REPORTS IN INFORMATICS

ISSN 0333-3590

Extended Doubly Adaptive Quadrature Routines

Terje O. Espelid

REPORT NO 266 February 2004

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/ps/2004-266.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

Extended Doubly Adaptive Quadrature Routines

Terje O. Espelid

February 2004

Abstract

In this paper we discuss how to modify a recently published Matlab code, `coteglob`, such that the excellent performance this code demonstrates for low and intermediate accuracy requests is retained while the performance is improved for high accuracy requests. `coteglob` is a globally adaptive code using a 5 and 9 point pair of Newton-Cotes rules. Combining an extended sequence of rules using 5, 9, 17 and 33 points with a doubly adaptive bisection strategy is the main focus of the paper. We also discuss local versus global adaptivity and conclude that globally adaptive codes are to be preferred. Based on this we develop several new globally adaptive codes that all compare favorably both with `coteglob` and with Matlab's best currently available quadrature software `qaudl`. We include the results from extensive testing using both a Lyness-Kaganove testing technique and 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, 5, 12, 20, 22, 10, 8].

Recently Espelid [7, 8] published a paper describing two new adaptive quadrature codes, `coteda` and `coteglob`. These two codes are written in Matlab and they were compared to Gander and Gautschy's Matlab codes `adaptsim` and `adaptlob`, [9, 10], through a Lyness-Kaganove testing technique,[16], and a battery test, [7, 8]. Through these tests `coteda` and `coteglob` demonstrate both better reliability and better efficiency than `adaptsim` and `adaptlob`. Especially for moderate to high accuracy requests the two new codes are superior comparing the number of function values needed to meet the accuracy request. Gander and Gautschi on the other hand states that their two codes are better than other codes available in different software packages in 2/3 of all cases in their battery test.

We know that when you have a high accuracy request it is usually better to use rules of high degree of precision, implying more points in the basic rule. A low accuracy request may be achieved cheaper with rules of low degree of polynomial precision, implying fewer points in the basic rule. Based on this observation we want to extend the sequence of rules from two to three or four in a doubly adaptive code: either 5, 9 and 17 or 5, 9, 17 and 33 point rules. We consider only closed rules based on equidistant points but they are not all Newton-Cotes rules. We will develop some new rules based on 17 and 33 points in this paper.

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. More than ten years ago Berntsen and Espelid, [3], 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. While the error estimator developed in [3] was based on a sequence of *symmetric null rules* of different polynomial degrees, Espelid in [6] 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.

The two codes, `coteda` and `coteglob`, use a sequence of four and eight null rules respectively in their error estimators and the good reliability observed are due to this type of error estimators.

In the following we will first describe how to extend the doubly adaptive strategy. Then we will construct the rules we are going to use in the new experimental codes and discuss which sequences of rules to include in these codes. Next we will give some sets of null rules, present three error estimating algorithms based on these null rules and take a close look at how these error estimators will handle a classical example. Then we will discuss local versus global adaptivity, present the extended globally adaptive algorithm and finally we will present results for both Lyness-Kaganove tests and a battery test for all new codes and in addition `coteglob` and `quadl` (Matlab's version of `adaptlob`) for comparison.

2 Extending the doubly adaptive strategy.

We define the integral to be computed by

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

We want to approximate this integral using doubly adaptive quadrature software.

Assume that we have $2n+1$ equidistant points $x_i = i/n$, $i = -n, \ldots, n$ in the interval $[-1, 1]$ and a symmetric 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. Symmetry implies that $w_i = w_{-i}$ and we assume furthermore that at least constants are integrated exactly by this rule, thus $\sum_{i=-n}^{n} w_i = 2$. A rule is said to have degree of precision d if it integrates exactly all monomials of degree $\leq d$ and fails to integrate exactly $f(x) = x^{d+1}$.

We want to compute estimates to $I[f]$ in an adaptive manner. Let us focus on a particular interval, applying the $2n+1$ point quadrature rule and an associated error estimator. If we are not satisfied with the error we will bisect the interval and apply the same rule on each half thus using a total of $4n+1$ function evaluations. Before we perform the bisection step we may first try a different rule using these $4n+1$ points over the whole interval. In order to do that we need to evaluate f in the $2n$ new nodes. If we are not satisfied with this error either, we may bisect the interval and apply the $2n+1$ point rule twice (no additional function evaluations are needed).

This is exactly how `coteda` and `coteglob` work with $n = 2$ in both cases: the difference between the two codes are that they are locally and globally adaptive respectively. Choosing $n = 2$ in these two codes is a compromise between adaptivity (few nodes in the rule) and reliability (Rule of thumb: the number of null-rules in the error estimator should at least be four).

Using this strategy in general we have applied two different quadrature rules Q_A and Q_B of different polynomial degrees over the same interval of length h. Let us assume that the local order of the error is respectively $\alpha < \beta$. Then it is reasonable to assume that for h small enough and f smooth

$$2K_A(h/2)^\alpha \geq K_B h^\beta, \tag{3}$$

implying that, at some point, the bisection strategy applying Q_A twice is a waste of effort and we should from this point on apply Q_B directly on the two halves and any subdivision that may be needed of this interval.

This observation is quite important and if it is possible to test, in a reliable manner, whether the situation (3) has occurred or not then we may take proper actions based on that. Using the error estimators to compute estimates to all three errors (the whole interval and both halves) we may test a similar relation (3) between these three quantities in the code.

Assume that the test indicates that (3) is satisfied: then there is no point in using the rule Q_A any longer neither on this interval nor on any subintervals of this interval. This opens the scene

for a new pair of rules, namely Q_B and Q_C: Q_B is now the preferred rule for this interval and if we have to process this interval further we have to bisect the interval and apply Q_B twice. This in turn opens for a new rule, Q_C, with $8n+1$ nodes which we will use before the bisection takes place. In our experiments with $n=2$ the C-rule therefore has 17 nodes.

At a later stage in this process a similar relation, (3), between the rules B and C may be satisfied which then opens for a new rule D using $16n+1$ nodes and thus the pair of rules B, C can be replaced by the pair C, D.

We will in our experiments try sequences of rules with 5, 9 and 17 nodes implying two pairs (A, B) and (B, C). In addition we will test several sequences of rules using 5, 9, 17 and 33 evaluation points implying three pairs of rules: (A, B), (B, C) and (C, D). We may of course continue this process, however based on the problems in the tests we have performed it seems doubtful that extending the sequence to five rules and thus four pairs is useful.

3 New quadrature rules and sequences of rules.

The rules we construct will be used in a modified doubly adaptive code and one may specify any wanted tolerance, either absolute or relative, therefore we may in many cases come close to machine precision. Therefore it is important how these rules propagate errors in the function values. Let us denote the error in $f(x_i)$ as δf_i then we have

$$|\delta Q[f]| = |\sum_{i=-n}^{n} w_i \delta f_i| \leq \sum_{i=-n}^{n} |w_i||\delta f_i|, \leq (\sum_{i=-n}^{n} w_i^2)^{1/2} (\sum_{i=-n}^{n} (\delta f_i)^2)^{1/2}. \tag{4}$$

The last inequality is the well known Cauchy-Schwarz inequality which states that the worst case error in $Q[f]$ is a product of the 2-norm to the weight vector and the 2-norm to the error-function vector. Given the symmetry we may define the 2-norm to a symmetric rule as follows

$$||Q||_2 = (w_0^2 + 2\sum_{i=1}^{n} w_i^2)^{1/2}.$$

and it is obviously important in the choice of rules to avoid large values in this 2-norm, (4). The theoretical minimum value of this 2-norm occurs when all weights are equal giving $w_i = 2/(2n+1)$ and thus rules of degree of precision at least zero satisfy $||w||_2 \geq 2/\sqrt{2n+1}$ for intervals of length 2. We have decided in this experiment to accept rules with 2-norm up to 100 and we are therefore prepared to loose up to 2 digits in order to achieve a higher degree of precision.

Here are two Newton-Cotes rules that are used by both `coteda` and `coteglob`. First the five point rule

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

and then the nine point rule

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

We observe that the nine point rule has some negative weights. The two rules have degree of precision 5 and 9 respectively and applying them to an interval of length h implies an error $O(h^7)$ and $O(h^{11})$ respectively if f is a sufficiently smooth function. Furthermore $||Q_B||_2 \approx 1.25$ and $||Q_A||_2 \approx 1.0634$ which is less than twice the minimum value in both cases ($2/3$ and $2/\sqrt{5} \approx 0.89$) and therefore acceptable. We will use these two rules in all sequences we test.

3.1 Rules based on 17 nodes.

The Newton-Cotes rule of degree 17 is found using Maple by integrating the Lagrange polynomials. We give the weights for this rule in column one in Table 1. The rule's 2-norm is 41.795 compared

Table 1: 17 nodes symmetric quadrature rules $(w_0, \ldots, w_8)$ with minimum 2-norm.

Degree of prec.: 17	15	13	11
-20.951950514333334	2.0738965950389237	0.01778721553184694	0.15179361661478292
19.180342211078733	-1.2870774416966075	0.14045687987840804	0.12928967978816338
-13.934614237197880	0.39257951974485817	0.23355426108135809	0.09795254569981574
8.5525299934402953	0.73769703510789268	0.06057557034342839	0.11351099934264011
-3.9501480717783930	-0.69396767247322527	0.03375273822962541	0.15644691064971614
1.7037379778090087	0.70183631648434168	0.30164637897318257	0.13674889192632305
-0.36795289329867606	-0.15325968015767598	-0.02133717890863580	0.07017585040233021
0.26128238288028031	0.23265662112814697	0.20779207320023156	0.18241207299513005
0.03079789423329901	0.03258700434280735	0.03466566943647826	0.03756624088848987
2-norm: 41.795	3.334	0.656	0.517

to the minimum value 0.485. Since we accept rules with 2-norm less than 100 in this experiment we have included this rule as rule C in two different sequences.

In order to create an alternative 17 nodes rule we relax the degree of precision to 15 and use w_0 as a free parameter. A symmetric rule of degree 15 has to satisfy the following 8 equations

$$\sum_{i=1}^{8} w_i = 1/2 - w_0, \text{and} \sum_{i=1}^{8} w_i x_i^{2j} = 1/(2j+1),\ j = 1, \ldots, 7,$$

giving a VanderMonde matrix, V, such that

$$Vw = u - w_0 e,$$

where $x_i = i/8$, $u = (1, 1/3, 1/5, 1/7, 1/9, 1/11, 1/13, 1/15)^T$, $e = (1/2, 0, 0, 0, 0, 0, 0, 0)^T$ and $w = (w_1, \ldots, w_8)^T$. Thus we get $w = V^{-1}u - w_0 V^{-1}e$. We are interested in the degree 15 rule with the minimum 2-norm thus minimizing $w_0^2 + 2\, w^T w$ which turns out to be a quadratic function in w_0 with a unique minimum value. Thus the degree 15 rule with minimum 2-norm is unique and we give the rule in column two in Table 1. This rule has 2-norm ≈ 3.334, thus reduced with a factor less than 1/10 from the degree 17 Newton-Cotes rule. We observe that we have negative weights in the degree 15 rule too.

Reducing the degree to 13 we get two free parameters (say w_0 and w_1) and we follow a similar procedure as above minimizing a function which is quadratic in w_0 and w_1 and construct the normal equations (2×2 system) and find a unique minimal rule, column three in Table 1, with 2-norm ≈ 0.656 (and some negative weights).

Reducing the degree to 11 implies three free parameters (say w_0, w_1 and w_2) and we follow a similar procedure, then a rule with positive weights appears and at the same time with minimal 2-norm ≈ 0.517 as we see only slightly smaller that the degree 13 rule's 2-norm, column four in Table 1

An interesting theoretical question is if there exist symmetric rules with non-negative weights of higher degree than 11 based on our 17 nodes? The answer to this question is probably no: we have investigated the following problem numerically, search for the rule with non-negative weights with *maximum* 2-norm of degree 13. Since the object function is quadratic we know that this maximum has to be attained in one of the extreme points in the set of feasible points (convex): having nine weights and two degrees of freedom implies $9 \times 8/2 = 36$ extreme points to inspect. An extreme point in this setting is a case where two of the nine weights are zero. Checking these 36 extreme-points by running a simple Matlab code shows that in none of these 36 extreme points we find all weights non-negative. This implies that there does not exist interior points with non-negative weights since if such a rule exists then the maximum has to be attained in an extreme point in the set. Thus degree 11 is the maximum degree we may get based on our 17 nodes if we ask for non-negative weights only. Observe also that we may construct rules with positive weights

Table 2: 33 nodes symmetric quadrature rules $(w_0, \ldots, w_{16})$ with minimum 2-norm.

