DECUHR: An Algorithm for Automatic Integration of Singular Functions over a Hyperrectangular Region. *

Terje O. Espelid †
Department of Informatics
University of Bergen
Høyteknologisenteret, N-5020 Bergen, Norway

Alan Genz
Department of Pure and Applied mathematics
Washington State University
Pullman WA 99164-3113

Abstract

We describe an automatic cubature algorithm for functions that have a singularity on the surface of the integration region. The algorithm combines an adaptive subdivision strategy with extrapolation. The extrapolation uses a non-uniform subdivision that can be directly incorporated into the subdivision strategy used for the adaptive algorithm. The algorithm is designed to integrate a vector function over an n-dimensional rectangular region and a FORTRAN implementation is included.
Keywords: automatic, adaptive, cubature, singularity, extrapolation, stability.

1 Introduction

There are only a few algorithms available designed specifically for automatic integration of singular integrands in higher dimensions. Implementations of three such algorithms have been given by de Doncker and Robinson [6] (TRIEX), Carino, de Doncker and Robinson [5] (TRILEV) and Carino [4] (ADLEV). These three routines are 2-dimensional and have the triangle as the integration region. They are all automatic, and ADLEV and TRIEX are also adaptive. They all use a *uniform* subdivision of the triangle combined with extrapolation. TRIEX applies the geometric subdivision sequence and is based on the ϵ-algorithm. TRILEV and ADLEV both use the harmonic subdivision sequence combined with the Levin u-transform [9] and the d-transform [10] respectively.

*Published by: *Numerical Algorithms 8(1994)201-220*
†Supported by the Norwegian Research Council for Science and the Humanities

In two recent papers Espelid [7, 8] described a new method for extrapolating a sequence of estimates produced by a *non-uniform* subdivision of the initial integration region. In [8] the technique is applied to problems with vertex singularities and internal point singularities. The integration region can be a hyperrectangle, simplex or sphere. In [7] the technique is generalized for vertex, line, face and more general subregion singularities. To achieve this generalization one has to restrict the region of integration to a hyperrectangle.

In this paper we will discuss an implementation of the new approach given in [7]. The paper is organized as follows: in the next section we describe the problem and the basic error expansion from [7]. Then we describe the basic algorithm and discuss some design questions for the algorithm and the numerical stability of the algorithm. Then we present some examples and give some concluding remarks. In Section 7 we briefly describe the code, and present the list of parameters in the Appendix.

2 The Basic Error Expansion

A function $f(\mathbf{x})$, where $\mathbf{x} \in R^n$, is said to be *homogeneous* of degree α (about the origin) if

$$f(\lambda \mathbf{x}) = \lambda^\alpha f(\mathbf{x}), \quad \text{for all } \lambda > 0 \text{ and for all } \mathbf{x} \not\equiv \mathbf{0}.$$

We will use the notation introduced by Lyness [11], and denote such a function by $f_\alpha(\mathbf{x})$. Homogeneous functions satisfy two simple rules: $f_\alpha f_\beta$ is of homogeneous of degree $\alpha + \beta$ and $(f_\alpha)^\beta$ is of homogeneous of degree $\alpha\beta$.

We consider n-dimensional integration problems where the singularity, which involves exactly s of the n variables, is caused by a function homogeneous about the origin. In order to simplify the presentation, we assume that the variables are numbered so that these s variables are $x_1, x_2, \ldots, x_s$. Because an affine transformation from the given rectangular region on to the unit n-cube does not changes the degree of a homogeneous function and does not change the degree of any given cubature rule, we further assume that the integration is the n-dimensional unit hypercube, C_n. We therefore define

$$I(f) = \int_{C_n} f(\mathbf{x}) d\mathbf{x} = \int_0^1 \int_0^1 \cdots \int_0^1 f(x_1, x_2, \ldots, x_n)\, dx_1\, dx_2 \ldots dx_n. \tag{1}$$

We will describe how to compute numerical estimates to integration problems of type (1), where the function involved, f, is a product of a homogeneous function $f_\alpha(x_1, x_2, \ldots, x_s)$ and a function $g(\mathbf{x})$ which is regular in C_n. The function $f_\alpha(x_1, x_2, \ldots, x_s)$ may have a singularity at the origin of C_s, from the homogeneity. Now define $\mathbf{x}_0 = (0, 0, \ldots, 0, x_{s+1}, x_{s+2}, \ldots, x_n)$, so that $f(\mathbf{x}) = f_\alpha(x_1, x_2, \ldots, x_s) g(\mathbf{x})$ with $\alpha > -s$ and $g(\mathbf{x}_0) \not\equiv 0$ for $\mathbf{x}_0 \in C_n$. We also assume that the origin $(x_1, \ldots, x_s) = \mathbf{0}_s$ is the only point in C_s where f_α is non-analytic.

We begin the nonuniform subdivision by cutting C_n into $s + 1$ new subregions. First, we divide the region in half by cutting at $x_1 = 0.5$. We then cut in half the first of the two halves at $x_2 = 0.5$ and continue until we have one region containing the singularity and s regions where

the function is assumed well-behaved. Considering these s subdivisions as one stage, we may do j such stages. The hyper-rectangle containing the singularity is now $H_n^{(s)}(h) = [0,h]^s \times [0,1]^{n-s}$, with $h = 1/2^j$. Now define

$$I_{H_n^{(s)}(h)}(f) = \int_0^1 \cdots \int_0^1 (\int_0^h \cdots \int_0^h f(x_1, x_2, \ldots, x_n)\, dx_1 \ldots dx_s) dx_{s+1} \ldots dx_n, \tag{2}$$

and suppose that both $\mathbf{x}_0$ and $\mathbf{x}$ are points in C_n. Expand $g(\mathbf{x})$ in a Taylor series with respect to the coordinates $x_1, \ldots, x_s$ around $\mathbf{0}_s$ with p basic terms and a remainder term, multiply this expression by $f_\alpha(x_1, x_2, \ldots, x_s)$ and integrate over $H_n^{(s)}(h)$. Changing integration variables $x_i = hv_i,\ i = 1, 2, \ldots, s$, in each integration term moves all integrals to C_n. The result is

$$I_{H_n^{(s)}(h)}(f_\alpha g) = c_0\, h^{\alpha+s} + \sum_{\ell=1}^{p-1} c_\ell\, h^{\alpha+s+\ell} + O(h^{\alpha+s+p}), \tag{3}$$

with

$$\begin{cases} c_0 = \int_{C_n} f_\alpha(x_1, x_2, \ldots, x_s) g(\mathbf{x}_0) d\mathbf{x}, \\ c_\ell = \frac{1}{\ell!} \int_{C_n} f_\alpha(x_1, x_2, \ldots, x_s) \sum_{i_1,\ldots,i_\ell=1}^{s} x_{i_1} \cdots x_{i_\ell}\, \frac{\partial^\ell g}{\partial x_{i_1} \ldots \partial x_{i_\ell}}(\mathbf{x}_0)\, d\mathbf{x}, \\ \ell = 1, 2, \ldots p-1. \end{cases}$$

