Journal of Universal Computer Science, vol. 4, no. 5 (1998), 486-500
submitted: 25/11/97, accepted: 3/5/98, appeared: 28/5/98 © Springer Pub. Co.

A Hybrid Subdivision Strategy for Adaptive Integration Routines

Ronald Cools
(Katholieke Universiteit Leuven, Belgium
ronald.cools@cs.kuleuven.ac.be)

Bart Maerten
(Katholieke Universiteit Leuven, Belgium
bart.maerten@cs.kuleuven.ac.be)

Abstract: In this paper we propose a modification of a part of the global adaptive integration algorithm that is usually taken for granted: the subdivision strategy. We introduce a subdivision strategy where the routine decides whether it is best to divide a hyper-rectangular region or a n-simplex in 2 or 2^n parts or something in between.

Key Words: quadrature; cubature; adaptive integration software

Category: G.1, G.4

1 A globally adaptive cubature algorithm

The goal of quadrature (and cubature) software is to produce estimates for (multiple) integrals. Most software that is available nowadays is adaptive, i.e., the software selects at run-time the points where the integrand is evaluated based on the behaviour of the integrand. We repeat here the high level description of the classical globally adaptive scheme given in [Cools, Haegemans 92].

The algorithm is presented in [Alg. 1]. The region collection, which contains all information about the regions that is needed for future reference, is organised using some data structure. The region collection management routines create and maintain the region collection. The algorithm controller takes region(s) from the collection and passes them to the region processor. In a globally adaptive algorithm the region with the largest absolute error is selected from the collection for further processing. The region processor tries to improve the estimates for the integrals $\hat{Q}_k$ and their error $\hat{E}_k$ over the regions it receives from the algorithm controller. Usually the region processor first divides the region in parts with equal volume and same shape as the given region and then computes new estimates $\hat{Q}_k^{(i)}$ for the integrals over each subregion using a fixed local integral estimator and error estimator. The results from the region processor are returned to the algorithm controller, which passes all relevant information to the region collection management routines. The algorithm controller decides when to terminate the algorithm. This is usually done when the total estimated error $\hat{E}$ becomes smaller than the requested error ϵ.

In most previously developed algorithms the statement 'Process these region(s)' is implemented as described in [Alg. 1]. The region processor consists of 2 independent parts that are executed sequentially: first divide, then compute new approximations for each region using an a priori chosen quadrature (cubature) formula and error estimator. The quadrature (cubature) formulas that one uses, are weighted sums of function values. In [Cools, Haegemans 92] we described alternative region processors, e.g., where the region processor is allowed to decide whether a region should be subdivided or not. In the following section we describe a region processor that decides which subdivision to choose.

Algorithm 1: A globally adaptive quadrature/cubature algorithm with standard Region Processor.

Initialise the collection of regions with the M given regions;
Produce $\hat{Q}_k$ and $\hat{E}_k$ for $k = 1, 2, \ldots, M$;
Put $\hat{Q} = \sum_{k=1}^{M} \hat{Q}_k$ and $\hat{E} = \sum_{k=1}^{M} \hat{E}_k$;
while $\hat{E} > \epsilon$ **do**
begin
    Take some region(s) from the collection;
    Process these region(s)
        Divide the region into s subregions;
        Compute $\hat{Q}_k^{(i)}$ and $\hat{E}_k^{(i)}$,$i = 1, 2, \ldots, s$ using an 'a priori' chosen integration rule and error estimator;
    Update $\hat{Q}$ and $\hat{E}$
        $\hat{Q} = \hat{Q} + (\sum_{i=1}^{s} \hat{Q}_k^{(i)} - \hat{Q}_k)$;
        $\hat{E} = \hat{E} + (\sum_{i=1}^{s} \hat{E}_k^{(i)} - \hat{E}_k)$;
    Put the new regions in the collection
    and put $M = M + s - 1$;
end

2 The subdivision strategy

In the adaptive routines that are widely available for integration over an interval, the region processor starts with bisecting the interval into two equal parts. In the literature one can find only few articles that investigate irregular subdivisions or subdivisions into more than two parts, see [Hanke 82] and [Berntsen, Espelid, Sørevik 91].

For the n-cube the available adaptive routines start with bisecting the cube into two equal parts. For the n-simplex most available adaptive routines start with dividing a simplex into 2^n simplices. In the following sections we will describe this and propose a modified subdivision strategy. The 2-dimensional case will be discussed in detail and a possible extension to higher dimensions will be described.

The subdivision strategy described in this paper is implemented in Cubpack++, a C++ package for approximating integrals over a large variety of 2-dimensional regions [Cools, Laurie, Pluym 97a] [Cools, Laurie, Pluym 97b]. For the triangle this was done recently and thus not mentioned in the paper describing Cubpack++. The package (source code, User Manual and example files) is available at `http://www.cs.kuleuven.ac.be/~ronald` .

