A Comparison of Three High-Precision Quadrature Schemes

David H. Bailey, Karthik Jeyabalan, and Xiaoye S. Li

CONTENTS

The authors have implemented three numerical quadrature schemes, using the Arbitrary Precision (ARPREC) software package. The objective here is a quadrature facility that can efficiently evaluate to very high precision a large class of integrals typical of those encountered in experimental mathematics, relying on a minimum of a priori information regarding the function to be integrated. Such a facility is useful, for example, to permit the experimental identification of definite integrals based on their numerical values. The performance and accuracy of these three quadrature schemes are compared using a suite of 15 integrals, ranging from continuous, well-behaved functions on finite intervals to functions with infinite derivatives and blow-up singularities at endpoints, as well as several integrals on an infinite interval. In results using 412-digit arithmetic, we achieve at least 400-digit accuracy, using two of the programs, for all problems except one highly oscillatory function on an infinite interval. Similar results were obtained using 1,012-digit arithmetic.

1. INTRODUCTION

Numerical quadrature has a long and distinguished history, including contributions by Newton, who devised the basis of what is now known as the Newton-Cotes scheme, and Gauss, who devised Gaussian quadrature. In the 20th century, numerous additional schemes were devised, including extended Simpson rules, adaptive quadrature, Romberg integration, Clenshaw-Curtis integration, and others [Davis and Rabinowitz 84, Krommer and Ueberhuber 98]. In addition, numerous "kernels" were devised that permit these schemes to efficiently compute definite integrals of functions that include a particular expression as a factor.

Virtually all of the modern literature on these techniques, as well as their practical implementations on computers, have been targeted at computing definite integrals to the accuracy of 15 digits or less, namely the limits of ordinary IEEE 64-bit floating-point data, which has 53

2000 AMS Subject Classification: Primary 65D30

Keywords: Numerical quadrature, numerical integration, arbitrary precision

1058-6458/2005 $0.50 per page
Experimental Mathematics **14**:3, page 317

mantissa bits. Little attention has been paid to the issues of very high precision quadrature, in part because few serious applications have been known for such techniques, and also because techniques that work well for standard machine precision often do not scale well to the realm of high precision. The software packages Mathematica and Maple include arbitrary precision arithmetic, together with numerical integration to high precision. These facilities are generally quite good, although in many cases they either fail or require unreasonably long run times.

In the past few years, computation of definite integrals to high precision has emerged as a useful tool in experimental mathematics. In particular, it is often possible to recognize an otherwise unknown definite integral in analytic terms, provided its numerical value can be calculated to high accuracy. Such experimental evaluations of integrals often involve integer relation detection, which means finding integers a_i, not all zero, such that for a given n-long real vector (x_i), we have $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$. Integer relation computations are used here to determine whether the numerical value of a definite integral is given by a formula of a certain type with unknown integer or rational coefficients. The most frequently used integer relation detection algorithm is the PSLQ algorithm [Bailey and Broadhurst 00]. It and other integer relation schemes require very high precision (often hundreds or thousands of decimal digits) in both the input data and in the operation of the algorithm to obtain meaningful results.

As one example, recently one of the authors, together with Jonathan Borwein and Greg Fee of Simon Fraser University in Canada, were inspired by a problem in the *American Mathematical Monthly* [Ahmed 02]. They found by using one of the quadrature routines described in this paper, together with a PSLQ integer relation detection program, that if $C(a)$ is defined by

$$C(a) = \int_0^1 \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)},$$

then

$$\begin{aligned} C(0) &= \pi\log 2/8 + G/2, \\ C(1) &= \pi/4 - \pi\sqrt{2}/2 + 3\sqrt{2}\arctan(\sqrt{2})/2, \\ C(\sqrt{2}) &= 5\pi^2/96, \end{aligned}$$

where $G = \sum_{k\ge 0}(-1)^k/(2k+1)^2$ is Catalan's constant (the third of these results is the result from the *Monthly*). These experimental results then led to the following general result, rigorously established, among others:

$$\int_0^\infty \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)} = \frac{\pi}{2\sqrt{a^2-1}}\left[2\arctan(\sqrt{a^2-1}) - \arctan(\sqrt{a^4-1})\right].$$

As a second example, recently Borwein and one of the present authors empirically determined that

$$\begin{aligned} \frac{2}{\sqrt{3}}\int_0^1 \frac{\log^6(x)\arctan[x\sqrt{3}/(x-2)]}{x+1}\,dx = \\ \frac{1}{81648}\big[&-229635L_3(8) + 29852550L_3(7)\log 3 \\ &- 1632960L_3(6)\pi^2 + 27760320L_3(5)\zeta(3) \\ &- 275184L_3(4)\pi^4 + 36288000L_3(3)\zeta(5) \\ &- 30008L_3(2)\pi^6 - 57030120L_3(1)\zeta(7)\big], \end{aligned}$$

where $L_3(s) = \sum_{n=1}^\infty [1/(3n-2)^s - 1/(3n-1)^s]$. General results have been conjectured but not yet rigorously established.