Now we use an L point integration rule Q_H for a hyperrectangle H,

$$Q_H(f) = \sum_{i=1}^{L} w_i\, f(\mathbf{x}_i),$$

with $\sum_i w_i$ equal to the volume of hyperrectangle H. If we apply this rule to f over $H_n^{(s)}(h)$ we obtain

$$Q_{H_n^{(s)}(h)}(f_\alpha g) = b_0\, h^{\alpha+s} + \sum_{\ell=1}^{p-1} b_\ell\, h^{\alpha+s+\ell} + O(h^{\alpha+s+p}), \tag{4}$$

with $b_0 = Q_{C_n}(f_\alpha(x_1, x_2, \ldots, x_s) g(\mathbf{x}_0))$ and

$$b_\ell = \frac{1}{\ell!} Q_{C_n}(f_\alpha(x_1, x_2, \ldots, x_s) \sum_{i_1,\ldots,i_\ell=1}^{s} x_{i_1} \cdots x_{i_\ell}\, \frac{\partial^\ell g}{\partial x_{i_1} \ldots \partial x_{i_\ell}}(\mathbf{x}_0)), \ell = 1, 2, \ldots p-1.$$

Subtracting (3) from (4), we finally obtain the error expansion

$$E_n^{(s)}(h) = Q_{H_n^{(s)}(h)}(f_\alpha g) - I_{H_n^{(s)}(h)}(f_\alpha g) = \sum_{\ell=0}^{p-1} a_\ell\, h^{\alpha+s+\ell} + O(h^{\alpha+s+p}), \tag{5}$$

where $a_\ell = b_\ell - c_\ell$ for $\ell = 0, 1, \ldots, p-1$ and b_ℓ is the numerical estimate produced by the rule Q of the integral for c_ℓ. Because the functions involved should become smoother with increasing

values of ℓ, it is reasonable to expect $|a_\ell| >> |a_{\ell+1}|$, at least as long as $\alpha + s + \ell \leq \text{degree}(Q)$, the degree of the integration rule Q.

The extrapolation scheme is based on the error expansion (5). In fact, we assume that the error given in (5) will be the major error source in the estimate of $I(f)$. The error contribution from the rest of the original region C_n also has to be kept under control, and we will now consider how it will influence the extrapolation process and the final global estimate of $I(f)$.

Suppose we use a standard globally adaptive subdivision strategy which at each stage subdivides the subregion for the most difficult part of the integration problem (1). Each time the subregion containing the singularity needs subdivision we divide this into $s+1$ new subregions following the previously described procedure. Thus, at any time, the collection of subregions will contain only one hyperrectangle containing the singularity, namely $H_n^{(s)}(h)$. Assume that at a given time we have already subdivided this hyperrectangle k times. Now, using $h_i = 1/2^i$, define

$$\begin{aligned} &I_i = I_{H_n^{(s)}(h_i)}(f_\alpha g),\ i = 0, 1, 2, \ldots, k, \\ &U_i = I_{i-1} - I_i,\ i = 1, 2, \ldots, k, \\ &S_k = \textstyle\sum_{i=1}^k U_i \text{ and } \hat{S}_k = \sum_{i=1}^k \hat{U}_i, \end{aligned} \tag{6}$$

with $\hat{U}_i$ an approximation to U_i. The elements in the sequence $U_1, U_2, \ldots$ are by definition (6) integrals over a sequence of subregions, $L_n^{(s)}(h) = [h, 2h]^s \times [0, 1]^{n-s}$ for a given value of h. We illustrate, in Figure 1, two subregion sequences in 2 dimensions:

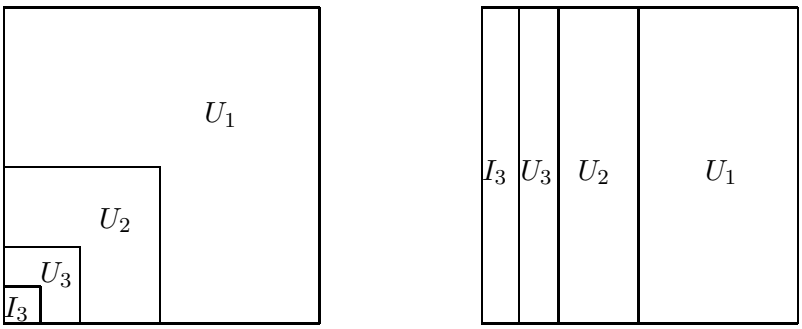

Figure 1. 2-D (a)corner and (b)line singularity U and I-sequences.

Using the definition of U_i we get $S_k = I_0 - I_k$. Thus S_k is an approximation to I_0 with error $-I_k$. We intend to use the expansion for $I_{H_n^{(s)}(h)}$ with $h = h_k = 1/2^k$ given in (3) with extrapolation to approximate I_k, and use standard numerical integration rules to approximate each U_i. We have already described a subdivision for each of the U_i subregions into s hyperrectangular pieces. We might need to adaptively subdivide some of these pieces further to achieve the necessary accuracy. First we consider the extrapolation.

If we use the rule Q to estimate $I_i = I_{H_n^{(s)}(h_i)}$, then $Q_i = Q_{H_n^{(s)}(h_i)}(f_\alpha g)$ is an estimate of the

tail $I_i = \sum_{\ell>i} U_\ell$. Now define

$$T_{i0} = Q_i + \sum_{\ell=1}^{i} \hat{U}_\ell, \text{ for } i = 0, 1, 2, \ldots, k.$$

T_{i0} is an approximation to I_0 and we have according to (5)

$$T_{i0} - I_0 = \sum_{\ell=1}^{i} E_{U_\ell} + \sum_{\ell=0}^{p-1} a_\ell h_i^{\alpha+s+\ell} + O(h_i^{\alpha+s+p}), \tag{7}$$

where $E_{U_\ell} = \hat{U}_\ell - U_\ell$. We use standard linear extrapolation to eliminate the first k terms in the sum of h-powers in (7). Let $n_1 = 2^{\alpha+s} - 1$ and $n_{j+1} = 2n_j + 1$ for $j > 0$ and define extrapolation table entries T_{ij} by

$$T_{ij} = T_{i,j-1} + (T_{i,j-1} - T_{i-1,j-1})/n_j, i = 1, 2, \ldots, k, \ j = 1, 2, 3, \ldots, i. \tag{8}$$

Now put the T_{ij}'s in an extrapolation table:

$$\begin{array}{llll} T_{00} & & & \\ T_{10} & T_{11} & & \\ \vdots & \vdots & \ddots & \\ T_{k0} & T_{k1} & \cdots & T_{kk} \end{array}$$

After j extrapolation steps in row k we have

$$T_{kj} = I_0 + \sum_{\ell=j}^{p-1} a_\ell^{(j)} h_k^{\alpha+s+\ell} + O(h_k^{\alpha+s+p}) + \sum_{\ell=1}^{k} (1 - \beta_{k+1-\ell,j}) E_{U_\ell}. \tag{9}$$

We refer the reader to [8] for more details about the $\beta-$coefficients. These coefficients can be computed and allow us to monitor the effect of the errors in the approximate $\hat{U}_\ell$ values. If at some stage one of these errors becomes large relative to the extrapolation error, then we can compute a better approximation in the associated subregion by further subdividing that subregion.

