Adaptive Quadrature - Revisited

WALTER GANDER AND WALTER GAUTSCHI
ETH Zürich

Dedicated to Cleve B. Moler on his 60th birthday

Abstract

First, the basic principles of adaptive quadrature are reviewed. Adaptive quadrature programs being recursive by nature, the choice of a good termination criterion is given particular attention. Two Matlab quadrature programs are presented. The first is an implementation of the well-known adaptive recursive Simpson's rule; the second is new and is based on a four-point Gauss-Lobatto formula and two successive Kronrod extensions. Comparative test results are described and attention is drawn to serious deficiencies in the adaptive routines `quad` and `quad8` provided by Matlab.

1 The basic idea of adaptive quadrature

Let $[a, b]$ be the interval of integration, assumed to be bounded, and f a real integrable function. To compute

$$I = \int_a^b f(x)dx,$$

one generally proceeds as follows (see also, e.g., [2, Ch. 6]). First one integrates f using two different numerical integration methods, thus obtaining the approximations I_1 and I_2. Typically, one, say I_1, is more accurate than the other. If the relative difference of the two approximations is smaller than some prescribed tolerance, one accepts I_1 as the value of the integral. Otherwise the interval $[a, b]$ is divided, e.g., in two equal parts $[a, m]$ and $[m, b]$, where $m = (a + b)/2$, and the two respective integrals are computed independently:

$$I = \int_a^m f(x)dx + \int_m^b f(x)dx.$$

One now again computes recursively two approximations for each integral and, if necessary, continues to subdivide the smaller intervals.

As is well known, it is impossible to construct a program that is foolproof, i.e., that correctly integrates any given function [1]. It is easy to construct a function f which a given program will not integrate correctly [10]. Our task is therefore to design an algorithm that works well for as many functions as possible.

M.T. Heath writes in his recent textbook [9]:

> If the integrand is noisy, or if the error tolerance is unrealistically tight relative to the machine precision, then an adaptive quadrature routine may be unable to meet the error tolerance and will likely expend a large number of function evaluations only to return a warning message that its subdivision limit was exceeded. Such a result should not be regarded as a fault of the adaptive routine but as a reflection of the difficulty of the problem or unrealistic expectations on the part of the user, or both.

This is true for the two adaptive quadrature functions `quad` and `quad8` provided by Matlab. However, our routines will show that Heath's assessment is overly pessimistic in general – we can do much better than what he says.

2 Termination criterion

If I_1 and I_2 are two estimates for the integral, a conventional stopping criterion is

```
if abs(i1-i2) < tol*abs(i1),
```
(1)

where *tol* is some prescribed error tolerance. This criterion by itself is not sufficient. Consider for example $f(x) = \sqrt{x}$, the integration interval $[0, 1]$, I_2: Simpson's rule using the step size h, I_1: Simpson's rule for the step size $h/2$ and $tol = 10^{-4}$. If implemented in Matlab, this procedure terminates fatally with the error message `Segmentation fault`. In this example the two Simpson values never agree to 4 decimals in the first interval containing 0.

The function `quad` in Matlab is based on an adaptive recursive Simpson's rule. The stopping criterion (1) is supplemented by a limitation on the number of recursion levels. This prevents failure in the example above. However, it is not clear how many recursion levels should be allowed, and the value `LEVMAX = 10` used in `quad` is often inadequate. A warning error message is given if the recursion level limit is reached. In case of $f(x) = \sqrt{x}$ and $[a, b] = [0, 1]$ we obtain with the call `quad('f',0,1,1e-12)` the warning `Recursion level limit reached 1024 times` and the value 0.666665790717630 is returned, which is correct to only 6 digits instead of the requested 12 digits.

We somehow have to terminate the recursion if the magnitude of the partial integral I_1 or I_2 is negligible compared to the whole integral (see [4, p. 209], [3]). Therefore, we have to add the criterion

$$|I_1| < \eta \left| \int_a^b f(x)dx \right|, \tag{2}$$

where η is another prescribed tolerance and where we have to use an estimate for the unknown integral. With both criteria (1) and (2) used together, and with some reasonable choices of tol and η, a working algorithm can be obtained. If, e.g., for the above example we use

```
if (abs(i1-i2) < 1e-4*abs(i1)) | (abs(i1<1e-4))   (3)
```

as stopping criterion, we obtain $I = 0.666617217$ in 41 function evaluations.

The stopping criterion (3), however, is still not satisfactory because the user has to choose tol and η, which depend on the machine and on the problem. A wrong selection is easily possible.

To improve the criterion, we first need a rough estimate

$$\texttt{is},\ \texttt{is} \neq 0, \tag{4}$$

of the modulus of the integral I. The stopping criterion (2) would then be $|I_1| < \eta \cdot is$. In order to eliminate η, we stop the recursion machine-independently by

```
if is + i1 == is.                                    (5)
```

In the same spirit we may as well replace the criterion (1) by

```
if is + (i1-i2) == is.                               (6)
```

The criterion (6) will in general be met before the criterion (5) and therefore we shall require only (6). There are cases, e.g., $\int_0^1 \frac{1}{\sqrt{1-x^2}}\,dx$, where, when ignoring the singularity, the subdivision will continue until an interval contains no machine number other than the end points. In this case we also need to terminate the recursion. Thus, our termination criterion is

```
if (is + (i1-i2) == is) | (m <= a) | (b<=m),         (7)
```

where $m = (a+b)/2$. This, in particular, guarantees termination of the program, and an explicit limitation on the number of recursion levels is no longer necessary.

Using the stopping criterion (7), we attempt to compute the integral to machine precision. If we wish to compute the integral with less accuracy, say within the tolerance *tol* , it suffices to magnify the estimated value is:

```
is = is*tol/eps,
```

where `eps` denotes the machine precision.

