Numerical Integration in S-PLUS or R: A Survey

Diego Kuonen*

This paper reviews current quadrature methods for approximate calculation of integrals within S-PLUS or R. Starting with the general framework, Gaussian quadrature will be discussed first, followed by adaptive rules and Monte Carlo methods. Finally, a comparison of the methods presented is given. The aim of this survey paper is to help readers, not expert in computing, to apply numerical integration methods and to realize that numerical analysis is an art, not a science.

Key Words: Adaptive method; Comparison; Gaussian quadrature; Monte Carlo method; Quadrature.

1 Introduction

Numerical integration, also called *quadrature*, is the study of how the numerical value of an integral can be found. The purpose of this paper is to discuss quadrature methods for approximate calculation of integrals. All are based, in one way or another, on the obvious device of adding up the value of the integrand at a sequence of points within the range of integration. Hence, most of the approximations we consider have the form

$$\int_{R_m} \cdots \int w(x_1, \ldots, x_m) f(x_1, \ldots, x_m) dx_1 \cdots dx_m \approx \sum_{i=1}^{M} W_i f(y_{i,1}, \ldots, y_{i,m}), \qquad (1.1)$$

where R_m is a given region in a m-dimensional Euclidean space E_m and $w(x_1, \ldots, x_m)$ is a given weight function. The $(y_{i,1}, \ldots, y_{i,m})$ lie in E_m and are called the *points* of the formula. The W_i are constants which do not depend on $f(x_1, \ldots, x_m)$ and are called the

*Diego Kuonen, PhD, is CEO and applied statistician, Statoo Consulting, PO Box 107, 1015 Lausanne, Switzerland (E-mail: kuonen@statoo.com). The author's research was performed while writing his PhD thesis under the supervision of Professor A. C. Davison at the Swiss Federal Institute of Technology in Lausanne.

coefficients of the formula. We say that formula (1.1) has *degree* r (or *degree of exactness* r) if it is exact for all polynomials in $x_1, \ldots, x_m$ of degree $\leq r$ and there is at least one polynomial of degree $r+1$ for which it is not exact. See also Stroud (1971) or Evans (1993, chap. 6).

The theory of integration formulae for functions of one variable ($m = 1$) is well developed. A great deal of this theory can be found in the books by Engels (1980), Davis and Rabinowitz (1984), Evans (1993) or Press *et al.* (1993, chap. 4). For $m = 1$ equation (1.1) can be written as

$$\int_R w(x)f(x)dx = \int_a^b w(x)f(x)dx \approx \sum_{i=1}^{M} W_i f(y_i). \tag{1.2}$$

In the classical formulae the integral of a function is approximated by the sum of its values at a set of equally spaced points, multiplied by certain aptly chosen coefficients of the formula. Examples include the trapezoidal and Simpson's rules. Hence only the W_i are free to be used to force the quadrature rule to have a certain degree of exactness. The freedom to fix the points y_i has been thrown away, presumably in the interests of getting linear equations for the W_i. If the y_i are also left free, the result is a set of non-linear equations which can be shown to have solutions based on the zeros of the associated sets of orthogonal polynomials for the given interval $[a, b]$ and weight function $w(x)$. This leads to the elegant theory of Gaussian quadrature, which will be discussed in Section 3. Gaussian quadratures are formulae which are said to be *progressive*, as the points for any point-number M are in general quite different from those for any other point-number. Another term used to describe quadrature rules is *adaptive*. A rule is adaptive if it compensates for a difficult subrange of an integrand by automatically increasing the number of quadrature points in the awkward region. As we will see in Section 4 adaptive rules are usually based on a standard underlying quadrature rule, often a progressive one. For very high dimensionality Monte Carlo or random sampling methods (Section 5) can begin to be competitive, though in this regime all methods tend to be very inaccurate for a reasonable computer effort. Finally, a comparison of the presented methods is given in Section 6. But, first we describe the computing environment used.

2 Computing Environment

Calculations were made on a Sun SPARC Ultra 60 workstation with 1Gb RAM using S-PLUS or R. S-PLUS is a value-added version of S (Becker *et al.*, 1988; Chambers, 1998) sold by Insightful Corporation. The S language is often the vehicle of choice for research in statistical methodology and R (Ihaka and Gentleman, 1996; `www.R-project.org`) provides a free software route to participation in that activity. Good reference books for S-PLUS and R are Venables and Ripley (2000, 2002).

The tools available in S-PLUS to measure resources differ between versions. The principal resource considered in this paper is CPU time. It is clear that more resources, like memory usage, should be considered but we found it sufficient to consider CPU time

in order to illustrate our remarks. To do so we used the S-PLUS function `resources` given in Venables and Ripley (2000, pp. 151–152).

3 Gaussian Quadrature

The idea of Gaussian quadrature is to give ourselves the freedom to choose not only the coefficients W_i, but also points at which the function is to be evaluated. Moreover, the formula (1.2) is forced to have degree of exactness $2M-1$. Because of the computational expense of generating a new Gaussian formula, only commonly used combinations of the interval and weight functions are normally tabulated (Evans, 1993, sec. 2.3). The most commonly used rule is the Gauss–Legendre rule with interval $[-1, 1]$ and weight function $w(x) = 1$.

