An Adaptive Algorithm for the Approximate Calculation of Multiple Integrals *

Jarle Berntsen[†] and Terje O. Espelid[‡]
Department of Informatics
University of Bergen
Allegt. 55, N-5000 Bergen
Norway

Alan Genz[§]
Mathematics Department
Washington State University
Pullman, WA 99164-3113
U. S. A.

Abstract

An adaptive algorithm for numerical integration over hyperrectangular regions is described. The algorithm uses a globally adaptive subdivision strategy. Several precautions are introduced in the error estimation in order to improve the reliability. In each dimension more than one integration rule is made available to the user. The algorithm has been structured to allow efficient implementation on shared memory parallel computers.

Keywords: automatic integration, adaptive, cubature, multidimensional integration, fully symmetric rules, null rules, parallel algorithms.

*Published in *ACM Trans. Math. Softw.* **17** (1991),pp. 437–451.
[†]Supported by the Norwegian Research Council for Humanities and Sciences and STATOIL.
[‡]Supported by the Norwegian Research Council for Humanities and Sciences.
[§]Supported by the Norwegian Marshall Fund.

1 Introduction

Adaptive algorithms are now used widely for the numerical calculation of multiple integrals. These algorithms have been developed for a variety of integration regions, including hyper-rectangles, spheres and simplicies. In this paper we describe an algorithm for groups of integrals over a common n-dimensional hyper-rectangular region. The integrals have the form

$$\mathbf{I}[\mathbf{f}] \;=\; \int_{a_1}^{b_1} \int_{a_2}^{b_2} \ldots \int_{a_n}^{b_n} \mathbf{f}(\mathbf{x}) d\mathbf{x},$$

where $\mathbf{x}$ is an n-vector, and $\mathbf{f}$ is an s-vector of integrands. The algorithm input is n, $\mathbf{a}$, $\mathbf{b}$, s, $\mathbf{f}$, an error tolerance *tol* and a limit L_{max}, on the allowed number of evaluations of $\mathbf{f}$. The output is an s-vector of integral estimates $\hat{\mathbf{I}}$ and an s-vector of error estimates $\hat{\mathbf{G}}$. If possible, the algorithm computes $\hat{\mathbf{I}}$ with $\|\hat{\mathbf{G}}\|_\infty < tol$ using at most L_{max} values of $\mathbf{f}$.

The basic algorithm that we present is similar to a globally adaptive algorithm for single integrands first described by van Dooren and de Ridder [25], and improved by Genz and Malik [17], and Berntsen [2]. Implementations of the Genz and Malik modified algorithm have appeared in the NAG [1] library. Test results (Genz [15], Berntsen [3]) have shown that this algorithm may often be significantly more efficient than other algorithms for a variety of integrands when n is in the range 2-8, but the algorithm is sometimes unreliable. However, Berntsen [2, 5, 6] and Berntsen and Espelid [8] and Espelid [13] have shown how the use of problem specific rules and a more sophisticated error estimation method can increase the reliability of the original algorithm, and these ideas are incorporated into the new algorithm.

A second problem with the original algorithm is a serial organization that prevents its efficient implementation on parallel computers. Modifications to the algorithm structure at the subregion selection level that allow for good parallelization on a number of parallel computers have recently been described by Genz [16] and Berntsen [4]. By allowing for a vector integrand instead of a single integrand the performance pf the new algorithm may also be better for some large scale calculations where a large number of similar integrals, possibly differing only by the value(s) of some parameter(s), are needed over a common integration region. In this case it often happens that a significant part of the computation required for each integrand is the same for all of the integrands. These common calculations need be done only once for each integrand evaluation point, and it may often be possible to do the remaining calculations with some parallelism. Thus, the new algorithm allows for user controlled parallelism at the lowest level during the integrand evaluations and, also at the subregion level. An additional saving with the new algorithm may occur because all of the integral calculations use the

same subdivision of the integration region, and so the work for finding a good subdivision is shared among all of the integral calculations.