In some cases, Maple or Mathematica is able to evaluate a definite integral analytically, but the resulting expressions are quite complicated, and thus not very enlightening. For example, although the integrals

$$\begin{aligned} I_1 &= \int_0^1 \frac{t^2\log(t)\,dt}{(t^2-1)(t^4+1)}, \\ I_2 &= \int_0^{\pi/4} \frac{t^2\,dt}{\sin^2(t)}, \\ I_3 &= \int_0^\pi \frac{x\sin x\,dx}{1+\cos^2 x}, \end{aligned}$$

are successfully evaluated by Maple and Mathematica, the results are somewhat lengthy expressions involving advanced functions and complex entities. In the third problem, for instance, the expression produced by Mathematica continues for more than 30 lines. We suspect that there are considerably simpler closed-form versions of these integrals. Indeed, we can obtain the following, based solely on the high-precision numerical values of these integrals, combined with integer relation computations:

$$\begin{aligned} I_1 &= \pi^2(2-\sqrt{2})/32, \\ I_2 &= -\pi^2/16 + \pi\log(2)/4 + G, \\ I_3 &= \pi^2/4. \end{aligned}$$

These and numerous other examples that we could cite underscore the need for a practical, general-purpose, high-precision quadrature facility for experimental mathematics, by which we mean a computer program that can

numerically evaluate a large class of definite integrals to high precision, given nothing other than the function definition in a separate user subprogram. In other words, we seek a practical, self-contained tool that does not rely on symbolic manipulation, the presence or absence of certain "kernels" in the integrand, or knowledge of the behavior of the function or its derivatives. We also seek a scheme that is well suited to highly parallel implementation, so that a parallel computer system can be utilized when required for significantly faster runtimes. This latter requirement by itself rules out reliance on commercial products such as Mathematica and Maple, since these are not yet available for highly parallel platforms.

The only a priori assumptions that we grant are that the function to be integrated has a finite definite integral and is infinitely differentiable within the given finite interval. It may have a singularity (either a blow-up singularity or an infinite derivative) at one or both endpoints. We also consider functions defined on an infinite interval, such as $(0, \infty)$, with the proviso that a linear transformation such as $x \to 1/(t+1)$ converts the problem to the above-mentioned class. Note that definite integrals of functions with a finite set of discontinuities or other singularities within an interval may be computed as a sum of definite integrals on subintervals, so that the assumption given above encompasses a broad range of functions of interest.

We acknowledge, however, that it is most likely not possible to fashion a single numerical technique that works for all functions of this class. Further, even a reasonable problem may require an unreasonable amount of computer time given current technology. Nonetheless we aim to do as well as possible, within these constraints, particularly within the domain of problems that commonly arise in experimental mathematics.

2. THE ARPREC SOFTWARE

The quadrature techniques we describe below have been implemented using the Arbitrary Precision (ARPREC) computation package [Bailey et al. 02]. This software is based in part on the multiprecision package MPFUN90 (written in Fortran-90) [Bailey 95], which in turn is based on the earlier MPFUN-77 package (written in Fortran-77) [Bailey 93]. In MPFUN90, object-oriented facilities built into the Fortran-90 language, namely custom datatypes and operator overloading, were exploited to permit Fortran programmers to utilize the MPFUN90 library by making only a few minor changes (mostly type statement changes) to existing Fortran application programs.

The ARPREC library extends the functionality of the MPFUN packages to the realm of C/C++ programs. In particular, the ARPREC package combines the following features, which we believe to be unique for currently available software of this type:

- code written in C++ for high performance and broad portability;
- both C++ and Fortran-90 translation modules, which permit existing C++ and Fortran-90 application programs to use the arbitrary precision library by making only a few minor changes to existing source code;
- arbitrary precision integer, floating, and complex datatypes;
- support for datatypes with differing precision levels;
- interoperability with conventional integer and floating-point datatypes;
- numerous common algebraic and transcendental functions (sqrt, exp, log, sin, cos, tan, arccos, arcsin, arctan, erf, gamma, and others);
- quadrature programs (for numerical integration);
- PSLQ programs (for integer relation detection);
- polynomial root programs, for both real and complex roots;
- special routines, utilizing FFT-based multiplication, for extra-high-precision (over 1,000 digits) computation.

The ARPREC package is based on the IEEE 64-bit floating-point arithmetic standard, which is now implemented on virtually all computer systems, thus permitting a high degree of portability. It includes "configure" and "make" scripts that for most Unix systems automatically detect the software environment and perform a valid installation. The software and documentation is freely available on the Internet [Bailey et al. 02].

3. THE THREE QUADRATURE SCHEMES

We describe here three numerical quadrature schemes that we have found suitable for computing definite integrals to very high precision. We considered several other schemes, including at least one adaptive method, but

found that they are not competitive with these schemes when high-precision results are required—their runtimes scale too rapidly with the numeric precision level. Also, most of these other schemes fail for integrands with infinite derivatives or singularities at the endpoints (a difficulty, as we shall see, that is shared by one of the schemes we describe below).

QUADGS: A Gaussian quadrature scheme. Gaussian quadrature certainly is not new, although most descriptions in the literature do not address the requirements of arbitrary precision implementation. This scheme approximates an integral on $[-1, 1]$ as the sum $\sum_{0 \le j < n} w_j f(x_j)$, where the abscissas x_j are the roots of the nth degree Legendre polynomial $P_n(x)$ on $[-1, 1]$, and the weights w_j are

$$w_j = \frac{-2}{(n+1)P'_n(x_j)P_{n+1}(x_j)}$$

[Atkinson 93, page 187]. We compute an individual abscissa by using a Newton iteration root-finding algorithm with a dynamic precision scheme. The starting value for x_j in these Newton iterations is given by $\cos[\pi(j - 1/4)/(n + 1/2)]$, which may be calculated using ordinary 64-bit floating-point arithmetic [Press et al. 86, page 125]. Our Newton iterations start with 40-digit precision and iterate until convergence is achieved at this level. Thereafter our program nearly doubles the working precision with each subsequent Newton iteration until the full precision, p digits, desired for quadrature results is achieved. Using this approach, the total runtime for an individual abscissa computation is only about three times the runtime of the final iteration. We compute the Legendre polynomial function values using an n-long iteration of the recurrence $P_0(x) = 0$, $P_1(x) = 1$, and

$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x)$$

for $k \ge 2$. The derivative is computed as $P'_n(x) = n(xP_n(x) - P_{n-1}(x))/(x^2 - 1)$. For functions defined on intervals other than $[-1, 1]$, a linear scaling is used to convert the Gaussian abscissas to the given interval.

