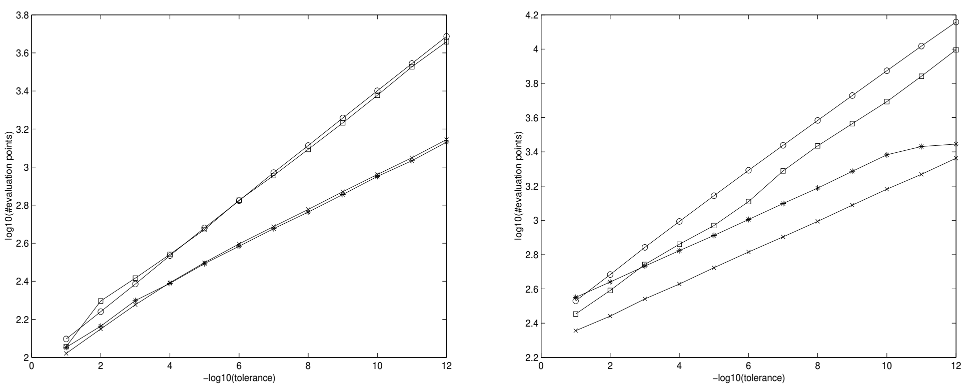
(a) Test family 1: Singularity. (b) Test family 2: Discontinuous.

re 7:



(a) Test family 3: C_0 function. (b) Test family 4: One peak.

Figure 8: Work.



(a) Test family 5: Four peaks. (b) Test family 6: Non-linear oscillatory.

Figure 9: Work.

`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, and conversely an overestimate has the opposite effect.

Finally one should add that codes based on rules with less ten nodes, all codes in this paper use less than ten nodes, may have trouble in handling more difficult problems 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 are able to handle this particular 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 discussed in this paper on the 23 test problems used by Gander and Gautschi in their battery test. Gander and Gautschi [9, 10] have picked a total of 23 different test problems from two different sources: the 21 first comes from Kahaner [14] and the last two are picked from [11].

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. $\int_0^{10} \sqrt{50}\exp(-50\pi x^2)dx$.
15. $\int_0^{10} 25\exp(-25x)dx$.
16. $\int_0^{10} 50/(\pi(2500x^2 + 1))dx$.
17. $\int_{0.01}^1 50(\sin(50\pi x)/(50\pi x))^2 dx$.
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. $\int_0^1 1/(1 + (230x - 30)^2)dx$.

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 order to summarize the results of this battery test we have constructed the following table. Here we give, for each of the 23 problems, the following information:

1. A blank position: for all twelve accuracies we have success.

2. 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.

3. 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.

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)

Summarizing the results for the battery test.

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 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.

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 hand 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.

Appendix: the numerical experiments.

We report tests on the following six Matlab routines

- `adaptsim`: this code is developed by Gander and Gautschi, [9, 10].
- `modsim`: this code is a modification of `adaptsim` presented in this paper.
- `adaptlob`: this code is developed by Gander and Gautschi, [9, 10];
- `modlob`: this is a modification of the previous code presented in this paper.
- `coteda`: this code is developed in this paper.
- `coteglob`: this code is developed in this paper.

All six Matlab codes are 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/.

The Lyness-Kaganove test

The testing technique we have used in this part is due to Lyness and Kaganove [17]. This technique is based on a selection of test families of integrands. Each test family has a special attribute, a difficulty parameter and one or more random parameters. The test families used in these experiments are given below, and they are picked from [1] ,[17] and [20]. These test families have furthermore been used by Berntsen and Espelid in [2].

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-lin. Oscill.

In our experiments we have chosen the difficulty parameters $\alpha_i, i = 1,...,6$, 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 $\lambda_i, i = 1,...,4$, for test family 5), are picked randomly from the region of integration using the Matlab code random('unif',. . .). The global code is designed to accept a maximum number of allowed function values and this value is set in all experiments in this paper equal to 30 000. We have tested the codes for error tolerances tol = $10^{-1}, 10^{-2}, ..., 10^{-12}$. (For 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 but a few cases the routines report that the returned values satisfies the error request. An exception here is the code coteglob which gives a warning about too small intervals on a number of occasions, all cases appear when asking for the maximum precision. The experiments have been run on a SUN Ultra 10.

In the tables below we report on the performance of the different routines for each test family and for each error request. For each routine the four numbers from left to right are:

1. The average number of integrand evaluations. Only the cases where the error request is satisfied are included in the average.
2. The average number of correct digits in the returned value. Here too, only the cases where the error request is satisfied are included in the average.
3. Failures in percent: that is cases where the actual error is greater than the error tolerance.
4. The average number of wrong digits in the cases described above.