For a vector of integrals, where the components vary continuously as functions of some parameter, the estimates produced by our algorithm will also vary continuously when the same subdivision is applied to all components. This will generally not be the case when different components are given separate treatment (see Lyness and Kaganove [20]).

In the next three sections we describe in detail the new algorithm which incorporates these improvements. In Section 2 the adaptive subdivision strategy is described, including modifications that facilitate parallelization at the subregion level and allow for more than one integrand. In Section 3 we describe several different integration rules that can be selected for use with the algorithm, and in Section 4 we describe the new error estimation method. A FORTRAN implementation of the algorithm, called DCUHRE [10], was used for the test results described in Section 5. Some concluding remarks are given in Section 6.

2 The Globally Adaptive Algorithm

In this section we describe the adaptive integration algorithm in detail. All of the adaptive algorithms for numerical integration have a general structure that consists of the following four main steps:

i) Choose some subregion(s) from a set of subregions.
ii) Subdivide the chosen subregion(s).
iii) Apply an integration rule to the new subregions; update the subregion set.
iv) Update global integral and error estimates; check for convergence.

The corresponding components that distinguish these adaptive algorithms are i) a data structure for the subregion set, ii) a subdivision strategy and iii) an integration rule and an error estimator.

The original van Dooren and de Ridder algorithm was initialized by applying the integration rule to the integrand on the whole integration region. The integration rule provided an estimate for the integral, as well as an error estimate and a coordinate axis for the next subdivision. In the original algorithm, the subregion set was an ordered list with the subregions ordered according to the corresponding error estimates, and the algorithm proceeded from one stage to the next by always choosing the subregion with largest estimated error for subdivision. This subregion was divided in half along one coordinate axis to produce two new subregions, and then the algorithm continued with steps iii) and iv) above. Following the recommendations made by Malcolm and Simpson [21] for one dimensional globally adaptive algorithms, Genz and Malik changed this subregion

set data structure to a heap, in order to reduce the overheads associated with maintenance of the set. The algorithm in this paper uses a heap data structure for the subregion set.

The new algorithm uses the same integration rule for all of the integrands; in a particular subregion the use of this rule gives an s-vector $\mathbf{R}$ of integration rule results for that subregion, an s-vector $\hat{\mathbf{E}}$ of error estimates and a coordinate axis k for a subsequent subdivision of that subregion. The original and improved algorithms divided one selected subregion into two pieces along the coordinate axis where the integrand had largest fourth divided difference. The new algorithm uses this same strategy, with two modifications. The first modification is introduced because the new algorithm applies to a vector of integrands. The same subdivision is used for all of the integrands, so the subdivision axis for the next subdivision of a particular subregion should be determined using information from all of the integrands. Let $\mathbf{u}$ be the center of a chosen subregion with dimensions given by the components of the width vector $\mathbf{v}$, and let $\mathbf{u}(\alpha)_i$ be the vector obtained from $\mathbf{u}$ by adding $\alpha v_i/2$ to the ith component of $\mathbf{u}$. Given some positive parameters α_1 and α_2, define a fourth difference operator along coordinate axis i by

$$D_i\mathbf{f} = \|\mathbf{f}(\mathbf{u}(\alpha_1)_i) + \mathbf{f}(\mathbf{u}(-\alpha_1)_i) - 2\mathbf{f}(\mathbf{u}) - \frac{{\alpha_1}^2}{{\alpha_2}^2}(\mathbf{f}(\mathbf{u}(\alpha_2)_i) + \mathbf{f}(\mathbf{u}(-\alpha_2)_i) - 2\mathbf{f}(\mathbf{u}))\|_1.$$

The subdivision axis chosen for that subregion is a coordinate axis k where $D_k\mathbf{f} = \|\mathbf{Df}\|_\infty$. The integrand values used for these difference operators are also used in the integration rule, so the calculation of the differences does not significantly contribute to the computation time. The computation of these differences is done at the same time as the integration rule and error estimate computations are done. In order to ensure that several different axes are chosen for subsequent subdivision when the differences are near zero, any term in the sum for $D_i\mathbf{f}$ is set to zero if that term is less than four times the machine epsilon, relative to the absolute value of the integrand at the center of the subregion. If there are several axes where the maximum occurs, k is chosen from this group to be the axis where the subregion width (v_k) is largest.