2.1 Subdivision strategies for the cube

In all global adaptive routines available for integration over hyper-rectangular regions (HALF [Van Dooren, De Ridder 76], ADAPT [Genz, Malik 80], DCUHRE [Berntsen, Espelid, Genz 91a] [Berntsen, Espelid, Genz 91b]) the region processor starts with dividing the target region into 2 equal halves based on the direction with the largest fourth divided difference. This is computed using the integrand evaluations of 5 points on each coordinate axis used by the cubature formula. This subdivision strategy was first used in 1974 by Luc De Ridder and Paul Van Dooren in their master's thesis [De Ridder, Van Dooren 74] from which HALF [Van Dooren, De Ridder 76] was derived.

It is assumed that a division in 2 is more adaptive than a division in 2^n regions congruent with the given hyper-rectangle. Van Dooren and De Ridder already wrote: "To improve adaptivity R is not divided into 2^n, but only into 2." Besides, if n is large, a 2^n-division might be too expensive.

Let us first think about the worst thinkable cases for these two types of subdivision. We assume the user gave one region.

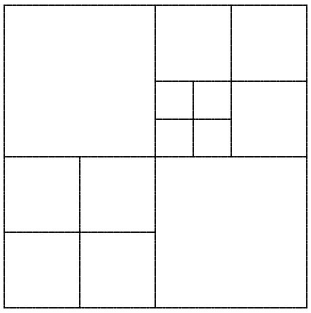

Figure 1: Ideal 4-division for a square

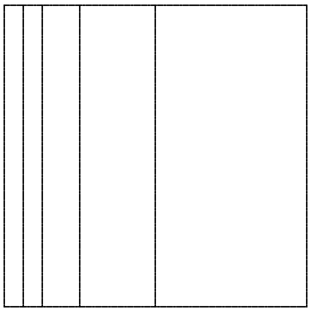

Figure 2: Ideal 2-division for a square

The worst thinkable scenario for a 2-division is that the final subdivision corresponds with that of a 2^n-division as in [Fig. 1]. Indeed, after m steps of a 2^n-division, the number of regions for which the integral was estimated using a cubature formula $= R = 1 + 2^n m$ and $A = 1 + (2^n - 1)m$ of them are in the region collection (the Active regions). Obtaining this final subdivision with a 2-division requires m_2 steps:

$$A = 1 + m_2 = 1 + (2^n - 1)m \Rightarrow m_2 = m(2^n - 1)$$

so the integral is estimated for $R_2 = 1 + 2m_2 = 1 + 2m(2^n - 1)$ regions. Hence

$$\frac{R_2}{R} = \frac{1 + 2m(2^n - 1)}{1 + 2^n m} \approx \frac{2m(2^n - 1)}{2^n m} = 1 + (1 - 2^{1-n}).$$

E.g. for 2 dimensions it follows from the above equation that a 2-division requires 50% more work than a 4-division, if we assume that the amount of work is the same for each subregion.

The worst thinkable scenario for a 2^n-division is that the ideal subdivision corresponds with always dividing in the same direction, e.g., due to a serious problem along one of the faces, as in [Fig. 2]. Then the ideal 2-division computes the integral estimate for $R_2 = 1 + 2m$ regions of which $A_2 = 1 + m$ remain in the region collection. The 2^n-division cannot obtain this ideal subdivision. The corresponding 2^n-division will divide in each of the m steps all subregions that touch the face causing the difficulty. The 2^n-division would compute integral estimates for R regions

$$\begin{aligned} R &= 1 + 2^n + 2^n 2^{n-1} + 2^n 2^{2n-2} + 2^n 2^{3n-3} + \cdots \\ &= 1 + \sum_{i=0}^{m-1} 2^n 2^{i(n-1)} \\ &= 1 + 2^n \sum_{i=0}^{m-1} \left(2^{n-1}\right)^i \\ &= 1 + 2^n \left(\frac{2^{m(n-1)} - 1}{2^{n-1} - 1} \right) \\ &= \frac{2^n 2^{m(n-1)} - 2^{n-1} - 1}{2^{n-1} - 1}. \end{aligned}$$

So,

$$\frac{R}{R_2} = \frac{2^n 2^{m(n-1)} - 2^{n-1} - 1}{(2^{n-1} - 1)(2m + 1)} \approx \frac{2^{m(n-1)}}{m}.$$

We see that the worst case for a 2^n-division is more dramatic than the worst case for the 2-division.

But why should one restrict to either a 2-division or a 2^n-division? We suggest that the region processor not just decides on the subdivision direction but also on the number of subregions 2^i with $1 \leq i \leq n$. In the sequel we will illustrate this idea for 2-dimensional regions and suggest how this can be generalised to higher dimensions.

Experiments with alternative subdivision strategies are reported in some master thesises. We are however not aware of publication of such an investigation in the international available literature and there is no sign of such a strategy in the available software, except in Cubpack++.

If one wants to divide a cube into 2 parts, one has to find a criterion to decide how to cut the cube. As mentioned in the beginning of this section, Van Dooren and De Ridder introduced the fourth divided difference for this purpose and since then, everybody uses this. The idea behind this is that a difference in a particular direction is a measure for the difficulty in that direction. Because the basic cubature formula used by Van Dooren and De Ridder has 5 points on each coordinate axis [see Fig. 3], they could compute the fourth divided difference in all n coordinate directions without any additional function evaluation.