Degree of prec.: 27	25	23	21
17.596473481403099	-2.3679310141365321	0.51795540546616861	-0.04772069442118868
-5.4208814966182699	0.37760363260096110	0.08052708940656544	0.041012950076198569
-14.231233070221511	2.4449165672292395	-0.39853034620283325	0.16606739510630740
15.326402464090806	-0.96619838066865647	0.01948308219652956	0.12326785610056442
3.9483050394538197	-2.0838413254855428	0.54468348497948632	-0.02495999082878566
-19.331291092852420	2.0265820380044810	0.09171154726689796	-0.02893092126444607
13.501113227549911	1.4253416273287707	-0.46429465456793815	0.13487007901085610
7.4467704632794576	-2.9496548828792714	0.13778904251757138	0.17615372270394189
-24.354757093571235	0.82539270581532494	0.63627224435758999	-0.012384096078534824
27.624501654321348	2.8450268150499947	-0.31542179983747938	-0.06496030124326712
-20.246745867038546	-4.3489327423708479	-0.35551576249713175	0.18664930487482380
11.036738075229429	3.7432030265649247	0.98324562571214049	0.15969788707935383
-4.4611013899969840	-2.0184707237853084	-0.76465819010968196	-0.19199392034620951
1.5082956203676327	0.91934415211438517	0.53212453603657755	0.29187122541332239
-0.30306061552847280	-0.20556316027229057	-0.12642677329136230	-0.0636830189089325
0.14401180552105430	0.13404457571286155	0.12431335118942319	0.11482155932993915
0.01469553531243093	0.01517158210924014	0.01571982011056061	0.0163606161854625
2-norm: 79.485	12.306	2.583	0.775

of degree 11 that only use a subset of the 17 points: three degrees of freedom makes it possible to construct symmetric rules using only 11 of the original 17 points with all weights positive. However, in this software project these rules do not appear to be interesting.

3.2 Rules based on 33 nodes.

The Newton-Cotes rule with 33 nodes turns out to have a 2-norm ≈ 450669.534. A rule with such a high 2-norm is not acceptable in this software project. We then reduce the degree of precision to 31, get one the degree of freedom and construct the rule of minimum 2-norm ≈ 11408.245. Reducing the degree of precision to 29 we get two degrees of freedom and the minimum 2-norm ≈ 739.410 still too large. Reducing the degree of precision to 27 we get three degrees of freedom, w_0, w_1 and w_2. We solve the following system with $x_i = i/16$, for $i = 0, 1, \ldots, 16$.

$$\sum_{i=3}^{16} w_i = 1 - w_0/2 - w_1 - w_2, \text{and} \sum_{i=3}^{16} w_i x_i^{2j} = 1/(2j+1) - w_1 x_1^{2j} - w_2 x_2^{2j}, j = 1, \ldots, 13$$

giving a VanderMonde matrix, V, such that

$$Vw = u - w_0 v_0 - w_1 v_1 - w_2 v_2,$$

where u, v_0, v_1, v_2, w are 14 elements column vectors; $u_j = 1/(2j+1)$, for $j = 0(1)13$, $v_0 = (1/2, 0, \ldots, 0)^T, v_1 = (1, x_1^2, \ldots, x_1^{26})^T$, $v_2 = (1, x_2^2, \ldots, x_2^{26})^T$ and $w = (w_3, w_4, \ldots, w_{16})^T$. Thus we get $w = V^{-1}u - w_0 V^{-1}v_0 - w_1 V^{-1}v_1 - w_2 V^{-1}v_2$. Here we find a rule with 2-norm 79.485 which is acceptable in our experiment. We give this rule in column number one in Table 2. Next we construct minimum 2-norm rules of degree of precision 25, 23 and 21 and we give these rules in Table 2 in column 2, 3 and 4 respectively. We note that all these rules have negative weights. Constructing the minimum 2-norm rule of degree 19 we get the norm 0.426 and this rule too has negative weights. The minimum 2-norm rule of degree 17 however has all weights positive and 2-norm 0.369 which is close to the theoretical minimum value $2/\sqrt{33} \approx 0.348$

A natural question to ask is if there exist rules of degree 19 with positive weights? In this case we have seven degrees of freedom out of the seventeen weights. Thus the set of extreme points

Table 3: The maximum degree of precision of symmetric quadrature rules.

Nodes	Romberg	Positive weights	2-norm ≤ 1.25	2-norm < 10	2-norm < 100
5	5	5	5	5	5
9	7	7	9	9	9
17	9	11	13	15	17
33	11	19	21	23	27

consists of 19448 possible vertices. We have checked all these extreme points and this search confirms that for 52 of these there do exist rules with non-negative weights of degree 19 with 2-norms between 0.532 and 0.608. The fact that the maximum 2-norm (0.608) for rules of degree 19 with non-negative weights are less than the minimum 2-norm (0.775 in Table 2) for rules of degree 21 (no restrictions on the weights) implies that no rule of degree 21 can have non-negative weights. Therefore the maximum degree has to be 19 for a rule with non-negative weights. Since seven of the weights may be zero we find rules using only a subset $19 = 33 - 2 \times 7$ of the 33 nodes we are searching among. However, in this software project these rules do not appear to be interesting.

We summarize this section by stating that we have constructed four rules which all have 2-norm less than 100 with decreasing degrees (and 2-norms) from 27(-2) 21. In order to achieve the rule's 2-norm less than 100 we had to reduce the degree of precision from 33 to 27 while searching for non-negative weights we have to reduce the degree of precision to 19.

3.3 Choosing the quadrature rule sequences.

Observe that the sequence of equidistant nodes, $S = \{5, 9, 17, 33\}$, are a subset of the sequence of nodes in the classical Romberg tableau which is based on the composed Trapezoidal method and has a point sequence $2, 3, 5, 9, 17, 33, 65, \ldots$. The Romberg method is using Richardson h^2-extrapolation and the diagonal in the Romberg tableau consists of a set of symmetric quadrature rules with positive weights and degree of precision $1, 3, 5, 7, 9, 11, 13, \ldots$. The Romberg method is a non-adaptive strategy which approximately doubles the number of points used in each step.

We want to experiment with several sequences of quadrature rules based on the S-sequence of nodes. The four diagonals in the Romberg tableau that we associate with these number of nodes we denote as the Romberg sequence. The first rule in the Romberg sequence is identical to rule Q_A, the five point Newton-Cotes rule. The three next rules we denote as Romberg's 9, 17 and 33 point rules. In Table 3 we summarize the results from the previous three sections. In column one we give the S-sequence and then we give the maximum degree of precision for sequences of rules satisfying: positive weights, 2-norm less than 1.25 (we do not want to discard rule Q_B) and so on. Now, observe that the rules are going to operate in pairs in our modified doubly adaptive algorithm. A quadrature rule with degree of precision equal to d applied to a smooth function over an interval of length h has a local error of $O(h^{d+2})$. Thus the difference in order between the rules in such a pair is equal to the difference in the rules' degree of precision. Taking these differences between all three pairs in the Romberg's sequence we get $\{2, 2, 2\}$, while the positive weights sequence of rules have differences $\{2, 4, 8\}$. The three next sequences in the table have differences $\{4, 4, 8\}$, $\{4, 6, 8\}$ and $\{4, 8, 10\}$.

We feel that having differences in the sequence of degrees as small as 2 will effect the adaptive code's ability to switch to a new pair of rules. Therefore we have chosen not to include the Romberg sequence or the positive weights sequence in this experiment. All the three other sequences are included: The code 1 is based on three rules only: rules of degrees 5, 9 and 17, implying a set of three Newton-Cotes rules. The code 2 is based on all four rules in the last column: rules of degrees 5, 9, 17 and 27, here the last rule is a non-interpolatory quadrature rule. The code 3 is based on four rules: rules of degrees 5, 9, 15 and 23 and finally the code 4 is based on four rules: rules of degrees 5, 9, 13 and 21.

We conclude this section by stating that Patterson [19] has suggested to start with a Gauss or Lobatto rule say based on 65 points and then drop every second point to obtain a sequence of 3,

5, 9, 17, 33 points and use this sequence in an automatic integrator. This gives a sequence of rules of order 3, 5, 9, 17 and 33 and it turns out that the rules' weights are all positive. The drawback in an adaptive setting is that when we bisect an interval, then very few of the function values in this sequence are possible to re-use implying a much higher cost per node in the subdivision tree.

We may also start by using a Lobatto rule based on 33 nodes and Patterson's idea to construct a sequence 5, 9, 17 and 33 which will have degrees of precision 5, 9, 17 and 63. We do not know anything about the sign of the weights to the rules in this sequence. It is not likely that the higher degree in the most accurate rule can overcome this extra cost in building the subdivision tree in this case.

4 The null rules, error estimation and phase factors.

4.1 The null rules.

The term *null rule* was first used in 1965 by J.N. Lyness, [14]. We refer the reader interested in a presentation of sequences of null rules and how to construct error estimators based on null rules to [3] or [8]. The following definition of a null rule is useful in this context.

Definition 1 *A rule*

$$N[f] = \sum_{i=-n}^{n} u_i f(x_i) \tag{5}$$

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

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

A null rule based on $2n + 1$ nodes has furthermore degree of precision at most $2n - 1$. 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.

In Table 4 we give four orthogonal null rules associated with 5 nodes in S. We have used Maple and the definition of null rules to construct this set of null rules. These four null rules have degree

Table 4: The four orthogonal null rules based on five nodes.

Nodes:	-1	-1/2	0	1/2	1
N_1 :	1	-4	6	-4	1
N_2 :	-1	2	0	-2	1
N_3 :	2	-1	-2	-1	2
N_4 :	-2	-1	0	1	2

Table 5: The eight orthogonal null rules based on nine nodes.

Nodes:	-1	-3/4	-1/2	-1/4	0	1/4	1/2	3/4	1
N_1 :	1	-8	28	-56	70	-56	28	-8	1
N_2 :	-1	6	-14	14	0	-14	14	-6	1
N_3 :	4	-17	22	1	-20	1	22	-17	4
N_4 :	-4	11	-4	-9	0	9	4	-11	4
N_5 :	14	-21	-11	9	18	9	-11	-21	14
N_6 :	-14	7	13	9	0	-9	-13	-7	14
N_7 :	28	7	-8	-17	-20	-17	-8	7	28
N_8 :	-4	-3	-2	-1	0	1	2	3	4

of precision 3, 2, 1, 0 respectively. N_1 and N_3 are symmetric while N_2 and N_4 are anti-symmetric. Since the nodes are all rational numbers the null rules have to have rational weights and since a null rule may be scaled with any number it is possible to represent these null rules using integers only. In Table 5 we give eight orthogonal null rules associated with the 9 nodes in S which have the degree of precision $7, 6, \ldots, 1, 0$ respectively. In the Appendix A we present the 15 of the 16 null rules associated with 17 nodes and 15 of the totally 32 null rules associated with the 33 nodes.

4.2 Error estimation and phase factors.

In our codes all these null rules are initially normalized through the same 2-norm, a natural choice is $||N_j||_2^2 = ||Q||_2^2 = \sum_{i=-n}^{n} w_i^2$, where Q is the actual quadrature rule used by the code. This relation holds for any interval of size h implying that the weights of the rule and the null rules are scaled with the same factor $h/2$ (initially all rules and null rules are given over an interval of length 2).

Through this choice and scaling one can show [3] that when the null rules are applied to a smooth function f over a small enough interval of length h then $N_j[f] = O(h^{2n+1-j})$ for $j = 1, 2, \ldots, 2n$. Thus we expect for such a function, at least asymptotically for small h, that the absolute values of this sequence of null rules applied to this function form an increasing sequence.

This observation may be turned around: we may test if the expected order is true for a given function and interval and take different actions based on this test. This is the basis for the error estimation algorithms given in [3] and [8].

Many codes use only the absolute value of a single null rule applied to f, equivalent to the scaled difference between to different quadrature rules, to estimate the local error. Lyness and Kaganove [15] have pointed out that phase factor effects may ruin $|N[f]|$ as an estimate of the true error. The classical Lyness-Kaganove example, [15] page 72, is as follows

$$I = \int_1^2 f(x;\lambda)\, dx, \quad f(x;\lambda) = 0.1/((x-\lambda)^2 + 0.01)\,, 1 \le \lambda \le 2, \tag{7}$$

in which each integrand function has a peak height 10 and a half-width 0.1. The value of these integrals lie between 1.46 and 2.46. As Lyness and Kaganove state the problems are not unlike each other; none are trivial and none are pathological.

We will use the three Newton-Cotes rules with nodes, 5, 9 and 17, on (7) and plot the absolute values of the true error, $|Q-I|$, for 1000 sample values of the parameter λ in order to visualize these 'phase factor' effects. In addition we want to illustrate the behavior of the null rules, combinations of null rules and the error estimators, for all three local rules for the same sample values of the parameter λ. For the three different set of nodes we define

$$\begin{aligned} E_j^{(5)} &= |N_j[f]|,\ j = 1(1)4, \\ E_j^{(9)} &= (N_{2j-1}^2[f]^2 + N_{2j}^2[f])^{1/2},\ j = 1(1)4, \\ E_j^{(17)} &= (N_{3j-2}^2[f] + N_{3j-1}^2[f] + N_{3j}^2[f])^{1/2},\ j = 1(1)5. \end{aligned}$$

Observe that we have three different E-sequences which for a smooth function and a small interval will have orders h^4, h^3, h^2, h; h^7, h^5, h^3, h and finally $h^{14}, h^{11}, h^8, h^5, h^2$. Therefore the ratio between successive E-values in a sequence will be $O(h)$ for the 5 nodes, $O(h^2)$ for the 9 nodes and finally $O(h^3)$ for the 17 nodes. In [8] The local error estimating algorithm A, which assumes $K+1$ null rules, appears as

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

When applying the NC 5 node rule we use this algorithm with $K = 3$, $r_{critical} = 1/2$, $C = 32$ and $\alpha = 4$. Observe that the algorithm checks the error versus the *local noise level*. This noise level is defined as follows

$$noise = 50\epsilon\ h/2\ \sum_{i=-n}^{n} |w_i|\ |f_i|, \tag{8}$$

where ϵ is the machine epsilon and the sum represents a factor which appears if we assume that all errors in the function evaluations are relative errors and with an upper bound equal to 50ϵ. To compute the *global noise level* we simply sum all these local noise levels in a globally adaptive code. (8) is easily computed simultaneously with the quadrature estimate as just another inner product. An interesting observation is that if all rules used by the code have positive weights only then we may interpret the final global noise level as an approximation to what we may naturally phrase as *the problem's noise level* (recalling that the rounding errors are in focus)

$$50\epsilon \int_a^b |f(x)|\ dx.$$

When applying the NC 9 node rule we use instead algorithm B from [8].