A second modification to the previous algorithms is introduced to allow the new algorithm to run more efficiently on a parallel computer with p processors. At each stage the p/2 subregions with the largest errors are divided into two pieces along a (possibly different) selected coordinate axis. This produces p subregions for the parallel computation of the integration rule results.

The error assigned to each subregion for the purpose of determining the position of the subregion in the subregion heap is the maximum of the estimated errors from the integration rule results from that subregion for the different components of the vector integrand. In addition to this maximum error, the information that

is maintained for a subregion in the heap is the subregion center $\mathbf{u}$ (n components), the subregion width vector $\mathbf{v}$ (n components), the subregion integration rule results $\mathbf{R}$ (s components), the subregion error estimates $\hat{\mathbf{E}}$ (s components) and the coordinate axis k chosen for the next subdivision. A heap of M subregions therefore requires M(2n+2s+2) storage locations. In our implementation of the algorithm there must be a user specified limit M_{max} on the working storage space; the algorithm described here does not continue if the next step requires more than this amount of heap storage. The global variables that need to be maintained by the algorithm are the number of subregions M, the number of integrand evaluations L_{tot}, the integral estimates $\hat{\mathbf{I}}$ and the global error estimates $\hat{\mathbf{G}}$.

We finish this section with a more complete description of the algorithm steps. The integration rule and error estimation are assumed to require L integrand evaluations.

if $L_{max} \geq L$ **then**
  use the integration rule in the whole integration region,
    returning $\mathbf{R}$, $\hat{\mathbf{E}}$ and a subdivision axis k
  initialize the heap, M, L_{tot}, $\hat{\mathbf{I}}$ and $\hat{\mathbf{G}}$
  set P = 1
  do while ($M + P \leq M_{max}$ **and** $L_{tot} + 2PL \leq L_{max}$ **and** $\|\hat{\mathbf{G}}\|_\infty \geq tol$)
    take P subregions from the heap
    divide the P subregions in half along the chosen axes
    use the integration rule in 2P new subregions (in parallel),
      returning $\mathbf{R}$, $\hat{\mathbf{E}}$ and k for each new subregion
    update the heap, M, L_{tot}, $\hat{\mathbf{I}}$ and $\hat{\mathbf{G}}$
    set P = max(1, min(p/2, M, M_{max}-M))
  endo
endif.

The new algorithm retains the features that made the original and improved algorithms efficient, while allowing for parallelism at both the subregion and integrand levels. The heap maintenance could also be parallelized, but test results from implementations of earlier versions of this algorithm have shown that the time for heap maintenance is usually insignificant compared to the integrand evaluation time, even when the integrand evaluations are done in parallel. The other significant changes to the previous algorithms occur in the integration rule and error estimation components, and these changes are discussed in the next two sections.

3 Integration Rules and Null Rules

All rules used by the algorithm are fully symmetric (FS-) rules. For each possible choice of the parameter "key" a set of five FS-rules is used. In such a set there is one *integration rule* R of polynomial degree 2m+1

$$I[f] = \int_H f(\mathbf{x})d\mathbf{x} \simeq R[f] = \sum_{j=1}^{L} w_j f(\mathbf{x}_j) \tag{1}$$

where H is the region of integration, $\mathbf{x}_j$ the evaluation points and w_j the corresponding weights, $j = 1, \ldots, L$. In addition there are four *null rules* N_i of structure

$$N_i[f] = \sum_{j=1}^{L} w_j^{(i)} f(\mathbf{x}_j), \;\; i = 1, 2, 3, 4. \tag{2}$$

All rules in such a set are based on the same set of evaluation points $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$. We have $w_j^{(i)} \neq 0$ (with $w_j^{(0)} = w_j$) for at least one value of $i \in (0, 1, 2, 3, 4)$ for every $j \in (1, 2, \ldots, L)$. The null rules have polynomial degree $2m-1$, $2m-1$, $2m-3$ and $2m-5$ respectively. The integration rule is used to estimate the integral over each subregion while the null rules are used to produce four estimates of the error

$$E[f] = R[f] - I[f] \tag{3}$$

over each subregion. More details about the construction of the final error-estimate are given in the next section.