The use of the fourth divided difference is one of the holy cows of adaptive integration. Recent routines use the fourth divided difference although they use cubature formulas with more than 5 points on each coordinate axis. DCUHRE uses fourth divided differences although the number of points on each axis varies between 7 and 11.

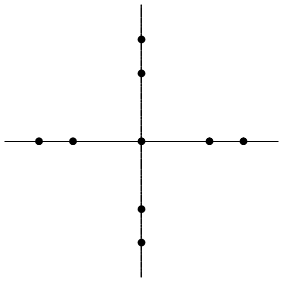

Figure 3: Distribution of cubature points on the coordinate axes.

Using CUBPACK's Fortran framework [Cools, Haegemans 92], we wrote a routine for integration over a collection of parallelograms. For this purpose we implemented a new subroutine for the approximation of an integral with an error estimator, using the 37-point cubature formula of degree 13 constructed by Rabinowitz and Richter [Rabinowitz, Richter 69]. This cubature formula has 5 points on each coordinate axis. This cubature formula and error estimator are incorporated in Cubpack++. We compared our results with DCUHRE using the same tests as Berntsen, Espelid and Genz used [Berntsen, Espelid, Genz 91a] [Berntsen, Espelid, Genz 91b] [Berntsen, Espelid, Genz 88]. This test is based on the test-families listed in [Tab. 1]. Each test-family has a particular kind of difficulty (attribute). The parameters β_1, β_2 are picked randomly from $[0,1]$. For families 2 and 3, these are scaled according to

$$\beta_i \leftarrow \frac{1}{2} + (\frac{1}{2} - \frac{1}{d_j})\beta_i \text{ for } i = 1,2 \text{ and } j = 2,3.$$

For family 7, γ_{i1} and γ_{i2} are picked randomly from $[0,1]$ and are used to shift the place of the difficulty of the integrand. The parameters α_1, α_2 are picked randomly from $[0,1]$ and then scaled according to

$$2^{e_j}(\alpha_1 + \alpha_2) = d_j.$$

In our experiments we used

$$\mathbf{d} = (2.7, 2.5, 2.5, 300, 200, 300, 300, 200, 200, 15)$$

$$\mathbf{e} = (0, 0, 0, 2, 1, 1.5, 1.5, 2, 2, 0)$$

which are the same numbers as used for DCUHRE. These numbers determine how difficult a problem is. For each family 100 random choices are made. Our error estimator was tuned such that its reliability is as good as for DCUHRE. In this context a routine is called reliable if

$$\textit{real absolute error} \leq \textit{estimated absolute error} \leq \textit{requested absolute error}.$$

The subdivision strategy we use is presented in [Alg. 2] and [Fig. 4]. Note that for $\alpha = 0$ a 4-division is obtained, while for $\alpha = 1$ a traditional 2-division is obtained.

Table 1: Families of integrands for integration over $[0,1]^2$.

Test-families	Attributes
$f_1(x,y) = (\beta_1 x + \beta_2 y)^{\frac{-2}{d_1}}$	Corner singularity
$f_2(x,y) = \begin{cases} 0 \text{ if } x < \beta_1 - d_2^{-1} \text{ or } x > \beta_1 + d_2^{-1} \\ \quad \text{or } y < \beta_2 - d_2^{-1} \text{ or } y > \beta_2 + d_2^{-1} \\ 1 \text{ otherwise} \end{cases}$	Discontinuous rectangle
$f_3(x,y) = \begin{cases} 0 \text{ if } \sqrt{(x-\beta_1)^2 + (y-\beta_2)^2} > \frac{1}{d_3} \\ 1 \text{ otherwise} \end{cases}$	Discontinuous sphere
$f_4(x,y) = \exp(-\alpha_1 \lvert x-\beta_1 \rvert - \alpha_2 \lvert y-\beta_2 \rvert)$	C_0 function
$f_5(x,y) = \exp(-\alpha_1^2 (x-\beta_1)^2 - \alpha_2^2 (y-\beta_2)^2)$	Gaussian
$f_6(x,y) = \left((\alpha_1^{-2} + (x-\beta_1)^2)(\alpha_2^{-2} + (y-\beta_2)^2)\right)^{-1}$	Inner product peak
$f_7(x,y) = \sum_{i=1}^{2} \left((\alpha_1^{-2} + (x-\gamma_{i1})^2)(\alpha_2^{-2} + (y-\gamma_{i2})^2)\right)^{-1}$	2 inner product peaks
$f_8(x,y) = (1 + \alpha_1 x + \alpha_2 y)^{-3}$	Corner peak
$f_9(x,y) = \left((d_9^{-2} + x^2)(1 + (y-\beta_2)^2)\right)^{-1}$	Peak at x=0
$f_{10}(x,y) = \cos(2\pi\beta_1 + \alpha_1 x + \alpha_2 y)$	Oscillatory

Algorithm 2.