The number of wrong digits is computed by

$$\text{Wrong digits} = \lvert -log_{10}(tol) + log_{10}(\epsilon_{act})\rvert,$$

where ϵ_{act} is the actual relative accuracy in the returned value.

tol	adaptsim				modsim				coteda			
10^{-1}	12	1.4	78.4	0.3	63	2.6	0.0	0.0	55	2.5	1.3	0.2
10^{-2}	31	2.4	98.4	1.0	105	3.8	0.6	0.1	97	3.6	2.2	0.3
10^{-3}	63	3.3	98.9	1.3	163	4.7	0.2	0.1	151	4.6	2.0	0.3
10^{-4}	106	4.5	99.0	1.3	239	5.5	0.6	0.4	222	5.4	1.9	0.3
10^{-5}	181	5.4	98.7	1.4	347	6.3	0.3	0.6	314	6.3	1.6	0.3
tol	adaptlob				modlob				coteglob			
10^{-1}	59	1.4	43.2	0.2	90	2.0	0.4	0.2	59	2.5	2.3	0.2
10^{-2}	152	2.5	57.4	0.5	171	3.0	1.7	0.3	110	3.8	1.2	0.2
10^{-3}	252	3.5	58.1	0.6	263	4.0	3.3	0.3	178	4.8	0.5	0.2
10^{-4}	385	4.5	56.8	0.7	374	5.0	2.2	0.3	264	5.8		
10^{-5}	566	5.6	54.8	0.7	505	6.0	2.8	0.2	379	6.8	0.6	4.3

Test Family 1: Singularity.

tol	adaptsim				modsim				coteda			
10^{-1}	11	1.4	50.0	0.4	41	3.4	0.1	0.2	37	3.4	0.1	0.2
10^{-2}	22	2.4	72.7	0.6	53	4.2	0.1	0.6	49	4.2	0.1	0.6
10^{-3}	33	3.5	78.5	0.6	68	5.4	0.1	0.8	64	5.4	0.1	0.8
10^{-4}	48	4.5	75.6	0.6	81	6.4	0.1	0.4	77	6.4	0.9	0.4
10^{-5}	61	5.4	77.6	0.6	94	7.3			90	7.3		
10^{-6}	73	6.4	78.6	0.6	108	8.3	0.1	0.6	103	8.4	0.1	0.7
10^{-7}	89	7.4	79.6	0.6	121	9.4			119	9.3		
10^{-8}	104	8.4	75.4	0.6	133	10.3			131	10.3		
10^{-9}	122	9.4	78.6	0.6	154	11.3			147	11.2		
10^{-10}	150	10.4	78.5	0.6	167	12.3			161	12.3		
10^{-11}	172	11.5	76.0	0.6	193	13.3			175	13.3		
10^{-12}	213	12.4	78.0	0.6	207	14.3	0.1	0.2	189	14.2	0.1	0.2
tol	adaptlob				modlob				coteglob			
10^{-1}	56	2.1	1.4	0.1	62	2.8	0.1	0.9	35	3.2		
10^{-2}	95	3.1	0.9	0.3	95	3.8	0.0	0.0	48	4.1		
10^{-3}	137	4.0	0.8	0.1	127	4.8	0.1	0.6	62	5.2		
10^{-4}	174	5.0	1.5	0.2	162	5.8	0.1	0.3	75	6.2		
10^{-5}	217	6.0	0.3	0.6	195	6.8	0.1	0.6	88	7.2		
10^{-6}	256	7.1	0.4	0.5	227	7.7	0.1	0.8	101	8.2		
10^{-7}	295	8.0	0.9	0.2	261	8.8	0.1	0.7	116	9.2		
10^{-8}	336	9.1	0.5	0.4	295	9.8			130	10.2		
10^{-9}	376	10.1	0.7	0.2	327	10.7			144	11.2		
10^{-10}	415	11.0	0.7	0.2	360	11.8			159	12.2		
10^{-11}	455	12.1	0.6	0.2	394	12.7			172	13.2		
10^{-12}	500	13.1	0.6	0.2	426	13.7			187	14.1		

Test Family 2: Discontinuous.