Neither S-PLUS or R offer the Gauss–Legendre rule, nor any other standard Gaussian quadrature rule, by default. Nevertheless, the `integrate2` library for S-PLUS (`lib.stat.cmu.edu/S/integrate2`) contains the function `intgauss`, which performs the numerical integration of a function over a given region using a classical 10 point Gaussian formula. To give more flexibility for the choice of M and to extend it to higher dimensions, we wrote a S-PLUS function `GL.integrate.1D` to compute (1.2) by means of the Gauss–Legendre rule and based on a modified version of the C function GAULEG given in Press *et al.* (1993, p. 151). The function is entirely written in S-PLUS; a C version is available. We used the fact that any finite range quadrature on the interval $[a, b]$, can be transformed using the linear transformation

$$x = \frac{b-a}{2}t + \frac{b+a}{2}$$

to the standard interval $[-1, 1]$. The function `GL.integrate.1D` is given in Kuonen (2001, app. A, Table A.1). It uses the function `GL.YW` which computes, if needed, the points y_i and the weights W_i for the interval $[\texttt{low}, \texttt{upp}]$, for $i = 1, \ldots, M$, where $M = \texttt{order}$. The function `GL.YW` is given in Kuonen (2001, app. A, Table A.2).

For the multi-dimensional case we reduce the multiple integral on the left-hand side of (1.1) into repeated integrals over $[-1, 1]$,

$$\int_{-1}^{1} dx_1 \int_{-1}^{1} dx_2 \cdots \int_{-1}^{1} f(x_1, \ldots, x_m) dx_m. \tag{3.1}$$

Then we apply a classical quadrature formula to each integral in (3.1), which yields using the right-hand side of (1.1) a product rule of the form

$$\sum_{i_m=1}^{M} \cdots \sum_{i_1=1}^{M} W_{i_1} \cdots W_{i_n} f(y_{i_1}, \ldots, y_{i_m}), \tag{3.2}$$

where the weights W_{i_j} and the points y_{i_j}, $j = 1, \ldots, m$, are chosen to be appropriate for the specific dimension to which they are applied (Evans, 1993, chap. 6). The number of function evaluations using the M^m integration points may be quite large. For $m = 2$ this

is illustrated by the S-PLUS function `GL.integrate.2D` given in Kuonen (2001, app. A, Table A.3). For example

```
> tmp.fct <- function(x,y) {1/(1-x*y)}
> GL.integrate.2D(tmp.fct, low=c(0,0), upp=c(1,1), order=128)
[1] 1.644886
```

took 5.48 seconds CPU time to perform 16,384 function evaluations when the calculation of the array containing the points and the weights was needed, and 0.09 seconds CPU time when previously tabulated values were taken.

Multi-dimensional Gaussian quadrature up to $m = 20$ over hyper-rectangles could also be computed with the subroutine D01FBF from the commercial NAG Fortran library (`www.nag.co.uk`), and loaded dynamically into S-PLUS or R. There is ongoing development of a free software library of routines for numerical computing: the 'GNU Scientific Library' (GSL) available at `sources.redhat.com/gsl/`. Current releases look promising.

There are many different ways in which the Gaussian quadrature has been extended. An example is the Gauss–Kronrod formulae; see, for instance, Davis and Rabinowitz (1984, sec. 2.7.1.1). An optimal extension can be found for Gauss–Legendre quadrature, giving a degree of exactness of $3M + 1$. This is for instance the case with the S-PLUS or R function `integrate`, which implements uni-dimensional adaptive 15-point Gauss–Kronrod quadrature based on the Fortran functions DQAGE, and DQAGIE from QUADPACK (Piessens *et al.*, 1983; `www.netlib.org/quadpack/`). This function is the only numerical integration function implemented in the S-PLUS or R standard packages. A similar function, `gkint` from the S-PLUS library `integrate2` (`lib.stat.cmu.edu/S/integrate2`) uses a (7–15)-point Gauss–Kronrod pair by means of the routine DQAG from QUADPACK.

As mentioned, for M quadrature points in each dimension the sum in (3.2) is over M^m terms. Therefore the numerical effort of Gaussian quadrature techniques increases exponentially with the integral dimension. Hence when m is large this method is nearly useless. Furthermore, the trouble with Gaussian quadrature is that you have no real idea of how accurate the answer is. You can always increase the accuracy by using a higher order Gauss method or by applying it piecewise over smaller periods but you still do not know the accuracy in terms of correct decimal places. To get a prescribed accuracy one needs adaptive integration, which keeps reducing the step size until a specified error has been achieved.

4 Adaptive Methods

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 simplices. A globally adaptive algorithm for integration over hyper-rectangles was first described by van Dooren and de Ridder (1976) and programmed as a Fortran function HALF. It was improved by Genz and Malik (1980).