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** $r_{critical} \leq r$ **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** $\hat{E} = 0$

We use this algorithm with $K = 3$, $r_{critical} = 1/4$, $C = 32$ and $\alpha = 2$. We also give a new local error estimating algorithm which we use when applying the 17 and the 33 node rules. This algorithm is a modification of Algorithm B's *Compute* part only, therefore we give only the initial part of algorithm C

The local error estimating algorithm C

Compute: $e_j = N_j[f],\ j = 1, 2, \ldots, 3K+3;$
$E_j = \sqrt{e_{3j-2}^2 + e_{3j-1}^2 + e_{3j}^2},\ j = 1, 2, \ldots, K+1;$
$r_j = E_j / E_{j+1},\ j = 1, 2, \ldots, K;$
$r = \max_{j=1,2,\ldots,K}\ r_j;$
Non-asymptotic: ...

When applying the NC 17 node rule we use this algorithm with $K = 4$, $r_{critical} = 1/8$, $C = 32$ and $\alpha = 5/3$. In our experiments we also use rules based on the 17 nodes with reduced order of precision (and smaller 2-norm) and in those cases we have to adjust α accordingly. Finally, when applying rules with 33 nodes we also use Algorithm C with $K = 4$, $r_{critical} = 1/8$, $C = 32$ and α chosen to fit the order of the actual rule used, the order of r and the order of E_1.

First we apply all three rules of degrees 5, 9 and 17 respectively once on the classical Lyness-Kaganove example (7) for each value of the parameter $\lambda \in [1, 2]$. Then we apply the three local error estimates for each rule and plot in the figures 1, 2 and 3 part (a) the E- sequences used in these three local error estimators and the absolute values of the true error versus the parameter $\lambda \in [1, 2]$. In part (b) of these figures we plot the absolute values of the true errors and the estimated errors for each of the three rules.

Comments: all six plots demonstrate that both the problem, the quadrature rules and the E-sequences are symmetric around the parameter value $\lambda = 3/2$. Degree 5: $N_1[f]$ is zero for four different values of λ. In Figure 1 (a) these zeros are indicated by four dips in the plotted $\log_{10} |N_1[f]|$. This implies that $|N_1[f]|$, or a scaled version, is not well suited as an upper bound on the true error everywhere in the interval $\lambda \in [1, 2]$ (Lyness and Kaganove, 'phase factors'). We observe that the sequence $|N_1[f]|, |N_2[f]|, |N_3[f]|$ and $|N_4[f]|$ give a confusing impression in the interval and the tested order $|N_1[f]| < |N_2[f]| < |N_3[f]| < |N_4[f]|$ is true in a small part of the interval (reflected in the estimated error for the degree 5 rule) when λ is near the end points.

Note that for the degree 9 rule then E_1, E_2, E_3 and E_4 all have dips, however these dips do not appear to be associated with zeros and the tested order $E_1 < E_2 < E_3 < E_4$ appears to be true in a larger part of the interval (reflected in the estimated error for the degree 9 rule).

Finally for the degree 17 rule the order $E_1 < E_2 < E_3 < E_4 < E_5$ appears to be true almost everywhere in the interval (No 'Table mountains' in the estimated error plot for the degree 17 rule, as observed in the degree 9 plot).

We are going to use the three error estimating algorithms in adaptive codes. Therefore it is interesting to see how the three algorithms handle a bisection of the interval. We focus on the left part of the integral of length h

$$I_h = \int_1^{1+h} f(x; \lambda)\ dx, \text{ with } \lambda \in [1, 2] \tag{9}$$

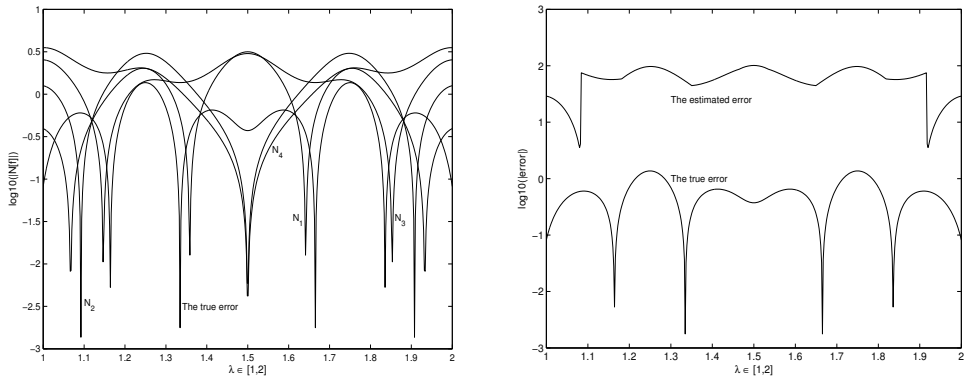
Transforming this integral to the original interval length, $x = 1 + ht$ and $\lambda = 1 + h\gamma$, we find that (9) has a peak of height $H = 10h$ and half-width $1/H$ for all values of h ($h = 1$ gives us Lyness and Kaganove's original problem). Therefore this subproblem becomes easier to handle with smaller values of h. In addition we may have $\lambda > h$ (or $\gamma > 1$) which means that the peak is outside this subinterval making this part of the problem quite easy to integrate for all rules.

In the figures 4, 5 and 6 we give plots for the three rules applied to (9) with $h = 1/2$ and in the figures 7, 8 and 9 we give the plots for the three rules applied to (9) with $h = 1/4$, which is the next size of the intervals in an bisecting strategy.

Observe that for the degree 5 rule the order test $|N_1[f]| < |N_2[f]| < |N_3[f]| < |N_4[f]|$ is satisfied for all values of $\lambda > h$ (both the figures 4 and 7) and in this region of the parameter λ the estimated error bounds the true error nicely and seems to have the same slope. When $\lambda \leq h$ on the other hand the tested order is not true most of the time and therefore the non-asymptotic error estimate is used giving the 'Table mountain' appearance.

The degree 9 rule seems to have the order $E_1 < E_2 < E_3 < E_4$ true almost everywhere in the interval $[-1, 1]$, figures 5 and 8, giving rise to an optimistic error estimate based on E_1. The dips we can observe in the plot of E_1 are visible in the error estimate plots too, however the error estimate appears to be a true upper bound everywhere for this rule. We observe that the error estimate bounds the true error nicely everywhere in the interval. The slopes seems as the same when $\lambda > h$. The peak inside the integration interval effects both the true error and the error estimate indicating that the difficulty is detected.

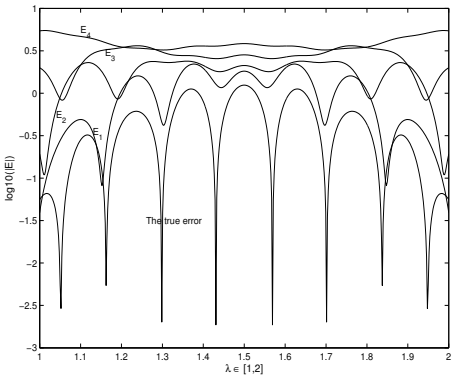
The degree 17 rule has the order $E_1 < E_2 < E_3 < E_4 < E_5$ true everywhere giving rise to an optimistic error estimate based on E_1, figures 6 and 9. Remarks similar to those for the degree 9 rule are valid for this rule too. Note that both the true error and the estimated error becomes close to machine precision in the case when $h = 1/4$ and $\lambda > 1.6$ (Figure 9).



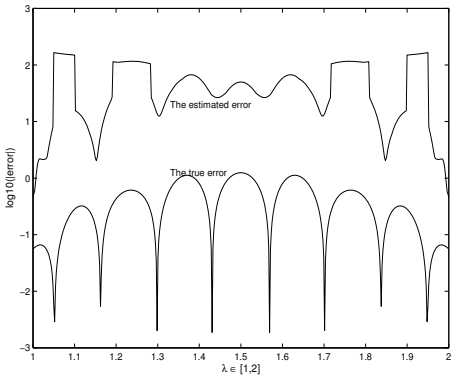
(a) The N-sequence

(b) The true and estimated error

Figure 1: The Degree 5 rule applied to the whole interval.

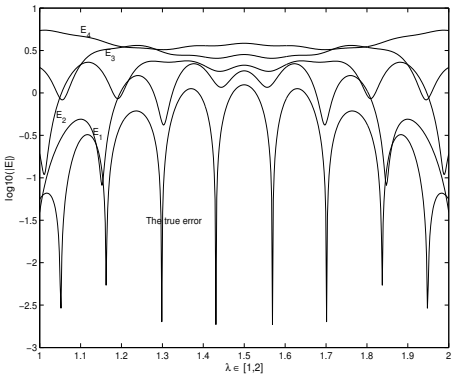


(a) The E-sequence

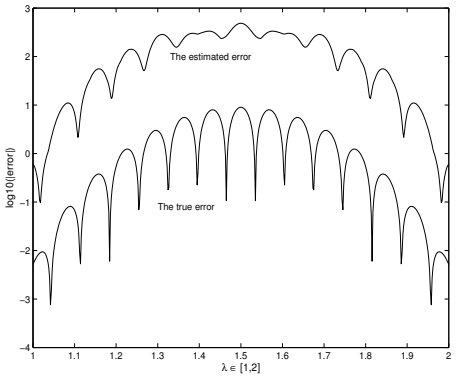


(b) The true and estimated error

Figure 2: The Degree 9 rule applied to the whole interval.

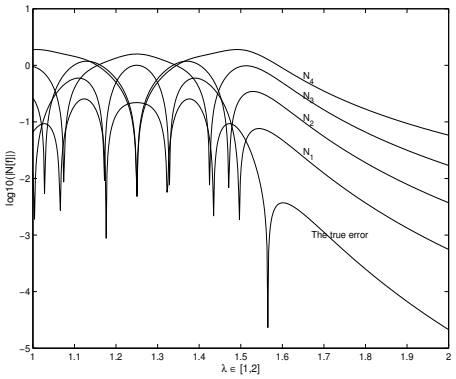


(a) The E-sequence

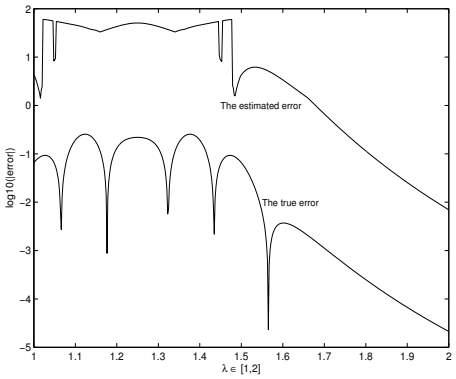


(b) The true and estimated error

Figure 3: The Degree 17 rule applied to the whole interval.

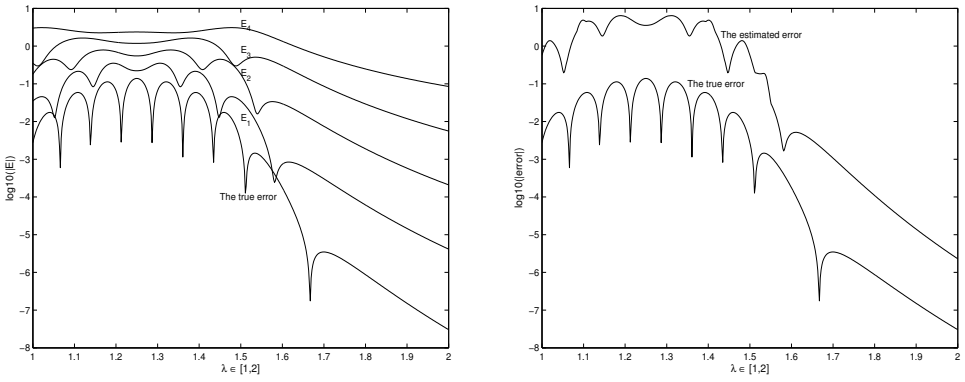


(a) The N-sequence



(b) The true and estimated error

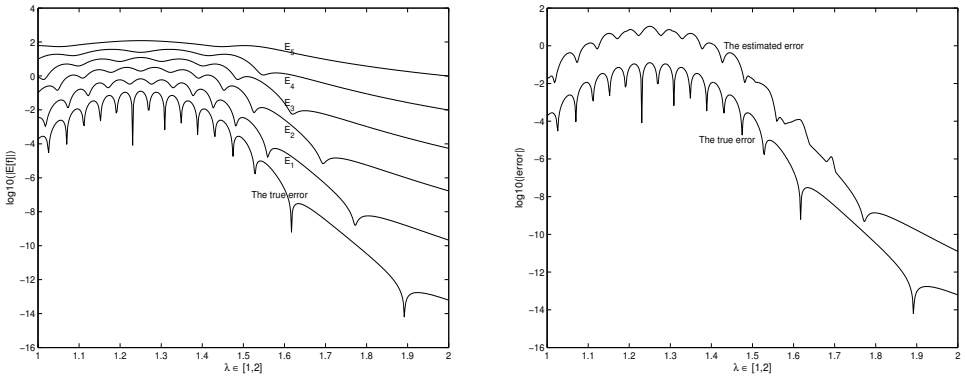
Figure 4: The Degree 5 rule applied to half of the interval.