tol	adaptsim				modsim				coteda			
10^{-1}	10	1.7			23	3.8			17	3.5		
10^{-2}	14	2.5	33.3	0.2	30	4.9			24	4.5		
10^{-3}	21	3.5	31.5	0.3	36	6.0			30	5.5		
10^{-4}	28	4.6	24.8	0.3	42	6.7			41	6.4		
10^{-5}	38	5.5	36.2	0.2	51	7.9	0.1	0.3	49	7.5	0.1	0.3
10^{-6}	47	6.5	31.9	0.2	68	9.0			58	8.5		
10^{-7}	64	7.6	24.1	0.2	75	9.6			66	9.5		
10^{-8}	97	8.5	37.1	0.2	109	10.9			75	10.5		
10^{-9}	125	9.5	33.8	0.3	116	11.7			90	11.4		
10^{-10}	171	10.5	27.0	0.2	182	12.7			101	12.4		
10^{-11}	300	11.5	37.9	0.2	189	13.5			107	13.5		
10^{-12}	402	12.5	32.8	0.2	319	14.7			136	14.4		
tol	adaptlob				modlob				coteglob			
10^{-1}	18	2.3			26	3.4			16	3.3		
10^{-2}	38	3.4	2.5	0.1	41	4.3			22	4.2		
10^{-3}	56	4.2	3.8	0.4	60	5.5			30	5.4		
10^{-4}	81	5.4	12.1	0.5	75	6.3			39	6.2		
10^{-5}	99	6.4	11.9	0.4	91	7.3			46	7.3	0.2	0.4
10^{-6}	117	7.3	8.2	0.5	109	8.5			57	8.3		
10^{-7}	139	8.3	10.8	0.2	124	9.3			63	9.3		
10^{-8}	158	9.4	10.3	0.5	141	10.4			73	10.4		
10^{-9}	195	10.4	10.5	0.6	160	11.4			82	11.4		
10^{-10}	276	11.4	10.2	0.5	184	12.3			97	12.5		
10^{-11}	314	12.3	11.4	0.5	201	13.4			106	13.2		
10^{-12}	360	13.3	10.8	0.6	219	14.4			114	14.4		

Test Family 3: C_0 function.

tol	adaptsim				modsim				coteda			
10^{-1}	106	3.9	73.0	1.0	175	5.4	1.1	0.9	147	5.6	2.0	0.9
10^{-2}	145	4.8	33.9	2.0	235	6.3	0.2	0.7	188	6.5	0.4	1.6
10^{-3}	198	5.7	12.1	2.9	322	7.1			244	7.4		
10^{-4}	299	6.8	5.1	3.2	446	8.0	0.1	0.7	310	8.4	0.1	0.3
10^{-5}	463	7.9	2.7	1.7	622	8.8	0.2	0.3	388	9.5	0.2	0.4
10^{-6}	715	9.0	3.2	0.9	860	9.7	0.4	0.5	476	10.3	0.3	0.7
10^{-7}	1136	10.3	2.2	0.9	1201	10.5	0.4	0.5	588	11.3	0.2	0.7
10^{-8}	1808	11.4	2.1	1.1	1661	11.4	0.4	0.7	726	12.3	0.3	0.6
10^{-9}	2786	12.7	1.3	1.0	2320	12.2	0.5	0.6	888	13.1	0.2	0.8
10^{-10}	4465	13.8	1.3	0.9	3216	13.1	0.6	0.7	1097	14.0	0.2	0.5
10^{-11}	7111	14.8	0.9	1.2	4486	13.9	0.8	0.6	1354	14.8	0.3	0.9
10^{-12}	11046	15.2	0.9	1.1	6212	14.7	0.6	1.0	1662	15.2	0.2	0.9
tol	adaptlob				modlob				coteglob			
10^{-1}	240	5.8	24.9	1.0	244	6.5	3.9	1.0	87	3.6		
10^{-2}	304	7.0	9.8	1.9	286	7.3	1.4	2.0	112	4.7		
10^{-3}	393	8.4	4.6	2.9	348	8.2	0.7	3.0	149	5.7		
10^{-4}	518	9.7	2.1	3.9	426	9.2	0.2	4.0	192	6.6		
10^{-5}	695	11.0	0.3	5.0	516	10.2	0.2	0.6	243	7.3		
10^{-6}	954	12.3			640	10.9			304	8.4		
10^{-7}	1314	13.0			795	11.9			373	9.5		
10^{-8}	1825	13.7			975	13.0	0.1	0.1	453	10.3		
10^{-9}	2516	13.8			1211	13.8			558	11.0		
10^{-10}	3474	13.9			1502	14.5			686	12.2		
10^{-11}	4861	14.1			1857	14.7	0.2	0.1	841	13.4		
10^{-12}	6734	14.3			2285	15.0	0.2	0.2	1035	13.7		

Test Family 4: One Peak.