3 Adaptive Simpson quadrature

The idea of adaptive Simpson quadrature is old [12]. However, in order to obtain good performance, a careful implementation is necessary. The Matlab function `quad` compares two successive Simpson values (relative and absolute difference) and has a limitation on the number of recursion steps. If we compute $\int_0^1 \sqrt{x}\,dx$ with $tol = 10^{-8}$ we obtain the message "`Warning: Recursion` `level limit reached in quad.  Singularity likely.`". The routine returns 0.6666657907152264 (a value correct to only 6 digits) and needs 800 function evaluations.

First, we propose to use for is a Monte Carlo estimate which also uses the function values in the middle and at the end points of the interval (those values are used for Simpson's rule):

$$is = \frac{b-a}{8}(f(a) + f(m) + f(b) + \sum_{i=1}^{5} f(\xi_i)). \tag{8}$$

Here $m = (a+b)/2$ and $\xi = a + [.9501\ .2311\ .6068\ .4860\ .8913](b-a)$ is a vector of random numbers in (a, b). If by accident we get $is = 0$, then we use the value $is = b - a$.

With this choice of is, we adopt the stopping criterion (7). Furthermore, we do not compare successive Simpson values $i1 = S(h)$ and $i2 = S(h/2)$ but overwrite $i1$ with one step of Romberg extrapolation:

```
i1 = (16*i2 - i1)/15.
```

In order to avoid recomputation of function values, we pass $fa = f(a)$, $fm = f((a+b)/2)$ and $fb = f(b)$ as parameters. In every recursion step, only two new function evaluations are necessary to compute the approximations $i1$ and $i2$. The following Matlab function `adaptsim` has the same structure as `quad`. For $\int_0^1 \sqrt{x}\ dx$ with $tol = 10^{-8}$ we obtain with `adaptsim` the value 0.6666666539870345 (correct to almost 8 digits) using only 126 function evaluations.

```
function Q = adaptsim(f,a,b,tol,trace,varargin)
%ADAPTSIM  Numerically evaluate integral using adaptive
%   Simpson rule.
%
%   Q = ADAPTSIM('F',A,B) approximates the integral of
%   F(X) from A to B to machine precision.  'F' is a
%   string containing the name of the function. The
%   function F must return a vector of output values if
%   given a vector of input values.
%
%   Q = ADAPTSIM('F',A,B,TOL) integrates to a relative
```

```
%   error of TOL.
%
%   Q = ADAPTSIM('F',A,B,TOL,TRACE) displays the left
%   end point of the current interval, the interval
%   length, and the partial integral.
%
%   Q = ADAPTSIM('F',A,B,TOL,TRACE,P1,P2,...) allows
%   coefficients P1, ... to be passed directly to the
%   function F:  G = F(X,P1,P2,...). To use default values
%   for TOL or TRACE, one may pass the empty matrix ([]).
%
%   See also ADAPTSIMSTP.
%
%   Walter Gander, 08/03/98
%   Reference: Gander, Computermathematik, Birkhaeuser, 1992.

global termination2

  termination2 = 0;
  if (nargin < 4), tol = []; end;
  if (nargin < 5), trace = []; end;
  if (isempty(tol)), tol = eps; end;
  if (isempty(trace)), trace = 0; end;
  if tol<eps
      tol= eps;
  end
  x = [a (a+b)/2 b];
  y = feval(f, x, varargin{:});
  fa = y(1); fm = y(2); fb = y(3);
  yy = feval(f, a+[.9501 .2311 .6068 .4860 .8913]*(b-a), ...
       varargin{:});
  is = (b - a)/8*(sum(y)+sum(yy));
  if is==0, is = b-a; end;
  is = is*tol/eps;
  Q = adaptsimstp(f,a,b,fa,fm,fb,is,trace,varargin{:});


function Q = adaptsimstp (f,a,b,fa,fm,fb,is,trace,varargin)
```

```
%ADAPTSIMSTP  Recursive function used by ADAPTSIM.
%
%   Q = ADAPTSIMSTP('F',A,B,FA,FM,FB,IS,TRACE) tries to
%   approximate the integral of F(X) from A to B to
%   an appropriate relative error. The argument 'F' is
%   a string containing the name of f. The remaining
%   arguments are generated by ADAPTSIM or by recursion.
%
%   See also ADAPTSIM.
%
%   Walter Gander, 08/03/98

global termination2

  m = (a + b)/2; h = (b - a)/4;
  x = [a + h, b - h];
  y = feval(f, x, varargin{:});
  fml = y(1); fmr = y(2);
  i1 = h/1.5 * (fa + 4*fm + fb);
  i2 = h/3 * (fa + 4*(fml + fmr) + 2*fm + fb);
  i1 = (16*i2 - i1)/15;
  if (is + (i1-i2) == is) | (m <= a) | (b<=m),
    if ((m <= a) | (b<=m)) & (termination2==0);
       warning(['Interval contains no more machine number. ',...
                'Required tolerance may not be met.']);
       termination2 =1;
    end;
    Q = i1;
    if (trace), disp([a b-a Q]), end;
  else
    Q = adaptsimstp (f,a,m,fa,fml,fm,is,trace,varargin{:}) + ...
        adaptsimstp (f,m,b,fm,fmr,fb,is,trace,varargin{:});
  end;
```

The minimal number of function evaluations is 10, which is attained if the error test is met in the very first call to `adaptsimstp`.

Discontinuous functions are integrated quite well by `adaptsim`.