One legitimate question that can be raised here is whether the resulting Legendre polynomial function values are accurate for large n, due to the recurrence used in the generation algorithm. Fortunately, we have found that this error appears to be minor, based on our implementations and tests with p up to 1,000 digits and with n up to 12,288. As we shall see in Section 7, we have obtained quadrature results accurate to over 1,000-digit accuracy, for several problems suitable for Gaussian quadrature, using a working precision (for both initialization and quadrature calculations) of only 1,012 digits.

In our implementation, we precompute multiple levels (i.e., multiple sets) of abscissa-weight pairs, where each level has twice as many abscissa-weight pairs as the level before. In particular, the number of abscissa-weight pairs at level k in our program is $3 \cdot 2^k$, so that the total for m levels is $\sum_{k \le m} 3 \cdot 2^k \approx 6 \cdot 2^m$. When evaluating an integral using this program, we start with the first level, obtain a quadrature result, and continue to apply additional levels until we are satisfied with the estimated accuracy of our result (see Section 5), or else we exhaust our sets of precomputed abscissa-weight pairs.

Our program saves additional time by dynamically increasing the working precision in the quadrature calculation, in a similar manner as is used in calculating abscissas. We use modest precision (80 digits) for the first two levels. Thereafter, if the estimated number of correct digits for the quadrature result (see Section 5) at a given level is more than half the current working precision, then the working precision is doubled before proceeding to the next level, until the full precision of p digits has been reached. This modification reduces runtimes by up to 35% for problems well suited for Gaussian quadrature, with even greater savings for problems that are not well suited, since full-precision arithmetic is not wasted on such problems.

The cost of computing abscissa-weight pairs using this scheme increases quadratically with n, since each Legendre polynomial evaluation requires n steps. The abscissa-weight pairs can alternately be computed using an eigenvector scheme due to Golub and Welsch [Golub and Welsch 69], although this scheme requires considerably more memory, and its computational cost also increases quadratically with n. We know of no scheme for generating Gaussian abscissa-weight pairs that avoids this quadratic dependence on n [Kahan 04]. For many well-behaved integrand functions (as we shall see), Gaussian quadrature achieves quadratic convergence, meaning that doubling n achieves roughly twice as many correct digits in the quadrature result, after a few initial levels. Assuming this behavior is achieved for a given function, this means that to achieve accuracy of p digits, one needs n to scale linearly with p. With FFT-based multiplication (available in the ARPREC package, for instance), the cost of basic arithmetic scales as $p \log p$ for practical precision p (i.e., up to several million digits; in general an additional factor of $\log \log p$ is involved). Thus the overall cost of the Gaussian initialization process scales as roughly $p^3 \log p$.

In a similar manner, we can estimate the cost of performing a quadrature calculation for a given integrand function once the initialization has been done. Fairly simple quadratically convergent algorithms (in which the number of accurate digits approximately doubles with each successive iteration) are known for all algebraic functions and many of the common transcendental functions, including the exponential, logarithm, trigonometric and inverse trigonometric functions [Bailey 93, Bailey 95]. For such functions, the cost of a single evaluation scales roughly as $p \log^2 p$, using FFT-based arithmetic, in the range of practical precision (for algebraic functions, it is only $p \log p$). Assume for a moment that a transcendental integrand function is well suited for Gaussian quadrature, so that n scales linearly with p, as described in the previous paragraph. Then the cost of evaluating the integral of such a function scales roughly as $p^2 \log^2 p$.

QUADERF: An error function-based quadrature scheme. This program approximates an integral on $[-1, 1]$ as a sum $\sum w_j f(x_j)$, as with Gaussian quadrature, but here the abscissas x_j are given by $\mathrm{erf}(hj)$, where $\mathrm{erf}(x) = (2/\sqrt{\pi}) \int_0^x e^{-t^2}\ dt$, and the weights w_j are given by $(2/\sqrt{\pi})e^{-(hj)^2}$. To compute the error function $\mathrm{erf}(x)$, we use the following formula for $\mathrm{erfc}(t) = 1 - \mathrm{erf}(t)$ given by Crandall [Crandall 96, page 85] (who in turn attributes it to a 1968 paper by Chiarella and Reichel [Chiarella and Reichel 68]),

$$\mathrm{erfc}(t) = \frac{e^{-t^2}\alpha t}{\pi}\left(\frac{1}{t^2} + 2\sum_{k\ge 1}\frac{e^{-k^2\alpha^2}}{k^2\alpha^2 + t^2}\right) + \frac{2}{1 - e^{2\pi t/\alpha}} + E,$$

where $t > 0$ and $|E| < 3e^{-\pi^2/\alpha^2}$. The parameter α is chosen small enough to ensure that the error E is sufficiently small. Given a precision of p digits, let α be defined by the formula $10^{-p} = 10e^{-\pi^2/\alpha^2}$, so that $E < 10^{-p}$. Then provided that $t < \sqrt{p \log 10}$, the formula above is also accurate to a relative error of 10^{-p}. The generation of x_j and w_j should be performed to a relative accuracy of at least the primary precision, p_1 digits, desired for quadrature results.