Compute the fourth divided differences D_x and D_y
case

$\max(D_x, D_y) < \varepsilon$:	divide in 4
$D_y < \alpha D_x$:	divide in 2 by halving in the x-direction
$D_x < \alpha D_y$:	divide in 2 by halving in the y-direction
$\frac{1}{\alpha} \geq \frac{D_x}{D_y} \geq \alpha$:	divide in 4

end case

We have run the subdivision strategy for a number of values $\alpha \in [0,1]$ and for accuracy requests $10^{-1}, \ldots, 10^{-5}$. With the results we made graphs that represent the number of function evaluations used as a function of the parameter α. Some typical graphs are presented in section 2.2 for the simplex. In this way we could easily see for the different function families and error requests for which α we obtained a minimum of function evaluations. The best overall value for α is not the same for every family and error request, and the same holds for ε. There is not one best solution, but there are several good ones.

We decided to choose $\alpha = 0.55$ and $\varepsilon = 10^{-4}$.

In [Tab. 2] we list for each test-family and several requested relative accuracies ε_{rel} the average number of function evaluations. DCUHRE allows a user to choose between 3 integration rules: a rule of degree $d = 7$, 9 or 13. For our routine, we list the results for $\alpha = 0.55$. These results show that our routine in

Table 2: Comparison between Cubpack++ $\alpha = 0.55$ and DCUHRE.

Test-family 2

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	3040	2500	3909	3319
10^{-2}	7758	8832	48703	9019

Test-family 3

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	10737	7045	6696	4306
10^{-2}	117249	78805	95843	49202

Test-family 1

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-2}	517	576	2464	364
10^{-3}	986	1276	3854	844
10^{-4}	1509	2004	5177	1232
10^{-5}	2491	2722	6528	1620
10^{-8}	18396	8664	10791	4082

Test-family 4

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	1733	1843	2076	1553
10^{-2}	3677	4070	5119	3421
10^{-3}	6872	7832	12444	6446
10^{-4}	11841	13504	32752	11249
10^{-5}	20602	21620	90348	19988

Test-family 5

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	1010	1291	1777	996
10^{-2}	1588	2033	2528	1412
10^{-3}	2417	3205	3343	2000
10^{-4}	3924	4906	4345	2908
10^{-5}	6631	7491	5541	4282
10^{-8}	44179	23414	11381	13144

Test-family 6

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	2135	2215	2207	1688
10^{-2}	3038	4212	3887	2550
10^{-3}	4184	6777	6052	3502
10^{-4}	6771	9513	8763	4675
10^{-5}	12685	12587	12001	6357
10^{-8}	110344	39872	24664	16227

Test-family 7

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	3509	3466	3246	2395
10^{-2}	5448	7259	6332	4067
10^{-3}	7821	12256	10624	6041
10^{-4}	12829	17328	16136	8465
10^{-5}	23487	22927	23081	11619
10^{-8}	197204	75194	52133	30069

Test-family 8

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	493	592	625	479
10^{-2}	627	853	1097	598
10^{-3}	787	1044	1475	671
10^{-4}	1445	1213	1762	748
10^{-5}	2958	1739	2025	1272
10^{-8}	28664	7205	3308	3031

Test-family 9

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	421	564	715	481
10^{-2}	537	669	1235	481
10^{-3}	732	1048	1495	481
10^{-4}	853	1795	1632	617
10^{-5}	1310	2190	1799	882
10^{-8}	9966	4585	2697	2763

Test-family 10

	DCUHRE			Cubpack++
ε_{rel}	$d=7$	$d=9$	$d=13$	$\alpha=0.55$
10^{-1}	455	309	195	372
10^{-2}	992	453	201	515
10^{-3}	2150	757	261	681
10^{-4}	4701	1239	456	941
10^{-5}	10078	2034	621	1310
10^{-8}	101663	10164	1691	3244

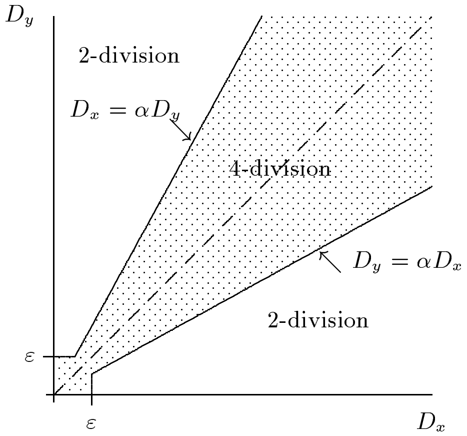

Figure 4: the new subdivision strategy

almost all cases is more efficient than DCUHRE.

Extending this subdivision strategy to $n > 2$ is straightforward. One computes n divided differences, $D_i, i = 1, \ldots, n$, re-using the function values on each coordinate axis. If they are all very small ($< \epsilon$) or "comparable" ($\frac{1}{\alpha} \geq \frac{D_i}{D_j} \geq \alpha, \forall i, j$), then the cube is divided in 2^n congruent parts. If one of them is significantly larger than all others, then the cube is divided in 2 parts. If two of them are comparable and both significantly larger than all others, then the cube is divided in 4. Et cetera. Preliminary results show this is a very promising subdivision strategy.