(a) The E-sequence

(b) The true and estimated error

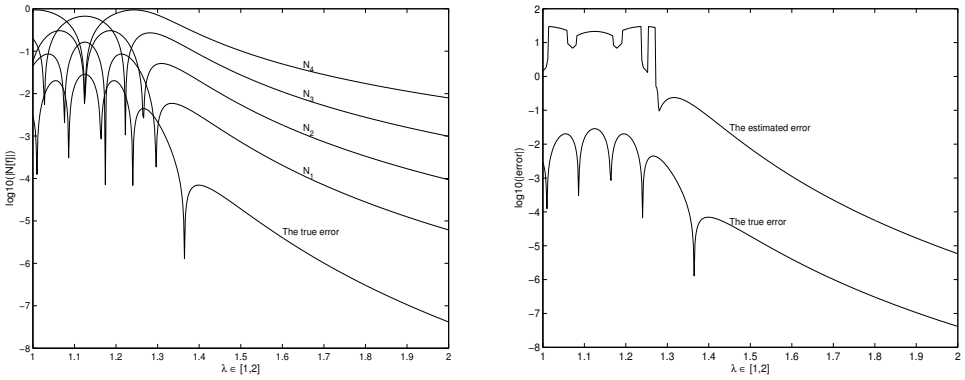
Figure 5: The Degree 9 rule applied to half of the interval.



(a) The E-sequence

(b) The true and estimated error

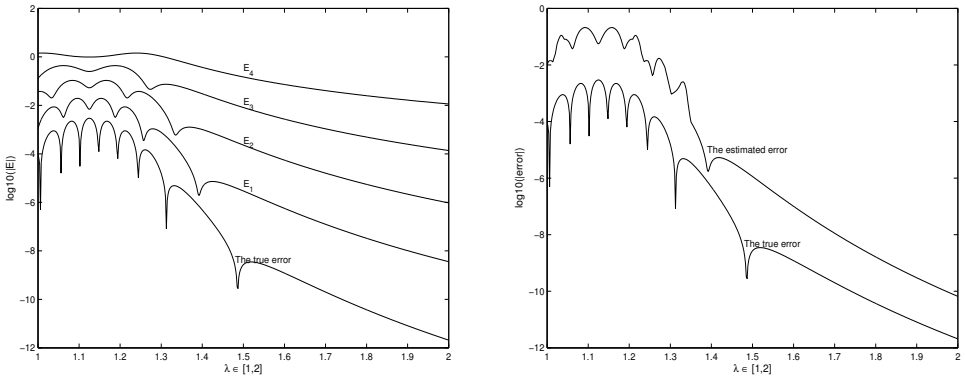
Figure 6: The Degree 17 rule applied to half of the interval.



(a) The N-sequence

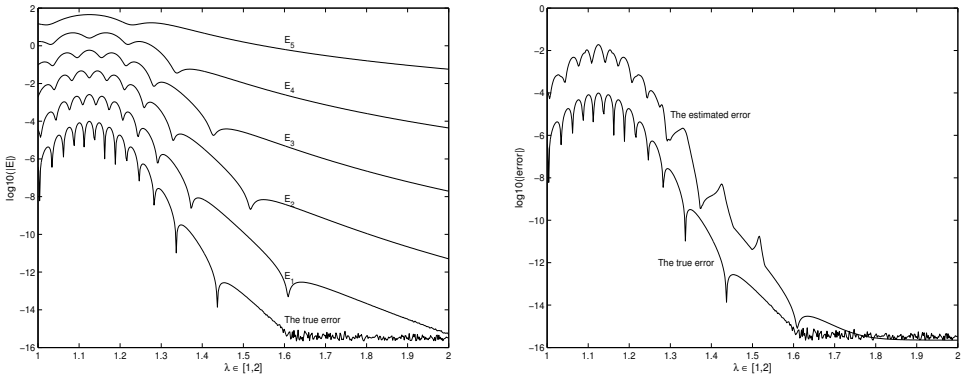
(b) The true and estimated error

Figure 7: The Degree 5 rule applied to a quarter of the interval.



(a) The E-sequence (b) The true and estimated error

Figure 8: The Degree 9 rule applied to a quarter of the interval.



(a) The E-sequence (b) The true and estimated error

Figure 9: The Degree 17 rule applied to a quarter of the interval.

5 Locally versus globally adaptive quadrature algorithms.

In Matlab's `quadl` the Gander and Gautschy code `adaptlob` is implemented with an absolute rather than a relative tolerance request. This decision is wise in my opinion because relative tolerance requests in a *locally* adaptive code is a risky task since you have to 'pre-compute' an estimate, *est*, to the integral to be used to decide whether you are finished or not. Therefore, if you underestimate *est* then $tol * |est|$ becomes too small and if you overestimate then this number becomes too large. This under/over-estimate may be several orders of magnitude wrong and thus effect either the efficiency of the code or the reliability of the code. In my opinion an absolute tolerance specification fits best to a locally adaptive code.

Gander and Gautschy claim that one should be allowed to try to achieve machine precision and in a few examples they demonstrate that the code is able to achieve this. The cost of such an option is quite high and no guarantee can be issued (Too small intervals is their stopping criterion if things go wrong). In `quadl` one uses Gander and Gautschy's locally adaptive strategy but chooses to count the number of function evaluations and stop when this number becomes too large. This is not a wise decision in a *locally* adaptive code. When you stop due to the function count then you

have only a good estimate for the integral over a subinterval and therefore may have no significant digits in your approximation to the true value of the integral. The only valuable answer may therefore be a warning.

I have recently modified `coteda` such that this locally adaptive code allows for an absolute tolerance specification. This code has two extra stopping criteria: the code checks for too small intervals and the code 'pre-compute' an estimate of the *global noise level* and uses this estimate to modify the stopping criteria if *tol* is below *noise*. A locally adaptive code cannot udate this global noise estimate and there is always a risk that we may under/over-estimate this value too. However, the code is much less sensitive to this estimate than to the 'pre-computed' estimate to the integral itself which is needed when a relative tolerance is allowed in a locally adaptive code.

In a globally adaptive code none of these issues cause problems and therefore it is natural to include all three elements; (1) stop when the number of function evaluations becomes too high will give the best possible estimate to the integral using this number of function evaluations, (2) updating *noise* makes it possible to modify the stopping criteria when *tol* is less than *noise*, and (3) stop subdividing an interval when the size of the interval becomes too small (unable to distinguish between the points in the interval).

A drawback in a globally adaptive algorithm is the need for an explicit data structure. It is well known, Malcolm and Simpson [17], that a heap (or rather a partially ordered list) in general is the best way to organize such a data structure since we need to find the interval with the largest error estimate each time we are ready to process a new interval. I have implemented the heap-algorithm tailored to this problem, published by Malcolm and Simpson, in Matlab. Based on timings with `tic` and `toc` it turns out that for problems of the size I have studied this heap algorithm is unable to compete with an unordered list data structure and using Matlab's simple `max` function.

With M intervals the heap approach is of $O(M \log M)$ operations while the simple `max` function needs $O(M^2)$ so this result is therefore surprising. The explanation to this observation is that the `max` function is much faster in Matlab than the indexing operations needed in the partially ordered list. Therefore M has to be very large in order to observe the expected difference in Matlab. In addition, having a sequence of rules prevents to a certain extent the growth in the number of intervals since we often prefer to use a higher degree rule instead. Example: 10^4 function evaluations means 2500 intervals using our five node rule while it means ≈ 312 intervals when we use one of our 33 node rules everywhere.

On the other hand a locally adaptive code makes use of recursive programming in Matlab and we therefore do not have to worry about an explicit data structure (handled by Matlab using a stack) and is therefore much faster in the administration of the computations than a global code.

Another issue that has been discussed in the literature is locally adaptive codes' ability to integrate singular functions. A much used strategy in locally adaptive codes is to test the local error, e_h, as follows

$$e_h \leq tol \frac{h}{b-a} \tag{10}$$

in order to decide if we are finished with the interval of size h. Thus, every interval is allowed an error proportional to the relative length of the interval. The weakness in such an approach becomes clear when we attempt to apply this idea to the integral

$$\int_0^h 1/\sqrt{x}\, dx.$$

The true integral is $2\sqrt{h}$ and any null rule will have a value $\hat{C}\sqrt{h}$ for different values of the constant $\hat{C}$. The maximum of the E-sequences (assuming $r \geq 1$) therefore gives $e_h = O(\sqrt{h})$ independent of which estimates we have used. This implies that (10) cannot be satisfied when $h \to 0$ since we expect $O(\sqrt{h}) < O(h)$. `coteda` handles this problem as follows: instead of introducing $h/(b-a)$ explicitly on the righthand side in (10) we reduce the expected order in the error estimate on the lefthand side by one when we are in the strong asymptotic region. In the non-asymptotic region, we apply

$$C \max_{j=1,2,\ldots,K+1} E_j \leq tol$$

with a guarding constant $C \geq 10$ which means that this region gets a smaller portion of *tol*. Thus we require $O(\sqrt{h}) < tol$ and this test will be passed when $h \to 0$. By proper guarding constants in the asymptotic region as well this implies reliability for these problems too. The globally adaptive codes are much more dynamic in this respect since a small error estimate in one subinterval means that we are allowed a larger error in a different subinterval.

To conclude this section I feel that in general, aiming for reliable codes and including stopping criteria based on function counts still giving a meaningful answer, the globally adaptive strategy is to be preferred in quadrature software in spite of the fact that explicit handling of data structures may slow down these codes compared to locally adaptive codes.

6 An extended globally doubly adaptive algorithm.

A doubly adaptive algorithm is characterized by two types of adaptivity: (1) a subdivision strategy e. g. bisecting the interval and (2) applying a sequence of rules $Q_1, Q_2, \ldots, Q_L$ on a new interval where the code decides how many of the available L rules to use. In `coteda` and `coteglob` $L = 2$ with $Q_1 = Q_A$ and $Q_2 = Q_B$. In this modified algorithm we will extend the number of rules from two and up to four. Thus $Q_3 = Q_C$ using 17 nodes and $Q_4 = Q_D$ using 33 nodes.

To each stored interval we store a flag which inform the codes which rule we have used. In addition we store the left and right end points of the interval, the quadrature approximation, the error estimate, the local noise level and finally all function values used to produce these estimates.

When the codes are based on four rules then flag may take the integer values from 0 to 7 and indicate which rule has been used on the interval according to column two in Table 6. Only two flag values are associated with rule A and D, that is 0 and 7 respectively. While three flag values are associated with the rules B and C.

In addition Table 6, column three, tells which rule(s) to use next on this interval and how the flag parameter will be set for this interval (or possibly the two half intervals). When flag=0 we are going to compute four new function values in order to use rule Q_B and then set flag to 1 or 2 depending on the parameter r, the maximum ratio of two consecutive E-elements in the Local error estimating algorithm B. A similar decision is taken when flag=3, depending on r in algorithm C. In both these cases the strong asymptotic test is positive and we are willing to consider changing the pair from (A,B) to (B,C) in the first case and from (B,C) to (C,D) in the second case. However, at this point we just store this interval with the new value of flag and all other updated values for this interval.

For flag=2 and flag=5 it is unclear what will happen next according to Table 6. In both cases we know that the test on strong asymptotic behavior is satisfied for the nodes used and the remaining question is whether one should apply a lower degree rule, say Q_{i-1}, twice or apply a higher degree rule, Q_{i+1}, once. In order to check this we compute the left error, e_L, and right error, e_R, which a bisection with a lower degree rule will give (no extra function evaluations is needed) and then compare the sum to the error we have stored for this interval, e. If

$$e_L + e_R \geq e \tag{11}$$

we prefer to apply a higher order rule on this interval (more function evaluations needed) and then update the data structure with the new values. Otherwise we prefer bisection using the lower degree rule twice (no extra function values needed). Thus, we simply view (11) as the practical implementation of the test (3). In the case where the test is true, for this interval and for all subintervals, it is a waste of effort to try the lower degree rule and we therefore use a higher degree rule in combination with the rule we already have used.

In our extended globally doubly adaptive algorithm we use the quadrature rule notation $Q_1, Q_2, \ldots$ instead of $Q_A, Q_B, \ldots$. We also assume that we as input have got a collection of $M \geq 1$ intervals and therefore start by applying rule Q_2 to all these M intervals. The advantage with such an option is that a user of a code may have insight in a given quadrature problem (e. g. position of a difficulty) which is useful to the code and should therefore be allowed to specify several subdivision points and one tolerance for the whole problem instead of splitting this into M problems using a code M times.

Table 6: The globally doubly adaptive quadrature algorithm's use of flag.

flag	Rule used	Apply new rule(s) and set flag
0	Q_A	Q_B once, flag $\leftarrow 1 + (r < 1/4)$;
1	Q_B	Q_A twice, flag $\leftarrow 0$;
2	Q_B	Q_A twice, flag $\leftarrow 0$; or apply Q_C once, flag $\leftarrow 4$;
3	Q_B	Q_C once, flag $\leftarrow 4 + (r < 1/8)$;
4	Q_C	Q_B twice, flag $\leftarrow 3$;
5	Q_C	Q_B twice, flag $\leftarrow 3$; or apply Q_D once, flag $\leftarrow 7$;
6	Q_C	Q_D once, flag $\leftarrow 7$;
7	Q_D	Q_C twice, flag $\leftarrow 6$;

Applying Q_2 in the first step is consistent with how `coteda` and `coteglob` are designed. In the presented algorithm we give an absolute error request, `tol`, while in the codes we develop both an absolute and a relative error request are allowed.