The described approach is based on the error expansion (7) using the h-exponent sequence to define the denominators n_j in (8). This approach may be made more general: suppose that the function f in (1) is a product of a homogeneous function f_α, a logarithmic function $\ln g_\beta$ and a function $g(\mathbf{x})$ which is regular in C_n. The functions $f_\alpha(x_1, x_2, \ldots, x_s)$ and $g_\beta(x_1, x_2, \ldots, x_s)$ are both assumed to have a point singularity at the origin $\mathbf{0}_s$ of C_s due to the homogeneous property.

$$f(\mathbf{x}) = f_\alpha(x_1, x_2, \ldots, x_s) \ (\ln g_\beta(x_1, x_2, \ldots, x_s)) \ g(\mathbf{x}), \text{with } \alpha > -s.$$

Furthermore we assume that the origin is the only point in C_s where f_α and g_β are non-analytic. Repeating the line of arguments that led to (7) we get a new error expansion

$$T_{i0} - I_0 = \sum_{\ell=1}^{i} E_{U_\ell} + \sum_{\ell=0}^{p-1} (\check{a}_\ell + \hat{a}_\ell \ln h_i)\, h_i^{\alpha+s+\ell} + O(h_i^{\alpha+s+p} \ln h_i), \tag{10}$$

We refer the reader to [7] for the $\check{a}_\ell$ and $\hat{a}_\ell$ expressions. The extrapolation scheme can now be based on the error expansion (10). Using linear extrapolation just as before: it is well known that considering each h-exponent to appear twice in (10) will remove both the constant and the $\ln h$ term in front of the $h^{\alpha+s+\ell}$. In the code we have included the option of having such a logarithmic factor.

3 The Basic Algorithm

At each stage in our algorithm we consider two different alternatives if we are not satisfied with the current global approximation. If we believe the error for some $\hat{U}_\ell$ is too large we can compute a better value for $\hat{U}_\ell$ and update the T-table, or if we believe the extrapolation error is too large, we can increase k, compute $\hat{U}_{k+1}$ and Q_{k+1} and create a new row in the T-table.

An algorithm combining adaptivity and extrapolation for this problem has as inputs: the dimension n, the corners of the region of integration, the dimension of the singularity s, the indices for those s variables responsible for the singularity (assumed to be $1, 2, 3, \ldots, s$ in this paper), the coordinates for one singular corner (assumed to be the origin in this paper), the degree of the singularity α (optional), the function f and finally an error tolerance ϵ. The output will normally be an estimate of the integral, $\hat{Q}$, and an error estimate $\hat{E}$. If possible the algorithm computes $\hat{Q}$ with $\hat{E} \leq \epsilon$ (here $\epsilon = \max(\epsilon_a, \epsilon_r|\hat{Q}|)$, where ϵ_a and ϵ_r are user specified absolute and relative error tolerances respectively). This does not ensure that $|I - \hat{Q}| \leq \epsilon$, but it does suggest that it is likely. Basic elements in this globally adaptive cubature algorithm are:

- A collection of hyperrectangles organized in a heap: at a certain stage we have M hyperrectangles in the heap, say $R_j,\ j = 1, ..., M$. We also have the hyperrectangle containing the singularity separate from the heap. (Keeping the singular region separate simplifies maintenance of the heap.)

- An local integration rule, Q, used to produce local estimates, $\hat{Q}_j$, for the integrals over each hyperrectangle R_j in the heap, and Q_i for the integrals I_i over the singular regions: our implementation provides at least two local rules of polynomial degree 7 and 9 in each dimension (see [1, 2]).

- A procedure to estimate the error in the local approximations over the regions in the heap produced by Q: this error estimate, $\hat{E}_j$, is constructed using several null rules (see [1, 2]). For the singular region, we use the pure extrapolation error as an error estimate.

- A strategy for picking the next hyperrectangle to be processed: in our algorithm this will be the hyperrectangle with the greatest error estimate. Therefore the heap is organized so that the hyperrectangle with the greatest error estimate is the root element. The error in the singular region is compared with greatest error from the subregion heap before proceeding with next subdivision. If we integrate a vector of integrands, then the error for a particular subregion is the maximum of the error estimates for all of the integrands.

- A strategy for how to divide such a region: we follow [1, 2] and divide a regular region into two halves by cutting the edges in the direction with greatest variation at the midpoint (measured by the fourth divided difference). A singular region has to be cut in $s + 1$ new parts: we cut the remaining singular region s times, sequentially, in two parts orthogonal to one of the remaining singular directions. In each step we choose to cut orthogonally to the direction having the greatest variation.

- An extrapolation procedure: we chose to use linear extrapolation and therefore α is needed to compute the denominators n_j. Alternatively, we can estimate α before the extrapolation starts.

- A global error estimate: we compute this using (9). Note that it is important to know, for each region in the heap, which U_ℓ it belongs to. Having computed the β-coefficients, the global error can be computed by adding the last sum in (9) (all terms in absolute values) to the estimate of the pure extrapolation error.

We now give the basic algorithm:

A Globally Adaptive Cubature Algorithm with Extrapolation

Initialize: Compute Q_0 and cut the given region in $s+1$ parts;
Compute Q_1, $\hat{Q}_\ell$, $\hat{E}_\ell$, $\ell = 1, 2, \ldots, s$,
$\hat{U}_1 = \sum_{\ell=1}^{s} \hat{Q}_\ell$ and $\hat{E}_{U_1} = \sum_{\ell=1}^{s} \hat{E}_\ell$;
Initialize the heap and set $M = s$;
Initialize extrapolation table with $T_{00} = Q_0$, $T_{10} = Q_1 + U_1$;
Set $k = 1$, extrapolate and estimate the pure extrapolation error;
Compute the global error estimate $\hat{E}_{T_{11}}$;

Control: **while** $\hat{E}_{T_{kk}} > \epsilon$ **do**
if the pure extrapolation error > the root of heap error **then**
Subdivide the singular region into $s+1$ parts;

Process regions: Compute: Q_{k+1}, $\hat{Q}_{M+j}$ and $\hat{E}_{M+j}$, $j = 1, 2, \ldots, s$,
$\hat{U}_{k+1} = \sum_{j=1}^{s} \hat{Q}_{M+j}$ and $\hat{E}_{U_{k+1}} = \sum_{j=1}^{s} \hat{E}_{M+j}$;
$T_{k+1,0} = T_{k,0} - Q_k + Q_{k+1} + \hat{U}_{k+1}$ and set $k = k+1$;
Extrapolate and estimate the pure extrapolation error;
Compute the global error $\hat{E}_{T_{kk}}$;

Update: Update the heap, replace the singular region and set $M = M + s$.
else
Delete the root region from the heap, say region R_i;
Subdivide this region into two new subregions;

Process regions: Compute for the two subregions: $\hat{Q}_i^{(\ell)}$ and $\hat{E}_i^{(\ell)}$, $\ell = 1, 2$;

Update: Update the heap, $\hat{U}_j$ and $\hat{E}_{U_j}$ associated with the region R_i;
Update the extrapolation table and the pure extrapolation error;
Compute the global error $\hat{E}_{T_{kk}}$ and set $M = M + 1$;
end if
end while

All of the previous analysis depends on an exact value for α, the degree of the singularity. For problems where α is unknown or difficult to determine, α could be estimated numerically, and this estimated value could be used in the extrapolation process. In our implementation of the algorithm we allow the user to flag situations where α is unknown. In these cases we estimate α by considering values of the integrand at points $\mathbf{x}_j = (2^{-j}, 2^{-j}, ..., 2^{-j})$, for $j \geq 0$. Letting $f_j = |f(\mathbf{x}_j)|$, and $\alpha_j = \log(f_j/f_{j+1})/\log(2)$, we have

$$\alpha_j = \log\left(\frac{2^{-j\alpha} f(\delta, ..., \delta) g(\mathbf{x}_j)}{2^{-(j+1)\alpha} f(\delta, ..., \delta) g(\mathbf{x}_{j+1})}\right)/\log(2) \approx \log(2^\alpha)/\log(2) = \alpha,$$

and the approximation should improve as j increases, but the convergence can be slow. For our implementation, we used a combination of linear and nonlinear (see Bjørstad, Dahlquist and Grosse [3]) sequence extrapolation techniques to produce an estimate for α that should be

accurate to at least six or seven digits even when a logarithmic singularity is present, at a cost of approximately thirty f_j values. For problems where we wish to integrate a vector of integrands we define $f_j = ||\mathbf{f}(\mathbf{x}_j)||_1$.