For example, if we integrate

$$f(x) = \begin{cases} x+1, & x<1 \\ 3-x, & 1 \le x \le 3 \\ 2, & x>3 \end{cases} \tag{9}$$

on $[0,5]$ with `adaptsim('f',0,5,1e-6)`, i.e., with $tol = 10^{-6}$, we obtain instead of the exact value 7.5 the value 7.49996609147638 with 98 function evaluations. Using `quad` with the same tolerance $tol = 10^{-6}$, one obtains the value 7.50227769215902 (correct to only 3 digits) with 88 function evaluations. The difference in performance is due to the different termination criteria and the artificial limitation to 10 recursion levels used in `quad`.

4 Adaptive Lobatto quadrature

4.1 The basic quadrature rule

As basic quadrature rule we use the Gauss-Lobatto rule with two (symmetric) interior nodes. On the canonical interval $[-1,1]$, the two interior nodes are the zeros of $\pi_2(x)$, where

$$\int_{-1}^{1} (1-x^2)\pi_2(x)p(x)dx = 0 \quad \text{for all} \;\; p \in \mathbb{P}_1.$$

Thus, up to a constant factor, π_2 is the Jacobi polynomial $P_2^{(\alpha,\beta)}$ of degree 2 corresponding to parameters $\alpha = \beta = 1$. Since $P_2^{(1,1)}(x) = \text{const} \cdot (x^2 - \frac{1}{5})$, the interior nodes are $x_{\pm 1} = \pm\frac{1}{\sqrt{5}}$. By symmetry, the formula has the form

$$\int_{-1}^{1} f(x)dx = \;\; a[f(-1)+f(1)] + b[f(-\tfrac{1}{\sqrt{5}}) + f(\tfrac{1}{\sqrt{5}})] + R^{GL}(f),$$

where

$$R^{GL}(f) = 0 \quad \text{for} \;\; f \in \mathbb{P}_5.$$

Exactness for $f(x) = 1$ and $f(x) = x^2$ yields

$$\left.\begin{array}{rcl} 2a + 2b & = & 2 \\ 2a + \frac{2}{5}b & = & \frac{2}{3} \end{array}\right\} \quad \text{hence} \quad a = \tfrac{1}{6},\; b = \tfrac{5}{6}.$$

Thus, the basic quadrature rule on $[-1,1]$ is

$$\int_{-1}^{1} f(x)dx = \tfrac{1}{6}[f(-1)+f(1)]+\tfrac{5}{6}[f(-\tfrac{1}{\sqrt{5}})+f(\tfrac{1}{\sqrt{5}})]+R^{GL}(f). \quad (10)$$

We note that using Maple one can compute the basic quadrature rule directly by means of the ansatz $a(f(-1)+f(1)) + b(f(-x1)+f(x1))$ and requiring that it be exact for $f(x) = 1, x^2$ and x^4:

```
u1 := 2*a +2*b:
u2 := 2*a + 2*b*x1^2:
u3 :=  2*a + 2*b*x1^4:
solve({u1=2, u2=2/3, u3=2/5},{a,b,x1});
```

The result is:

$$\left\{ \mathit{x1} = \mathit{RootOf}(5\_Z^2-1),\ a = 1/6,\ b = 5/6 \right\},$$

in agreement with (10).

4.2 Kronrod extension of the Gauss-Lobatto formula

To estimate the error of (10) we construct the Kronrod extension of (10). (For background information and history on Gauss-Kronrod extensions see, e.g., [7].) By a well-known theorem on quadrature rules of maximum algebraic degree of exactness (cf. [8, Theorem 3.2.1]), the three Kronrod points are the zeros of $\pi_3^*(x)$, a (monic) polynomial of degree 3 satisfying

$$\int_{-1}^{1} (1-x^2)\pi_2(x)\pi_3^*(x)p(x)dx = 0 \quad \text{for all} \quad p \in \mathbb{P}_2.$$

Here, $\pi_2(x) = x^2 - \frac{1}{5} =: \pi_1(x^2)$, and by symmetry

$$\pi_3^*(x) = x\pi_1^*(x^2)$$

for some $\pi_1^* \in \mathbb{P}_1$. It suffices, therefore, to choose $\pi_1^*(x^2)$ such that

$$\int_{-1}^{1} (1-x^2)\pi_1(x^2)\pi_1^*(x^2)x^2dx = 0.$$

Putting $x^2 = t$ yields

$$\int_0^1 (1-t)\pi_1(t)\pi_1^*(t)t^{\frac{1}{2}}dt = 0,$$

and, with $\pi_1^*(t) = t - c$, we obtain

$$\int_0^1 (1-t)(t-\tfrac{1}{5})(t-c)t^{\frac{1}{2}}dt = 0,$$

that is,

$$c\int_0^1 t^{\frac{1}{2}}(t^2 - \tfrac{6}{5}t + \tfrac{1}{5})dt = \int_0^1 t^{\frac{3}{2}}(t^2 - \tfrac{6}{5}t + \tfrac{1}{5})dt,$$

$$32c = \tfrac{64}{3}, \quad c = \tfrac{2}{3}.$$

The three Kronrod points, therefore, are

$$x_{\pm 1}^* = \pm\sqrt{\tfrac{2}{3}}, \quad x_0^* = 0.$$

By symmetry, the Kronrod extension has the form

$$\begin{aligned}\int_{-1}^1 f(x)dx = \; & A[f(-1)+f(1)] + B[f(-\sqrt{\tfrac{2}{3}}) + f(\sqrt{\tfrac{2}{3}})] \\ & +C[f(-\tfrac{1}{\sqrt{5}}) + f(\tfrac{1}{\sqrt{5}})] + Df(0) + R^{GLK}(f),\end{aligned}$$

where

$$R^{GLK}(f) = 0 \quad \text{for} \quad f \in \mathbb{P}_9.$$

Exactness for $f(x) = 1, x^2, x^4, x^6$ yields

$$\begin{array}{lllll} 2A & +2B & +2C & +D & = 2, \\ 2A & +2\cdot\tfrac{2}{3}B & +2\cdot\tfrac{1}{5}C & & = \tfrac{2}{3}, \\ 2A & +2\cdot\tfrac{4}{9}B & +2\cdot\tfrac{1}{25}C & & = \tfrac{2}{5}, \\ 2A & +2\cdot\tfrac{8}{27}B & +2\cdot\tfrac{1}{125}C & & = \tfrac{2}{7}. \end{array}$$