In a straightforward implementation of the error function quadrature scheme, the calculation of abscissa-weight pairs, for a given h, can be terminated when $w_j < \epsilon_1 = 10^{-p_1}$. However, we have found that it is advantageous to compute additional abscissa-weight pairs, continuing until $w_j < \epsilon_2$, where the secondary epsilon $\epsilon_2 = 10^{-p_2}$, and $p_2 = 2p_1$ (i.e., $\epsilon_2 = \epsilon_1^2$). These calculations may still be done with a relative accuracy of the primary working precision p_1, provided that α in the formula above for erfc is selected based on p_2 rather than p_1. Additionally, we store the values $\mathrm{erfc}(hj)$ for subsequent quadrature computation, rather than $x_j = \mathrm{erf}(hj) = 1 - \mathrm{erfc}(hj)$, since the latter, many of which are very close to 1, lose accuracy in subtraction. Then when we evaluate integrand functions in a quadrature computation, we perform linear scaling of precomputed abscissas using a secondary (higher) precision of p_2 digits (a linear scaling of abscissas is required whenever the interval of integration differs from $[-1, 1]$, as explained in Section 4). In this way, we can use more accurate input values for an expression such as $1 - t$ appearing in a problem such as $\int_0^1 e^t(1 - t)^{-1/2}\, dt$. The function itself does not need to be computed using this higher precision, so the added computational cost of this secondary precision procedure is negligible.

This optional modification permits the use of abscissas that are closer to endpoints than the primary epsilon ϵ_1 would normally permit, thus achieving significantly higher accuracy in the quadrature results for problems with a blow-up singularity at an endpoint (see Section 4). For the test problems below, we found $p_2 = 2p_1$ (corresponding to $\epsilon_2 = \epsilon_1^2$) to be adequate; with more extreme singularities, an even smaller value of ϵ_2 may be needed, in order that the function-weight products $w_j f(x_j)$, for abscissas x_j very close to the endpoints, are smaller than ϵ_1. If our program encounters the need for a smaller ϵ_2 during a quadrature calculation (because $|w_j f(x_j)| > \epsilon_1$ for x_j close to an endpoint), it outputs a message. This secondary epsilon procedure and the usage of additional abscissa-weight pairs are not needed for integrands that do not have a blow-up singularity at an endpoint.

As with the Gaussian scheme, multiple "levels" of abscissa-weight pairs are typically precomputed, with each level having approximately twice as many pairs as the previous level. In our implementation, this is controlled by setting $h = 2^{2-k}$ for level k. With h defined in this manner, the even-indexed abscissa-weight pairs at one level are merely the full set at the previous level. Thus only the odd-indexed pairs need to be computed at each level (after the first level), and, more importantly, the function to be integrated needs to be evaluated only at the odd-indexed abscissas at each level. Additional time can be saved for many functions by terminating the summation $\sum_j w_j f(x_j)$ once the terms $w_j f(x_j)$ are consistently smaller than ϵ_1.

With the primary epsilon set to 10^{-400} and the secondary epsilon set to 10^{-800}, as in the tests in Table 1 below, roughly $n = 5.5 \cdot 2^k$ abscissa-weight pairs are generated at level k, so that the total required for m levels is approximately $\sum_{k \le m} 5.5 \cdot 2^k \approx 11 \cdot 2^m$. As we shall see, error function quadrature achieves quadratic convergence for many problems, so that the number n of abscissa-weight pairs needed for p-digit accuracy scales roughly linearly with p. The cost of computing an individual abscissa-weight pair is dominated by the cost of the series evaluation in the formula for erfc. The number of terms that need to be calculated and summed for an erfc evaluation is linearly proportional to p. Thus the cost of the error function initialization process, using FFT-based arithmetic, scales as $p^3 \log p$. Even though this is the same general scaling formula as with Gaussian quadrature, in practice the error function initialization is much faster at a given precision and level.

For error function quadrature there is no point in attempting to dynamically increase precision during a quadrature computation. This is because doing so would sacrifice the advantage of needing to evaluate the integrand function only at the odd-indexed abscissas at each level (which presumes that all previous function evaluations are fully accurate). It is often useful, though, to first try a given problem with modest precision, say 100 digits, thus not wasting high-precision computation on a problem not well suited for this scheme.

As we will see below, the error function quadrature scheme works very well for all of our test problems except one highly oscillatory integrand. Assuming that a given problem is well suited for this scheme, so that the number of function evaluations needed scales linearly with the precision p, then the cost of evaluating the integral of such a function using this scheme scales roughly as $p^2 \log^2 p$ within a range of practical precision, or in other words with the same scaling formula as Gaussian quadrature.

QUADTS: A tanh-sinh quadrature scheme. This scheme is similar to the error function scheme. In this case, the abscissas are chosen as $x_j = \tanh u_2$ and the weights $w_j = u_1 / \cosh^2 u_2$, where $u_1 = \pi/2 \cdot \cosh(hj)$ and $u_2 = \pi/2 \cdot \sinh(hj)$.

In a straightforward implementation, the generation of abscissa-weight pairs should be performed with the primary precision p_1 digits desired for quadrature results and continues, for a given h, until $w_j < \epsilon_1 = 10^{-p_1}$. In our implementation, as with the error function quadrature scheme, at each level we calculate additional abscissa-weight pairs, continuing until $w_j < \epsilon_2$, where $\epsilon_2 = 10^{-p_2}$ and $p_2 = 2p_1$ (i.e., $\epsilon_2 = \epsilon_1^2$). Also, as before, we actually store $1 - x_j = 1/(e^{u_2} \cosh u_2)$, and, during a quadrature calculation, we perform linear scaling of these precomputed values using the secondary precision p_2.