Tests with a limited number of examples indicate that it is difficult to determine highly accurate α values in all possible situations, but that our strategy provides α values accurate enough to be used with the integration extrapolation methods that we have outlined. We wish to emphasize, however, that the preferred mode of use for our software is for the user to provide an exact α value obtained from theoretical analysis.

4 Numerical Stability

In [8] it was shown that T_{kk} can be written

$$T_{kk} = \sum_{i=1}^{k} \gamma_i^{(k)} \hat{U}_i + \sum_{i=0}^{k} \delta_i^{(k)} Q_i,$$

and the condition number for T_{kk} was defined as

$$\tau_s^{(k)} = (1 - \frac{1}{2^s}) \sum_{i=1}^{k} |\gamma_i^{(k)}| (\frac{1}{2^s})^{i-1} + \sum_{i=0}^{k} |\delta_i^{(k)}| (\frac{1}{2^s})^i.$$

Note that the volumes associated with the estimates $\hat{U}_i$ and Q_i enter this expression. We can compute $\tau_s^{(k)}$ once the h-exponent sequence in (7) is known (see [8]). For example, when $s + \alpha = 1/2$, we have $\tau_s^{(k)}$ values that are given in the table,

s / k	1	2	3	4	7	10
1	5.83	6.92	6.30	4.49	1.55	1.07
2	5.83	4.28	3.13	1.80	1.02	1.00
3	5.83	2.96	2.15	1.24	1.00	1.00
4	5.83	2.30	1.81	1.09	1.00	1.00
5	5.83	1.97	1.67	1.04	1.00	1.00

Table 1: τ-table for $s + \alpha = 1/2$.

The extrapolation procedure is remarkably stable, and increasing the number of extrapolation steps has a positive effect on the stability of the method. $H_n^{(s)}(h)$ has volume h^s and this volume will decrease in each step with a factor $\frac{1}{2^s}$. This decrease in volume counter-effects the influence of the coefficients $\gamma_i^{(k)}$ and $\delta_i^{(k)}$ on the condition number.

The technique can also be applied to a combination of an algebraic and logarithmic singularity (see [7]) by assuming that each exponent in the error expansion appears twice. In the case

when $s+\alpha = 1/2$ this gives the sequence: 1/2, 1/2, 3/2, 3/2, 5/2, 5/2, ... We illustrate this in Table 2

s / k	1	2	3	4	7	10
1	5.83	22.31	30.23	39.15	17.94	4.51
2	5.83	16.48	16.55	14.29	2.37	1.05
3	5.83	13.57	11.40	7.08	1.19	1.00
4	5.83	12.11	9.25	4.50	1.05	1.00
5	5.83	11.38	8.28	3.43	1.02	1.00

Table 2: τ-table for $s+\alpha = 1/2$ with logarithmic singularity.

These two examples indicate that the numerical stability is very good when the singularity is not too strong. In the case of a very strong singularity we have considered changing the subdivision procedure as follows: cut the singular region by a factor $0 < \mu < 1$ in each of the singular directions. By reducing this cutting factor we will reduce the singular region volume faster and will therefore achieve improved stability. Recall that the first denominator in the extrapolation process is $n_1 = 2^{s+\alpha} - 1 > 0$. The closer n_1 is to zero the greater the condition number becomes. A cutting ratio equal to μ gives $n_1 = \mu^{-(s+\alpha)} - 1$ and forcing $n_1 = 1$, say, gives $\mu = 2^{-\frac{1}{s+\alpha}}$. This implies very good stability, but the cost is higher. The U_1-region will get the size $1-\mu$ in each of the singular directions and this means that this region comes closer to the singularity. Thus we will have to do more work on this region before it is useful in the extrapolation process. Therefore, what we gain in stability we may loose in the ability of Q to provide good estimates for the U values. In our implementation we have decided to use a fixed value of $\mu = 1/2$.

Another way to improve stability is to use a transformation of the original integration problem. Define $x_i = t_i^m$ for $m > 1$ and $i = 1, 2, \ldots, s$. The problem (1) now becomes

$$I(f) = m^s \int_{C_n} (t_1 t_2 \cdots t_s)^{m-1} f(t_1^m, \ldots, t_s^m, x_{s+1}, \ldots, x_n)\, dt_1 \cdots dt_s dx_{s+1} \cdots dx_n. \qquad (11)$$

We still integrate over C_n after the transformation, however the homogeneous function f_α has changed to $\tilde{f}_{\tilde{\alpha}}$ and the homogeneous degree to $\tilde{\alpha}$

$$\tilde{f}_{\tilde{\alpha}}(t_1, t_2, \ldots, t_s) = m^s (t_1 t_2 \cdots t_s)^{m-1} f_\alpha(t_1^m, t_2^m, \ldots, t_s^m),$$

with $\tilde{\alpha} = \alpha m + s(m-1)$. Note that $\tilde{\alpha} > \alpha > -s$ when $m > 1$ and thus (11) is an easier problem. Let us proceed as in Section 2 and expand $\tilde{g}$ around $\mathbf{x}_0$, myltiply by $\tilde{f}_{\tilde{\alpha}}$ and integrate over $H_n^{(s)}$. This gives

$$\tilde{E}_n^{(s)}(h) = Q_{H_n^{(s)}(h)}(\tilde{f}_{\tilde{\alpha}}\tilde{g}) - I_{H_n^{(s)}(h)}(\tilde{f}_{\tilde{\alpha}}\tilde{g}) = \sum_{\ell=0}^{p-1} \tilde{a}_\ell\, h^{\tilde{\alpha}+s+\ell m} + O(h^{\tilde{\alpha}+s+pm}), \qquad (12)$$

with $\tilde{a}_\ell = \tilde{b}_\ell - \tilde{c}_\ell$ for $\ell = 0, 1, \ldots, p-1$, $\tilde{b}_0 = Q_{C_n}(\tilde{f}_{\tilde{\alpha}}(t_1, t_2, \ldots, t_s)g(\mathbf{x}_0))$,
$\tilde{c}_0 = \int_{C_n} \tilde{f}_{\tilde{\alpha}}(t_1, t_2, \ldots, t_s)g(\mathbf{x}_0)dt_1 \cdots dt_s dx_{s+1} \cdots dx_n$, and

$$\tilde{b}_\ell = \tfrac{1}{\ell!} Q_{C_n}(\tilde{f}_{\tilde{\alpha}}(t_1, t_2, \ldots, t_s) \textstyle\sum_{i_1,\ldots,i_\ell=1}^{s} (t_{i_1} \cdots t_{i_\ell})^m \, \frac{\partial^\ell g}{\partial x_{i_1} \ldots \partial x_{i_\ell}}(\mathbf{x}_0))$$
$$\tilde{c}_\ell = \tfrac{1}{\ell!} \int_{C_n} \tilde{f}_{\tilde{\alpha}}(t_1, t_2, \ldots, t_s) \textstyle\sum_{i_1,\ldots,i_\ell=1}^{s} (t_{i_1} \cdots t_{i_\ell})^m \, \frac{\partial^\ell g}{\partial x_{i_1} \ldots \partial x_{i_\ell}}(\mathbf{x}_0) \, dt_1 \cdots dt_s dx_{s+1} \cdots dx_n,$$

for $\ell = 1, 2, \ldots p-1$. We can see from (12) that the exponent sequence in h starts with $\tilde{\alpha} + s$ and then proceeds in steps of m. Using $\mu = 1/2$ and taking the expansion (12) into account gives the denominator sequence defined by $n_1 = 2^{\tilde{\alpha}+s} - 1, n_{j+1} = 2^m n_j + 2^m - 1$ for $j > 0$.