Gauss elimination gives

$$A = \tfrac{11}{210}, \quad B = \tfrac{72}{245}, \quad C = \tfrac{125}{294}, \quad D = \tfrac{16}{35}.$$

Thus,

$$\int_{-1}^{1} f(x)dx = \tfrac{11}{210}[f(-1)+f(1)] + \tfrac{72}{245}[f(-\sqrt{\tfrac{2}{3}}) + f(\sqrt{\tfrac{2}{3}})] + \tfrac{125}{294}[f(-\tfrac{1}{\sqrt{5}}) + f(\tfrac{1}{\sqrt{5}})] + \tfrac{16}{35}f(0) + R^{GLK}(f). \qquad (11)$$

Again, we can compute this extension directly, using Maple and the ansatz $A[f(-1)+f(1)]+B[f(-x1)+f(x1)]+C[f(-1/\sqrt{5})+f(1/\sqrt{5})]+Df(0)$, requiring exactness for $f(x)=1, x^2, x^4, x^6$ and x^8:

```
x2 := 1/sqrt(5);
u1 := 2*A + 2*B +2*C +D = 2;
u2 := 2*A + 2*B*x1^2 + 2*C*x2^2 = 2/3;
u3 := 2*A + 2*B*x1^4 + 2*C*x2^4 = 2/5;
u4 := 2*A + 2*B*x1^6 + 2*C*x2^6 = 2/7;
u5 := 2*A + 2*B*x1^8 + 2*C*x2^8 = 2/9;
solve({u1,u2,u3,u4,u5},{A,B,C,D,x1});
```

The result, as above, is:

$$\left\{B = \frac{72}{245},\ A = \frac{11}{210},\ D = \frac{16}{35},\ C = \frac{125}{294},\ x1 = RootOf(3\_Z^2 - 2)\right\}.$$

4.3 Kronrod extension of (11)

It will be desirable to estimate how much more accurate (11) is compared to (10). We try to estimate the respective errors by constructing a Kronrod extension of (11), hoping that one exists with real nodes and positive weights. There will be six symmetrically located Kronrod points $\pm x_1, \pm x_2, \pm x_3$, which, it is hoped, interlace with the nodes of (11). Again, on the basis of [8, Theorem 3.2.1], with $n = 13$ and

$$\omega_n(x) = (x^2-1)(x^2-\tfrac{2}{3})(x^2-\tfrac{1}{5})x\pi_6^*(x),$$
$$\pi_6^*(x) = (x^2-x_1^2)(x^2-x_2^2)(x^2-x_3^2),$$

the 13-point quadrature rule to be constructed will have degree of exactness $d = 12 + k$ provided that π_6^* is chosen to satisfy the "orthogonality" condition

$$\int_{-1}^{1}(x^2-1)(x^2-\tfrac{2}{3})(x^2-\tfrac{1}{5})x\pi_6^*(x)p(x)dx=0 \quad \text{for all} \quad p\in\mathbb{P}_{k-1}.$$

The optimal value of k is $k=6$, yielding a formula of degree $d=18$ (actually, $d=19$ because of symmetry). If we let

$$\alpha_i=x_i^2,\quad i=1,2,3,$$

and make the substitution $x^2=t$, $\pi_6^*(x)=\pi_3^*(x^2)$, the orthogonality relation becomes

$$\int_{0}^{1}(t-1)(t-\tfrac{2}{3})(t-\tfrac{1}{5})\sqrt{t}\pi_3^*(t)p(t)dt=0 \quad \text{for all} \quad p\in\mathbb{P}_2,$$

where

$$\pi_3^*(t)=(t-\alpha_1)(t-\alpha_2)(t-\alpha_3)=t^3-at^2+bt-c.$$

Putting $p(t)=1,t,t^2$ in this relation, one finds, after some tedious calculations, that the coefficients a, b, c must satisfy

$$\begin{array}{rrrl} 30a & -13b & & =35,\\ 595a & -510b & +221c & =588,\\ 11172a & -11305b & +9690c & =10395. \end{array}$$

Solving for a, b, c gives

$$a=\tfrac{37975}{27987},\quad b=\tfrac{4095}{9329},\quad c=\tfrac{9737}{475779},$$

and then solving the cubic equation $\pi_3^*(t)=0$ yields, with the help of Maple, to 38 decimal digits,

$$\begin{aligned}\alpha_1&=.88902724982774341965844097377815423496,\\ \alpha_2&=.41197571308045073755318461761021278774,\\ \alpha_3&=.05587701708251581527560062078156901902 6,\end{aligned}$$

and hence

$$\begin{aligned}x_1=\sqrt{\alpha_1}&=.94288241569547971905635175843185720232,\\ x_2=\sqrt{\alpha_2}&=.64185334234578130578123554132903188354,\\ x_3=\sqrt{\alpha_3}&=.23638319966214988028222377349205292599.\end{aligned}$$

It is a fortunate circumstance that the α_i turn out to be all positive, hence also the x_i, and moreover the $\pm x_i$ interlace with the nodes of (11).