2.2 Subdivision strategies for the simplex

In all published global adaptive routines available for integration over the n-simplex (TRIADA [Haegemans 77], CUBTRI [Laurie 82], DCUTRI [Berntsen, Espelid 92] , DCUTET [Berntsen, Cools, Espelid 93]) the region processor starts with dividing the target region in 2^n parts of equal volume. Alan Genz [Genz 91] suggested dividing a simplex in 2 equal parts, based on the direction of the largest fourth divided difference. This is computed using 5 integrand evaluations on lines through the centre parallel to the edges of the simplex. So, there is an extra cost involved. In [Espelid, Genz 92] a subdivision strategy for a triangle is suggested that chooses between 3 or 4 parts. Extending this 3-division to higher dimensions does not look promising [Genz, Cools 97]. We will only consider divisions in 2 or 2^n parts.

For simplices a 2^n-division and a 2-division will never get the same subdivision. This makes it difficult to compare both. We need a reasonable definition of corresponding division in order to make a comparison. Let us first look at the worst cases. We assume the user gave one region.

The worst thinkable scenario for a 2-division is that the final subdivision corresponds with that of a 2^n-division. In this case we say that two subdivisions correspond if they have an equal number of subregions of the same volume. The number of regions obtained after m steps with a 2^n-division $= R = 1 + 2^n m$ and $A = 1 + (2^n - 1)\, m$ of them are in the active region collection. Obtaining a corresponding subdivision with a 2-division requires m_2 steps:

$$A = 1 + m_2 = 1 + (2^n - 1)\, m \Rightarrow m_2 = m\, (2^n - 1)\,.$$

Hence, the integral estimation is done for $R_2 = 1 + 2m\, (2^n - 1)$ regions. A 2-division requires some extra function evaluations for the divided differences. These are done after a region is selected for further subdivision. Hence it is not sufficient to compare R and R_2. We define the fraction ρ as the number of extra function evaluations for the differences over the number of function evaluations used for the integration of the two new subregions. Now we can compare the number of function evaluations used by both strategies.

$$\begin{aligned}\frac{N_2}{N} &= \frac{1 + 2m\,(2^n - 1)\,(1+\rho)}{1 + 2^n m} \\ &\approx \frac{2m\,(2^n - 1)}{2^n m}\,(1+\rho) = 1 + \rho + (1 - 2^{1-n})\,(1+\rho)\,.\end{aligned}$$

E.g. for 2 dimensions it follows from the above equation that a 2-division requires more than 50% more work than a 4-division and the additional cost to choose the direction of the subdivision cannot be ignored.

The worst thinkable scenario for a 2^n division is that the ideal subdivision corresponds to always dividing the same edge, e.g., due to a serious problem along one of the faces. The ideal 2-division computes for $R_2 = 1 + 2m$ regions integral estimates of which $A_2 = 1 + m$ remain in the region collection. Analogously to this situation for the n-cube, what we now call the corresponding 2^n-division will divide in each step all subregions that touch the face causing the difficulty. It computes integral estimates for R regions

$$\begin{aligned}R &= 1 + 2^n + 2^n \sum_{i=1}^{m-1} \left(2^{n\cdot i} - 1\right) \\ &= 1 + 2^n \sum_{i=0}^{m-1} 2^{n\cdot i} - 2^n\,(m-1) \\ &= 1 + 2^n \frac{2^{n\cdot m} - 1}{2^n - 1} - 2^n\,(m-1)\,.\end{aligned}$$

Therefore,

$$\frac{R}{R_2} \approx \frac{2^{n\cdot m} - 2^n m}{2m} \Rightarrow \frac{N}{N_2} \approx \frac{2^{n\cdot m-1} - 2^{n-1} m}{m \cdot (1+\rho)}.$$

We see that the worst case for a 2^n-division is much worse than for a 2-division, just as for the cube.

The criterion introduced in [Genz 91] for deciding which edge to cut is based on fourth order divided differences, centered at the centroid of the selected simplex (with the largest current error estimate). Denote the vertices of this simplex with $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n$. The edge directions are given by $\mathbf{d}_{ij} = \mathbf{v}_j - \mathbf{v}_i$,

$\forall\, 0 \leq i < j \leq n$. Now we define $f_{ij}(\gamma) = f(\mathbf{c} + \gamma \cdot \mathbf{d}_{ij}/(5(n+1)))$ with $\mathbf{c} = \sum_{i=0}^{n} \mathbf{v}_i/(n+1)$. A fourth divided difference operator is given by:

$$D_{ij} = \|\mathbf{d}_{ij}\|_1 \cdot |6f_{ij}(0) - 4(f_{ij}(-2) + f_{ij}(2)) + (f_{ij}(-4) + f_{ij}(4))|\,.$$

We divide the edge with direction $\mathbf{d}_{lm}$ for which $D_{lm} = \max_{i<j} D_{ij}$. The multiplication with the 1-norm of the direction vector is done in order to provide a mechanism to prevent too elongated simplices. The differences are rescaled (division by $\|\mathbf{d}_{lm}\|_1$) to obtain a less size dependent measure.

The subdivision strategy we use for 2 or 4-division of a triangle is presented in [Alg. 3]. We have done the same type of experiments as for the square. Our experiments showed it is better to use D_2/D_0 than D_1/D_0. The edges with the two biggest differences are the ones for which the direction vector is the 'most equal to each other', so in a way they are not independent enough. The results for the 1-norm were slightly better than those with the 2-norm.