In our tanh-sinh quadrature program, each level k of abscissa-weight pairs uses $h = 2^{-k}$. As with the error function scheme, the even-indexed abscissa-weight pairs at one level are merely the full set of pairs at the previous level, and the integrand function needs to be evaluated only at the odd-indexed abscissas at each level. Additional time can be saved for many functions by terminating the summation $\sum_j w_j f(x_j)$ once the terms $w_j f(x_j)$ are consistently smaller than ϵ_1. In the tests shown in Table 1 below, where the primary epsilon is set to 10^{-400} and the secondary epsilon is set to 10^{-800}, we find that roughly $3.6 \cdot 2^k$ abscissa-weight pairs are generated at level k, so that the total required for m levels is approximately $\sum_{k \le m} 3.6 \cdot 2^k \approx 7.2 \cdot 2^m$.

As with the other two schemes, the tanh-sinh scheme achieves quadratic convergence for many integrand functions, so that the number n of abscissa-weight pairs required to achieve an accuracy of p digits scales, for these functions, roughly linearly with p. The cost of computing an individual pair with this scheme is dominated by the cost of exponential function evaluation, for which simple quadratically convergent algorithms are known [Bailey 93, Bailey 95] (one is implemented in ARPREC). With FFT-based arithmetic, and within a practical range of precision, the cost of one exponential evaluation scales as $p \log^2 p$. Thus the cost of the tanh-sinh initialization process scales roughly as $p^2 \log^2 p$, which is a more slowly growing rate than that of the other two schemes.

As with error function quadrature, there is no advantage in attempting to dynamically increase precision during a quadrature computation, since this would sacrifice the advantage of needing to evaluate the function only at odd-indexed abscissas at each level.

In practice, as we shall see, tanh-sinh quadrature achieves quadratic convergence for many integrand functions. Assuming that a given problem is of this class, so that the number of function evaluation scales linearly with precision p, then the cost of evaluating the integral of such a function using this method scales roughly as $p^2 \log^2 p$, within a range of practical precision, or in other words with the same scaling formula as with the other two methods. The tanh-sinh scheme was first introduced by Takahasi and Mori [Takahasi and Mori 74, Mori 91].

4. THE EULER-MACLAURIN SUMMATION FORMULA

The error function and tanh-sinh quadrature schemes are based on the Euler-Maclaurin summation formula, which implies that for certain bell-shaped integrands, approximating the integral by a simple step-function summation is remarkably accurate, much more so than one would normally expect. The Euler-Maclaurin summation formula can be stated as follows [Atkinson 93, page 180]. Let $m \geq 0$ and $n \geq 1$ be integers, and define $h = (b-a)/n$ and $x_j = a + jh$ for $0 \leq j \leq n$. Further assume that the function $f(x)$ is at least $(2m+2)$-times continuously differentiable on $[a, b]$. Then

$$\int_a^b f(x)\,dx = h\sum_{j=0}^{n} f(x_j) - \frac{h}{2}\left(f(a)+f(b)\right) - \sum_{i=1}^{m} \frac{h^{2i}B_{2i}}{(2i)!}\left(f^{(2i-1)}(b) - f^{(2i-1)}(a)\right) - E,$$

where B_{2i} denote the Bernoulli numbers, and

$$E = \frac{h^{2m+2}(b-a)B_{2m+2}f^{2m+2}(\xi)}{(2m+2)!}$$

for some $\xi \in (a, b)$.

In the circumstance where the function $f(x)$ and all of its derivatives are zero at the endpoints a and b (as in a smooth, bell-shaped function), note that the second and third terms of the Euler-Maclaurin formula are zero. Thus for such functions the error of a simple step-function approximation to the integral, with interval h, is simply E. But since E is less than a constant times $h^{2m+2}/(2m+2)!$, for any m, we conclude that the error goes to zero more rapidly than any power of h. For a function defined on $(-\infty, \infty)$, the Euler-Maclaurin summation formula still applies to the resulting doubly infinite sum approximation, provided as before that the function and all of its derivatives tend rapidly to zero for large positive and negative arguments.

This principle is utilized in the error function and tanh-sinh schemes by transforming the integral of $f(x)$ on the interval $[-1, 1]$ to an integral on $(-\infty, \infty)$ using the change of variable $x = g(t)$. Here $g(x)$ is some monotonic, infinitely differentiable function with the property that $g(x) \to 1$ as $x \to \infty$ and $g(x) \to -1$ as $x \to -\infty$, and also with the property that $g'(x)$ and all higher derivatives rapidly approach zero for large positive and negative arguments. In this case we can write, for $h > 0$,

$$\int_{-1}^{1} f(x)\,dx = \int_{-\infty}^{\infty} f(g(t))g'(t)\,dt = h\sum_{j=-\infty}^{\infty} w_j f(x_j) + E,$$

where $x_j = g(hj)$ and $w_j = g'(hj)$. If $g'(t)$ and its derivatives tend to zero sufficiently rapidly for large t, positive and negative, then even in cases where $f(x)$ has an infinite derivative or an integrable singularity at one or both endpoints, the resulting integrand $f(g(t))g'(t)$ will be a smooth bell-shaped function for which the Euler-Maclaurin argument applies. Thus, in these cases, the error E in this approximation decreases faster than any power of h.

The error function integration scheme uses $g(t) = \mathrm{erf}(t)$ and $g'(t) = (2/\sqrt{\pi})e^{-t^2}$. Note that $g'(t)$ is merely the bell-shaped probability density function, which is well known to converge rapidly to zero, together with all of its derivatives, for large arguments. The tanh-sinh scheme uses $g(t) = \tanh(\pi/2 \cdot \sinh t)$ and $g'(t) = \pi/2 \cdot \sinh t/\cosh^2(\pi/2 \cdot \sinh t)$, for which the convergence to zero is compound exponential, even faster than the probability density function.