Implementations of the Genz and Malik modified algorithm (programmed as a Fortran function ADAPT) have appeared in the NAG Fortran library (`www.nag.co.uk`, subroutine D01FCF). The routine operates by repeated subdivision of the hyper-rectangular region into smaller hyper-rectangles. In each subregion, the integral is estimated using a rule of degree seven, and an error estimate is obtained by comparison with a rule of degree five which uses a subset of the same points. These subdivisions are designed to dynamically concentrate the computational work in the subregions where the integrand is most irregular, and thus adapt to the behaviour of the integrand. Genz (1991) gives a detailed description in the context of adaptive numerical integration for simplices.

Berntsen *et al.* (1991a) improved the reliability of previous algorithms, and developed a new algorithm for adaptive multidimensional integration. Tests (Berntsen *et al.* 1988) of a Fortran implementation, DCUHRE (Berntsen *et al.*, 1991b), have shown that the improvement has been successful.

Both DCUHRE and ADAPT can be dynamically loaded into S-PLUS or R. The Fortran routine DCUHRE is implemented in the S-PLUS function `dcuhre`, which is contained in the `integrate2` library (`lib.stat.cmu.edu/S/integrate2`), and ADAPT comes with the S-PLUS function `adapt` included in the S-PLUS library `adapt` (`lib.stat.cmu.edu/S/adapt`). The function `adapt` is in the R package `integrate`, which is available on CRAN (`cran.r-project.org`).

Genz (1992) suggested that such subregion adaptive integration algorithms can be used effectively in some multiple integration problems arising in statistics. The key to good solutions for these problems is the choice of an appropriate transformation from the infinite integration region for the original problem to a suitable finite region for the subregion adaptive algorithm. Genz (1992, sec. 3.2) also discussed different types of such transformations; see also Davis and Rabinowitz (1984) for further examples of possible transformations.

Traditional quadrature methods (even newer adaptive ones) have been almost forgotten in the recent rush to 'Markov Chain Monte Carlo' (MCMC) methods; Evans and Swartz (1995) provided a nice recent summary focusing on these methods. They indicate that significant progress has been made using five general techniques: asymptotic methods, importance sampling, adaptive importance sampling, multiple quadrature and Markov chain methods. More recently, Genz and Kass (1997) argued that the reason why existing quadrature methods have been largely overlooked in statistics, even though they are known to be more efficient than Monte Carlo methods for well-behaved problems of low dimensionality, may be that when applied they are poorly suited for peaked-integrand functions. Hence they proposed transformations based on split-t distributions to allow integrals to be efficiently computed using a subregion-adaptive numerical integration algorithm. Fortran routines are already available (BAYESPACK at `www.sci.wsu.edu/math/faculty/genz/genzhome/software.html`) and work on constructing a version for use with S-PLUS is underway.

5 Monte Carlo Methods

Numerical methods known as Monte Carlo (MC) methods can be loosely described as statistical simulation methods. For a complete introduction to MC integration we refer to Stroud (1971, chap. 6), Kalos and Whitlock (1986) or Robert and Casella (1999, chap 3). The classical MC method for approximating a multiple integral as given in the left-hand side of (1.1) with $w(x_1, \ldots, x_m) = 1$, denoted by $\mathcal{I}(f)$, is as follows. We choose M set of points $\{y_{1,1}, \ldots, y_{1,m}\}, \ldots, \{y_{M,1}, \ldots, y_{M,m}\}$ at random, uniformly distributed in R_m. The integral is then estimated using $W_i = V/M$ in the right-hand side of (1.1),

$$\mathcal{I}(f) \approx \hat{\mathcal{I}}(f) = \frac{V}{M} \sum_{i=1}^{M} f(y_{i,1}, \ldots, y_{i,m}), \tag{5.1}$$

where $V = \mathcal{I}(1)$ is the m-dimensional volume of R_m. The basic MC method iteratively approximates a definite integral by uniformly sampling from the domain of integration, and averaging the function values at the samples. The integrand is treated as a random variable, and the sampling scheme yields a parameter estimate of the mean, or expected value of the random variable. Since $\hat{\mathcal{I}}(f)$ in the right-hand side of (5.1) estimates $\mathcal{I}(f)$ the absolute error ϵ in this mean can be evaluated by considering the corresponding standard error of the mean,

$$\epsilon = \left| \hat{\mathcal{I}}(f) - \mathcal{I}(f) \right| \approx \frac{\sigma}{M^{1/2}}, \tag{5.2}$$

where σ^2 is $V\mathcal{I}(f^2) - \mathcal{I}^2(f)$. If $\{y_{1,1}, \ldots, y_{1,m}\}, \ldots, \{y_{M,1}, \ldots, y_{M,m}\}$ are regarded as independent random variables then $\hat{\mathcal{I}}(f)$ is a random variable with mean $\mathcal{I}(f)$ and variance σ^2/M, which can also be estimated from the random sample through

$$\frac{V}{M^2} \sum_{i=1}^{M} \{f(y_{i,1}, \ldots, y_{i,m}) - \hat{\mathcal{I}}(f)\}^2.$$