Algorithm 3.

```
Compute the 4th divided differences D_0, D_1, D_2 (rescaled)
Relabel them such that D_0 ≥ D_1 ≥ D_2
case
        D_0 < ϵ:        divide in 4
        D_2/D_0 < α:    divide in 2 by halving the side belonging to D_0
        D_2/D_0 ≥ α:    divide in 4
end case
```

Suppose one tries to integrate a function with a singularity on one of the sides of the triangle. In such a situation, the subdivision strategy will always cut the same direction in 2 equal parts, because here the factor $\|\mathbf{d}_{lm}\|_1$ will not be enough to prevent a degenerate triangle. Hence, special action is needed to prevent the generation of degenerate triangles.

There are two types of degenerate triangles [see Fig. 5]. In the first case two

Figure 5: 2 degenerate triangles

angles are almost equal to $\pi/2$ and the edge opposite to the very small angle is excluded for subdivision. The second case has one angle almost equal to π and the edge in opposite position has to be bisected. Obviously this involves a heuristic tolerance that depends on the accuracy of the floating point numbers used.

We compared our results with DCUTRI using the same tests as in [Berntsen, Espelid 92] . This test is based on TRITST [Berntsen 89] and uses the test families described in [Tab. 3]. Note that for these tests we use the same local cubature formula as is used in DCUTRI, so the subdivision strategy is basically the only part of the implementation that differs. If $\alpha = 0$ the subdivision strategy is the same as for DCUTRI. The only difference then is the error estimator because Cubpack++ uses an extra heuristic. For this we do not expect to get exactly the same results for DCUTRI and Cubpack++ with $\alpha = 0$. Other reasons to expect different results for $\alpha = 0$ is the difference in programming language (FORTRAN vs. C++) and the way the region collection is used. For test families 1, 2, 3 and 7 the integral of these functions over the unit triangle $\{(0,0),(0,1),(1,0)\}$ is approximated. For test families 4, 5 and 6 a second triangle $\{(1,1),(0,1),(1,0)\}$ is used to get an approximation over the unit square. We treat these two triangles as a region collection. Berntsen [Berntsen 89] approximated each triangle separately and added the approximations afterwards. For family 4, we simulated his approach, which is not really globally adaptive. The parameters β_1, β_2 are picked randomly:

for test families 1, 3, 4, 5, 6 and 7, β_1 from [0,1]
for test family 2, β_1 from $[d_2, 1-(1+\sqrt{2})d_2]$

for test family 2, β_2 from $[d_2, 1-\beta_1-\sqrt{2}d_2]$
for test family 3, β_2 from $[0, 1-\beta_1]$
for test families 4 and 6, β_2 from $[0,1]$

The parameters α_1 and α_2 are first picked from $[0,1]$ and then scaled according to

$$\alpha_1 + \alpha_2 = d_j, \quad j = 3,4,5,6 \text{ and } 7.$$

We used the same difficulty parameters as Espelid and Genz:

$$\mathbf{d} = (-0.9, 0.25, 75, 100, 150, 100, 30).$$

Table 3: Families of integrands for integration over triangles.

Test-families	Attributes
$f_1(x,y) = (\lvert x-\beta_1\rvert + y)^{d_1}$	Singularity on x-axis
$f_2(x,y) = \begin{cases} 1 \text{ if } \sqrt{(x-\beta_1)^2 + (y-\beta_2)^2} < d_2 \\ 0 \text{ otherwise} \end{cases}$	Discontinuous sphere
$f_3(x,y) = \exp(-\alpha_1\lvert x-\beta_1\rvert - \alpha_2\lvert y-\beta_2\rvert)$	C_0 function
$f_4(x,y) = \exp\left(-\alpha_1^2(x-\beta_1)^2 - \alpha_2^2(y-\beta_2)^2\right)$	Gaussian
$f_5(x,y) = \left(\alpha_1^{-2} + (x-\beta_1)^2\right)^{-1}\left(\alpha_2^{-2} + y^2\right)^{-1}$	Peak on x-axis
$f_6(x,y) = \left(\alpha_1^{-2} + (x-\beta_1)^2\right)^{-1}\left(\alpha_2^{-2} + (y-\beta_2)^2\right)^{-1}$	Internal peak
$f_7(x,y) = \cos(2\pi\beta_1 + \alpha_1 x + \alpha_2 y)$	Oscillatory

After a lot of experiments, similar to those described in the previous section, we decided to choose $\alpha = 0.45$ and $\epsilon = 10^{-3}$. Some typical graphs are shown in [Fig. 6], [Fig. 7], [Fig. 8], and [Fig. 9]. In [Tab. 4] we list for each test-family and several requested relative accuracies ϵ_{rel} the average number of function evaluations used by DCUTRI and Cubpack++ with this new subdivision strategy for the triangle. We took 500 random samples for each family and the maximum allowed number of function evaluations was 10^5.