The doubly infinite sum in the formula above can be approximated by a finite sum provided one takes reasonable care to insure that the truncated tails are insignificant. This is the rationale for the secondary epsilon scheme mentioned in Section 3: in cases where the integrand has a blow-up singularity at an endpoint, this scheme permits one to sum additional terms, with abscissas very close to the endpoints, until the rapidly decreasing weights overwhelm the large function values. If this is done properly, the finite sum will be within the target tolerance of the full doubly infinite summation. Along this line, whenever the given interval of integration is other than $[-1, 1]$, a linear scaling must be performed on the precomputed abscissas during the quadrature computation. As we mentioned in Section 3, when using the secondary epsilon scheme it is important to perform this scaling using a (higher) secondary precision, so that arguments for the integrand function evaluation near the endpoints are as accurate as possible.

5. ERROR ESTIMATION

As mentioned above, we seek a practical, general purpose, high-precision numerical integration facility that does not

depend on a priori bounds of the function or its derivatives. Rigorous error bounds are not possible for any quadrature scheme without such knowledge [Davis and Rabinowitz 84, page 420]. Instead, we use the following heuristic error estimation scheme, which is inspired by (although it does not rely on) the quadratically convergent behavior often achieved by these schemes. In spite of its heuristic nature, it appears to work well in practice, both on problems for which highly accurate quadrature results are obtained, as well as for those for which highly accurate results are not obtained.

Let S_k be the computed approximations of the integral for levels k up to level n. Then the estimated error E_n at level n is one if $n \le 2$, zero if $S_n = S_{n-1}$, and otherwise 10^d, where $d = \max(d_1^2/d_2,\ 2d_1,\ d_3,\ d_4)$ (except d is not set greater than 0). In this formula,

$$\begin{aligned} d_1 &= \log_{10} |S_n - S_{n-1}|, \\ d_2 &= \log_{10} |S_n - S_{n-2}|, \\ d_3 &= \log_{10}(\epsilon_1 \cdot \max_j |w_j f(x_j)|) \\ d_4 &= \log_{10} \max(|w_l f(x_l)|, |w_r f(x_r)|) \end{aligned}$$

Here x_l is the closest abscissa to the left endpoint and x_r is the closest abscissa to the right endpoint. The term d_4 is not present for Gaussian quadrature. In our Gaussian quadrature program, $\epsilon_1 = 10^{-q}$, where q is the current working precision in digits, which for a given problem starts at 80 digits and then is dynamically increased, as described in Section 3, until it achieves the full precision p. For the error function and tanh-sinh programs, $\epsilon_1 = 10^{-p_1}$, where p_1 is the primary precision, in digits, as described in Section 3. Calculations of d may be done to ordinary double precision accuracy (i.e., 15 digits), and the resulting value may be rounded to the nearest integer.

The rationale for the four terms in the formula for d is as follows. The first term is a simple multiplicative projection based on the differences between the quadrature result at the current level and the past two levels. The second term stems from the observation that the best one can hope for is quadratic convergence; in other words, the number of correct digits cannot be more than twice the previous level. The third term derives from the observation that the error cannot be less than the current epsilon times the largest function-weight element that is being summed. The fourth term is based on the fact that in the two Euler-Maclaurin-based schemes, the accuracy of the quadrature result is limited by the sizes of the final function-weight terms near the two endpoints, since the infinite sum mentioned in Section 4 is truncated there.

One does not need to rely on this estimation scheme if one is willing to continue computation until the quadrature results from two successive levels agree to within the full primary precision (or the final precomputed set of abscissa-weight pairs is exhausted). This would significantly increase the runtime for many problems, since it would be necessary to compute with at least one additional level of abscissas and weights. Also, even two consecutive values in full agreement still do not constitute a mathematically rigorous guarantee of correctness.

6. TEST PROBLEMS

The following 15 integrals were used as a test suite to compare these three quadrature schemes. In each case an analytic result is known, as shown below, facilitating the checking of results. The 15 integrals are listed in five groups:

- 1–4: continuous, well-behaved functions (all derivatives exist and are bounded) on finite intervals;
- 5–6: continuous functions on finite intervals, but with an infinite derivative at an endpoint;
- 7–10: functions on finite intervals with an integrable singularity at an endpoint;
- 11–13: functions on an infinite interval;
- 14–15: oscillatory functions on an infinite interval.

$$\begin{aligned}
1 &: \int_0^1 t\log(1+t)\,dt = 1/4 \\
2 &: \int_0^1 t^2 \arctan t\,dt = (\pi - 2 + 2\log 2)/12 \\
3 &: \int_0^{\pi/2} e^t \cos t\,dt = (e^{\pi/2} - 1)/2 \\
4 &: \int_0^1 \frac{\arctan(\sqrt{2+t^2})}{(1+t^2)\sqrt{2+t^2}}\,dt = 5\pi^2/96 \\
5 &: \int_0^1 \sqrt{t}\log t\,dt = -4/9 \\
6 &: \int_0^1 \sqrt{1-t^2}\,dt = \pi/4 \\
7 &: \int_0^1 \frac{\sqrt{t}}{\sqrt{1-t^2}}dt = 2\sqrt{\pi}\Gamma(3/4)/\Gamma(1/4) \\
8 &: \int_0^1 \log^2 t\,dt = 2
\end{aligned}$$