This will give a more stable extrapolation procedure. Because we now have steps of size m in the exponent sequence, we also expect faster convergence of the expansion and thus the extrapolation. However, for the transformed integrands, U_j may be more difficult to approximate because we loose some of the polynomial degree for Q with increasing values of m. This transformation also requires more work. It is also difficult to pick the optimal m. Advice to a user might be to do some experiments on a particular problem class before using the code. The code has been designed so that a constant step size, m (integer), in the exponent sequence greater than one is allowed, with a default value of $m = 1$. We will illustrate, in an example, the behavior for different values of m.

Note that in the case of an additional logarithmic factor (12) has to be modified by replacing $\tilde{a}_\ell$ by $\check{b}_\ell + \hat{b}_\ell \ln h$ in front of $h^{s+\tilde{\alpha}+m\ell}$. Thus linear extrapolation can be performed as before by assuming that each h-exponent appears twice.

5 Numerical Examples

In this section we will demonstrate, with a few examples in 2, 3 and 4 dimensions, how efficient the new algorithm, implemented as a Fortran subroutine called DECUHR, can be. All computations have been done in double precision arithmetic on a SUN Sparc 10 workstation.

The first example is a line singularity in C_2 combined with a smooth function $g(x, y)$. For this example we used a local integration rule of degree 13 with 65 evaluation points per rectangle, in DECUHR. The integral for this example is

$$\int_{C_2} x^{-1/2} e^{2x+y(1-x)}(1-x) \, dxdy \approx 3.22289\ 15389\ 1637. \tag{13}$$

The presence of the 2-dimensional line singularity implies that both the singular and regular regions are cut in two halves in each subdivision step with the cost 130 function evaluations per step. The true and estimated relative errors are plotted in the following graph.

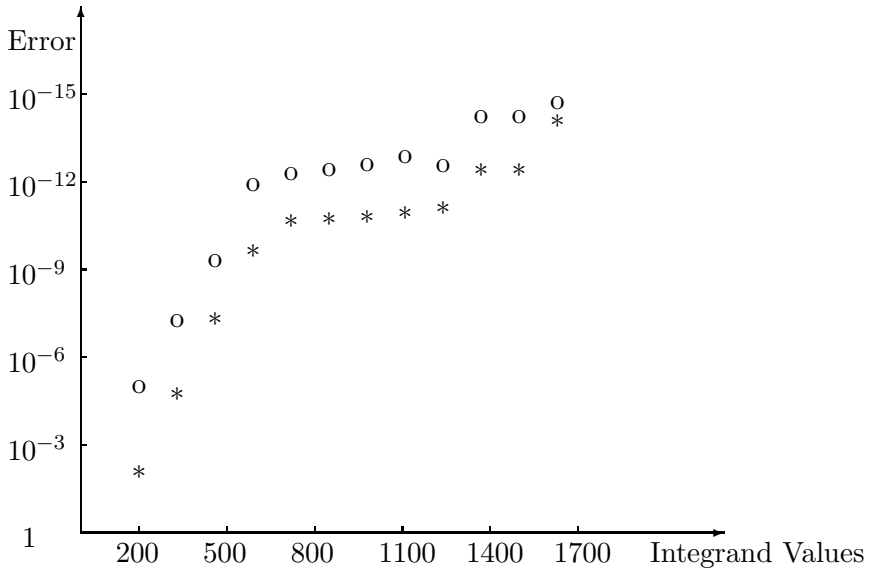

Figure 2. Actual (o) and estimated (*) relative errors with DECUHR for (13).

DECUHR performs quite well on this problem. The maximum number of extrapolation steps was six, and the actual sequence of extrapolation steps for the successive stages of the algorithm was 1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 6, 6. After 5 extrapolation steps DECUHR needed to work adaptively to improve the accuracy in the U-sequence. The precision level of the degree 13 rule had been reached for the five terms in the U-sequence and in this case each one of these U-regions has to be subdivided once to get better U-estimates.

Finally the code decides that it is ready for a new extrapolation step (it tries to take this step, however due to an increased error estimate, which is probably caused by the new U-term, the code decides not to update the result, but the next time this is done). The good numerical stability property is also demonstrated in this example. We note that the estimated error is much greater than the true error. This is due to a cautious error estimate which is intended to give a reliable code. We refer the reader to [1, 2] for further details. We can also use a rule of lower degree on this problem.

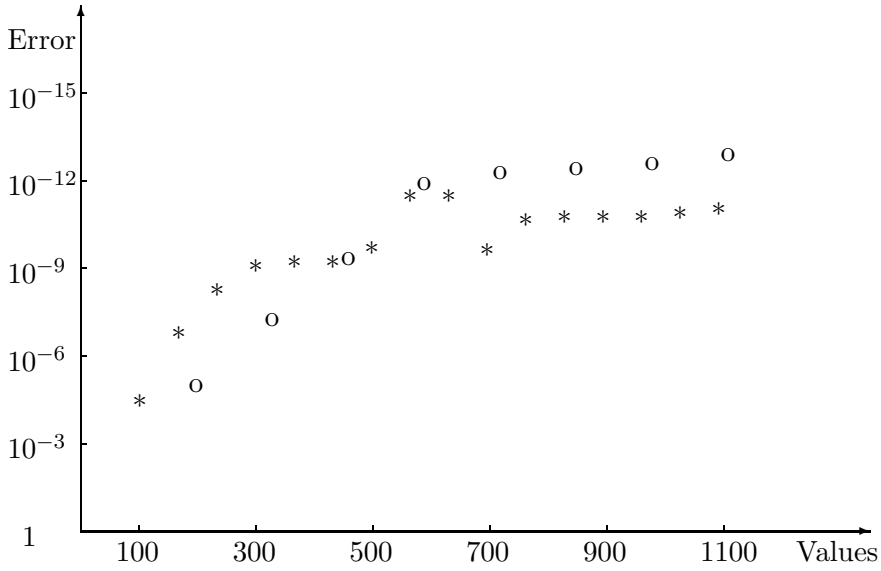

Figure 3. Errors from degree 13 (o) and 9 (*) rules with DECUHR for (13).

We see from this plot that there is, as we should expect, a turnover point: for low accuracies the degree 9 rule is better, but for high accuracies the degree 13 rule is to be better.

The second example is a 3 dimensional problem with a singularity along an edge in C_3:

$$\int_{C_3} (x+y)^{-1/2}\, e^{x+xy+z/3}\, dxdydz \approx 2.7878925361\ldots \tag{14}$$

The dimension of the singularity is two so the singular region will be subdivided into three subregions with each new extrapolation step. We use a 127 point degree 11 rule.

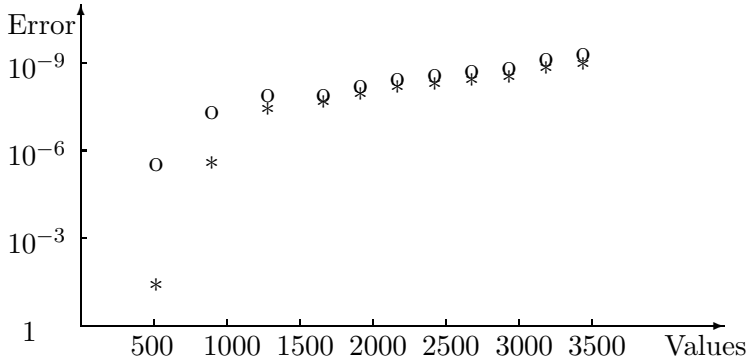

Figure 4. Actual (o) and estimated (*) relative errors with DECUHR for (14).