Furthermore, the error estimate (5.2) may be inverted to show the number of samples needed to yield a desired error, $M = \sigma^2/\epsilon^2$. For $m = 1$ this is illustrated by the S-PLUS functions `MC.integrate.1D` and for the two-dimensional case by `MC.integrate.2D`; both given in Kuonen (2001, app. A, Tables A.5–A.6). An example of their use is

```
> MC.integrate.1D(function(z) sqrt(z), 0, 1, 1000, 2/3)
To achieve an error of 0.0001 you need at least 314516 points.
[1] 0.6670135
```

This clearly reflects the slow convergence of the MC methods; the absolute error (5.2) has an average magnitude of $O(M^{-1/2})$. Hence to reduce the error, for example, by a factor of 10 requires a 100-fold increase in the number of sample points. In the previous example one would need $M = 314,516$ points to get an accuracy of 0.0001. Therefore, other methods have been studied for decreasing the error. Such approximations are called 'Quasi Monte Carlo' (QMC) methods. The QMC method uses a formula which is

formally identical to that of the MC method, except that the points used for evaluating the function are generated deterministically. Unlike the MC method, the QMC method has a deterministic error bound, and the accuracy of the integral is generally significantly better than in the MC method. Many different QMC methods are known. One method makes use of results from the the theory of numbers and is called the *number-theoretic* method; see Stroud (1971, sec. 6.3), Fang and Wang (1994) or Fang *et al.* (1994).

Additional methods have been employed to reduce the error of the MC method, such as importance sampling, stratified sampling, antithetic variates and non-random sequences (Press *et al.*, 1993, sec. 7.6–7.8; Evans and Swartz, 1995). These methods are mostly concerned with finding sets of points that yield smaller integration errors. Importance sampling concentrates samples in the area where they are more effective by using *a priori* knowledge of the function. Stratified sampling tries to distribute samples evenly by subdividing the domain into subregions such as grids. It is possible to combine some of these techniques, or to apply them adaptively (Press *et al.*, 1993, sec. 7.8). For example, uniformly distributed samples generated by stratification can be employed for importance sampling.

As described above, MC integration draws samples from the required distribution, and then forms sample averages to approximate expectations. MCMC methods draw these samples by running a constructed Markov chain for a long time. An example of a way to construct such a chain is the Gibbs sampler. An introduction to MCMC methods and their applications is given in Gilks *et al.* (1996) or Robert and Casella (1999). But questions on convergence of the chains and efficient implementation remain unresolved (Cappé and Robert, 2000).

It is well known that for high-dimensional integrals MC techniques should be preferred to the standard quadrature methods given in Sections 3 and 4 since the sum in (5.1) is only over M terms instead of the M^m terms in (1.1). Nevertheless, we do not feel so comfortable using the MC methods mentioned in this section for mainly one reason: one needs too many function evaluations to get a certain accuracy. So we will consider their simplest versions (Kuonen, 2001, app. A, Tables A.5–A.6) in the next section only for purposes of illustration.

6 Comparison

The testing of numerical quadrature methods involves the practical realization of the theoretical claims, and is well illustrated by applying a method to a set of well-designed examples. For $m = 1$ two extensive sets of test integrals which appear in the numerical analysis literature have been used to make a comparative computation. The first set is due to Casaletto *et al.* (1969) and contains 50 functions ranging from polynomials up to degree 20 through functions with discontinuities; see also Evans (1993, Table 2.3). The second set of 21 examples is due to Kahaner (1971) which includes in addition some harder examples (Evans, 1993, Table 2.4). These 71 test examples have been integrated using the various S-PLUS functions described in the previous sections. Namely, from Section 3 the default S-PLUS function `integrate`, the function `intgauss` using a 10-

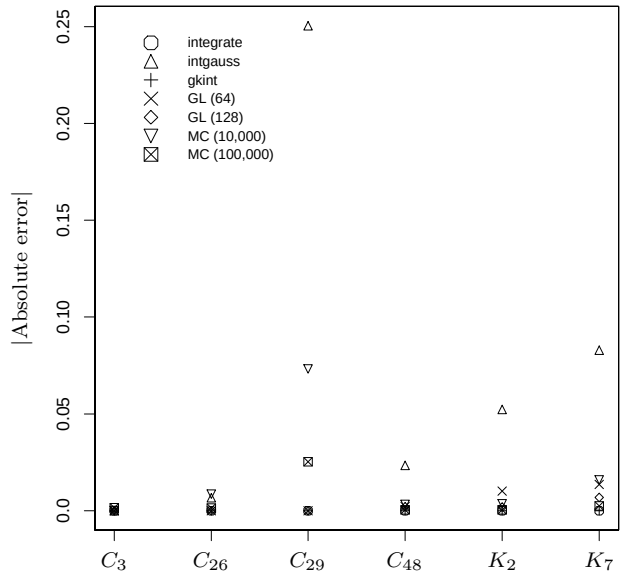

Figure 1 *Absolute integration errors for* $C_3, C_{26}, C_{29}, C_{48}, K_2, K_7$ *(given in Table 1) using the* S-PLUS *functions* `integrate`*,* `intgauss`*,* `gkint`*,* `GL.integrate.1D` *(GL) and* `MC.integrate.1D` *(MC) with the number of points in brackets.*

point Gaussian formula, the (7–15)-point Gauss–Kronrod method implemented in `gkint` and `GL.integrate.1D` given in Kuonen (2001, app. A, Table A.1) which uses the Gauss–Legendre (GL) rule with M points. And from Section 5 the function `MC.integrate.1D` (Kuonen, 2001, app. A, Table A.5).