Alternatively, Maple can be used to compute the zeros of π_3^* directly as follows:

```
restart;
Digits :=40;
pis := t->(t-a1)*(t-a2)*(t-a3);
sols := solve({int((t-1)*(t-2/3)*(t-1/5)*sqrt(t)*pis(t),t=0..1)=0,
       int((t-1)*(t-2/3)*(t-1/5)*sqrt(t)*pis(t)*t,t=0..1)=0,
       int((t-1)*(t-2/3)*(t-1/5)*sqrt(t)*pis(t)*t^2,t=0..1)=0},
       {a1,a2,a3}):
evalf(sols);
```

The desired Kronrod extension has the form

$$\begin{aligned}\int_{-1}^{1} f(x)dx &= A[f(-1)+f(1)] + B[f(-x_1)+f(x_1)] + C[f(-\sqrt{\tfrac{2}{3}}) + f(\sqrt{\tfrac{2}{3}})] \\ &\quad + D[f(-x_2)+f(x_2)] + E[f(-\tfrac{1}{\sqrt{5}}) + f(\tfrac{1}{\sqrt{5}})] + F[f(-x_3)+f(x_3)] \\ &\quad + Gf(0) + R^{GLKK}(f), \quad R^{GLKK}(\mathbb{P}_{19}) = 0. \qquad (12)\end{aligned}$$

Exactness for the first seven powers of x^2 yields, after division by 2, the system

$$\begin{aligned}
A + B + C + D + E + F + \tfrac{1}{2}G &= 1, \\
A + \alpha_1 B + \tfrac{2}{3}C + \alpha_2 D + \tfrac{1}{5}E + \alpha_3 F &= \tfrac{1}{3}, \\
A + \alpha_1^2 B + \tfrac{4}{9}C + \alpha_2^2 D + \tfrac{1}{25}E + \alpha_3^2 F &= \tfrac{1}{5}, \\
A + \alpha_1^3 B + \tfrac{8}{27}C + \alpha_2^3 D + \tfrac{1}{125}E + \alpha_3^3 F &= \tfrac{1}{7}, \\
A + \alpha_1^4 B + \tfrac{16}{81}C + \alpha_2^4 D + \tfrac{1}{625}E + \alpha_3^4 F &= \tfrac{1}{9}, \\
A + \alpha_1^5 B + \tfrac{32}{243}C + \alpha_2^5 D + \tfrac{1}{3125}E + \alpha_3^5 F &= \tfrac{1}{11}, \\
A + \alpha_1^6 B + \tfrac{64}{729}C + \alpha_2^6 D + \tfrac{1}{15625}E + \alpha_3^6 F &= \tfrac{1}{13}.
\end{aligned}$$

The solution is, to 38 digits,

$$
\begin{aligned}
A &= .01582719197348018308716998673330551059 1,\\
B &= .09427384021885004553128250507710817196 0,\\
C &= .15507198733658539625363597980210298680,\\
D &= .18882157396018245442000533937297167125,\\
E &= .19977340522685852679206802206648840246,\\
F &= .22492646533333952701601768799639508076,\\
G &= .24261107190140773379964095790325635233.
\end{aligned}
$$

By good fortune, it consists of entirely positive entries.

Note that even here we can use Maple to obtain the result by "brute force". Using the ansatz (12) with unknown knots a_1, a_2 and a_3, and requiring that it be exact for the monomials $1, x^2, x^4, \ldots x^{18}$, we obtain 10 nonlinear equations in 10 unknowns:

```
x1:=sqrt(2/3); x2:=sqrt(1/5);
u1:=2*A+2*B+2*C+2*D+2*E+2*F+G = 2;
u2:=2*A+2*B*a1^2+2*C*x1^2+2*D*a2^2+2*E*x2^2+2*F*a3^2 = 2/3;
u3:=2*A+2*B*a1^4+2*C*x1^4+2*D*a2^4+2*E*x2^4+2*F*a3^4 = 2/5;
u4:=2*A+2*B*a1^6+2*C*x1^6+2*D*a2^6+2*E*x2^6+2*F*a3^6 = 2/7;
u5:=2*A+2*B*a1^8+2*C*x1^8+2*D*a2^8+2*E*x2^8+2*F*a3^8 = 2/9;
u6:=2*A+2*B*a1^10+2*C*x1^10+2*D*a2^10+2*E*x2^10+2*F*a3^10 = 2/11;
u7:=2*A+2*B*a1^12+2*C*x1^12+2*D*a2^12+2*E*x2^12+2*F*a3^12 = 2/13;
u8:=2*A+2*B*a1^14+2*C*x1^14+2*D*a2^14+2*E*x2^14+2*F*a3^14 = 2/15;
u9:=2*A+2*B*a1^16+2*C*x1^16+2*D*a2^16+2*E*x2^16+2*F*a3^16 = 2/17;
u10:=2*A+2*B*a1^18+2*C*x1^18+2*D*a2^18+2*E*x2^18+2*F*a3^18 = 2/19;
sols:=solve({u1,u2,u3,u4,u5,u6,u7,u8,u9,u10},
            {A,B,C,D,E,F,G,a1,a2,a3});
```

Maple solves this system in 7 minutes on a SUN Sparcstation 20/514 (50 Mhz SuperSparc processor) and gives a solution containing very complicated expressions (several pages long). However, evaluating the expressions as floating point numbers (`Digits:=15; evalf(sols);`) yields (rounded to 10 digits)

```
{a1 = -.2363831997, a2 = -.6418533423, E = .1997734052,
 a3 = -.9428824157, D = .1888215742, F = .09427384020,
 G = .2426110719, B = .2249264653, C = .1550719873,
 A = .01582719197},
```

a permutation of the solution given above.