tol	`adaptsim`				`modsim`				`coteda`			
10^{-1}	26	1.4	88.6	0.6	125	3.9	0.1	0.2	105	3.5	0.5	0.2
10^{-2}	71	2.8	81.6	1.2	174	4.9			141	4.6		
10^{-3}	100	3.6	51.2	1.1	243	5.7			189	5.6		
10^{-4}	150	4.8	41.0	0.7	343	6.7			247	6.6		
10^{-5}	232	5.8	31.3	0.5	479	7.4			315	7.6		
10^{-6}	345	7.1	16.7	0.7	666	8.4			395	8.6		
10^{-7}	543	8.1	17.3	0.5	935	9.1			487	9.5		
10^{-8}	870	9.2	14.4	0.5	1298	10.2			599	10.5		
10^{-9}	1339	10.3	7.6	0.4	1811	10.8			743	11.6		
10^{-10}	2126	11.4	8.0	0.4	2518	11.9			914	12.2		
10^{-11}	3387	12.5	2.1	0.6	3501	12.5			1120	13.0		
10^{-12}	5287	13.6	4.3	0.4	4862	13.6			1395	14.2		
tol	`adaptlob`				`modlob`				`coteglob`			
10^{-1}	114	2.3	28.7	0.4	177	4.2	1.5	0.2	113	3.8	0.1	0.6
10^{-2}	198	4.2	17.6	0.8	243	5.5			146	4.8		
10^{-3}	261	5.7	4.7	1.3	323	6.8	0.1	0.6	199	5.8		
10^{-4}	348	7.2	1.6	0.8	393	7.1			245	6.4		
10^{-5}	470	8.7	0.5	1.3	501	8.7			311	7.7		
10^{-6}	670	9.9	0.6	1.2	620	9.5			384	8.4		
10^{-7}	903	11.4	0.1	0.1	749	9.9			475	8.9		
10^{-8}	1239	12.6			955	11.4			580	10.2		
10^{-9}	1707	14.1			1197	12.4			718	11.5		
10^{-10}	2383	15.1	0.1	0.8	1468	12.5			893	12.0		
10^{-11}	3368	15.4	0.1	1.8	1862	14.0			1080	12.8		
10^{-12}	4558	15.5			2338	15.0			1352	14.5		

Test Family 5: Four Peaks.

tol	adaptsim				modsim				coteda			
10^{-1}	157	1.9	99.1	2.6	339	3.1	1.1	1.1	227	3.0	1.4	2.0
10^{-2}	240	2.8	97.4	3.1	483	3.9	1.3	0.5	276	3.9	1.3	0.7
10^{-3}	285	3.6	92.8	2.3	696	4.9	0.6	0.8	348	4.8	1.2	0.4
10^{-4}	391	4.7	88.7	1.5	986	5.8	1.2	0.4	425	5.8	0.8	0.6
10^{-5}	583	5.6	84.7	1.2	1393	6.7	1.6	0.5	529	7.0	2.1	0.5
10^{-6}	847	6.7	74.4	1.1	1962	7.8	0.8	0.6	654	7.7	0.9	0.6
10^{-7}	1329	7.7	71.8	1.0	2744	8.7	1.5	0.5	802	8.5	2.4	0.5
10^{-8}	2065	8.7	60.1	1.0	3835	9.7	1.2	0.5	987	9.6	2.5	0.5
10^{-9}	3198	9.7	54.5	0.9	5354	10.7	1.4	0.6	1228	10.6	3.2	0.5
10^{-10}	5038	10.8	48.6	0.8	7478	11.6	1.1	0.7	1522	11.4	5.3	0.3
10^{-11}	7825	11.9	41.6	0.8	10413	12.5	1.2	0.5	1857	12.4	2.8	0.1
10^{-12}	12321	12.9	32.8	0.8	14386	13.4	2.1	0.4	2307	13.2	6.5	0.5
tol	adaptlob				modlob				coteglob			
10^{-1}	284	3.5	65.6	2.2	386	4.5	0.3	1.2	355	4.9	0.5	3.1
10^{-2}	390	4.3	18.6	2.8	509	5.7	0.2	3.0	437	5.8		
10^{-3}	553	5.7	4.6	3.1	644	6.5			541	7.1		
10^{-4}	726	7.1	2.5	1.2	796	7.6			666	7.7		
10^{-5}	933	8.6	1.6	1.3	1005	8.4			817	8.5		
10^{-6}	1288	9.9	0.3	0.6	1285	9.3			1012	9.6		
10^{-7}	1945	11.2	0.1	1.0	1649	10.3	0.1	0.2	1255	10.5		
10^{-8}	2723	12.5			2087	11.2			1543	11.4		
10^{-9}	3671	13.4			2631	12.1			1933	12.4		
10^{-10}	4937	13.8			3295	13.1			2416	13.3		
10^{-11}	6946	13.9	0.1	0.5	4113	13.8	0.1	0.3	2700	13.6	0.1	0.3
10^{-12}	9912	14.0	0.5	0.6	5091	14.0	0.6	0.5	2790	13.8	2.5	0.4

Test family 6: Nonlinear Oscillatory.

The battery test

Gander and Gautschi [9, 10] have picked a total of 23 different test problems from two different sources: the 21 first comes from Kahaner [14] and the last two are picked from [11]. In the following 23 tables we report on the result of testing the six codes on these 23 problems. We specify twelve different error tolerances tol $=10^{-1}, 10^{-2}, \ldots, 10^{-12}$ and report on the number of function evaluations used by each of the six codes. A minus sign in front of this number indicates that the requested error bound was NOT met by the actual code in this particular case.