In our third example, a corner singularity is combined with a logarithmic singularity in C_3. With $r = \sqrt{x^2+y^2+z^2}$ compute

$$-\int_{C_3} r^{-0.5} \log r\, e^{xy+z}\, dxdydz \approx 0.11763\ 64548\ ... \tag{15}$$

Again we use a 127 point degree 11 rule. A corner singularity requires a subdivision of the singular region into four subregions for each new extrapolation step.

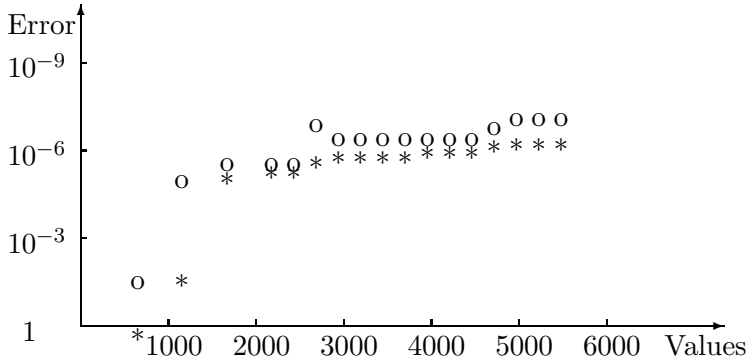

Figure 5. Actual (o) and estimated (*) relative errors with DECUHR for (15).

DECUHR used 1, 2, 3 and 4 extrapolation steps initially and then the rest of the work is spent on improving these four U-terms in the series.

For example 4 we integrate a face singularity in C_4:

$$\int_{C_4} x^{-3/2}\, sin(x)\, e^{(xy+z+2u)} dxdydzdu \approx 12.72764\ 93571... \tag{16}$$

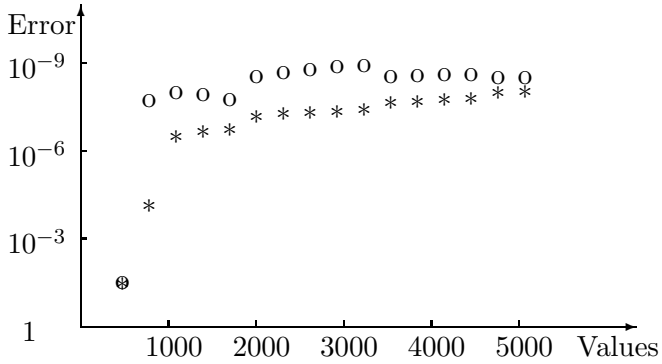

Figure 6. Actual (o) and estimated (*) relative errors for DECUHR with (16).

DECUHR used 1, 2 and 3 extrapolation steps initially and then the rest of the work is spent on improving these three U-terms in the series.

For example 5 a line singularity in C_2 from a Hölder singularity is combined with a peak in the center of the square. With $r = \sqrt{x^2 + y^2}$ compute

$$\int_{C_2} \frac{x}{r((x-1/2)^2 + (y-1/2)^2 + 0.01)} dxdy \approx 7.3872... \tag{17}$$

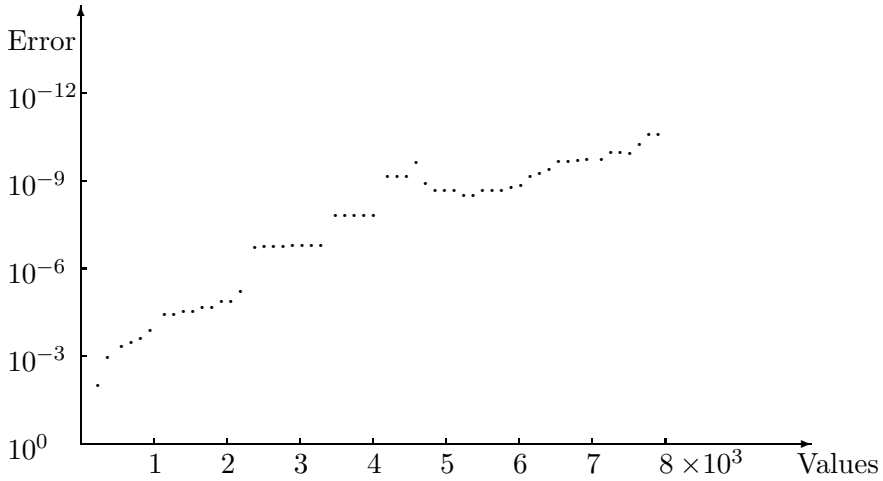

Figure 7. Actual errors from degree 13 rule (.) with DECUHR for (17).

This example illustrates the interaction between extrapolation and adaptivity. DECUHR starts by doing one extrapolation step, creating $\hat{U}_1$. Because of the peak DECUHR decides

immediately that $\hat{U}_1$ has to be improved. After one subdivision it is ready for a new extrapolation step. Then follow three adaptive steps before the third extrapolation step is taken. Extrapolation steps 4, 5 and 6 all gives jumps in the true precision, the 7th and last extrapolation step is not that easy to spot.

We use the last example to illustrate the effect of applying the transformation $x \leftarrow x^m$ (m integer) giving

$$\int_{C_2} m\, x^{0.3m-1}\, e^{(2x^m+y)} dxdy \approx 10.94423\ 78571\ 7145... \tag{18}$$

independent of $m \geq 1$. $m < 10/3$ implies a line singularity along the y-axis and the exponent sequence starts with $\alpha = 0.3m - 1$ and continues in steps of m. Running the code with $m = 1$, 2 and 3 gives the following plots.

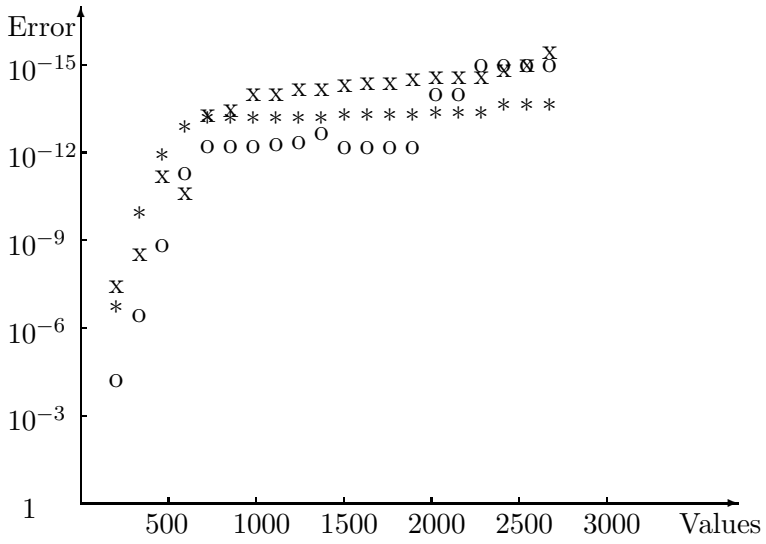

Figure 8. Actual errors for $m = 1$ (o), 2 (*) and 3 (x) from DECUHR for (18).