Globally adaptive quadrature algorithms all need an interval collection where essential information about the interval is stored. Numbering the intervals with index k we denote $\hat{Q}_k$ and $\hat{E}_k$ the quadrature approximation and error estimate for interval number k of length h_k. On the other hand Q_i is the quadrature rule number i in the sequence of rules.

In the algorithm a logical variable `bisect` appears: we simply assume that the result of consulting Table 6 is put into `bisect`, that is either bisect using a lower degree rule twice or else apply a higher degree rule.

An Extended Globally Doubly Adaptive Quadrature Algorithm

Initialize: Initialize the interval collection for the M given intervals;
Apply rule Q_2 to produce $\hat{Q}_k$, $\hat{E}_k$, $k = 1, 2, \ldots, M$;
Put $\hat{Q} = \sum_{k=1}^{M} \hat{Q}_k$; $\hat{E} = \sum_{k=1}^{M} \hat{E}_k$;
Control: **while** $\hat{E} > tol$ **do**
begin
Pick the interval, h_k, with the largest error estimate
from the collection; Assume that rule Q_i has been applied
to this interval previously; Use Table 6 and the test (11)
to set `bisect`
Process this interval: **if** `bisect` = 'true' **then**
Apply rule Q_{i-1} twice: Compute $\hat{Q}_k^{(1)}$, $\hat{E}_k^{(1)}$, $\hat{Q}_k^{(2)}$, $\hat{E}_k^{(2)}$; Put $m = 2$; Set flags;
else
Apply rule Q_{i+1} once: Compute $\hat{Q}_k^{(1)}$ and $\hat{E}_k^{(1)}$; Put $m = 1$; Set flag;
end
Update: $\hat{Q} = \hat{Q} + \sum_{j=1}^{m} \hat{Q}_k^{(j)} - \hat{Q}_k$;
$\hat{E} = \hat{E} + \sum_{j=1}^{m} \hat{E}_k^{(j)} - \hat{E}_k$;
Let these m intervals replace interval h_k in
the collection and put $M = M + m - 1$;
end

In this algorithm we have suppressed a number of details: a test on too small intervals, the counting of the number of function evaluations, the option to stop when a given maximum number of function evaluations is reached, the computation of the *local noise level* and finally the continuously updating of the *global noise level* including a modification of the stopping criteria when *tol* is below the global *noise* value. The last feature is essential in order to avoid that adaptive codes use a great effort to try to improve estimates that are almost impossible to improve with the available machine precision.

Table 7: The six test families.

1.	$\int_0^1 (\|x-\lambda\|)^{\alpha_1}\, dx$	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.	$\int_0^1 exp(-\alpha_3 \|x-\lambda\|)dx$	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

7 Testing of the six codes.

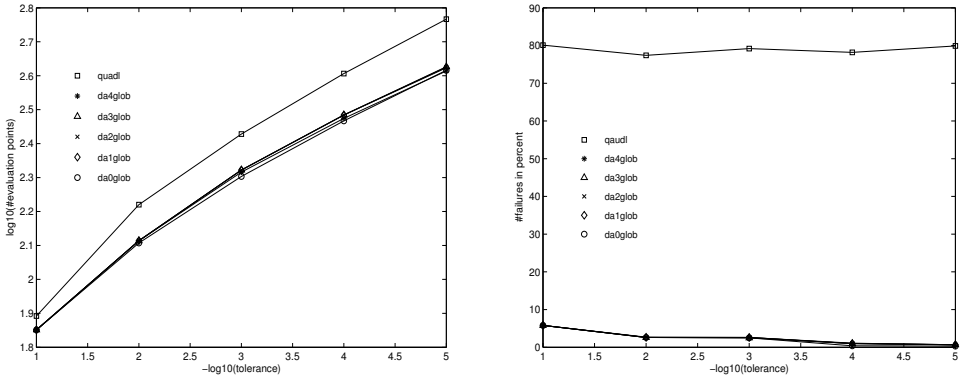
We report tests on the following six Matlab codes

- `da0glob`: Globally doubly adaptive code based on two rules of degrees 5 and 9.
- `da1glob`: Globally doubly adaptive code based on three rules of degrees 5, 9 and 17.
- `da2glob`: Globally doubly adaptive code based on four rules of degrees 5, 9, 17 and 27.
- `da3glob`: Globally doubly adaptive code based on four rules of degrees 5, 9, 15 and 23.
- `da4glob`: Globally doubly adaptive code based on four rules of degrees 5, 9, 13 and 21.
- `quadl`: Locally adaptive code found in Matlab. Based on `adaptlob` in [10].

`da0glob` is just a minor modification of `coteglob` [8]. The five first Matlab codes are available on the Web via T. O. Espelid's homepage: http://www.ii.uib.no/~terje/.

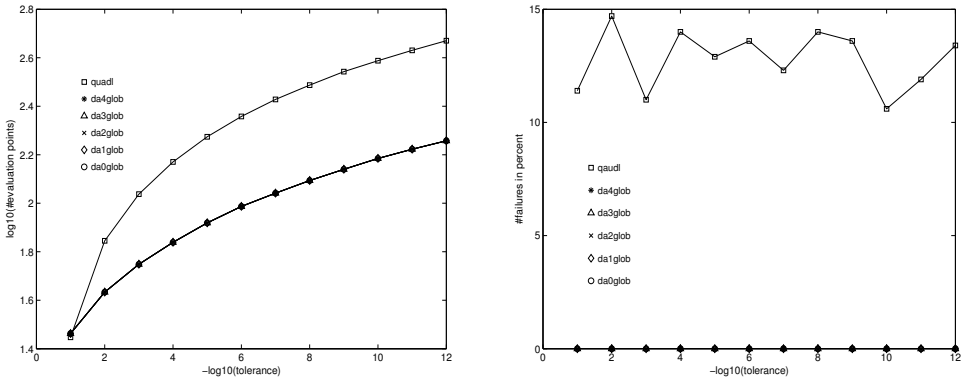
7.1 Results from the Lyness-Kaganove tests.

The test families used in our Lyness-Kaganove testing are given in Table 7, and they are picked from [2], [16] and [21]. 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 absolute 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 most cases all the six codes report that the returned values satisfies the error request. Exceptions: For families 5 and 6 some of the routines give a warning when tol $= 10^{-12}$ indicating that tol is below the global noise level and `quadl` on the other hand gives warnings that the maximum number of function evaluations is reached. For the complete test results we refer to the Appendix B where we list six tables containing all the results from these tests.



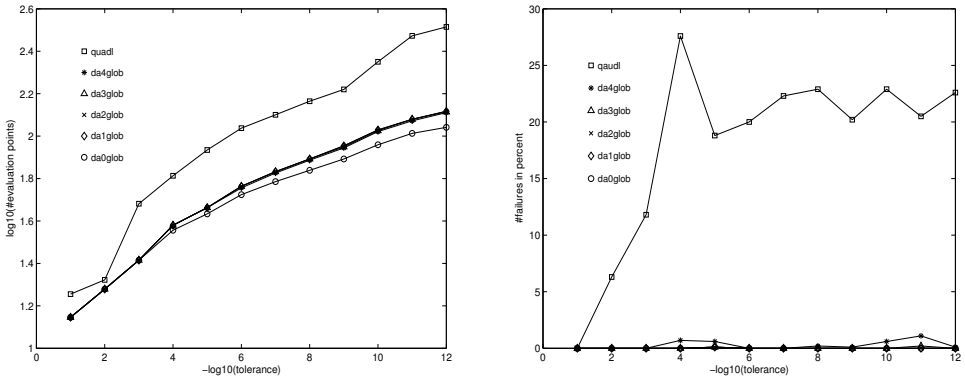
(a) work (b) failures

Figure 10: Test family 1 (Singularity).



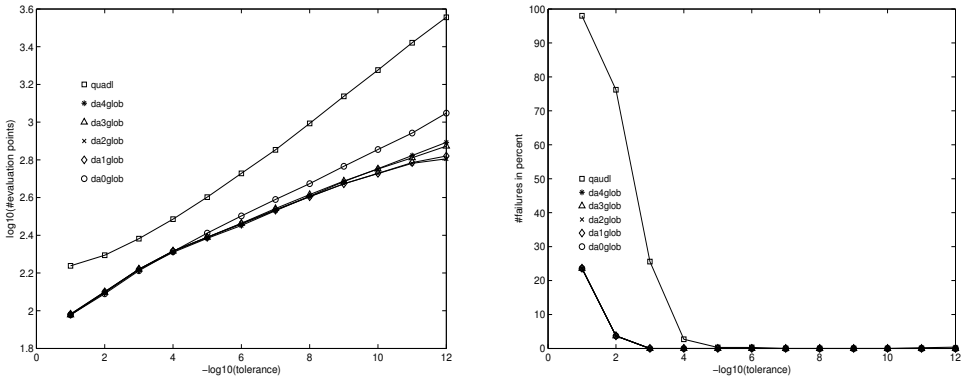
(a) work (b) failures

Figure 11: Test family 2 (Discontinuous).



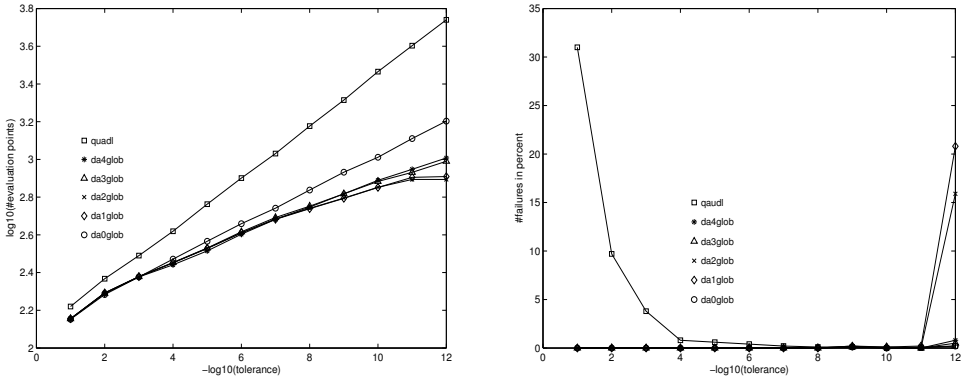
(a) work (b) failures

Figure 12: Test family 3 (C_0 function).



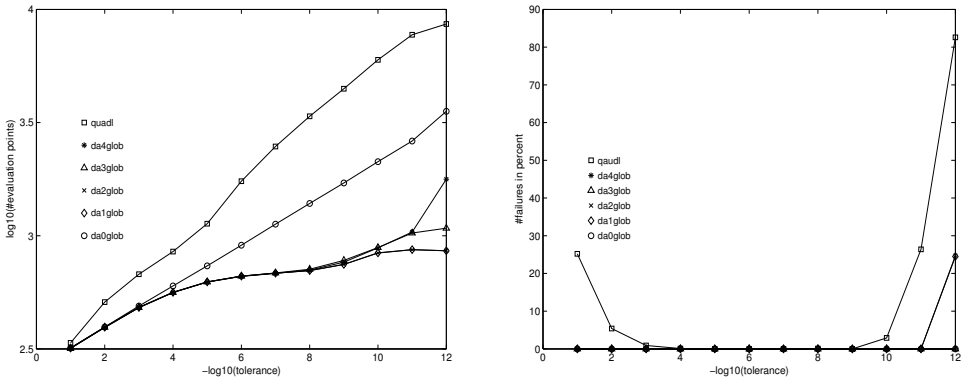
(a) work (b) failures

Figure 13: Test family 4 (One peak).



(a) work (b) failures

Figure 14: Test family 5 (Four peaks).



(a) work (b) failures

Figure 15: Test family 6 (Non-linear oscillatory).

We will give comments to these test results, given in the figures 10-15 and in the tables in Appendix B, for each family separately. We use `da0glob` as the basis code and relate both `quadl` and the four extended codes to this code's performance.

Family 1 (Singularity, Figure 10): `quadl` has ≈80% failures for all tolerances compared to between 2% to 7% for the five other codes. `quadl` uses in addition from 1.10 to 1.41 times the number of function evaluations as does `da0glob`. The main problem for the codes for this family is to detect and to shrink the interval containing the singularity. `da0glob` is marginally better than the other four codes indicating that the extended codes do not make much use of the higher degree rules. Furthermore we achieve, on the average, one extra digit of accuracy with all five codes compared to `quadl` in spite of fewer function evaluations. The cases of 80% failures for `quadl` on the other hand have for most accuracies an average of 0.8/.9 digits wrong.

Family 2 (Discontinuity, Figure 11): `quadl` has between 10% and 15% failures for all tolerances compared to practically no failures for the five other codes. `quadl` uses in addition from 0.97 to 2.59 times the number of function evaluations as does `da0glob`. The main problem for the codes for this family is to detect and to shrink the interval containing the discontinuity. No differences can be observed between the five other codes which all behaves as `da0glob`. Furthermore we achieve one extra digit of accuracy with all five codes compared to `quadl` in spite of much fewer function evaluations for high accuracy requests. The cases of failures for `quadl` on the other hand have for most accuracies an average of 0.2 digits wrong.