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	10	17	18	17	9	9
10^{-5}	14	17	18	17	9	9
10^{-6}	22	17	18	17	9	9
10^{-7}	22	29	18	17	9	9
10^{-8}	38	33	18	17	9	9
10^{-9}	58	49	18	17	17	9
10^{-10}	70	65	18	17	17	17
10^{-11}	134	81	18	17	17	17
10^{-12}	218	129	18	27	33	17

Test problem 1: $\int_0^1 \exp(x)dx$.

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-10	33	48	57	29	37
10^{-2}	-18	49	78	87	45	45
10^{-3}	-34	65	108	127	61	61
10^{-4}	-42	73	168	157	69	69
10^{-5}	-58	89	198	187	85	85
10^{-6}	-74	105	198	227	101	101
10^{-7}	-82	113	258	257	109	109
10^{-8}	-98	129	288	287	125	125
10^{-9}	-114	145	318	327	141	141
10^{-10}	-122	153	378	357	149	149
10^{-11}	-138	169	408	387	165	165
10^{-12}	-154	185	438	427	181	181

Test problem 2: $\int_0^1 f(x)dx$, where $f = 1$ if $x > 0.3$ else $f = 0$.

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	10	17	18	17	9	9
10^{-2}	-10	25	18	37	21	17
10^{-3}	-18	37	48	57	29	29
10^{-4}	-26	45	78	87	41	37
10^{-5}	-38	57	78	107	57	53
10^{-6}	-54	73	108	127	73	69
10^{-7}	-82	101	168	147	89	85
10^{-8}	-126	141	228	187	113	109
10^{-9}	-190	201	288	237	161	149
10^{-10}	-290	265	438	277	193	181
10^{-11}	-498	365	558	347	241	229
10^{-12}	-750	505	708	437	297	285

Test problem 3: $\int_0^1 \sqrt{x}dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	-10	17	18	37	9	9
10^{-5}	-10	33	18	37	9	9
10^{-6}	-10	41	18	57	9	9
10^{-7}	30	57	18	57	17	17
10^{-8}	54	73	18	77	17	17
10^{-9}	94	129	48	77	33	33
10^{-10}	126	153	48	117	33	33
10^{-11}	198	217	48	137	33	33
10^{-12}	342	305	48	217	65	49

Test problem 4: $\int_{-1}^1 (\frac{23}{25}\cosh(x) - \cos(x))dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	10	17	18	37	9	9
10^{-2}	14	17	18	37	17	17
10^{-3}	14	41	18	37	17	17
10^{-4}	22	41	18	37	33	33
10^{-5}	38	49	48	57	33	33
10^{-6}	46	89	48	77	33	33
10^{-7}	70	129	48	77	49	41
10^{-8}	118	177	48	117	65	57
10^{-9}	174	249	108	197	81	65
10^{-10}	254	337	168	237	113	81
10^{-11}	486	457	168	277	129	113
10^{-12}	662	649	228	337	129	129

Test problem 5: $\int_{-1}^{1} 1/(x^4 + x^2 + 0.9)dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	18	21	48	37	17	17
10^{-5}	22	29	48	47	25	21
10^{-6}	30	37	48	57	37	29
10^{-7}	54	49	78	77	45	41
10^{-8}	70	69	108	87	57	49
10^{-9}	106	93	138	107	77	65
10^{-10}	190	137	198	147	93	89
10^{-11}	266	177	258	177	121	109
10^{-12}	414	261	318	207	157	129

Test problem 6: $\int_{0}^{1} \sqrt{x^3}dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-10	65	48	117	61	61
10^{-2}	-30	93	108	187	89	85
10^{-3}	-54	117	168	247	113	121
10^{-4}	-86	161	-198	317	161	177
10^{-5}	-130	221	348	377	217	229
10^{-6}	-190	305	-438	447	269	289
10^{-7}	-322	417	-588	567	361	389
10^{-8}	-490	585	888	697	473	493
10^{-9}	-730	801	1158	837	581	613
10^{-10}	-1262	1101	1578	1047	745	781
10^{-11}	-1918	1541	2358	1277	945	977
10^{-12}	-2894	2133	3138	1567	1165	1217

Test problem 7: $\int_{0}^{1} 1/\sqrt{x}dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	21	18	17	9	9
10^{-2}	10	21	18	17	9	9
10^{-3}	14	25	18	17	17	17
10^{-4}	14	29	18	17	17	17
10^{-5}	22	45	18	27	17	17
10^{-6}	34	65	18	67	33	25
10^{-7}	46	93	48	67	33	33
10^{-8}	66	133	48	87	41	41
10^{-9}	126	193	48	97	49	41
10^{-10}	162	261	48	97	65	57
10^{-11}	254	357	108	147	89	73
10^{-12}	482	489	168	197	105	89