A rule for the cube $[-1, 1]^n$ is *fully symmetric* if ,whenever the rule contains a point $\mathbf{x} = (x_1, x_2, ..., x_n)$ with associated weight w, it contains all points that can be generated from $\mathbf{x}$ by permutations and/or sign-changes of the coordinates with the same associated weight. An integration rule has *polynomial degree* d if it integrates exactly all *monomials* $x_1^{k_1} x_2^{k_2} \ldots x_n^{k_n}$ with $\Sigma k_i \leq d$ and fails to integrate exactly at least one monomial of degree $d + 1$. A null rule, see Lyness [19], of polynomial degree d will integrate to zero all monomials of degree $\leq d$ and will fail to do so for at least one monomial of degree $d + 1$. Note that for any value of $\mu \neq 0$ then $\mu N_1[f]$ is a null rule too and

$$R[f] + \mu N_1[f] \tag{4}$$

is a FS integration rule of degree $2m - 1$. Thus each null rule may be thought

of as the difference between the basic integration rule $R[f]$ and an appropriate integration rule of lower degree.

A point $\mathbf{x}$ in the FS-set of evaluation points with $0 < x_i$ for i = 1, 2, ..., r and $x_i = 0$ for i = r+1, r+2, ..., n, $\mathbf{x}$ is a *generator* of all points in this FS-set which can be generated from $\mathbf{x}$ using the FS-property. In Table 1 we list all of the types of generators that are used in this algorithm.

Table 1. Types of n-dimensional generators.

Types	Generators	Number of points in set
(0,0,0,0, ...,0)	(0,0,0,0, ...,0) , $i = 1(1)K_0$	1
(α,0,0,0,...,0)	(α_i,0,0,0, ...,0) , $i = 1(1)K_1$	$2n$
(β,β,0,0, ...,0)	(β_i,β_i,0,0, ...,0) , $i = 1(1)K_2$	$2n(n-1)$
(γ,δ,0,0, ...,0)	(γ_i,δ_i,0,0, ...,0) , $i = 1(1)K_3$	$4n(n-1)$
(ε,ε,ε,0, ...,0)	(ε_i,ε_i,ε_i,0, ...,0) , $i = 1(1)K_4$	$4n(n-1)(n-2)/3$
(ζ,ζ,η,0, ...,0)	(ζ_i,ζ_i,η_i,0, ...,0) , $i = 1(1)K_5$	$4n(n-1)(n-2)$
(λ,λ,λ,λ, ...,λ)	(λ_i,λ_i,λ_i,λ_i, ...,λ_i) , $i = 1(1)K_6$	2^n

From Table 1 we see that 7 different types of generators are used, which we have numbered from 0 to 6. K_j is *a structure parameter*, see Mantel and Rabinowitz [22], which gives the number of different generators of type j, $j = 0, 1, \ldots, 6$, where K_0 has possible values 0 and 1 only.

Based on the observation that software, using one integration rule only, changes performance from one test problem to another, we decided to offer the user a choice of rule through an input parameter "key" in all dimensions. In dimensions 2 and 3 we offer three different sets of rules while in other dimensions we offer two sets of rules. In each dimension the main difference between the sets of rules we offer is the polynomial degree. In Table 2 we list the different options we have in this algorithm specifying in which dimension(s) the set of FS-rules of a certain structure is available. By the notation $d(K_0, K_1, K_2, K_3, K_4, K_5, K_6)L$ we give the degree $d = 2m + 1$ of the integration rule in the set, the value of the seven structure parameters and the number of points L for a given set of FS-rules in this algorithm.