4.4 The adaptive procedure

For an arbitrary interval $[a,b]$, the formulae (10) and (11) can be written respectively as

$$\int_a^b f(x)dx \approx \frac{h}{6}\{f(a)+f(b)+5[f(m-\beta h)+f(m+\beta h)]\} \quad (13)$$

and

$$\int_a^b f(x)dx \approx \frac{h}{1470}\{77[f(a)+f(b)]+432[f(m-\alpha h)+f(m+\alpha h)]$$
$$+625[f(m-\beta h)+f(m+\beta h)]+672f(m)\}, \quad (14)$$

where

$$h=\tfrac{1}{2}(b-a), \quad m=\tfrac{1}{2}(a+b),$$
$$\alpha=\sqrt{\tfrac{2}{3}}, \quad \beta=\tfrac{1}{\sqrt{5}}.$$

A similar reformulation holds for (12).

The adaptive Lobatto procedure is similar to the adaptive Simpson procedure of §3, with the second Kronrod extension (i.e., the formula (12) relative to the initial interval $[a,b]$) providing the estimate is, and (13) and (14) playing the roles of $i2$ and $i1$, respectively. There are three additional features, however:

(i) If the ratio ρ of the error of (14) and the error of (13), as determined for the initial interval $[a,b]$ by comparison with is, is less than 1, then the basic error tolerance tol is relaxed to tol/ρ, since we always accept the more accurate approximation (14). (A similar relaxation of the tolerance has already been suggested by Lyness in [12, Modification 1].)

(ii) At each recursive level, the current interval $[a,b]$ is subdivided into six subintervals when the error tolerance is not met, namely the intervals $[a,m-\alpha h]$, $[m-\alpha h,m-\beta h]$, $[m-\beta h,m]$, $[m,m+\beta h]$, $[m+\beta h,m+\alpha h]$, $[m+\alpha h,b]$ determined by (13) and (14). In this way, all function values computed are being reused.

(iii) Consistent with (ii), the termination criterion (7) is modified by replacing the last two conditions by $m - \alpha h \leq a$ and $b \leq m + \alpha h$, respectively.

The adaptive Lobatto procedure requires five new values of f to be computed at each level of the recursion.

4.5 Matlab code

The adaptive Lobatto procedure is implemented by the recursive Matlab program below.

```
function Q=adaptlob(f,a,b,tol,trace,varargin)
%ADAPTLOB  Numerically evaluate integral using adaptive
%   Lobatto rule.
%
%   Q=ADAPTLOB('F',A,B) approximates the integral of
%   F(X) from A to B to machine precision.  'F' is a
%   string containing the name of the function. The
%   function F must return a vector of output values if
%   given a vector of input values.
%
%   Q=ADAPTLOB('F',A,B,TOL) integrates to a relative
%   error of TOL.
%
%   Q=ADAPTLOB('F',A,B,TOL,TRACE) displays the left
%   end point of the current interval, the interval
%   length, and the partial integral.
%
%   Q=ADAPTLOB('F',A,B,TOL,TRACE,P1,P2,...) allows
%   coefficients P1, ... to be passed directly to the
%   function F:  G=F(X,P1,P2,...). To use default values
%   for TOL or TRACE, one may pass the empty matrix ([]).
%
%   See also ADAPTLOBSTP.

%   Walter Gautschi, 08/03/98
%   Reference: Gander, Computermathematik, Birkhaeuser, 1992.
```

```
global termination2

  termination2 = 0;
  if(nargin<4), tol=[]; end;
  if(nargin<5), trace=[]; end;
  if(isempty(tol)), tol=eps; end;
  if(isempty(trace)), trace=0; end;
  if tol < eps
    tol = eps;
  end
  m=(a+b)/2; h=(b-a)/2;
  alpha=sqrt(2/3); beta=1/sqrt(5);
  x1=.942882415695480; x2=.641853342345781;
  x3=.236383199662150;
  x=[a,m-x1*h,m-alpha*h,m-x2*h,m-beta*h,m-x3*h,m,m+x3*h,...
    m+beta*h,m+x2*h,m+alpha*h,m+x1*h,b];
  y=feval(f,x,varargin{:});
  fa=y(1); fb=y(13);
  i2=(h/6)*(y(1)+y(13)+5*(y(5)+y(9)));
  i1=(h/1470)*(77*(y(1)+y(13))+432*(y(3)+y(11))+ ...
     625*(y(5)+y(9))+672*y(7));
  is=h*(.0158271919734802*(y(1)+y(13))+.0942738402188500 ...
     *(y(2)+y(12))+.155071987336585*(y(3)+y(11))+ ...
     .188821573960182*(y(4)+y(10))+.199773405226859 ...
     *(y(5)+y(9))+.224926465333340*(y(6)+y(8))...
     +.242611071901408*y(7));
  s=sign(is); if(s==0), s=1; end;
  erri1=abs(i1-is);
  erri2=abs(i2-is);
  R=1; if(erri2~=0), R=erri1/erri2; end;
  if(R>0 & R<1), tol=tol/R; end;
  is=s*abs(is)*tol/eps;
  if(is==0), is=b-a, end;
  Q=adaptlobstp(f,a,b,fa,fb,is,trace,varargin{:});


function Q=adaptlobstp(f,a,b,fa,fb,is,trace,varargin)
%ADAPTLOBSTP  Recursive function used by ADAPTLOB.
```

```
%
%   Q = ADAPTLOBSTP('F',A,B,FA,FB,IS,TRACE) tries to
%   approximate the integral of F(X) from A to B to
%   an appropriate relative error. The argument 'F' is
%   a string containing the name of f.  The remaining
%   arguments are generated by ADAPTLOB or by recursion.
%
%   See also ADAPTLOB.

%   Walter Gautschi, 08/03/98

global termination2

  h=(b-a)/2; m=(a+b)/2;
  alpha=sqrt(2/3); beta=1/sqrt(5);
  mll=m-alpha*h; ml=m-beta*h; mr=m+beta*h; mrr=m+alpha*h;
  x=[mll,ml,m,mr,mrr];
  y=feval(f,x,varargin{:});
  fmll=y(1); fml=y(2); fm=y(3); fmr=y(4); fmrr=y(5);
  i2=(h/6)*(fa+fb+5*(fml+fmr));
  i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr) ...
     +672*fm);
  if(is+(i1-i2)==is) | (mll<=a) | (b<=mrr),
   if ((m <= a) | (b<=m)) & (termination2==0);
       warning(['Interval contains no more machine number. ',...
                'Required tolerance may not be met.']);
       termination2 =1;
    end;
    Q=i1;
    if(trace), disp([a b-a Q]), end;
  else
    Q=adaptlobstp(f,a,mll,fa,fmll,is,trace,varargin{:})+...
      adaptlobstp(f,mll,ml,fmll,fml,is,trace,varargin{:})+...
      adaptlobstp(f,ml,m,fml,fm,is,trace,varargin{:})+...
      adaptlobstp(f,m,mr,fm,fmr,is,trace,varargin{:})+...
      adaptlobstp(f,mr,mrr,fmr,fmrr,is,trace,varargin{:})+...
      adaptlobstp(f,mrr,b,fmrr,fb,is,trace,varargin{:});
  end;
```