Test problem 8: $\int_0^1 1/(1+x^4)dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	89	48	97	65	77
10^{-2}	38	113	138	177	89	97
10^{-3}	-54	193	198	277	129	145
10^{-4}	-94	281	198	317	177	177
10^{-5}	134	369	288	397	225	217
10^{-6}	198	497	498	557	257	265
10^{-7}	-358	761	648	617	337	353
10^{-8}	542	969	858	757	401	393
10^{-9}	-750	1449	1188	1057	497	497
10^{-10}	1438	1889	1698	1297	609	625
10^{-11}	2134	2753	2538	1657	769	761
10^{-12}	2982	3657	3258	1997	929	921

Test problem 9: $\int_0^1 2/(2+\sin(10\pi x))dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	14	17	18	17	9	9
10^{-5}	18	17	18	17	9	9
10^{-6}	26	25	18	17	17	17
10^{-7}	34	33	18	17	17	17
10^{-8}	54	45	48	27	25	17
10^{-9}	82	61	48	27	25	25
10^{-10}	118	81	48	37	33	25
10^{-11}	202	113	48	47	41	33
10^{-12}	302	161	48	57	49	41

Test problem 10: $\int_0^1 1/(1+x)dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	10	17	18	17	9	9
10^{-5}	14	17	18	17	9	9
10^{-6}	14	21	18	17	9	9
10^{-7}	22	29	18	17	9	9
10^{-8}	34	33	18	17	9	9
10^{-9}	38	53	18	27	9	9
10^{-10}	66	69	18	37	17	9
10^{-11}	122	97	48	47	17	17
10^{-12}	134	133	48	57	25	17

Test problem 11:$\int_0^1 1/(1+\exp(x))dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	17	18	17	9	9
10^{-2}	10	17	18	17	9	9
10^{-3}	10	17	18	17	9	9
10^{-4}	10	17	18	17	9	9
10^{-5}	10	17	18	17	9	9
10^{-6}	14	17	18	17	9	9
10^{-7}	14	17	18	17	9	9
10^{-8}	22	21	18	17	9	9
10^{-9}	38	25	18	17	9	9
10^{-10}	38	37	18	17	9	9
10^{-11}	70	49	18	17	9	9
10^{-12}	134	73	18	27	17	9

Test problem 12:$\int_0^1 x/(\exp(x)-1)dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	-22	669	-198	717	513	753
10^{-2}	-218	857	678	867	513	929
10^{-3}	-470	1273	768	1327	633	1001
10^{-4}	-658	1857	1038	1567	961	1313
10^{-5}	1014	2601	1368	1847	1025	1777
10^{-6}	-1706	3581	2328	2357	1089	1953
10^{-7}	2694	5213	3138	3127	1817	2169
10^{-8}	4030	6945	4398	3847	2017	3433
10^{-9}	6886	10157	5748	4787	2049	3849
10^{-10}	10790	13649	6918	6157	3273	4041
10^{-11}	16126	19593	12468	7597	3977	4897
10^{-12}	27614	26337	15288	9307	4081	4897

Test problem 13:$\int_{0.1}^1 \sin(100\pi x)/(\pi x)dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-14	37	48	77	33	37
10^{-2}	-26	49	78	87	37	37
10^{-3}	-34	61	78	87	45	45
10^{-4}	-38	65	108	97	49	53
10^{-5}	42	89	138	117	53	61
10^{-6}	54	105	168	127	69	85
10^{-7}	-70	165	168	157	93	97
10^{-8}	-102	197	228	167	97	105
10^{-9}	-134	293	288	197	125	137
10^{-10}	234	361	468	247	153	165
10^{-11}	342	557	528	357	173	181
10^{-12}	-458	697	678	397	193	245

Test problem 14: $\int_0^{10} \sqrt{50} \exp(-50\pi x^2) dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-14	37	78	57	29	33
10^{-2}	-26	37	78	67	33	37
10^{-3}	-34	53	78	67	41	49
10^{-4}	38	65	108	77	49	53
10^{-5}	-42	77	138	87	53	69
10^{-6}	-50	101	168	97	69	81
10^{-7}	-74	141	168	117	89	97
10^{-8}	-98	181	198	137	97	121
10^{-9}	-138	257	288	167	129	141
10^{-10}	-230	333	498	187	157	181
10^{-11}	-354	481	618	237	189	217
10^{-12}	-498	629	708	287	225	261