Table 2. Structure-parameters for the sets of FS-rules.

Key	Structure-parameters	Dimension
1	$13(1,5,5,3,0,0,0)65$	2
2	$11(1,5,2,0,3,2,0)127$	3
3	$9(1,4,2,1,0,0,0)33$	2
3	$9(1,4,1,1,1,0,1)l_n$	$n \geq 3$
4	$7(1,3,1,0,0,0,1)q_n$	$n \geq 2$

where

$$\begin{aligned} l_n &= 1 + 8n + 6n(n-1) + 4n(n-1)(n-2)/3 + 2^n, \\ q_n &= 1 + 6n + 2n(n-1) + 2^n. \end{aligned}$$

The set of rules for key= 1 is based on a pair of rules constructed by Eriksen [12] while the rules for key= 2 are based on a paper by Berntsen and Espelid [7, 8]. The set of rules for key ≥ 3 is based on a paper by Genz and Malik [18]. All rules used in the algorithm use evaluation points inside the region of integration.

In Table 3 we give the number of points used by the different structures for key values 3 and 4 for all dimensions between 2 and 10.

Table 3.

n	l_n	q_n
2	33	21
3	77	39
4	153	65
5	273	103
6	453	161
7	717	255
8	1105	417
9	1687	711
10	2605	1265

4 Error Estimation

We are interested in producing estimates of the error (3) when $R[f]$ is a degree $2m+1$ approximation. It is normally assumed (see Genz and Malik [17]) that $|E[f]|$ is bounded by an expression

$$|E[f]| \leq \hat{E}[f] = |N[f]| \tag{5}$$

for some null rule N of degree less than $2m+1$. Numerical experiments have shown (Berntsen [3, 5] and [15]) that a more reliable routine has to use a more sophisticated error estimation procedure.

The error estimates produced by the algorithm are based on the approximations delivered by the null rules (2), with a common scaling given by

$$\parallel N_i \parallel_1 \equiv \sum_{j=1}^{L} |w_j^{(i)}| = 2^n, \;\; i = 1, 2, 3, 4, \tag{6}$$

when $[-1,1]^n$ is the region of integration.

Lyness and Kaganove [20] have shown how "phase factors" may cause severe problems for the error estimation. In order to avoid the problems with "phase factors" we therefore use pairs of linearly independent null rules N_i and N_{i+1} and select for each subregion the error estimate

$$N_i^*[f] = 2^n \underset{\mu_i}{max}(|\mu_i N_i[f] + N_{i+1}[f]|/ \parallel \mu_i N_i + N_{i+1} \parallel_1), \;\; i = 1, 2, 3. \tag{7}$$

$N_i^*[f]$ will then be the greatest error estimate produced by a null rule in the space spanned by N_i and N_{i+1} with norm satisfying (6) (Espelid [14]). Furthermore the null rules $\mu_i N_i + N_{i+1}$, for i=1,2,3, are of degree $2m-1, 2m-3$ and $2m-5$ respectively. In [14] it is shown that the number of values of μ_i that have to be considered when computing the maximum in (7), is equal to the number of generators of the integration rule applied, and therefore the work involved when computing $N_i^*[f]$ is almost negligible.

In order to check the asymptotic assumption behind (5) we also check the inequalities([5, 6])

$$\frac{N_{i+1}^*[f]}{N_i^*[f]} \geq c_i > 1, \;\; i = 1, 2, \tag{8}$$

for suitable constants c_i. If the tests (8) are passed, we choose

$$\hat{E}[f] = c_3 N_1^*[f]; \tag{9}$$

otherwise we choose

$$\hat{E}[f] = c_4 max(N_1^*[f], N_2^*[f], N_3^*[f]), \tag{10}$$

where c_3 and c_4 are heuristic constants that are selected to achieve a reasonable balance between reliability and efficiency.