We see that there is no general conclusion to be drawn based on this experiment. The cost per function evaluation increases with m. If one has a certain precision in mind then there is an optimal m which may be valid for all functions in a certain class. Thus an initial experiment may give valuable insight in finding such a value of m.

When using the code, one has to trust the estimated errors. In Figure 9, we plot the estimated errors corresponding to the actual errors shown in Figure 8.

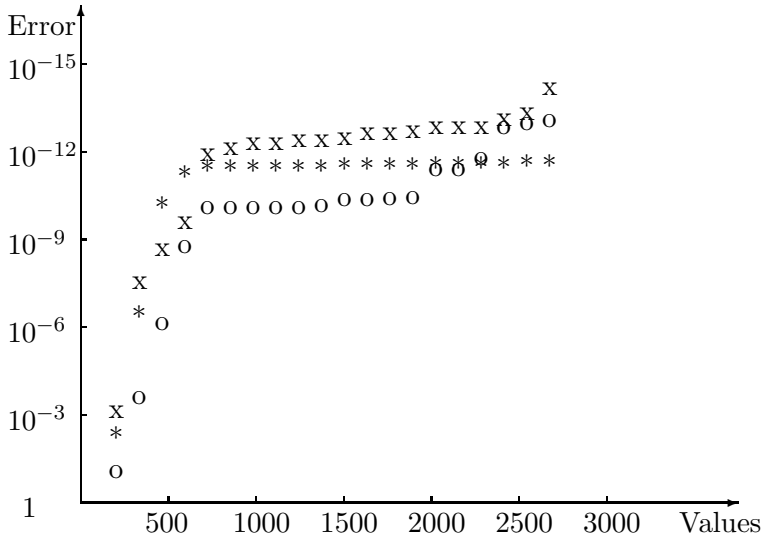

Figure 9. Estimated errors for $m = 1$ (o), 2 (*) and 3 (x) from DECUHR for (18).

We see basically the same behavior for the estimated errors as for the actual errors. Thus, we can use these estimated errors in choosing a near optimal value of m.

6 Concluding Remarks

The main goal in the development of the code DECUHR, for multidimensional integration, was to modify the general purpose code DCUHRE, [1, 2], to handle singularities effectively without ruining the good reliability properties of DCUHRE. The DCUHRE code does not use extrapolation, and therefore cannot efficiently adapt to singularities. The primary modifications to DCUHRE consisted of adding code to handle the subdivision and extrapolation for the singular regions. We feel that the power of this new code is clearly demonstrated through these examples and thus we claim that our main goal has been achieved.

In each dimension we have more than one rule available, just as in DCUHRE. The rule of thumb is that for low accuracy then a cheap rule may suffice, but for higher accuracy a more powerful rule is generally better.

The new algorithm can be applied to a vector of similar integrands over a common integration region. All of the integrands in the vector have to be singular in the same s variables with the same value of α. If there is a logarithmic singularity in one integrand we assume that all of the integrands in the vector have a logarithmic singularity. The homogeneous functions involved may be different. 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. For integrands that have enough similarity, the new algorithm may save time and space because of the common subdivisions.

7 DECUHR: A FORTRAN implementation

The algorithm described in Section 3, including the error-estimating procedure from [1, 2], has been implemented in a FORTRAN subroutine DECUHR, a double precision adaptive cubature routine with extrapolation. The software is organized so that a single precision version can be

obtained directly from DECUHR by using a global change of all DOUBLE PRECISION declarations to REAL declarations. The authors single precision version has the name SECUHR and the first letter in all subroutine names is changed from D to S. DECUHR has the calling sequence

```
CALL DECUHR (NDIM,NUMFUN,A,B,MINPTS,MAXPTS,FUNSUB,SINGUL,
             ALPHA,LOGF,EPSABS,EPSREL,KEY,EMAX,WRKSUB,NW,
             RESTAR,RESULT,ABSERR,NEVAL,IFAIL,WORK,IWORK)
```

DECUHR computes approximations RESULT to the vector integral

$$\text{I[FUNSUB]} = \int_H \text{FUNSUB}\, dx_1 dx_2 \cdots dx_{\text{NDIM}}$$

where FUNSUB is a vector of integrands of length NUMFUN and H is a NDIM dimensional hyperrectangular region with lower limits given by A and upper limits given by B. The singularity is assumed to involve the directions 1, 2,..., SINGUL. The code expects the specified vertex A to be a singular point. ALPHA is the degree of an algebraic singularity and LOGF flags a logarithmic singularity. DECUHR attempts to compute each component of RESULT with

$$|\ \text{I[K]-RESULT[K]}\ | \leq \max(\text{EPSABS,EPSRES*}|\ \text{I[K]}\ |), \text{ for K} = 1, ..., \text{NUMFUN},$$

where EPSABS and EPSREL are absolute and relative tolerances. The program has been successfully run on Sun SPARC 10, DECstation 5000 and HP 9000/840 computers.

The main integration subroutine and subprograms are organized according to the structure given in Figure 10.

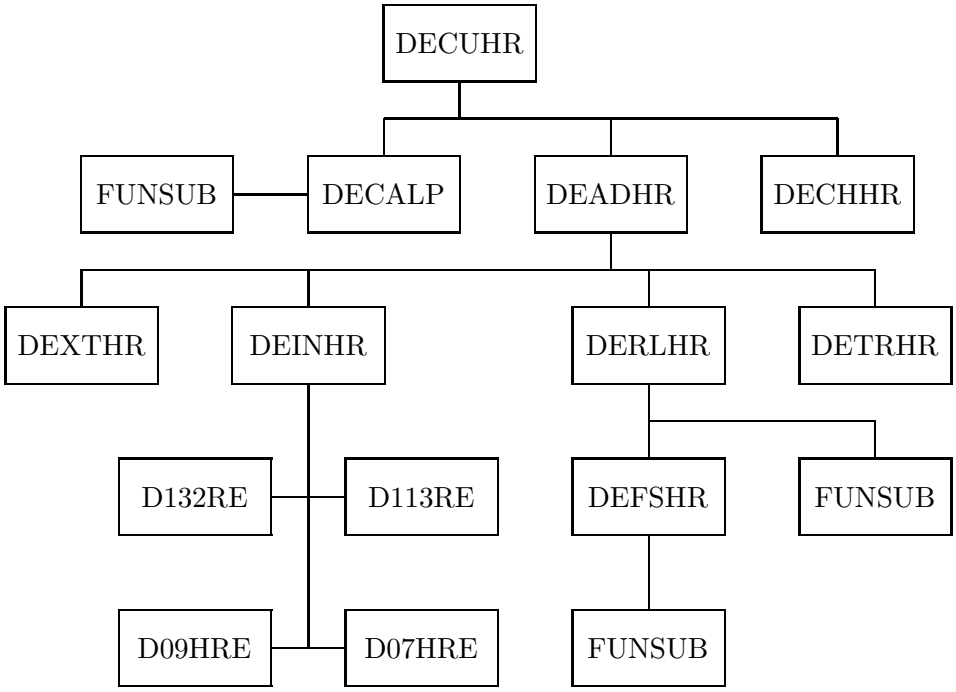

Figure 10. Subprogram organization

In the following list we briefly describe the purpose of DECUHR and supporting subroutines.