Test problem 15: $\int_0^{10} 25 \exp(-25x) dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-14	37	78	77	33	41
10^{-2}	-26	49	108	87	41	49
10^{-3}	-34	49	108	107	41	69
10^{-4}	-42	73	108	107	65	81
10^{-5}	-46	105	168	117	81	89
10^{-6}	-62	137	168	117	97	129
10^{-7}	-90	193	258	177	137	161
10^{-8}	-146	265	378	217	169	185
10^{-9}	-226	373	528	287	193	225
10^{-10}	346	509	648	347	249	281
10^{-11}	578	721	1008	427	305	353
10^{-12}	-870	993	1308	547	377	433

Test problem 16: $\int_0^{10} 50/(\pi(2500x^2+1)) dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-10	69	78	67	41	241
10^{-2}	-26	197	78	227	161	265
10^{-3}	-34	485	-168	557	-261	-321
10^{-4}	-42	-689	-498	947	-297	793
10^{-5}	-90	1237	918	1197	-393	961
10^{-6}	-238	1709	1008	1587	905	1081
10^{-7}	-534	2505	1458	1917	1065	1449
10^{-8}	990	3529	2718	2347	1185	1841
10^{-9}	-1438	4905	3588	3007	1625	2057
10^{-10}	-2282	6817	4518	3877	2017	2569
10^{-11}	-3962	9557	6318	4877	2273	3561
10^{-12}	5798	13433	8568	6187	2849	4009

Test problem 17:$\int_{0.01}^{1} 50(\sin(50\pi x)/(50\pi x))^2 dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	30	85	48	97	49	49
10^{-2}	34	117	78	137	57	57
10^{-3}	58	153	168	147	73	81
10^{-4}	102	245	198	227	113	105
10^{-5}	170	305	228	267	137	137
10^{-6}	218	425	258	417	177	169
10^{-7}	410	625	498	497	209	193
10^{-8}	658	869	618	587	257	241
10^{-9}	894	1201	768	697	329	297
10^{-10}	1626	1661	1128	837	385	353
10^{-11}	2614	2353	1368	1127	473	441
10^{-12}	3566	3253	2118	1437	569	537

Test problem 18:$\int_{0}^{\pi} \cos(\cos(x) + 3\sin(x) + 2\cos(2x) + 3\cos(3x))dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	-10	49	48	77	45	45
10^{-2}	-26	65	78	117	57	53
10^{-3}	-46	81	108	147	73	69
10^{-4}	-58	97	138	187	97	93
10^{-5}	-86	125	198	217	125	121
10^{-6}	-134	169	258	257	149	149
10^{-7}	-194	225	348	307	193	189
10^{-8}	-286	301	468	387	253	241
10^{-9}	498	417	648	447	305	297
10^{-10}	-730	565	858	557	385	377
10^{-11}	-1098	785	1188	677	489	469
10^{-12}	-1914	1085	1758	827	601	589

Test problem 19:$\int_{0}^{1} f(x)dx$, if $x > 10^{-15}$ then $f = \log(x)$ else $f = 0$.

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	10	17	18	17	9	9
10^{-2}	14	17	18	17	9	9
10^{-3}	14	25	18	17	17	17
10^{-4}	22	41	18	37	17	17
10^{-5}	38	65	18	37	33	25
10^{-6}	38	81	48	57	33	33
10^{-7}	70	121	48	57	49	33
10^{-8}	118	161	48	117	49	49
10^{-9}	142	217	48	137	65	57
10^{-10}	254	321	48	177	81	65
10^{-11}	318	457	168	197	113	89
10^{-12}	542	601	168	277	129	121

Test problem 20: $\int_{-1}^{1} 1/(1.005 + x^2)dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	18	65	48	47	49	53
10^{-2}	-22	97	138	97	77	73
10^{-3}	-46	-117	-168	-157	-93	-85
10^{-4}	-78	-169	-228	-227	-129	-121
10^{-5}	-110	-229	-258	-287	-161	-153
10^{-6}	-170	481	-378	-327	-193	-193
10^{-7}	-274	681	-528	-377	381	-233
10^{-8}	-478	905	1008	-507	461	445
10^{-9}	842	1293	1278	-667	581	573
10^{-10}	1330	1781	1788	-817	737	725
10^{-11}	-1862	2489	2388	-1037	913	873
10^{-12}	3306	3401	3378	1797	1117	1097

Test problem 21: $\int_{0}^{1} \sum_{i=1}^{3} 1/\cosh(20^i(x - 2i/10))dx$

tol	`adaptsim`	`modsim`	`adaptlob`	`modlob`	`coteda`	`coteglob`
10^{-1}	-10	185	168	187	129	129
10^{-2}	-10	229	198	257	129	209
10^{-3}	-10	297	288	367	177	249
10^{-4}	-10	465	588	447	257	257
10^{-5}	-10	689	678	607	257	377
10^{-6}	-10	981	1008	787	289	465
10^{-7}	-10	1417	1128	927	465	497
10^{-8}	-10	1913	1848	1097	505	641
10^{-9}	-10	2777	2928	1437	529	841
10^{-10}	-10	3761	3498	1857	833	953
10^{-11}	-10	5425	5058	2327	985	1113
10^{-12}	-10	7369	6078	2907	1033	1649