Family 3 (C_0-function, Figure 12): `quadl` has between 0% and 27% failures compared to almost no failures for the five other codes. `quadl` uses in addition from 1.11 to 2.97 times the number of function evaluations as does `da0glob`. The main problem for the codes for this family is to detect and to shrink the interval containing the C_0-part of the problem. Some differences can be detected between the five other codes with `da0glob` as the best, while all four remaining codes behave quite similar. The extended codes' attempts to use higher degree rules do not appear useful for this problem. However, the differences are small so the good behavior of `da0glob` is not ruined in the extensions. Furthermore we achieve almost one extra digit of accuracy with all five codes compared to `quadl` in spite of much fewer function evaluations. The cases of failures for `quadl` on the other hand have, for most accuracies, an average of 0.6 digits wrong.

Family 4 (One peak, Figure 13): `quadl` has between 0% and 98% failures compared to between 0 and 24% failures for the five other codes. These failures appear for all codes for low accuracies only: 10^{-1} and 10^{-2} for `da0glob` and all modifications, while for 10^{-1} up to 10^{-5} for `quadl`. `quadl` uses in addition from 1.48 to 3.22 times the number of function evaluations as does `da0glob`. Some differences can be observed between the five other codes with `da2glob` as the best: `da2glob` uses from 1.02 to 0.57 times the number of function evaluations as does `da0glob`. Furthermore we observe that we achieve almost the same accuracy with all six codes. The cases of failures for all codes have at least one digit wrong on the average.

Family 5 (Four peaks, Figure 14): `quadl` has between 0% and 31% failures: all severe failures appear initially and are damped with higher accuracy requests. `da0glob`, `da3glob` and `da4glob` all have very few failures. `da1glob` and `da2glob` have very few failures for tolerances larger than 10^{-12}. For tol $=10^{-12}$ both these two codes give warnings about too small tolerance value compared to the global noise level and therefore modify the stopping test. The reason for this is of course that both these codes have rules with 2-norms between 40 and 80 which implies that they will be effected by rounding noise at an earlier stage than the other codes in the test. The true values of these integrals are around 10 which implies a relative error $\approx 10^{-13}$. Therefore it is quite natural, with this warning in mind, that both codes have severe failures: 20.8% and 15.9% respectively with an average number of wrong digits equal to .5 and .4 respectively. `quadl` uses from 1.17 to 3.45 times the number of function evaluations as does `da0glob`. For accuracy requests between 1 to 11 decimals `da2glob` is the best code: it uses in addition from 1 to 0.61 times the number of function evaluations as does `da0glob`. Including all twelve accuracies `da3glob` is the best code using from 1 to 0.61 times the number of function evaluations as does `da0glob`.

Family 6 (Non-linear oscillatory, Figure 15): `quadl` has between 0% and 82.6% failures: these failures appear both for low accuracies and for high accuracy requests. For high accuracy requests

Table 8: The battery 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.	$\int_{0.01}^1 50(\sin(50\pi x)/(50\pi x))^2dx$.
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.	$\int_0^1 1/(1 + (230x - 30)^2)dx$.

the reason for the failures is that the maximum number of function evaluations is reached and we observe that the average number of wrong digits then is 10.5, 11.8 and 12.3 for the three highest accuracy requests respectively. The other five codes have no failures except that for tol $=10^{-12}$ here too `da1glob` and `da2glob` give warnings about too small tolerance which in turn results in 24.5% and 24.7% of failures with the average number of wrong digits equal to 0.2 in both cases. `quadl` uses in addition from 1.05 to 2.95 times the number of function evaluations as does `da0glob` (in spite of the fact that this code is stopped by the maximum number of function evaluations in many cases). For accuracy requests between 1 to 11 decimals `da1glob` and `da2glob` are the best codes: they use from 1 to 0.33 times the number of function evaluations as does `da0glob`. Including all twelve accuracies `da3glob` is the best using from 1 to 0.30 times the number of function evaluations as does `da0glob`.

7.2 Results from the battery test

We have also tested all six codes discussed in this paper on the 23 test problems, Table 8, used by Gander and Gautschi in their battery test. Gander and Gautschi [9, 10] picked 23 different test problems from two different sources: the 21 first comes from Kahaner [13] and the last two from [11]. We have tested the codes on twelve different *absolute* error tolerances tol $=10^{-1}, 10^{-2}, \ldots, 10^{-12}$ and the results can be found in the Appendix B. In order to summarize the results of this battery test we have constructed Table 9. Here we give, for each of the 23 problems, the following information:

1. An integer gives the number of cases out of the twelve tested accuracies where this code is both

Table 9: Summarizing the results for the battery test.

Problem	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
1	12	12	12	12	12	0
2	11	11	11	11	11	1(5)
3	10	12	12	12	11	0(2)
4	12	12	12	12	12	0
5	5	9	9	8	8	3
6	12	3	3	3	4	0
7	7	12	12	9	9	0(12)
8	7	11	11	11	11	1
9	3	8	7	5	8	1
10	8	12	12	11	11	0
11	12	12	12	12	12	0
12	12	12	12	12	12	0
13	4	11	11	8	8	1(3)
14	6	12	12	11	11	0
15	5	12	12	11	12	0
16	7	11	11	8	9	0
17	0(2)	6(2)	6(2)	2(2)	3(2)	4(3)
18	3	11	11	9	10	1
19	9	12	12	10	10	0(2)
20	6	9	9	9	9	3
21	5(5)	1(9)	1(9)	1(9)	1(9)	2(6)
22	2	10	10	8	9	2
23	6	10	10	8	9	1(3)
Sum	164(7)	231(11)	230(11)	203(11)	212(11)	20(36)

successful and uses the minimum number of function evaluations among all the successful codes in the test.

2. A number in paranthesis gives the number of cases out of the twelve tested accuracies where the accuracy request is not met for this code and problem.

3. Finally we sum over all 23 problems both the number of cases where a code is the best and the number of failures out of the $12 * 23 = 276$ cases totally.

According to Table 9 `quadl` is reasonably reliable with 36 failures out of the 276 cases, but only in 20 of the 276 cases this code is the best code in the test. In one of the failure cases `quadl` gives a warning that the maximum number of function evaluations has been reached (Problem 13, tol=10^{-12}) and the result has ten digits less accuracy than requested. This illustrates the problem with maximum function evaluations as a stopping criterion in a locally adaptive code.

`da0glob` has the fewest failures of all codes in the test and is the best code in 164 out of the 276 cases. The four new codes representing extensions of `da0glob` with either three (`da1glob`) or four rules all have 11 failures out of which 9 come from Problem 21. All four codes do well being best in more than 200 out of the 276 cases.

It is not possible to detect any difficulties for `da1glob` and `da2glob` due to the much larger 2-norms associated with the rules in the two codes based on these 276 tests. Furthermore, in spite of using only three rules in the sequence `da1glob` is the best code overall: 231 out of the 276 cases. Actually it is hard to see the need for more than three rules in the sequence based on this battery test.

Problem 21 deserves some attention. This problem is a three peak problem with the steepest peak located in $x = 0.6$ with height close to 1 and half-width $\approx 1.6 \times 10^{-4}$. All codes have trouble getting this integral correct due to the narrow half-width. In order to demonstrate how a user's

Table 10: Test problem 21, The number of function evaluations needed to achieve tol.

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	50	50	50	50	50	18
10^{-2}	114	114	114	114	114	48
10^{-3}	138	138	138	138	138	-78
10^{-4}	170	170	170	170	170	-168
10^{-5}	222	222	222	222	222	-228
10^{-6}	262	270	270	270	270	-228
10^{-7}	310	306	306	306	298	-348
10^{-8}	370	346	346	346	346	-438
10^{-9}	466	426	426	426	426	888
10^{-10}	590	506	514	530	530	1248
10^{-11}	698	586	594	602	602	1608
10^{-12}	870	642	658	666	666	2268

knowledge could improve this situation we have repeated the test on this problem using for five of the codes the option to give a division point in the interval: $[0, 0.6, 1]$ and we now get the results in Table 10

Observe that no failures are now observed for the five first codes (allowing an initial subdivision point in 0.6) in Table 10 while `quadl` has six failures (indicated by minus signs) for this problem. One may wonder why these codes do not demonstrate similar difficulties with handling Family 4, the one peak family. These problems have a peak height 10^4 with a half-width 10^{-4} and are therefore seemingly more difficult than Problem 21. However this is not true, Problem 21 has a much steeper peak located in 0.6 and even when we remove the two other peaks from this problem the peak is not detected (the codes will use the minimum number of function evaluations when we give an absolute error request).

An observation of a different kind is that when integrating Family 4 specifying a *relative* error request, then all five codes allowing such an error request will have no failures initially contrary to what we observe with an *absolute* error request with the same codes. The reason is simply that one typically will underestimate the true integral initially for these problems and therefore use a smaller number in the stopping criterion, this happens only initially in a globally adaptive code, implying a more reliable code. Therefore, sometimes mere luck produces good results!

8 Conclusions.

We conclude this testing by stating that which code to choose depends on the problem type at hand. The best overall code seems to be `da2glob` except for the weakness that when the accuracy request is high it may suffer from the rules' large 2-norms. Now, `da0glob` and all four modifications are monitoring the global noise level and will change the stopping criteria when tol is below the noise level. The point is that this happens at an earlier stage for `da1glob` and `da2glob` than for the other three doubly adaptive codes.

Furthermore the code `da3glob` is quite good and in the tests a similar weakness is not observed for this code. This code therefore seems well fitted for high accuracy requests too, but it is in general less efficient than `da2glob` for moderate accuracy requests. `da0glob` is still a very good code with a strength in detecting and isolating non-smooth problem areas. `quadl` seems to be too unreliable in general and is in addition much less efficient than all the other five codes both in the Lyness-Kaganove test and in the battery test.

I will conclude by stating that the extensions of the sequence of rules from two to three/four in a doubly adaptive algorithm have been quite successful. We have implemented four extensions which are all able to retain the good performance of `coteglob` for non-smooth functions and low accuracy requests and at the same time improve the performance of `coteglob` for smooth

functions and high accuracy requests. The tests clearly document that users of Matlab will benefit from replacing `quad` and `quadl` by say a triplet of codes: `da0glob`, `da2glob` and `da3glob`.

Appendix A: The 17 and 33 node null rules.

We give first the 15 null rules we are using in all codes based on the 17 nodes in Table 11. We specify the weights for the non-negative nodes in the interval $[-1, 1]$ only. Recall that all odd numbered null rules are symmetric and all even numbered null rules are anti-symmetric. These null rules have polynomial degrees $15, 14, \ldots, 1$ respectively.

Table 11: The fifteen orthogonal null rules based on the seventeen nodes

Nodes:	0	1/8	2/8	3/8	4/8	5/8	6/8	7/8	1
N_1:	12870	-11440	8008	-4368	1820	-560	120	-16	1
N_2:	0	-1430	2002	-1638	910	-350	90	-14	1
N_3:	-3432	1573	2002	-3913	3276	-1659	526	-97	8
N_4:	0	1001	-572	-663	1248	-915	372	-83	8
N_5:	924	-77	-935	365	846	-1137	631	-175	20
N_6:	0	-77	-11	81	4	-95	81	-29	4
N_7:	-504	-119	464	309	-428	-373	672	-329	56
N_8:	0	35	26	-19	-40	5	50	-37	8
N_9:	70	35	-37	-73	-25	65	65	-91	26
N_{10}:	0	-175	-215	-75	149	247	39	-325	130
N_{11}:	-480	-340	8	372	512	260	-312	-676	416
N_{12}:	0	55	88	83	36	-39	-104	-91	104
N_{13}:	36	31	17	-3	-24	-39	-39	-13	52
N_{14}:	0	-7	-13	-17	-18	-15	-7	7	28
N_{15}:	-24	-23	-20	-15	-8	1	12	25	40

In the tables 12 and 13 we give the 15 first orthogonal null rules based on 33 nodes. These null rules have polynomial degrees $31, 30, \ldots, 17$ respectively. We specify the weights $w_0, w_1, \ldots, w_{16}$ row wise in the two tables and have to use symmetry/anti-symmetry to construct the null rules. By row wise specification we get: in column two in the two tables w_0, w_5, w_{10} and w_{15} for each null rule, in column three w_1, w_6, w_{11} and w_{16} and so on.

Table 12: The eight first orthogonal null rules based on the thirty-three nodes.