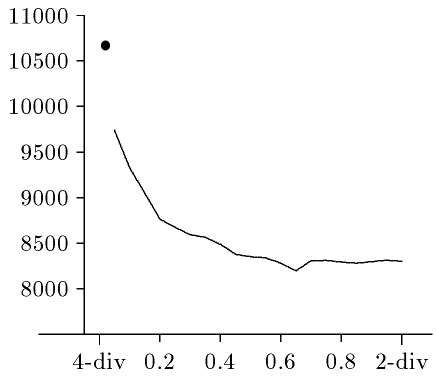

Figure 6: Family 5, $\epsilon_{rel} = 10^{-3}$, Peak on x-axis

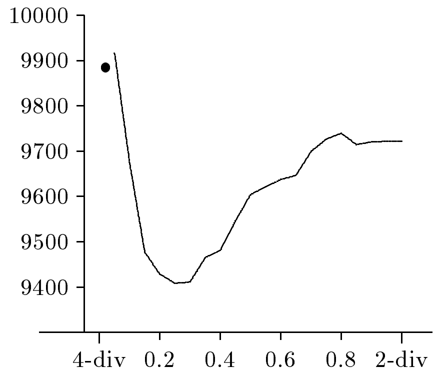

Figure 7: Family 6, $\epsilon_{rel} = 10^{-3}$, Internal peak function

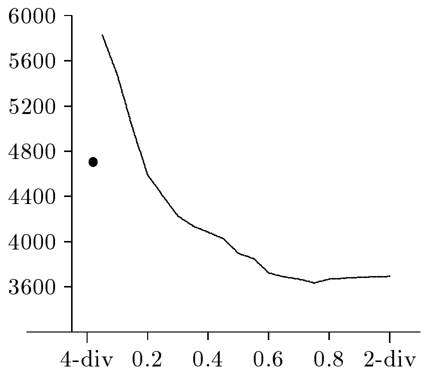

Figure 8: Family 1, $\epsilon_{rel} = 10^{-3}$, Singularity on x-axis

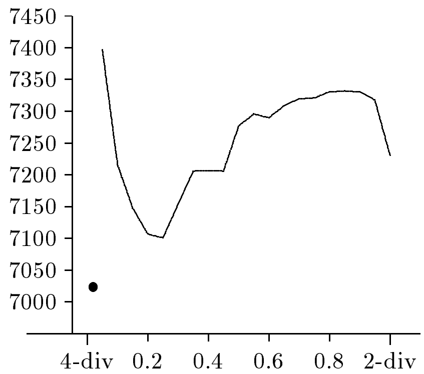

Figure 9: Family 4, $\epsilon_{rel} = 10^{-5}$, Gaussian function

The average cost is a continuous function of α. For $\alpha = 0$ we don't need to calculate the divided differences as they are not needed nor used. The average number of function evaluations is represented with a point for the normal 4-division. So, for α almost equal to 0, the gain in number of function evaluations by also allowing some 2-divisions is not large enough to compensate the overhead due to the calculation of the divided differences. A problem-dependent portion of 2-divisions is needed to do better than always using the 4-division (without calculation of the divided differences). In [Fig. 6] this is invisible, in [Fig. 7] this happens with $\alpha \simeq 0.1$, in [Fig. 8] this happens with $\alpha \simeq 0.2$, and in [Fig. 9] this never happens. These results show that our routine is in almost all cases more efficient than DCUTRI.

Table 4: Comparison between Cubpack++ and DCUTRI.

Test-family 1

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	442	377	373	349
10^{-2}	1291	1269	1186	1259
10^{-2}	4680	4652	3694	4026
10^{-2}	17271	17270	10133	11223
10^{-2}	48816	48840	25029	27714

Test-family 2

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	16243	6636	5523	6264
10^{-2}	99937	80823	45555	48668

Test-family 3

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	3248	3081	1960	1976
10^{-2}	11963	11889	5625	5779
10^{-3}	37069	37053	15708	16393
10^{-4}	79212	79193	41165	43332
10^{-5}	97546	97350	86387	88539

Test-family 4

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	2371	1362	1392	1396
10^{-2}	3324	2087	2195	2208
10^{-3}	4573	3128	3404	3394
10^{-4}	6571	4803	5017	5010
10^{-5}	9223	7014	7231	7206

Test-family 5

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	3681	1486	1845	1782
10^{-2}	6891	5325	4609	4714
10^{-3}	12136	10605	8301	8379
10^{-4}	19525	17367	12326	12389
10^{-5}	29726	26983	17624	17689

Test-family 6

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	4036	2537	2660	2644
10^{-2}	6922	6048	6281	6219
10^{-3}	10820	9871	9722	9547
10^{-4}	16122	14792	13615	13419
10^{-5}	23063	21437	18296	18126

Test-family 7

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	486	481	384	385
10^{-2}	757	756	534	541
10^{-3}	1083	1082	692	706
10^{-4}	1419	1419	864	882
10^{-5}	1736	1736	1117	1145

Test-family 4, two triangles apart

ε_{rel}	DCUTRI	Cubpack++ $\alpha = 0$	$\alpha = 1$	$\alpha = 0.45$
10^{-1}	2371	2080	2179	2185
10^{-2}	3324	3086	3314	3328
10^{-3}	4573	4355	4750	4747
10^{-4}	6571	6346	6723	6715
10^{-5}	9223	9003	9376	9346

Generalising this subdivision strategy to higher dimensions is much more difficult for the simplex than for the n-cube. To begin, we have $\frac{n(n+1)}{2}$ possible subdivision directions, instead of n. Consequently, if we do not want to restrict the region processor to choose only a 2-division or a 2^n-division, but allow a 2^i-division for $1 \le i \le n$, the heuristics involved in this choice will become complicated and expensive.