In an extensive comparative study we applied them to the 71 test examples. We used GL with $4, 8, 16, 32, 64$ and 128 points, and MC with $10^3, 10^4, 10^5$ and 10^6 points. The polynomials were easily integrated with lower order (> 4) GL rules. For the other examples it appeared that a 64-point GL procedure is necessary to get reliable results. For the MC methods choices of the number of points below $10,000$ were unsatisfactory.

The most interesting of the 71 test functions are given in the upper and middle blocks of Table 1, where C_i denote the selected test integrals from Casaletto *et al.* (1969) and K_j the ones of Kahaner (1971). Note that $i = 3, 26, 29, 30, 34, 48$ or $j = 7, 15, 16, 21$ correspond to n in Evans' Tables 2.3 or 2.4 respectively. The resulting absolute errors of the procedures compared to their analytical values (right column of Table 1) are given in Figure 1 for $C_3, C_{26}, C_{29}, C_{48}, K_2$ and K_7, and in Table 2 for $C_{30}, C_{34}, K_{15}, K_{16}$ and K_{21}. As illustrated in Figure 1 the polynomial C_3 was easily found, as well as C_{26} which contains an oscillation in the denominator (Figure 2, top middle panel). Integrals C_{29} and C_{30}, C_{34} (Table 2), represented in the top right, bottom left and bottom middle panels of Figure 2, all exhibit oscillatory behaviour which gives only very high order methods any chance of success. This is especially true with C_{34} when MC integration and orders inferior to 128 are used. Similarly the discontinuities in C_{48} and K_2 did not cause surprising results in Figure 1. An oddity occurred with K_{21} (Table 2) which appears to defeat all the methods due to its nature shown in the bottom right panel of

Table 1 *A selection of test integrals used in the comparative study.*

Integrals	Analytic values
$C_3 = \int_0^1 (x^2 - 2x + 3)\, dx$	2.333333
$C_{26} = \int_0^1 2/\{2 + \sin(10\pi x)\}\, dx$	1.154700
$C_{29} = \int_0^{2\pi} x \sin(30x) \cos x\, dx$	−0.209672
$C_{30} = \int_0^{2\pi} x \sin(30x) \cos(50x)\, dx$	0.117809
$C_{34} = \int_0^{100\pi} \{(100\pi)^2 - x^2\}^{1/2} \sin x\, dx$	298.435716
$C_{48} = \int_0^1 c_{48}(x)\, dx,\ c_{48}(x) = \begin{cases} 1/(x+2), & 0 \le x \le e-2 \\ 0, & e-2 < x \le 1 \end{cases}$	0.306852
$K_2 = \int_0^1 k_2(x)\, dx,\ k_2(x) = \begin{cases} 0, & 0 \le x < 0.3 \\ 1, & 0.3 \le < x \le 1 \end{cases}$	0.7
$K_7 = \int_0^1 x^{-1/2}\, dx$	2
$K_{15} = \int_0^{10} 25 \exp(-25x)\, dx$	1
$K_{16} = \int_0^{10} 50/\{\pi(1 + 2500x^2)\}\, dx$	0.499363
$K_{21} = \int_0^1 k_{21}(x)\, dx,\ k_{21}(x) = [1/\cosh\{10(x-0.2)\}]^2 + [1/\cosh\{100(x-0.4)\}]^4 + [1/\cosh\{1000(x-0.6)\}]^6$	0.210802
$E_1 = \int_0^1 \int_0^1 1/(1 - x_1 x_2)\, dx_1 dx_2$	$\pi^2/6$
$E_3 = \int_{-1}^1 \int_{-1}^1 (2 - x_1 - x_2)^{-1/2}\, dx_1 dx_2$	$16(2 - \sqrt{2})/3$
$E_4 = \int_{-1}^1 \int_{-1}^1 (3 - x_1 - 2x_2)^{-1/2}\, dx_1 dx_2$	$4\sqrt{2}(3\sqrt{3} - 2\sqrt{2} - 1)/3$
$E_6 = \int_{-1}^1 \int_{-1}^1 \lvert x_1^2 + x_2^2 - 0.25 \rvert\, dx_1 dx_2$	$5/3 + \pi/16$
$E_7 = \int_{-1}^1 \int_{-1}^1 \lvert x_1 - x_2 \rvert^{1/2}\, dx_1 dx_2$	8/15

Figure 2.

The default S-PLUS function `integrate`, which implements uni-dimensional adaptive 15-point Gauss–Kronrod quadrature, and a 128-point GL rule, as implemented in `GL.integrate.1D`, performed best, and this within the entire comparative computation of the 71 test examples. The S-PLUS function `intgauss` and the MC rules were less accurate.