N_1:	601080390	-565722720	471435600	-347373600	225792840
	-129024480	64512240	-28048800	10518300	-3365856
	906192	-201376	35960	-4960	496
	-32	1			
N_2:	0	35357670	-58929450	65132550	-56448210
	40320150	-24192090	12271350	-5259150	1893294
	-566370	138446	-26970	4030	-434
	30	-1			
N_3:	-155117520	110065005	-1900950	-108904425	171165540
	-171555735	130845390	-80046525	40037400	-16445871
	5521194	-1495501	319580	-51955	6046
	-449	16			
N_4:	0	-25662825	31175580	-13096545	-14567280
	34207095	-37978980	29728335	-17915040	8560539
	-3266676	989219	-233392	41483	-5236
	419	-16			
N_5:	55842307200	-27510548400	-28745405520	57750300720	-34116569760
	-18968679600	60452130960	-69405669840	53233727040	-30444656112
	13431080880	-4597644688	1206883488	-235835888	32422928
	-2804880	115072			
N_6:	0	30972375	-24548475	-11134875	34965060
	-22366695	-12194715	38591475	-42400800	30402567
	-15881319	6233557	-1837580	397345	-59783
	5611	-248			
N_7:	-20801200	6041525	17375120	-16975495	-6284980
	22253075	-12581920	-10832185	26235560	-25818091
	16564160	-7599239	2545012	-613637	101584
	-10385	496			
N_8:	0	-11399210550	4559684220	9616476870	-9251227440
	-4855613490	12763574460	-4768946910	-9086837760	15782062842
	-13253985108	7304839542	-2820989808	765452142	-140288148
	15690402	-812448			

Table 13: The seven next orthogonal null rules based on the thirty-three nodes

N_9:	61519549000	-6332894750	-60614849750	21012150250	56033272750
	-41246377250	-40282102250	65816227750	-4031553500	-67372496750
	85129203250	-58835071750	26728003250	-8254509250	1686980750
	-207444250	11687000			
N_{10}:	0	19541503800	-723759400	-19791826600	2719644200
	19988449000	-7622646200	-18379969400	17136506400	8197542600
	-26488371000	24800175400	-13635198200	4862018200	-1117503400
	151931000	-9349600			
N_{11}:	-3796635024	-265206123	3789397430	696569835	-3823262768
	-809713229	4021399902	257491325	-4353571560	1681016571
	3638836578	-5527386059	3843537360	-1612502515	421115786
	-63705837	4300816			
N_{12}:	0	-5716426716	-1693756064	5331192724	3045136336
	-4896243220	-3818405536	4985681756	3696585024	-6234750852
	-1321631520	7967143052	-7575989872	3838395572	-1153493792
	195604420	-14556608			
N_{13}:	44710952	10191467	-40437111	-27978951	29963472
	39806217	-19029483	-46185381	14703524	49586141
	-26510869	-43270689	67147872	-42709953	15008631
	-2877699	237336			
N_{14}:	0	5897900580	3538740348	-3911239332	-5816714904
	988554996	6522757956	1568971404	-6466121376	-2858390964
	7074216996	1639230516	-9099165240	7748312220	-3252257316
	712364004	-65504736			
N_{15}:	-329072606720	-122192401760	240997291392	299794823328	-34396276992
	-332515956960	-173035387008	252062005536	305613831424	-159149848352
	-370817363840	174219985632	379311525120	-491479216800	253789295232
	-64303815840	6637813248			

Appendix B: the numerical experiments.

We report tests on the six Matlab routines: `da0glob`, `da1glob`, `da2glob`, `da3glob`, `da4glob` and `quadl`.

The Lyness-Kaganove test.

The testing technique we have used in this part is due to Lyness and Kaganove [16]. 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 in Table 7, and they are picked from [2] ,[16] and [21]. These test families have furthermore been used by Berntsen and Espelid in [3].

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',...). All six codes are designed to accept a maximum number of function values and this value is set in all experiments in this paper equal to 10 000. We have tested the codes for absolute 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. The experiments have been run on a Dell Optiplex GX260 computer.

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

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

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

tol	da0glob				da1glob				da2glob			
10^{-1}	71	2.4	5.8	0.3	71	2.4	5.8	0.3	71	2.4	5.8	0.3
10^{-2}	128	3.7	2.6	0.3	130	3.7	2.6	0.3	130	3.7	2.6	0.3
10^{-3}	201	4.8	2.5	0.2	210	4.8	2.6	0.2	210	4.8	2.6	0.2
10^{-4}	293	5.7	0.4	0.3	305	5.7	1.0	0.3	305	5.7	1.0	0.3
10^{-5}	413	6.6	0.3	0.1	420	6.7	0.7	0.1	422	6.7	0.7	0.1
tol	da3glob				da4glob				quadl			
10^{-1}	71	2.4	5.8	0.3	71	2.4	5.8	0.3	78	1.4	80.1	0.5
10^{-2}	130	3.7	2.6	0.3	130	3.7	2.7	0.3	166	2.4	77.4	0.8
10^{-3}	120	4.8	2.5	0.2	207	4.7	2.6	0.2	268	3.5	79.2	0.9
10^{-4}	305	5.7	1.0	0.3	298	5.5	1.1	0.3	404	4.4	78.2	0.9
10^{-5}	423	6.7	0.7	0.1	411	6.4	0.6	0.1	584	5.5	79.9	0.9

Test Family 1: Singularity.

tol	da0glob		da1glob		da2glob	
10^{-1}	29	2.9	29	2.9	29	2.9
10^{-2}	43	3.9	43	3.9	43	3.9
10^{-3}	56	5.0	56	5.0	56	5.0
10^{-4}	69	5.9	69	5.9	69	5.9
10^{-5}	83	6.9	83	6.9	83	6.9
10^{-6}	97	8.0	97	8.0	97	8.0
10^{-7}	110	9.0	110	9.0	110	9.0
10^{-8}	124	9.9	124	9.9	124	9.9
10^{-9}	138	10.9	138	10.9	138	10.9
10^{-10}	153	12.0	153	12.0	153	12.0
10^{-11}	167	13.0	167	13.0	167	13.0
10^{-12}	181	14.0	181	14.0	181	14.0

tol	da3glob		da4glob		qaudl			
10^{-1}	29	2.9	29	2.9	28	1.7	11.4	0.1
10^{-2}	43	3.9	43	3.9	70	2.7	14.7	0.2
10^{-3}	56	5.0	56	5.0	109	3.6	11.0	0.2
10^{-4}	69	5.9	69	5.9	148	4.7	14.0	0.2
10^{-5}	83	6.9	83	6.9	188	5.7	12.9	0.2
10^{-6}	97	8.0	97	8.0	228	6.7	13.6	0.2
10^{-7}	110	9.0	110	9.0	268	7.7	12.3	0.2
10^{-8}	124	9.9	124	9.9	307	8.7	14.0	0.2
10^{-9}	138	10.9	138	10.9	349	9.7	13.6	0.2
10^{-10}	153	12.0	153	12.0	387	10.7	10.6	0.2
10^{-11}	167	13.0	167	13.0	427	11.7	11.9	0.2
10^{-12}	181	14.0	181	14.0	468	12.7	13.4	0.2

Test Family 2: Discontinuous.

tol	da0glob				da1glob				da2glob			
10^{-1}	14	3.2			14	3.2			14	3.2		
10^{-2}	19	4.1			19	4.1			19	4.1		
10^{-3}	26	5.0			26	5.0			26	5.0		
10^{-4}	36	6.1			38	6.1			38	6.1		
10^{-5}	43	7.0	0.1	0.2	46	7.0	0.1	0.2	46	7.0	0.1	0.2
10^{-6}	53	8.1			58	8.1			58	8.1		
10^{-7}	61	9.0			68	9.1			68	9.1		
10^{-8}	69	10.3			78	10.3			78	10.3		
10^{-9}	78	11.0			90	11.0			90	11.0		
10^{-10}	91	12.2			107	12.2			107	12.2		
10^{-11}	103	13.0			120	13.0			120	13.0		
10^{-12}	110	14.0			131	14.0			131	14.0		

tol	da3glob				da4glob				quadl			
10^{-1}	14	3.2			14	3.2			18	2.5		
10^{-2}	19	4.1			19	4.1			21	2.7	6.3	0.1
10^{-3}	26	5.0			26	5.0			48	4.1	11.8	0.4
10^{-4}	38	6.1			38	6.1	0.7	0.1	65	5.0	27.6	0.4
10^{-5}	46	7.0	0.1	0.2	46	7.0	0.6	0.1	86	6.0	18.8	0.5
10^{-6}	58	8.1			57	8.0			109	7.1	20.0	0.6
10^{-7}	68	9.0			67	9.0			126	8.1	22.3	0.6
10^{-8}	78	10.3			77	10.2	0.2	0.3	146	9.1	22.9	0.6
10^{-9}	89	11.0			88	10.9	0.1	0.2	166	10.1	20.2	0.6
10^{-10}	106	12.2			105	12.1	0.6	0.3	224	11.1	22.9	0.7
10^{-11}	120	13.0	0.2	0.5	118	12.9	1.1	0.2	297	12.0	20.5	0.6
10^{-12}	130	14.0			129	13.9	0.1	0.3	327	13.1	22.6	0.6

Test Family 3: C_0 function.

tol	da0glob				da1glob				da2glob			
10^{-1}	95	3.5	23.6	1.5	96	3.5	23.6	1.5	96	3.5	23.6	1.5
10^{-2}	123	4.7	3.7	2.5	126	4.6	3.7	2.5	126	4.6	3.7	2.5
10^{-3}	163	5.5			166	5.5			166	5.5		
10^{-4}	206	6.5			207	6.4			207	6.4		
10^{-5}	258	7.1			245	7.2			245	7.2		
10^{-6}	318	8.3			288	8.2			290	8.2		
10^{-7}	389	9.3			341	9.2			343	9.2		
10^{-8}	472	10.1			402	10.0			403	10.0		
10^{-9}	583	10.8			471	10.9			471	10.9		
10^{-10}	716	12.1			535	12.1			534	12.1		
10^{-11}	875	12.8			610	13.1			605	13.1	0.2	0.0
10^{-12}	1116	13.4			661	13.8			639	13.4	0.4	1.0
tol	da3glob				da4glob				quadl			
10^{-1}	95	3.5	23.6	1.5	95	3.5	23.6	1.5	173	3.2	98.0	1.5
10^{-2}	125	4.6	3.7	2.5	125	4.6	3.7	2.5	197	4.0	76.2	2.5
10^{-3}	166	5.4			165	5.4			241	5.4	25.6	3.4
10^{-4}	207	6.4			204	6.1			306	6.7	2.7	4.1
10^{-5}	245	7.2			242	6.9			400	8.0	0.3	1.1
10^{-6}	291	8.1			283	7.5			535	9.4	0.3	1.2
10^{-7}	348	9.1			339	8.3			713	10.7		
10^{-8}	413	9.9			405	9.3			985	12.1		
10^{-9}	487	10.8			484	10.1			1370	13.1		
10^{-10}	564	12.0			566	11.0			1889	13.3		
10^{-11}	647	12.8			665	12.0			2636	13.3		
10^{-12}	746	13.7			781	12.7			3596	13.5		

Test Family 4: One Peak.

tol	da0glob				da1glob				da2glob			
10^{-1}	142	3.5			143	3.5			143	3.5		
10^{-2}	192	4.5			196	4.5			193	4.5		
10^{-3}	238	5.3			239	5.3			239	5.3		
10^{-4}	296	6.4			283	6.3			283	6.3		
10^{-5}	368	7.2			337	7.3			338	7.3		
10^{-6}	456	7.9			408	8.0			408	8.0		
10^{-7}	551	8.8			481	8.8			484	8.8		
10^{-8}	688	10.5			548	10.3			550	10.3		
10^{-9}	855	10.8	0.1	0.4	621	10.8	0.2	0.4	624	11.0	0.2	0.4
10^{-10}	1027	11.5			710	11.6	0.1	0.1	711	12.0		
10^{-11}	1290	13.1	0.1	0.5	803	12.7	0.2	0.3	783	12.8		
10^{-12}	1596	13.5	0.2	0.2	811	13.2	20.8	0.5	783	13.0	15.9	0.4
tol	da3glob				da4glob				quadl			
10^{-1}	143	3.5			143	3.5			166	2.6	31.0	1.0
10^{-2}	195	4.5			195	4.5			233	4.1	9.7	1.1
10^{-3}	239	5.3			238	5.1			309	5.5	3.8	0.9
10^{-4}	284	6.2			276	5.9			416	7.1	0.8	1.0
10^{-5}	339	7.2			327	6.7			579	8.3	0.6	0.5
10^{-6}	413	8.0			401	7.7			796	9.8	0.4	0.2
10^{-7}	492	8.8			481	8.5			1074	11.1	0.2	0.8
10^{-8}	565	10.1			559	9.5			1502	12.4	0.1	0.5
10^{-9}	657	10.8	0.1	0.4	657	10.4	0.1	0.4	2063	14.0	0.1	0.5
10^{-10}	764	11.7	0.1	0.1	775	11.2			2921	14.7		
10^{-11}	854	12.6			886	12.3			4010	15.0		
10^{-12}	977	13.6	0.5	0.2	1018	13.0	0.8	0.2	5499	15.0	0.2	0.4

Test Family 5: Four Peaks.

tol	da0glob		da1glob				da2glob			
10^{-1}	319	4.7	319	4.7			319	4.7		
10^{-2}	395	5.6	394	5.6			394	5.6		
10^{-3}	489	6.6	482	6.5			482	6.5		
10^{-4}	600	7.7	562	7.6			562	7.6		
10^{-5}	736	8.3	625	7.8			625	7.8		
10^{-6}	908	9.4	663	8.6			663	8.6		
10^{-7}	1125	10.2	683	10.9			683	10.9		
10^{-8}	1389	11.0	701	12.1			701	12.1		
10^{-9}	1711	12.3	747	12.3			747	12.3		
10^{-10}	2125	13.0	840	12.3			839	12.3		
10^{-11}	2621	13.8	868	12.3			867	12.3		
10^{-12}	3546	14.2	858	12.5	24.5	0.2	858	12.5	24.7	0.2
tol	da3glob		da4glob				quadl			
10^{-1}	318	4.7	318	4.7			336	4.0	25.2	1.9
10^{-2}	394	5.5	393	5.4			509	5.4	5.4	3.1
10^{-3}	482	6.5	481	6.5			676	6.8	0.9	3.4
10^{-4}	562	7.6	561	7.5			851	8.4	0.1	5.4
10^{-5}	625	7.8	624	7.8			1130	9.7		
10^{-6}	663	8.6	661	8.4			1741	11.0		
10^{-7}	685	9.9	682	8.9			2479	12.4		
10^{-8}	711	11.1	705	9.4			3371	13.4	0.1	0.1
10^{-9}	778	12.2	764	10.4			4462	14.0		
10^{-10}	887	13.0	884	11.5			5988	14.1	2.9	10.5
10^{-11}	1028	13.4	1038	12.6			7729	14.2	26.4	11.8
10^{-12}	1080	13.5	1777	13.8			8623	14.4	82.6	12.3