Test problem 22: $\int_{0}^{1} 4\pi^2 x \sin(20\pi x)\cos(2\pi x)dx$

tol	adaptsim	modsim	adaptlob	modlob	coteda	coteglob
10^{-1}	50	57	108	117	49	49
10^{-2}	58	61	108	127	49	73
10^{-3}	94	81	168	167	73	93
10^{-4}	142	129	198	217	97	113
10^{-5}	198	153	288	247	121	145
10^{-6}	354	209	348	307	157	197
10^{-7}	530	317	468	377	209	233
10^{-8}	782	425	738	477	241	289
10^{-9}	1402	597	1038	567	305	345
10^{-10}	2118	817	1218	757	369	417
10^{-11}	3110	1165	1608	907	457	505
10^{-12}	5606	1601	2538	1157	545	625

Test problem 23: $\int_0^1 1/(1+(230x-30)^2)dx$

References

[1] J. Berntsen. A test of some well known quadrature routines. Reports in Informatics 20, Dept. of Informatics, Univ. of Bergen, 1986.

[2] J. Berntsen and T. O. Espelid. Error Estimation in Automatic Quadrature Routines. *ACM Trans. Math. Softw.*, 17(2):233–252, 1991.

[3] J. Berntsen, T. O. Espelid, and A. Genz. A test of ADMINT. Reports in Informatics 31, Dept. of Informatics, Univ. of Bergen, 1988.

[4] J. Berntsen, T. O. Espelid, and A. Genz. An Adaptive Algorithm for the Approximate Calculation of Multiple Integrals. *ACM Trans. Math. Softw.*, 17(4):437–451, 1991.

[5] R. Cools and A. Haegemans. Cubpack: Progress report. In T. O. Espelid and A. Genz, editors, *Numerical Integration, Recent Developments, Software and Applications*, NATO ASI Series C: Math. and Phys. Sciences Vol. 357, pages 305–315, Dordrecht, The Netherlands, 1992. Kluwer Academic Publishers.

[6] C. de Boor. On writing an automatic integration algorithm. In J. R. Rice, editor, *Mathematical Software*. 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, pages 341–342. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.

[8] T. O. Espelid and T. Sørevik. A discussion of a new error estimate for adaptive quadrature. *BIT*, 29:283–294, 1989.

[9] G. Gander and W. Gautschi. Adaptive Quadrature - Revisited. Report 306, Dept. Informatik., ETH, Zurich, 1998.

[10] G. Gander and W. Gautschi. Adaptive Quadrature - Revisited. *BIT*, 40(1):84–101, 2000.

[11] S. Garriba, L. Quartapelle, and G. Reina. Algorithm 36 - SNIFF: Efficient self-tuning algorithm for numerical integration. *Computing*, 20:363–375, 1978.

[12] A.C. Genz and A.A. Malik. An adaptive algorithm for numerical integration over an N-dimensional rectangular region. *J. Comp. Appl. Math.*, 6(4):295–302, 1980.

[13] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. North Oxford Academic, 1966.

[14] D.K. Kahaner. Comparison of numerical quadrature formulas. In J. Rice, editor, *Mathematical Software*, pages 229–259, New York, 1971. Academic Press.

[15] J.N. Lyness. Symmetric integration rules for hypercubes III. Construction of integration rules using null rules. *Math. Comp.*, 19:625–637, 1965.

[16] J.N. Lyness and J.J. Kaganove. Comments on the nature of automatic quadrature routines. *ACM Trans. Math. Softw.*, 2(1):65–81, 1976.

[17] J.N. Lyness and J.J. Kaganove. A technique for comparing automatic quadrature routines. *Computer Journal*, 20:170–177, 1977.

[18] W. M. McKeeman. Algorithm 145, adaptive numerical integration by Simpson's rule. *CACM*, 5(12):604, 1962.

[19] R. Piessens, E. de Doncker-Kapenga, C.W. Überhuber, and D.K. Kahaner. *QUADPACK, A Subroutine Package for Automatic Integration.* Series in Computational Math., 1. Springer-Verlag, 1983.

[20] T. Sørevik. Reliable and Efficient Algorithms for Adaptive Quadrature. Technical report, Thesis for the degree Doctor Scientiarum, Department of Informatics, University of Bergen, 1988.

[21] P. van Dooren and L. de Ridder. An adaptive algorithm for numerical integration over an N-dimensional cube. *J. Comp. Appl. Math.*, 2(3):207–217, 1976.