In order to enable the testing of the other adaptive methods described in Section 4 we considered the two-dimensional test integrals listed in Evans (1993, Table 6.2). A selection is given in the lower block of Table 1, where E_i denote the I_i in Evans' Table 6.2 for $i = 1, 3, 4, 6, 7$. We considered the following adaptive rules: the ADAPT routine (Genz and Malik, 1980) using the S-PLUS function `adapt` and DCUHRE (Berntsen *et al.*, 1991b) by means of the S-PLUS function `dcuhre`. We compared both with

Table 2 *Absolute integration errors for $C_{30}, C_{34}, K_{15}, K_{16}, K_{21}$ (given in Table 1) using the* S-PLUS *functions* `integrate`, `intgauss`, `gkint`, `GL.integrate.1D` *(GL) and* `MC.integrate.1D` *(MC) with the number of points in brackets. The values are rounded to three decimal places.*

	`integrate`	`intgauss`	`gkint`	GL (64)	GL (128)	MC (10,000)	MC (100,000)
C_{30}	0.000	3.186	0.000	0.239	0.176	0.228	0.064
C_{34}	0.000	5,476.455	0.000	3,425.142	0.000	886.795	131.527
K_{15}	0.000	0.681	0.000	0.000	0.000	0.022	0.024
K_{16}	0.000	0.362	0.000	0.001	0.000	0.061	0.047
K_{21}	0.211	0.211	0.210	0.211	0.211	0.210	0.210

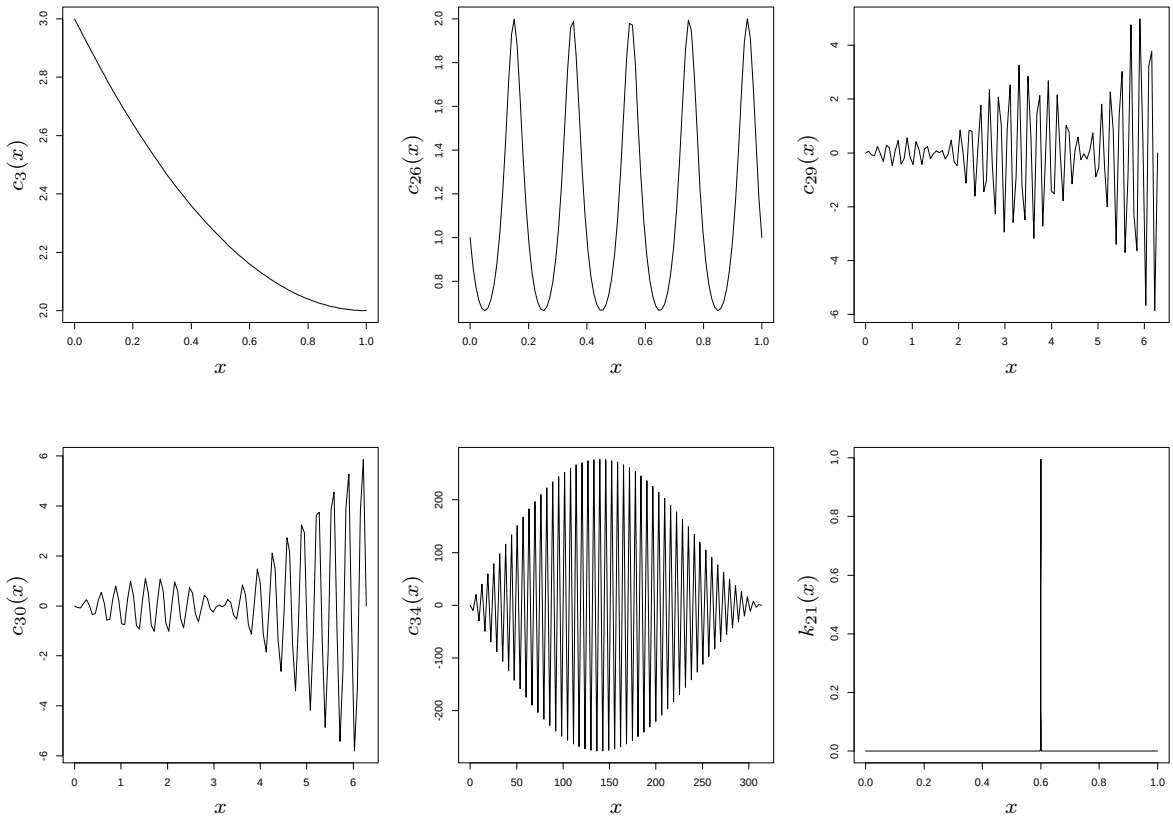

Figure 2 *Plot of the integrands of $C_3, C_{26}, C_{29}, C_{30}, C_{34}$ and K_{21} (clockwise from top left) given in Table 1.*

`GL.integrate.2D` (GL) and `MC.integrate.2D` (MC), given in Kuonen (2001, app. A,