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 [13] 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 absolute 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	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	9	9	9	9	9	18
10^{-5}	9	9	9	9	9	18
10^{-6}	9	9	9	9	9	18
10^{-7}	9	9	9	9	9	18
10^{-8}	9	9	9	9	9	18
10^{-9}	9	9	9	9	9	18
10^{-10}	17	17	17	17	17	18
10^{-11}	17	17	17	17	17	18
10^{-12}	17	17	17	17	17	18

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	29	29	29	29	29	18
10^{-2}	45	45	45	45	45	-48
10^{-3}	61	61	61	61	61	-78
10^{-4}	69	69	69	69	69	138
10^{-5}	85	85	85	85	85	168
10^{-6}	101	101	101	101	101	198
10^{-7}	109	109	109	109	109	-228
10^{-8}	125	125	125	125	125	258
10^{-9}	141	141	141	141	141	318
10^{-10}	149	149	149	149	149	-348
10^{-11}	165	165	165	165	165	-378
10^{-12}	181	181	181	181	181	408

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	13	13	13	13	13	18
10^{-3}	25	25	25	25	25	48
10^{-4}	33	33	33	33	33	48
10^{-5}	45	45	45	45	45	78
10^{-6}	65	65	65	65	65	-78
10^{-7}	81	81	81	81	81	138
10^{-8}	97	97	97	97	97	168
10^{-9}	129	129	129	129	129	-228
10^{-10}	169	169	169	169	169	348
10^{-11}	205	201	201	201	201	438
10^{-12}	257	237	237	237	241	588

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	9	9	9	9	9	18
10^{-5}	9	9	9	9	9	18
10^{-6}	9	9	9	9	9	18
10^{-7}	17	17	17	17	17	18
10^{-8}	17	17	17	17	17	18
10^{-9}	17	17	17	17	17	18
10^{-10}	33	33	33	33	33	48
10^{-11}	33	33	33	33	33	48
10^{-12}	33	33	33	33	33	48

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	17	17	17	17	17	18
10^{-3}	17	17	17	17	17	18
10^{-4}	33	33	33	33	33	18
10^{-5}	33	33	33	33	33	18
10^{-6}	33	33	33	33	33	48
10^{-7}	41	33	33	33	33	48
10^{-8}	57	57	57	57	57	48
10^{-9}	65	65	65	65	65	108
10^{-10}	81	65	65	65	65	168
10^{-11}	105	65	65	65	65	168
10^{-12}	129	65	65	113	129	168

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	13	17	17	17	17	18
10^{-5}	17	25	25	25	17	48
10^{-6}	25	33	33	33	41	48
10^{-7}	33	49	49	49	57	48
10^{-8}	45	57	57	57	73	78
10^{-9}	57	65	65	65	105	108
10^{-10}	69	81	81	89	137	168
10^{-11}	93	105	105	105	145	198
10^{-12}	113	121	121	121	177	258

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	69	69	69	69	69	-48
10^{-2}	93	93	93	93	93	-78
10^{-3}	141	141	141	141	141	-138
10^{-4}	193	193	193	193	193	-198
10^{-5}	245	245	245	245	245	-288
10^{-6}	305	305	305	305	305	-408
10^{-7}	405	397	397	397	397	-588
10^{-8}	509	509	509	509	509	-798
10^{-9}	629	613	613	613	613	-1128
10^{-10}	797	717	717	725	729	-1528
10^{-11}	993	837	837	901	917	-2068
10^{-12}	1233	997	997	1109	1133	-2848

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	17	17	17	17	17	18
10^{-4}	17	17	17	17	17	18
10^{-5}	17	17	17	17	17	18
10^{-6}	25	25	25	25	17	18
10^{-7}	33	33	33	33	33	48
10^{-8}	33	33	33	33	33	48
10^{-9}	41	33	33	33	33	48
10^{-10}	49	33	33	33	33	48
10^{-11}	65	33	33	33	33	78
10^{-12}	81	49	49	49	49	138

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	73	73	73	73	73	18
10^{-2}	89	89	89	89	89	138
10^{-3}	129	129	129	129	129	198
10^{-4}	177	177	177	177	177	198
10^{-5}	209	193	193	193	193	228
10^{-6}	249	241	241	241	209	468
10^{-7}	329	289	289	297	289	588
10^{-8}	385	353	353	353	353	768
10^{-9}	465	393	401	401	385	1038
10^{-10}	593	425	433	449	433	1548
10^{-11}	737	465	481	545	545	2358
10^{-12}	865	593	577	641	641	2808

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	9	9	9	9	9	18
10^{-5}	9	9	9	9	9	18
10^{-6}	9	9	9	9	9	18
10^{-7}	17	17	17	17	17	18
10^{-8}	17	17	17	17	17	18
10^{-9}	25	17	17	17	17	48
10^{-10}	25	17	17	17	17	48
10^{-11}	33	17	17	33	33	48
10^{-12}	41	33	33	33	33	48

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	9	9	9	9	9	18
10^{-5}	9	9	9	9	9	18
10^{-6}	9	9	9	9	9	18
10^{-7}	9	9	9	9	9	18
10^{-8}	9	9	9	9	9	18
10^{-9}	9	9	9	9	9	18
10^{-10}	9	9	9	9	9	18
10^{-11}	17	17	17	17	17	18
10^{-12}	17	17	17	17	17	48

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9		9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	9	9	9	9	9	18
10^{-4}	9	9	9	9	9	18
10^{-5}	9	9	9	9	9	18
10^{-6}	9	9	9	9	9	18
10^{-7}	9	9	9	9	9	18
10^{-8}	9	9	9	9	9	18
10^{-9}	9	9	9	9	9	18
10^{-10}	9	9	9	9	9	18
10^{-11}	9	9	9	9	9	18
10^{-12}	9	9	9	9	9	18

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	505	505	505	505	505	-18
10^{-2}	513	513	513	513	513	-78
10^{-3}	641	641	641	641	641	648
10^{-4}	889	889	889	889	889	738
10^{-5}	977	977	977	977	977	1038
10^{-6}	1065	1025	1025	1025	1025	1218
10^{-7}	1681	1305	1305	1305	1305	2208
10^{-8}	1905	1521	1521	1521	1521	3138
10^{-9}	2017	1617	1617	1617	1617	4068
10^{-10}	3113	1657	1657	1665	1665	5718
10^{-11}	3753	1841	1841	1953	1921	6978
10^{-12}	3977	2017	2017	2049	2049	-10068

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	33	33	33	33	33	48
10^{-2}	37	37	37	37	37	78
10^{-3}	37	37	37	37	37	78
10^{-4}	45	45	45	45	45	108
10^{-5}	53	53	53	53	53	108
10^{-6}	73	57	57	57	57	138
10^{-7}	93	77	77	77	77	168
10^{-8}	97	97	97	97	97	198
10^{-9}	121	105	105	105	105	228
10^{-10}	149	117	117	117	117	288
10^{-11}	173	125	125	125	125	498
10^{-12}	193	137	137	149	149	558

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	33	33	33	33	33	78
10^{-2}	37	37	37	37	37	78
10^{-3}	41	41	41	41	41	78
10^{-4}	49	49	49	49	49	78
10^{-5}	69	69	69	69	69	138
10^{-6}	77	69	69	69	69	168
10^{-7}	89	81	81	81	81	168
10^{-8}	113	89	89	89	89	198
10^{-9}	133	101	101	101	101	288
10^{-10}	165	109	109	109	109	438
10^{-11}	209	133	133	133	133	618
10^{-12}	249	145	145	161	145	708

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	41	41	41	41	41	78
10^{-2}	41	41	41	41	41	78
10^{-3}	61	65	65	65	65	108
10^{-4}	77	77	77	77	77	108
10^{-5}	81	81	81	81	81	138
10^{-6}	105	105	105	105	105	168
10^{-7}	137	137	137	137	137	258
10^{-8}	169	161	161	161	161	288
10^{-9}	201	161	161	177	161	438
10^{-10}	249	177	177	177	177	648
10^{-11}	305	209	209	257	257	888
10^{-12}	377	273	273	289	289	1188

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	93	93	93	93	93	48
10^{-2}	241	241	241	241	241	78
10^{-3}	261	261	261	261	261	78
10^{-4}	-297	-297	-297	-297	-297	-138
10^{-5}	-417	-417	-417	-417	-417	-468
10^{-6}	937	921	921	921	921	-768
10^{-7}	1049	1009	1009	1009	1009	978
10^{-8}	1313	1137	1137	1137	1137	1368
10^{-9}	1777	1465	1465	1473	1473	2208
10^{-10}	2001	1649	1649	1657	1649	3348
10^{-11}	2369	1729	1729	1745	1737	4398
10^{-12}	3353	1825	1825	1857	1841	5808

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	41	41	41	41	41	48
10^{-2}	57	57	57	57	57	48
10^{-3}	73	73	73	73	73	108
10^{-4}	97	97	97	97	97	168
10^{-5}	129	121	121	121	121	198
10^{-6}	161	129	129	129	129	228
10^{-7}	177	153	153	153	153	288
10^{-8}	225	153	153	169	153	588
10^{-9}	273	185	185	185	185	738
10^{-10}	337	217	217	217	217	948
10^{-11}	401	241	241	265	265	1188
10^{-12}	497	273	273	273	273	1698

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	45	45	45	45	45	-18
10^{-2}	53	53	53	53	53	-48
10^{-3}	65	65	65	65	65	108
10^{-4}	89	89	89	89	89	138
10^{-5}	117	117	117	117	117	168
10^{-6}	145	145	145	145	145	228
10^{-7}	173	173	173	173	173	318
10^{-8}	229	229	229	229	229	438
10^{-9}	285	285	285	285	285	558
10^{-10}	349	337	337	337	337	768
10^{-11}	441	389	389	397	401	1008
10^{-12}	557	457	457	505	525	1398

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	9	9	9	9	9	18
10^{-2}	9	9	9	9	9	18
10^{-3}	17	17	17	17	17	18
10^{-4}	17	17	17	17	17	18
10^{-5}	25	25	25	25	25	18
10^{-6}	33	33	33	33	33	48
10^{-7}	33	33	33	33	33	48
10^{-8}	49	33	33	33	33	48
10^{-9}	57	57	65	65	65	48
10^{-10}	65	65	65	65	65	48
10^{-11}	89	65	65	65	65	168
10^{-12}	121	65	65	65	65	168

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	37	37	37	37	37	18
10^{-2}	57	57	57	57	57	48
10^{-3}	73	73	73	73	73	-78
10^{-4}	-85	-85	-85	-85	-85	-168
10^{-5}	-121	-121	-121	-121	-121	-228
10^{-6}	-157	-161	-161	-157	-149	-228
10^{-7}	-185	-185	-185	-185	-185	-348
10^{-8}	-233	-233	-233	-233	-225	-438
10^{-9}	437	-249	-249	-249	-249	888
10^{-10}	565	-289	-289	-297	-289	1248
10^{-11}	717	-329	-337	-353	-353	1608
10^{-12}	857	-369	-369	-393	-401	2268

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	129	129	129	129	129	168
10^{-2}	169	169	169	169	169	168
10^{-3}	233	233	233	233	233	228
10^{-4}	257	257	257	257	257	408
10^{-5}	297	257	257	257	257	648
10^{-6}	425	297	297	297	297	798
10^{-7}	489	329	329	329	329	1038
10^{-8}	513	345	345	345	345	1248
10^{-9}	761	353	353	401	369	2208
10^{-10}	897	481	481	497	481	3018
10^{-11}	1001	513	513	513	513	3978
10^{-12}	1417	513	513	513	513	5418

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

tol	da0glob	da1glob	da2glob	da3glob	da4glob	quadl
10^{-1}	25	25	25	25	25	18
10^{-2}	41	41	41	41	41	-18
10^{-3}	53	53	53	53	53	-18
10^{-4}	73	73	73	73	73	-78
10^{-5}	93	93	93	93	93	108
10^{-6}	113	113	113	113	113	168
10^{-7}	137	137	137	137	137	198
10^{-8}	189	169	169	169	161	288
10^{-9}	217	201	201	201	201	348
10^{-10}	281	225	225	225	225	468
10^{-11}	337	249	249	257	257	648
10^{-12}	401	305	305	337	337	888

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

References

[1] J. Berntsen. On the subdivision strategy in numerical adaptive integration over the cube. Reports in Informatics 11, Dept. of Informatics, Univ. of Bergen, 1984.

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

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

[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] C. de Boor. On writing an automatic integration algorithm. In J. R. Rice, editor, *Mathematical Software*, New York, 1971. Academic Press.

[6] 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, Dordrecht, The Netherlands, 1992. Kluwer Academic Publishers.

[7] T. O. Espelid. Doubly Adaptive Quadrature Routines based on Newton-Cotes Rules. Reports in Informatics 229, Dept. of Informatics, Univ. of Bergen, 2002.

[8] T. O. Espelid. Doubly Adaptive Quadrature Routines based on Newton-Cotes Rules. *BIT*, 43:319–337, 2003.

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

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

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

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

[17] M. A. Malcolm and R. Bruce Simpson. Local versus global strategies for adaptive quadrature. *ACM Trans. Math. Softw.*, 1(2):129–146, 1975.

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

[19] T. N. L. Patterson. On some Gauss and Lobatto based quadrature formulae. *Math. Comput.*, 22:877–881, 1968.

[20] 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, Berlin, 1983.

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

[22] 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.