	QUADGS			QUADERF			QUADTS		
Prob.	Level	Time	Error	Level	Time	Error	Level	Time	Error
Init	12	25,130.59		12	224.51		12	51.50	
1	6	1.04	10^{-404}	9	7.93	10^{-404}	8	3.36	10^{-405}
2	6	0.92	10^{-403}	9	4.35	10^{-403}	8	2.48	10^{-404}
3	5	0.29	10^{-401}	9	5.07	10^{-402}	7	1.60	10^{-402}
4	6	0.81	10^{-403}	9	12.52	10^{-403}	8	4.99	10^{-403}
5	12	13.15	10^{-13}	9	7.62	10^{-404}	7	1.84	10^{-404}
6	12	1.23	10^{-15}	9	0.56	10^{-403}	8	0.25	10^{-403}
7	12	2.29	10^{-5}	9	1.15	10^{-401}	8	0.49	10^{-402}
8	12	12.55	10^{-8}	9	8.47	10^{-404}	7	1.72	10^{-402}
9	12	17.72	10^{-9}	9	9.87	10^{-402}	8	4.77	10^{-401}
10	12	6.56	10^{-5}	9	2.52	10^{-401}	8	1.32	10^{-400}
11	7	0.07	10^{-412}	10	0.94	10^{-403}	9	0.36	10^{-403}
12	12	7.51	10^{-5}	11	7.75	10^{-401}	10	4.95	10^{-402}
13	10	8.25	10^{-404}	12	8.66	10^{-403}	10	3.59	10^{-402}
14	12	62.35	10^{-404}	12	17.62	10^{-402}	11	14.54	10^{-402}
15	5/12	6.47	10^{-24}	9/12	2.05	10^{-22}	7/12	3.82	10^{-25}

TABLE 1. 400-digit runs.

$$9 \;:\; \int_0^{\pi/2} \log(\cos t)\,dt \;=\; -\pi\log(2)/2$$

$$10 \;:\; \int_0^{\pi/2} \sqrt{\tan t}\,dt \;=\; \pi\sqrt{2}/2$$

$$11 \;:\; \int_0^{\infty} \frac{1}{1+t^2}\,dt \;=\; \int_0^1 \frac{ds}{1-2s+2s^2} \;=\; \pi/2$$

$$12 \;:\; \int_0^{\infty} \frac{e^{-t}}{\sqrt{t}}\,dt \;=\; \int_0^1 \frac{e^{1-1/s}\,ds}{\sqrt{s^3-s^4}} \;=\; \sqrt{\pi}$$

$$13 \;:\; \int_0^{\infty} e^{-t^2/2}\,dt \;=\; \int_0^1 \frac{e^{-(1/s-1)^2/2}\,ds}{s^2} \;=\; \sqrt{\pi/2}$$

$$14 \;:\; \int_0^{\infty} e^{-t}\cos t\,dt \;=\; \int_0^1 \frac{e^{1-1/s}\cos(1/s-1)\,ds}{s^2} \;=\; 1/2$$

$$15 \;:\; \int_0^{\infty} \frac{\sin t}{t}\,dt \;=\; \int_0^{\pi} \frac{\sin t}{t}\,dt + 40320\int_0^{1/\pi} t^7\sin(1/t)\,dt - \frac{1}{\pi} + \frac{2}{\pi^3} - \frac{24}{\pi^5} + \frac{720}{\pi^7} \;=\; \frac{\pi}{2}$$

Problem 4, as was mentioned above, appeared in the September 2002 *American Mathematical Monthly* [Ahmed 02]. All are typical of the sorts of problems that the authors have encountered in experimental math research, except that in each of these cases, analytic solutions are well known. Problems 11–15 are integrals on an infinite interval, which is in each case here $[0,\infty)$. Except for Problem 15, such integrals are evaluated by using the transformation $s = 1/(t+1)$, as shown above. In Problem 15, the integral is written as the sum of integrals on $[0,\pi]$ and $[\pi,\infty)$. Then integration by parts is applied several times to the second integral of this pair, resulting in the expression shown above. This expression requires the evaluation of the integrals $\int_0^{\pi} t^{-1}\sin t\,dt$ and $\int_0^{1/\pi} t^7\sin(1/t)\,dt$, which are significantly better behaved than the original, resulting in faster convergence. Even with this transformation, however, Problem 15 remains the most difficult of the set, as we shall see.

7. RESULTS OF TESTS

The three quadrature programs, QUADGS, QUADERF, and QUADTS, were each implemented using the ARPREC arbitrary precision computation package [Bailey et al. 02], in a very similar programming style, with the primary user working precision set at 400 digits (the actual internal working precision employed by the software is roughly 412 digits). We sought results good to the target accuracy of the corresponding primary epsilon, namely 10^{-400}. A secondary precision of 800 digits and a corresponding secondary epsilon of 10^{-800} were employed in QUADERF and QUADTS, as described in Section 3, to achieve improved accuracy on problems with blow-up singularities (800-digit arithmetic is used here only in linear scaling of precomputed abscissas, and thus has negligible overall cost). One exception to these specifications is in Problem 15, where a primary precision level of 100 digits and a secondary precision of 200 digits