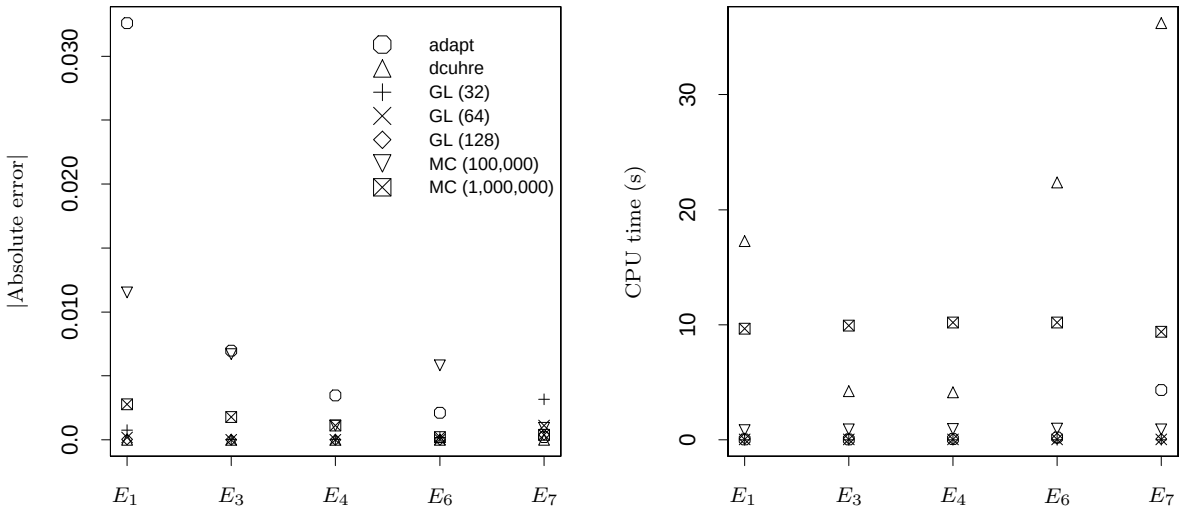

Figure 3 *Integration of E_1, E_3, E_4, E_6, E_7 (given in Table 1) using the* S-PLUS *functions* `adapt`*,* `dcuhre`*,* `GL.integrate.2D` *(GL) and* `MC.integrate.2D` *(MC) with the number of points in brackets. Left panel: Absolute integration errors. Right panel: CPU times in seconds.*

Tables A.3 and A.6), using several choices for the number of points per dimension. For example the use of a 32-point GL would result in 1024 function evaluations. The performance of these methods is illustrated in the left panel of Figure 3. The maximal absolute error achieved over all integrals was 0.0326 with E_1 using `adapt`. This illustrates that all methods work reasonably well when applied to E_i, $i = 1, 3, 4, 6, 7$. Nevertheless, `adapt` and the MC methods (even with one million points) perform slightly worse than the others, whereas `dcuhre` and a 128-point GL seem to be the most accurate methods under consideration. But the right panel of Figure 3 illustrates that the use of `dcuhre` results in significantly larger CPU times. This fact is not surprising for the MC methods as the number of points and hence the number of function evaluations is impractically large.

Once again, the use of a 128-point GL rule (`GL.integrate.2D`) delivered very accurate results within small CPU times as well as did `adapt`. Moreover, we noticed that `dcuhre` outperformed `adapt` in accuracy, but used much more CPU time. This was underlined with additional examples which are not given here.

7 Discussion

Numerical integration methods for use in S-PLUS or R were discussed and reviewed in this paper. However, questions on convergence and efficient implementation still remain. Some future approaches stated herein are promising.

As noted integrals over infinite domains should be transformed to a finite region in view of the accuracy and convergence of the quadrature method in use. This is especially true for adaptive integration algorithms which require repeated subdivision of

the integration region. When the integration is infinite, the point along the axis where the current subregion is to be cut is not clearly defined. Hence it is convenient to consider some appropriate transformations from the infinite integration region to a finite region. Then quadrature can be applied directly on the transformed integrand over the finite integration region. A number of simple one-variable transformations have been used for integration problems. Care must be taken in the selection of the transformation. As a check on consistency and efficiency we encourage use of several transformations for different computations of the same integral, and then comparison of their results.

An important point to realize is that when using quadrature methods, like the Gauss–Legendre rule, the only information such a method has about the integrand is a sequence of numerical values for it. To get a definite result for the integral, such a procedure then effectively has to make certain assumptions about the smoothness and other properties of the integrand. If a sufficiently pathological integrand is given, these assumptions may not be valid, and as a result, we may simply obtain the wrong answer. This problem may occur, for example, if one tries to integrate numerically a function which has a very thin peak at a particular position, like the integrand of K_{21} shown in the bottom right panel of Figure 2. The numerical integration routine samples the function at a number of points, and then assumes that the function varies smoothly between these points. As a result, if none of the sample points come close to the peak, then it will go undetected, and its contribution will not be correctly included. Therefore it is very important to get an idea of the effective range of the integrand in a preliminary analysis. But, if such a problem is thought to have arisen, one could bypass these problems using the split-t transformations proposed in Genz and Kass (1997) prior to the use of adaptive numerical integration algorithms.