In globally adaptive quadrature algorithms based on bisection of subregions, we build up binary trees of subregions as the computation proceeds. The final estimates of the integral and the error are based on the corresponding estimates from the subregions represented by the leaves of such a binary tree. Going from one level in the tree to the next, all information from the "old" function evaluations are often neglected. At least this is the case for the QUADPACK (Piessens et al. [23]) routine QAG and multidimensional ADAPT like routines ([17] and [25]). Any error estimate based on a finite number of integrand evaluations may be unreliable, and even though the actual error is very great, the estimated error may be negligible. The result of this may therefore be that a problem spot discovered through a large error estimate on one level of the tree, may be "lost" on the next level.

In order to try to avoid this problem we use two-level error estimates. Let $R_2[f]$ be an approximation to the integral over a given region H, and $R_1^{(j)}[f]$ for j = 1, 2, be two approximations over the two children subregions we get by bisecting H. We then get a two level error estimate by the expression

$$\hat{E}_2[f] = |R_2[f] - (R_1^{(1)}[f] + R_1^{(2)}[f])|. \tag{11}$$

This error estimate is combined with the local error estimates $\hat{E}_1^{(1)}[f]$ and $\hat{E}_1^{(2)}[f]$ to produce the final estimated errors, $\hat{E}^{(1)}$ and $\hat{E}^{(2)}$, over each of the two new regions (see Sørevik [24] and [5]),

$$\hat{E}^{(j)}[f] = \hat{E}_1^{(j)}[f] + c_5 \frac{\hat{E}_1^{(j)}[f]}{(\hat{E}_1^{(1)}[f] + \hat{E}_1^{(2)}[f])} \hat{E}_2[f] + c_6 \hat{E}_2[f], \;\; j = 1, 2. \tag{12}$$

We summarize the error estimation procedure as follows:

Error Estimation Procedure

For each of the children subregions we first compute

$$N_i^*[f] = 2^n \max_{\mu_i}(|\mu_i N_i[f] + N_{i+1}[f]|/ \parallel \mu_i N_i + N_{i+1} \parallel_1), \;\; i = 1, 2, 3,$$

and then we impose the test

if $c_1 N_1^*[f] \leq N_2^*[f]$ and $c_2 N_2^*[f] \leq N_3^*[f]$ **then**
$\hat{E}_1^{(j)}[f] = c_3 N_1^*[f]$
else
$\hat{E}_1^{(j)}[f] = c_4 max(N_1^*[f], N_2^*[f], N_3^*[f])$
endif,

where $\hat{E}_1^{(1)}[f]$ and $\hat{E}_1^{(2)}[f]$ are local error estimates over the children subregions. A two level error estimate $\hat{E}_2[f]$ is computed by

$$\hat{E}_2[f] = |R_2[f] - (R_1^{(1)}[f] + R_1^{(2)}[f])|.$$

The final estimates

$$\hat{E}^{(j)}[f] = \hat{E}_1^{(j)}[f] + c_5 \frac{\hat{E}_1^{(j)}[f]}{(\hat{E}_1^{(1)}[f] + \hat{E}_1^{(2)}[f])} \hat{E}_2[f] + c_6 \hat{E}_2[f], \;\; j = 1, 2,$$

are then computed.

The table below shows the values of the heuristic constants used by the different sets of rules in DCUHRE, the FORTRAN implementation of this algorithm.

Table of heuristic constants.

key	c_1	c_2	c_3	c_4	c_5	c_6
1	10	10	1	5	0.5	0.25
2	4	4	0.5	3	0.5	0.25
3	5	5	1	5	0.5	0.25
4	5	5	1	5	0.5	0.25

All heuristic constants are selected to achieve a reasonable level of reliability without increasing the cost too much. Our technical report [9] provides numerical evidence on how DCUHRE (note that the original name of this routine was ADMINT) performs with these choices of heuristic constants.