The minimal number of function evaluations is 18 and occurs if the error test is met in the very first call to `adaptlobstp`. This can be expected only in cases where f is very regular on $[a,b]$ and the tolerance tol is not too stringent. Discontinuities of f in the interior of $[a,b]$, on the other hand, cannot be expected to be handled efficiently by our routine; but the routine has been observed to cope rather efficiently with other difficult behavior, as long as f remains bounded on the interval $[a,b]$ and smooth in its interior.

5 Test results

In comparing adaptive quadrature routines, one must take into account a number of characteristics, of which the more important ones are:

(i) *efficiency*, as measured by the number of function evaluations required to meet a given error tolerance;

(ii) *reliability*, the extent to which the requested error tolerance is achieved; and

(iii) *tolerance responsiveness*, the extent to which the efficiency is sensitive to changes in the error tolerance.

We will try to convey these characteristics graphically by displaying a histogram over four tolerances: $tol = eps$ (the machine precision[1]), $tol = 10^{-9}$, $tol = 10^{-6}$, and $tol = 10^{-3}$, the height of each of the four bars in the histogram indicating the number of function evaluations in a logarithmic scale. A bar that is completely white signifies that the requested tolerance has been attained; a shaded bar means that the result produced has a relative error that exceeds the tolerance by a factor larger than 1 but less than or equal to 10. A black bar indicates a discrepancy by a factor larger than 10. Thus, a white bar identifies a routine that is reliable for the tolerance in question, a shaded bar one that is slightly unreliable, and a black bar one that might be severely

[1]The choice $tol = eps$ makes our routines, especially `adaptsim`, work much harder than necessary, without yielding any noticeable gain in accuracy compared to, say, $tol = 10\cdot eps$.

unreliable. The tolerance responsiveness can be seen from how rapidly the histogram falls off with decreasing tolerance. A histogram that is flat (or partially flat) at relatively high numbers of function evaluations indicates poor tolerance responsiveness.