Kuonen (2001, chap. 5–7) was devoted to the use of saddlepoint approximations in order to replace the computer-intensive bootstrap. The examples in the latter clearly illustrated the drawbacks of numerical integration for the computation of the distribution of studentized bootstrap distributions. They become useless in practice as their running time is outperformed by direct simulation of the bootstrap replicates. One may think that this is due to the use of interpreted languages like S-PLUS or R, but we do not think that this is the case as numerical integration in statistics, especially in multi-dimensional problems still raises many open questions. Nevertheless, it was illustrated in Kuonen (2001, sec. 6.5) that in such complex situations one should not use `integrate` as the computation may then become very time-intensive and may lead to inaccurate approximations. But, a GL rule with 128 points seems to be a good choice in practice.

Although the integrands discussed in Section 6 may not appear similar to integrands arising in statistics, the comparisons are nonetheless useful to statisticians. Further information on the numerical computation of multiple integrals in statistics may be found in Evans and Swartz (2000).

Acknowledgement

The author expresses his deep gratitude to A. C. Davison for his guidance when writing his PhD thesis (Kuonen, 2001) and thanks him for his suggestions relating to this paper. The work was supported by a grant from the Swiss National Science Foundation.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). *The New S Language*. London: Chapman & Hall.

Berntsen, J., Espelid, T. O. and Genz, A. (1988). *A Test of ADMINT*. Reports in Informatics 31, Department of Informatics, University of Bergen, Bergen, Norway.

Berntsen, J., Espelid, T. O. and Genz, A. (1991a). An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Transactions on Mathematical Software*, 17, 437–451.

Berntsen, J., Espelid, T. O. and Genz, A. (1991b). Algorithm 698: DCUHRE — An adaptive multidimensional integration routine for a vector of integrals. *ACM Transactions on Mathematical Software*, 17, 452–456.

Cappé, O. and Robert, C. P. (2000). Ten years and still running! *Journal of the American Statistical Association*, 95, 1282–1286.

Casaletto, J., Pickett, M. and Rice, J. R. (1969). A comparison of some numerical integration programs. *SIGNUM Newsletter*, 4, 30–40.

Chambers, J. M. (1998). *Programming with Data: A Guide to the S Language*. New York: Springer.

Davis, P. J. and Rabinowitz, P. (1984). *Methods of Numerical Integration*. New York: Academic Press.

Engels, H. (1980). *Numerical Quadrature and Cubature*. New York: Academic Press.

Evans, G. (1993). *Practical Numerical Integration*. New York: Wiley.

Evans, M. and Swartz, T. (1995). Methods for approximating integrals in statistics with special emphasis on Bayesian integration problems. *Statistical Science*, 10, 254–272.

Evans, M. and Swartz, T. (2000). *Approximating Integrals via Monte Carlo and Deterministic Methods*. Oxford: Oxford University Press.

Fang, K. T. and Wang, Y. (1994). *Number-theoretic Methods in Statistics*. London: Chapman & Hall.

Fang, K. T., Wang, Y. and Bentler, P. M. (1994). Some applications of number-theoretic methods in statistics. *Statistical Science*, 9, 416–428.

Genz, A. (1991). An adaptive numerical integration algorithm for simplices. In *Computing in the 90s, Proceedings of the First Great Lake Computer Science Conference, N. A. Sherwani, E. de Doncker and J. A. Kapenga (eds.)*. Lecture Notes in Computer Science, 507, New York: Springer, pp. 279–292.

Genz, A. (1992). Statistics applications of subregion adaptive multiple numerical integration. In *Numerical Integration: Recent Developments, Software and Applications, T. O. Espelid and A. Genz (eds.)*. Dordrecht: Kluwer Academic Publishers, pp. 267–280.

Genz, A. and Kass, R. E. (1997). Subregion-adaptive integration of functions having a dominant peak. *Journal of Computational and Graphical Statistics*, 6, 92–111.

Genz, A. and Malik, A. A. (1980). An adaptive algorithm for numerical integration over an N-dimensional rectangular region. *Journal of Computational and Applied Mathematics*, 6, 295–302.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J. (eds.) (1996). *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

Ihaka, R. and Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5, 299–314.

Kalos, M. H. and Whitlock, P. A. (1986). *Monte Carlo Methods Volume 1: Basics*. New York: Wiley.

Kahaner, D. K. (1971). Comparison of numerical quadrature formulae. In *Mathematical Software, J. R. Rice (ed.)*. New York: Academic Press, 229–259.

Kuonen, D. (2001). *Computer-intensive statistical methods: saddlepoint approximations with applications in bootstrap and robust inference*. PhD Thesis N. 2449, Department of Mathematics, Swiss Federal Institute of Technology, CH–1015 Lausanne. (`stat.kuonen.com/thesis/`)

Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. and Kahaner, D. (1983). *QUADPACK: A Subroutine Package for Automatic Integration*. Berlin: Springer.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1993). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

Robert, C. P. and Casella, G. (1999). *Monte Carlo Statistical Methods*. New York: Springer.

Stroud, A. H. (1971). *Approximation Caculation of Multiple Integrals*. New Jersey: Prentice–Hall.

Van Dooren, P. and de Ridder, L. (1976). An adaptive algorithm for numerical integration over an N-dimensional cube. *Journal of Computational and Applied Mathematics*, 2, 207–217.

Venables, W. N. and Ripley, B. D. (2000). *S Programming*. New York: Springer.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S-Plus* (4th ed.). New York: Springer.