- DECUHR is basically a driver for the algorithm controller DEADHR. Inside DECUHR the input to DECUHR is checked and the workspace for DEADHR is separated into mnemonic variables.
- DECALP computes estimates for ALPHA and LOGF, if required.
- DECHHR checks the validity of the input to DECUHR.
- DEADHR is the algorithm controller which at each subdivision step decides whether to stop or continue. This routine also decides whether a new extrapolation step will be taken or just an update of the extrapolation table will be done.
- DEXTHR does either the extrapolation or the update of the extrapolation table. The pure extrapolation error is estimated and the global error is then computed.
- DETRHR maintains the heap of subregions.
- DERLHR computes approximations to the integral and the error over each nonsingular subregion. The integration rules are based on fully symmetric sums, and DEFSHR is called to compute these sums.

- DEINHR selects the integration rule according to the value of KEY.
- D132RE is a two dimensional degree 13 rule.
- D113RE is a three dimensional degree 11 rule.
- D09HRE is a degree 9 rule for any dimension ≥ 2.
- D07HRE is a degree 7 rule for any dimension ≥ 2.

APPENDIX: The Parameters for DECUHR.

Input Parameters

NDIM The number of variables in the integrand(s). We require $1 < NDIM \leq 15$.

NUMFUN The number of components in the vector integrand function.

A The lower limits of integration. This array has dimension NDIM. This is assumed to be a singular vertex in the hyperrectangle.

B The upper limits of integration. This array has dimension NDIM.

MINPTS The minimum number of FUNSUB calls the code has to use.

MAXPTS The maximum number of FUNSUB calls the code is allowed to use. See the code for details.

FUNSUB An externally declared subroutine for computing all components in the vector function for the given evaluation point. It must have parameters (NDIM, X, NUMFUN, FUNVLS). The input parameters: NDIM is the dimension of the region, X is an array of dimension NDIM giving the coordinates of the evaluation point and NUMFUN is the number of components of the vector integrand. The output parameter FUNVLS is an array of dimension NUMFUN that stores NUMFUN values of the vector integrand.

SINGUL The dimension of the singularity. Thus the directions: 1,2,...,SINGUL are all involved in the singularity.

ALPHA The degree of the algebraic singularity. If input ALPHA $\leq$ -SINGUL ALPHA and LOGF are estimated and reset with estimated values. We refer to the code for details.

LOGF Signals whether a logarithmic singularity is present or not. LOGF = 0 implies no logarithmic singularity, while LOGF = 1 then implies that the logarithm to a homogeneous function appears in first power involving the SINGUL coordinates.

EPSABS Requested absolute error.

EPSREL Requested relative error.

KEY This selects the integration rule. KEY = 0 is the default value. The following table gives the polynomial degrees of the different integration rules for the possible values of KEY and for dimensions 2, 3 and > 3.

KEY	NDIM = 2	NDIM = 3	NDIM >3
0	13	11	9
1	13	-	-
2	-	11	-
3	9	9	9
4	7	7	7

EMAX The maximum allowed number of extrapolation steps.

WRKSUB The maximum allowed number of subregions.

NW Defines the length of the working array WORK. NW must satisfy
NW $\geq$ 3 + 17*NUMFUN + WRKSUB*(NDIM+NUMFUN+1)*2 + EMAX
+ NUMFUN*(3*WRKSUB+9+EMAX) + (EMAX+1)**2 + 3*NDIM)

RESTAR If RESTAR = 0, this is the first attempt to compute the integral.
If RESTAR = 1 , then we continue a previous attempt. In this case the only parameters for DECUHR that may be changed (with respect to the previous call of DECUHR) are MINPTS, MAXPTS, EPSABS, EPSREL and RESTAR.

Output Parameters

RESULT An array of dimension NUMFUN containing approximations to all components of the vector integral.

ABSERR An array of dimension NUMFUN of estimates for absolute errors for all components of the vector integral.

NEVAL The number of function evaluations used by DECUHR.

IFAIL IFAIL = 0 for normal exit, with
ABSERR(K) $\leq$ max(EPSABS, ABS(RESULT(K))*EPSREL)
for 1 $\leq$ K $\leq$ NUMFUN, using MAXPTS or fewer FUNSUB calls.
IFAIL = 1 if MAXPTS is too small for DECUHR to obtain the requested error. In this case DECUHR returns values of RESULT with estimated absolute errors ABSERR.
IFAIL = 2 if KEY $<$ 0 or KEY $>$ 4.

IFAIL = 3 if NDIM $<$ 2 or NDIM $>$ 15.
IFAIL = 4 if KEY = 1 and NDIM $\neq$ 2.
IFAIL = 5 if KEY = 2 and NDIM $\neq$ 3.
IFAIL = 6 if NUMFUN $<$ 1.
IFAIL = 7 if A(J) $\geq$ B(J), for some J.
IFAIL = 8 if MAXPTS $<$ MINPTS.
IFAIL = 9 if both EPSABS $\leq$ 0 and EPSREL $\leq$ 0.
IFAIL = 10 if RESTAR $>$ 1 or RESTAR $<$ 0.
IFAIL = 11 if SINGUL $>$ NDIM or SINGUL $<$ 1.
IFAIL = 12 if LOGF $<$ 0 or LOGF $>$ 1.
IFAIL = 13 if MAXPTS $<$ (SINGUL+2)*(The number of points in the rule).
IFAIL = 14 if ALPHA $\leq$ -SINGUL.
IFAIL = 15 if EMAX $<$ 1.
IFAIL = 16 if NW is too small.
IFAIL = 17 if WRKSUB is too small.

WORK A work array of dimension NW. See the code for details.

IWORK A work array of dimension NDIM + 2*WRKSUB. See the code for details.

References

[1] 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.

[2] J. Berntsen, T.O. Espelid, and A. Genz. Algorithm 698: DCUHRE: An Adaptive Multidimensional Integration Routine for a Vector of Integrals. *ACM Trans. Math. Softw.*, 17(4):452–456, 1991.

[3] P. Bjørstad, G. Dahlquist and E. Grosse. Extrapolation of Asymptotic Expansions by a Modified Aitken δ^2-Formula. *BIT*, 21(1):56–65, 1981

[4] R. Cariño. Numerical integration over finite regions using extrapolation by nonlinear sequence transformations, 1992. Ph.D. thesis, La Trobe University, Bundoora, Victoria, Australia.

[5] R. Cariño, I. Robinson, and E. de Doncker. An algorithm for automatic integration of certain singular functions over a triangle. In T. O. Espelid and A. Genz, editors, *Numerical Integration, Recent Developments, Software and Applications*, NATO ASI Series C: Math. and Phys. Sciences Vol. 357, pp. 295–304, Dordrecht, The Netherlands, 1992. Kluwer Academic Publishers.

[6] E.de Doncker and J.A. Kapenga. A parallelization of adaptive integration methods. In P. Keast and G. Fairweather, editors, *Proceedings of the NATO Advanced Research Workshop on Numerical Integration*. D. Reidel Publishing Company, 1987.

[7] T. O. Espelid. Integrating singularities using non-uniform subdivision and extrapolation. *Numerical Integration IV*, H. Brass and G. Hämmerlin (Eds.), Birkhäuser Verlag, Basel, Vol. 112:77-89, 1993.

[8] T. O. Espelid. On integrating vertex singularities using extrapolation. *BIT*, 34:62-79, 1994.

[9] D. Levin. Development of non-linear transformations for improving convergence of sequences. *J. Inst. Maths. Applics.*, 3:371–388, 1973.

[10] D. Levin and A. Sidi. Two classes of non-linear transformations for accelerating the convergence of infinite integrals and series. *Appl. Math. Comp.*, 9:175–215, 1981.

[11] J.N. Lyness. An error functional expansion for n-dimensional quadrature with an integrand function singular at a point. *Math. Comp.*, 30(133):1–23, 1976.