5 Testing

The FORTRAN implementation of the algorithm, DCUHRE, has been tested for efficiency and reliability, using the testing technique described in [20]. The testing technique is based on the use of a selection of test families of integrands. Each integrand in a test family has a particular feature (a peak, for example), some random parameters (the location of the peak) and a difficulty parameter (the sharpness of the peak). In each experiment the difficulty parameter is fixed, and the random parameters are varied, and we let the integration routine approximate the integral for each set of random parameters. In the technical report [9] we report on how DCUHRE (original name ADMINT) performs on 8 different test families for different dimensions. In that report we also give comparative test results for the routines ADAPT[17] and KROBOV (see Cranley and Patterson [11] and [15]). The tests were done with 2,3,4,5,6,7,8,10,12 and 15 dimensional integrals and took approximately 260 hours of CPU on an Alliant FX/8 (one processor).

We have selected some 2 dimensional test results of DCUHRE and ADAPT from that report, which we give here. For these results, the test integrand families were:

	Test families	Attributes
1	$f_1(\mathbf{x}) = \Pi_{i=1}^{2}(\tau_i^{-2} + (x_i - \xi_i)^2)^{-1}$	Product peak
2	$f_2(\mathbf{x}) = cos(2\pi\xi_1 + \sum_{i=1}^{2} \tau_i x_i)$	Oscillatory

The parameters ξ_1 and ξ_2 are picked randomly from [0,1]. τ_1 and τ_2 are first picked randomly from [0,1] and then scaled according to

$$\sum_{i=1}^{2} \tau_i = \rho_j, \quad j = 1, 2.$$

In the experiments reported here we have chosen $(\rho_1, \rho_2) = (300/2^{1.5}, 15)$. The region of integration for all test families is the unit square $[0, 1]^2$. MAXVLS, the maximum allowed number of integrand evaluations, was set equal to 200,000 in all experiments.

For each test family the test results are based on 200 samples of the random variables. In the figures 1, 2 and 3 we plot the test results for test family 1. In each of these three figures we plot the requested error, for values $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$ and 10^{-5}, against the average numbers of function evaluations (Fig. 1), the average numbers of correct digits in the computed approximations of the integrals (Fig. 2) and the number of failures out of 200 (Fig. 3). A failure is a case where

the code reports success and but the actual error is greater than the requested error.

In the figures 4 and 5 we present similar plots for test family 2. For test family 2 no failures were reported for any of the sampled functions and the different codes.

We plot the results for ADAPT and all three possible 2-dimensional choices of DCUHRE, where the number behind DCUHRE in the plots indicates which of the available 2-dimensional integration rules that has been applied. DCUHRE-1 uses the degree 13 rule, DCUHRE-3 the degree 9 rule and DCUHRE-4 the degree 7 rule.

Figure 1. Average numbers of function evaluations for test family 1 (Product peak).

Figure 2. Average numbers of correct digits for test family 1 (Product peak).

Figure 3. Numbers of failures out of 200 samples for test family 1 (Product peak).

Figure 4. Average numbers of function evaluations for test family 2 (Oscillatory).

Figure 5. Average numbers of correct digits for test family 2 (Oscillatory).

Note that for none of these 2000 sampled functions did any of the codes report an unsuccessful run. The results for test family 1 show the improved reliability (Fig. 3) of DCUHRE compared to ADAPT. From Figure 1 it is also clear that we have to pay for this improved reliability and the relative increase in cost is greatest in those cases where it is most important, namely, when we want only a few digits of accuracy.

The results for test family 2 show that in cases where ADAPT is reliable enough, there may not be an additional cost, or the additional cost may be very small, by introducing the error estimating procedures described in section 4. We also see in Figure 4 that on oscillatory problems there may be a substantial gain in efficiency by moving from a low degree rule to a higher degree rule and this gain is greater when we require more digits of accuracy.

6 Concluding Remarks

The main goal in the development of the new algorithm for adaptive multidimensional integration was to improve the reliability of some previous algorithms. Tests [9] of a FORTRAN implementation, DCUHRE, have shown that this goal has been achieved. The new algorithm has been structured to allow its efficient implementation on shared memory parallel computers and good speedups are demonstrated in the test report [9]. In each dimension more than one integration rule is made available to the user because the relative performance of different rules may differ from one problem to another. The other additional feature of the new algorithm are the modifications that allow its application to a vector of similar integrals over a common integration region. This feature should be used with caution, because the algorithm chooses a subdivision of the integration region that may be refined according to the behavior of particular integrands which have certain local regions of difficulty. If the integrands in a vector of integrands are significantly different, the vector should probably be split into smaller vectors of integrands and the algorithm applied separately to each subvector. For integrands that have enough similarity, the new algorithm may save both time and space because of the common subdivision, and will also allow for additional saving in time on many types of parallel computers where the integrand function evaluations can be done in parallel.