3 Conclusion

We described a new subdivision strategy for subregion-adaptive integration routines used to approximate integrals over squares and triangles, that is more adaptive than what was implemented prior to Cubpack++. It is illustrated that this new strategy is often more efficient.

We suggested how this strategy can be extended to higher dimensions and for the n-cube we believe this will increase the efficiency of adaptive integration routines.

References

[Berntsen 89] Berntsen, J.: "TRITST: A Subroutine for Evaluating the Performance of Subroutines for Automatic Integration over Triangles"; Reports in Informatics 34, Dept. of Inf., University of Bergen, Norway, 1989.

[Berntsen, Cools, Espelid 93] Berntsen, J., Cools, R., Espelid, T.O.: "Algorithm 720: An algorithm for adaptive cubature over a collection of 3-dimensional simplices"; ACM Trans. Math. Software, 19 (1993), 320–332.

[Berntsen, Espelid 92] Berntsen, J., Espelid, T.O.: "Algorithm 706: DCUTRI: An algorithm for adaptive cubature over a collection of triangles"; ACM. Trans. Math. Software, 18 (1992), 329–342.

[Berntsen, Espelid, Genz 88] Berntsen, J., Espelid, T.O., Genz, A.: "A Test of ADMINT"; Reports in Informatics 31, Dept. of Inf., Univ. of Bergen (1988).

[Berntsen, Espelid, Genz 91a] Berntsen, J., Espelid, T.O., Genz, A.: "An adaptive algorithm for the approximate calculation of multiple integrals"; ACM Trans. Math. Software 17 (1991), 437–451.

[Berntsen, Espelid, Genz 91b] Berntsen, J., Espelid, T.O., Genz, A.: "Algorithm 698: DCUHRE – an adaptive multidimensional integration routine for a vector of integrals"; ACM Trans. Math. Software 17 (1991), 452–456.

[Berntsen, Espelid, Sørevik 91] Berntsen, J., Espelid, T.O., Sørevik, T.: "On the subdivision strategy in adaptive quadrature algorithms"; J. Comput. Appl. Math. 35 (1991), 119–132.

[Cools, Haegemans 92] Cools, R., Haegemans, A.: "CUBPACK: Progress report"; Numerical Integration – Recent Developments, Software and Applications (T.O. Espelid and A. Genz, eds.), NATO ASI Series C: Math. and Phys. Sciences, vol. 357, Kluwer Academic Publishers, Dordrecht (1992), 305–315.

[Cools, Laurie, Pluym 97a] Cools, R., Laurie, D., Pluym, L.: "Algorithm 764: Cubpack++: A C++ package for automatic two-dimensional cubature"; ACM Trans. Math. Software 23, 1 (1997), 1–15.

[Cools, Laurie, Pluym 97b] Cools, R., Laurie, D., Pluym, L.: "A user manual for Cubpack++ version 1.1"; Report TW255, Dept. of Computer Science, K.U.Leuven, Belgium (March 1997).

[Espelid, Genz 92] Espelid, T.O., Genz, A.: "On the subdivision strategy in adaptive cubature algorithms for triangular regions"; Reports in Informatics 74, University of Bergen, Norway (December 1992).

[Genz 91] Genz, A.: "An adaptive numerical integration algorithm for simplices"; In N.A. Sherwani, E. de Doncker, and J.A. Kapenga, editors, Computing in the 90s – Proceedings of the First Great Lakes Computer Science Conference, Lecture Notes in Computer Science, vol. 507, Springer-Verlag, New York (1991), 279–292.

[Genz, Cools 97] Genz, A. and Cools, R.: "An adaptive numerical cubature algorithm for simplices"; Report TW273, Dept. of Computer Science, K.U.Leuven, Belgium (December 1997).

[Genz, Malik 80] Genz, A.C., Malik, A.A.: "An adaptive algorithm for numerical integration over an N-dimensional rectangular region"; J. Comput. Appl. Math. 6 (1980), 295–302.

[Haegemans 77] Haegemans, A.: "Algorithm 34: an algorithm for the automatic integration over a triangle"; Computing, 19 (1977), 179–187.

[Hanke 82] Hanke, W.: "Die optimale Sektion bei adaptiven Integrationsverfahren mit globaler Strategie"; ZAMM 62 (1982), T327–T329.

[Laurie 82] Laurie D.P.: "CUBTRI – automatic cubature over a triangle"; ACM Trans. Math. Software, 8 (1982), 210–218.

[Rabinowitz, Richter 69] Rabinowitz, P., Richter, N.: "Perfectly symmetric two-dimensional integration formulas with minimal number of points"; Math. Comp. 23 (1969), 765–799.

[De Ridder, Van Dooren 74] De Ridder, L., Van Dooren, P.: "Automatische meerdimensionale numerieke integratie", Master's thesis, K.U. Leuven (1974).

[Van Dooren, De Ridder 76] Van Dooren, P., De Ridder, L.: "An adaptive algorithm for numerical integration over an n-dimensional cube"; J. Comput. Appl. Math. 2 (1976), 207–217.