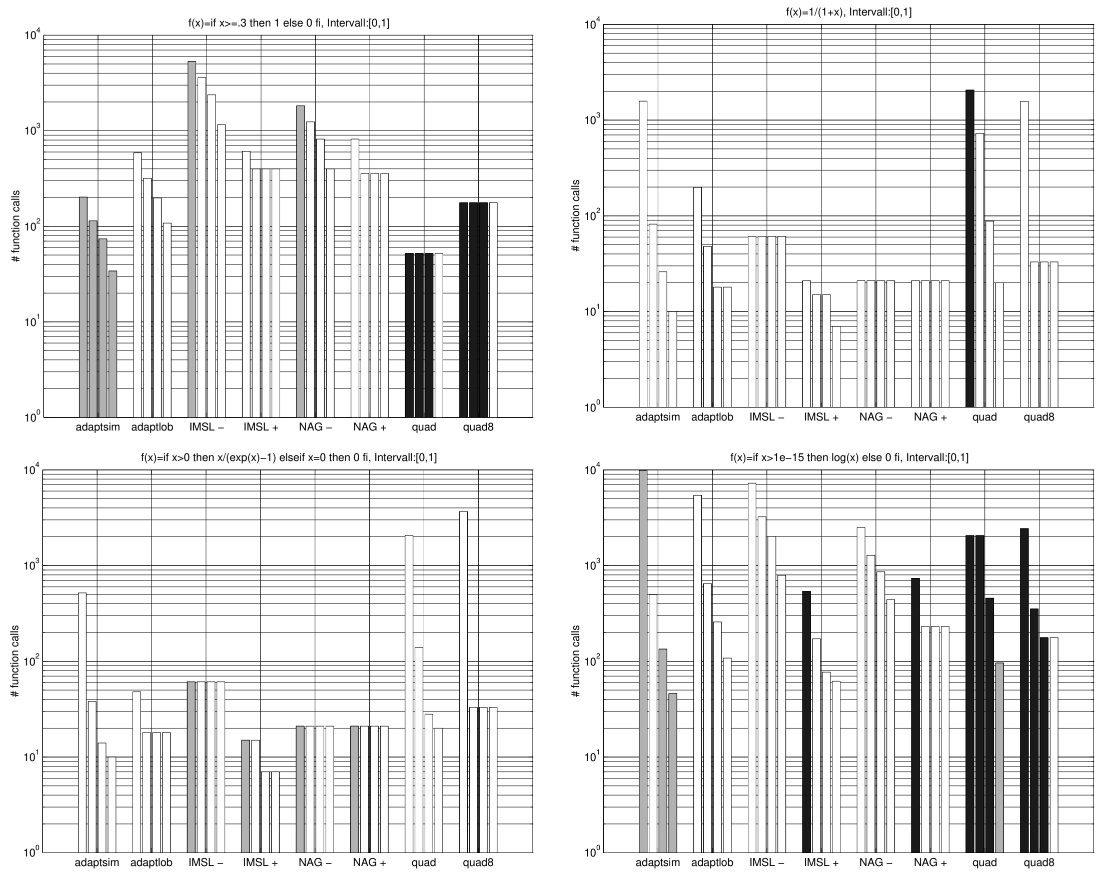

TABLE 2: Comparison with other adaptive quadrature routines

We compared our routines `adaptsim` and `adaptlob` with the worst and best routines in the IMSL library (`DQDAG`, `DQDAGS`), the worst and best routines of the NAG library (`D01AHF`, `D01AJF`, `D01AKF`), and with the routines `quad` and `quad8` from Matlab. The results are displayed in 23 histograms of Table 2 in [5, pp. 17–20], of which the first 21 refer to Kahaner's collection of test functions [11] and the last two to functions taken from [6]. Here, in Table

2, we present four typical examples. The tests were conducted on three different machines with four different versions of Matlab: SGI 02 R5000 (Matlab 5.0 and 5.1), HP A 9000/770 (Matlab 5.0), and SUN SPARCstation 20 (Matlab 5.0, 5.1, 5.2, and 5.2.1). The results were nearly identical. (The only significant difference was observed in connection with function #22, for which the routine `adaptsim` with $tol = eps$ — and only for this tolerance — on the machine SGI 02 returns a totally false answer with the minimum number 10 of function evaluations, whereas on the SUN SPARCstation it integrates the function correctly in 49926 function calls.) The graphics shown is based on the results obtained on the SUN SPARCstation with Matlab version 5.0. The tests of the NAG and IMSL routines were carried out in fortran on the HP/Convex Exemplar SPP2000/X-32 machine.

The following observations can be made.

• In terms of efficiency, the routine `adaptlob` performs distinctly better than `adaptsim` when the accuracy requirement is high. For machine precision *eps* it outperforms `adaptsim` in all but one example, and often significantly so. (The one exception is the discontinuous function #2, for which, however, `adaptsim` is slightly unreliable.) For the accuracy tolerance 10^{-9}, it does so in about half the cases. For lower tolerances, `adaptsim` is generally (but not always) more efficient than `adaptlob`, but less reliable.

• Compared with the other routines, those of the IMSL and NAG libraries are the most serious competitors. The best of them performs distinctly better than our routines in about one-third of the cases.

• In terms of reliability, the routine `adaptlob` is by far the best, exhibiting only one serious failure out of the $4 \cdot 23 = 92$ individual runs. It is followed by the IMSL and NAG library routines, which failed 6 or 7 times. The routines `quad` and `quad8` are by far the least reliable, having seriously failed in 30 resp. 15 cases. It is perhaps of interest to note that the second half of the termination criterion (7) for `adaptsim` and the analogous one for `adaptlob` has never been invoked in any of the 23 test cases. As already observed in §2, there are cases, however, for example the function $f(x) = \frac{1}{\sqrt{1-x^2}}$ for $0 \leq x < 1$ and $f(1) = 0$, where for $tol = eps$ that part of the stopping criterion is indeed activated, both in `adaptsim` and `adaptlob`. Also for the example (9) and $tol = eps$, one of our routines, `adaptlob` (but not the other), terminates in this manner.

• Both of our routines show excellent response to changes in the tolerance, in contrast to some of the other routines, where the response is more sluggish.

In view of these (admittedly limited) test results it would appear that the routines `adaptsim` and `adaptlob` are worthy contenders for inclusion in software libraries.

ACKNOWLEDGMENTS. The authors are indebted to Leonhard Jaschke for carrying out the extensive testing and for providing the graphical representation of the results. The second author is grateful to the first for the kind hospitality accorded him during his stays at the ETH.

References

[1] de Boor, Carl 1971. *On writing an automatic integration algorithm*, in: Mathematical Software, John R. Rice ed., Academic Press, New York, pp. 201–209.

[2] Davis, Philip J. and Rabinowitz, Philip 1984. *Methods of numerical integration*, 2d ed., Academic Press, Orlando.

[3] Gander, Walter 1993. *A simple adaptive quadrature algorithm*, Seminar für Angew. Math. ETH Zürich, Research Report No. 83-03.

[4] Gander, Walter 1992. *Computermathematik*, Birkhäuser, Basel.

[5] Gander, Walter and Gautschi, Walter 1998. *Adaptive quadrature — revisited*, Institut für Wissenschaftliches Rechnen ETH Zürich, Research Report #306.

[6] Garribba, S., Quartapelle, L., and Reina, G. 1978. *Algorithm 36 — SNIFF: Efficient self-tuning algorithm for numerical integration*, Computing **20**, 363–375.

[7] Gautschi, Walter 1988. *Gauss-Kronrod quadrature — a survey*, in *Numerical methods and approximation theory III* (G.V. Milovanović, ed.), Faculty of Electronic Engineering, Univ. of Niš, Niš, pp. 39–66.

[8] Gautschi, Walter 1997. *Numerical analysis: an introduction*, Birkhäuser, Boston.

[9] Heath, Michael T. 1997. *Scientific computing*, McGraw-Hill, New York.

[10] Kahan, William M. 1980. *Handheld calculator evaluates integrals*, Hewlett-Packard Journal **31**, no. 8, 23–32.

[11] Kahaner, D.K. 1971. *Comparison of numerical quadrature formulas*, in: Mathematical Software, John R. Rice ed., Academic Press, New York, 229–259.

[12] Lyness, J. N. 1969. *Notes on the adaptive Simpson quadrature routine*, J. Assoc. Comput. Mach. **16**, 483–495.