7 Acknowledgements

The authors thank T. Sørevik for participation in discussions in the initial phase of this work. The authors also thank the referees for valuable suggestions and comments.

References

[1] The Numerical Algorithm Group. Mayfield House, 256 Banbury Road, 294 Oxford OX2 7DE, United Kingdom.

[2] J. Berntsen. Cautious adaptive numerical integration over the 3-cube. Reports in Informatics 17, Dept. of Inf., Univ. of Bergen, 1985.

[3] J. Berntsen. A test of the NAG-software for automatic integration over the 3-cube. Reports in Informatics 15, Dept. of Inf., Univ. of Bergen, 1985.

[4] J. Berntsen. Adaptive multidimensional quadrature routines on shared memory parallel computers. Reports in Informatics 29, Dept. of Informatics, Univ. of Bergen, 1987.

[5] J. Berntsen. Practical error estimation in adaptive multidimensional quadrature routines. Reports in Informatics 30, Dept. of Informatics, Univ. of Bergen, 1988.

[6] J. Berntsen. Practical error estimation in adaptive multidimensional quadrature routines. *J. Comp. Appl. Math.*, 25:327–340, 1989.

[7] J. Berntsen and T.O. Espelid. On the construction of higher degree three dimensional embedded integration rules. Reports in Informatics 16, Dept. of Inf., Univ. of Bergen, 1985.

[8] J. Berntsen and T.O. Espelid. On the construction of higher degree three-dimensional embedded integrations rules. *SIAM J. of Numer. Anal.*, 25:222–234, 1988.

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

[10] J. Berntsen, T.O. Espelid, and A. Genz. An Adaptive Multidimensional Integration Routine for a Vector of Integrals. To appear in ACM Trans. Math. Software, 1990.

[11] R. Cranley and T.N.L. Patterson. Randomization of Number Theoretic Methods for Multiple Integration. *SIAM J. Numer. Anal.*, 13:904–914, 1976.

[12] S.S. Eriksen. On the development of embedded fully symmetric quadrature rules for the square, 1986. Dept. of Informatics, Univ. of Bergen, Thesis for the degree Cand. Scient.

[13] T. O. Espelid. On the construction of good fully symmetric integration rules. *SIAM J. Numer. Anal.*, 24:855–881, 1987.

[14] T.O. Espelid. Integration Rules, Null Rules and Error Estimation. Reports in Informatics 33, Dept. of Informatics, Univ. of Bergen, 1988.

[15] A.C. Genz. Testing Multiple Integration Software. In *Tools, Methods and Languages for Scientific and Engineering Computation*, B. Ford, J-C. Rault, and F. Thommaset (Eds.), pp. 208–217. North Holland, New York, 1984.

[16] A.C. Genz. The numerical evaluation of multiple integrals on parallel computers. In *Numerical Integration*, P. Keast and G. Fairweather (Eds.), pp. 219–230. D. Reidel, Dordrecht, Holland, 1987.

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

[18] A.C. Genz and A.A. Malik. An imbedded family of fully symmetric numerical integration rules. *SIAM J. Numer. Anal.*, 20:580–588, 1983.

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

[20] J.N. Lyness and J.J. Kaganove. Comments on the nature of automatic quadrature routines. *ACM Trans. Math. Software*, 1:65–81, 1976.

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

[22] F. Mantel and P. Rabinowitz. The application of integer programming to the computation of fully symmetric integration formulas in two and three dimensions. *SIAM J. Numer. Anal.*, 14:391–425, 1977.

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

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

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