	QUADGS			QUADERF			QUADTS		
Prob.	Level	Time	Error	Level	Time	Error	Level	Time	Error
Init	12	73,046.28		13	3,891.63		12	390.83	
1	7	6.86	10^{-1012}	10	97.16	10^{-1004}	9	37.33	10^{-1010}
2	7	9.13	10^{-1011}	11	112.11	10^{-1003}	9	32.64	10^{-1010}
3	7	10.01	10^{-1009}	10	90.29	10^{-1004}	9	41.23	10^{-1008}
4	7	9.31	10^{-1010}	11	453.92	10^{-1003}	9	67.39	10^{-1009}
5	12	14.70	10^{-13}	10	88.43	10^{-1004}	8	18.54	10^{-1010}
6	12	1.39	10^{-15}	10	6.75	10^{-1004}	9	2.29	10^{-1010}
7	12	2.49	10^{-5}	10	15.21	10^{-1001}	9	4.40	10^{-1002}
8	12	13.89	10^{-8}	10	98.25	10^{-1004}	8	19.19	10^{-1009}
9	12	18.66	10^{-9}	10	113.49	10^{-1004}	9	48.18	10^{-1008}
10	12	7.06	10^{-5}	10	35.80	10^{-1001}	9	15.55	10^{-1002}
11	8	0.41	10^{-1012}	11	10.41	10^{-1003}	10	3.03	10^{-1009}
12	12	7.98	10^{-5}	13	211.03	10^{-1001}	11	65.05	10^{-1002}
13	11	98.50	10^{-1011}	13	117.09	10^{-1003}	12	85.61	10^{-1007}

TABLE 2. 1,000 digit runs.

were used (these are more than ample, given the accuracy achieved). In each of the three programs, 12 levels (i.e., 12 sets) of abscissa-weight pairs were precomputed. Additional levels could have been precomputed, but this would not have materially changed these results. Each quadrature program was run blindly—beginning at level one and continuing at successively higher levels, each of which approximately doubles the runtime, until one of these two conditions was met: (1) the maximum level (level 12) was completed; or (2) the estimated error achieved the accuracy target, namely 10^{-400}. These runs were made on a 2 GHz Apple G5 system, using the IBM xlC and xlf90 compilers, with O3 optimization.

The results of these tests are given in Table 1 below. The first line gives the runtime, in seconds, for the initialization process. The initialization time is listed here separately from the integral evaluations, since we expect that in many practical applications, the abscissas and weights will be computed once and then stored for numerous subsequent evaluations. Initialization produced 24,670 abscissa-weight pairs for QUADGS, 43,951 pairs for QUADERF, and 28,965 pairs for QUADTS. Table 1 includes the number of levels used by each of the three programs for the various problems. In Problem 15, where two individual integrals are evaluated, the number of levels used for both steps are shown in the table. The errors are shown to within one order of magnitude and are based on the analytic evaluations given in the previous section.

We have also successfully run these three programs with 1,000-digit precision (1,012-digit internal precision). In Table 2, we include 1,000-digit results for Problems 1–13 (in other words, for all categories of problems except the last). For these problems, 24,670 abscissa-weight pairs (12 levels) were generated for QUADGS, 138,982 pairs (13 levels) for QUADERF, and 32,708 pairs (12 levels) for QUADTS. No modifications were made to the programs for these runs, other than to change the precision and epsilon levels: 1,000 digits primary precision, 2,000 digits secondary precision, and corresponding epsilons.

8. ANALYSIS

The Gaussian quadrature program (QUADGS) did extremely well on the first set of problems, namely integrals of bounded, well-behaved continuous functions on finite intervals. In both Table 1 and Table 2, it was between four and 40 times faster than QUADERF on these problems, and between three and seven times faster than QUADTS. Its accuracy on these problems was consistently less than the target tolerance. QUADGS also did well on Problems 11, 13, and 14, achieving the target tolerance in reasonable runtimes. But for the other problems, which are characterized by functions that are not well behaved at endpoints, its accuracy was quite poor, even when all 12 levels of abscissas and weights were utilized (it is well known that Gaussian quadrature is not very effective for such integrands [Kahan 04]). Another major drawback of the Gaussian scheme is that its initialization time is many times higher than that of the other two schemes.

Level	Prob. 2	Prob. 4	Prob. 6	Prob. 8	Prob. 10	Prob. 12	Prob. 14
1	10^{-1}	10^{-1}	10^{-1}	10^{0}	10^{-1}	10^{0}	10^{0}
2	10^{-2}	10^{-5}	10^{-3}	10^{-3}	10^{-3}	10^{-1}	10^{-1}
3	10^{-6}	10^{-6}	10^{-8}	10^{-10}	10^{-8}	10^{-3}	10^{-2}
4	10^{-13}	10^{-12}	10^{-17}	10^{-21}	10^{-16}	10^{-6}	10^{-3}
5	10^{-26}	10^{-25}	10^{-34}	10^{-43}	10^{-33}	10^{-11}	10^{-5}
6	10^{-52}	10^{-51}	10^{-68}	10^{-87}	10^{-66}	10^{-20}	10^{-10}
7	10^{-104}	10^{-102}	10^{-134}	10^{-173}	10^{-132}	10^{-37}	10^{-19}
8	10^{-206}	10^{-204}	10^{-266}	10^{-348}	10^{-264}	10^{-70}	10^{-37}
9	10^{-411}	10^{-409}	10^{-529}	10^{-696}	10^{-527}	10^{-132}	10^{-68}
10	10^{-821}	10^{-819}	10^{-1004}	10^{-1004}	10^{-1001}	10^{-249}	10^{-128}
11	10^{-1003}	10^{-1003}				10^{-472}	10^{-242}
12						10^{-896}	10^{-460}
13						10^{-1001}	10^{-876}

TABLE 3. QUADERF errors at each level of computation.

In the results in Table 1, the error function quadrature program (QUADERF) produced highly accurate answers, each less than the target tolerance of 10^{-400}, on all problems except the last one, including all problems with infinite derivatives and blow-up singularities. It was several times slower than QUADGS on the first set of problems, but its timing was comparable to QUADGS on Problems 13 and considerably faster than QUADGS on Problem 14. In the results in Table 2, it achieved the full target tolerance of 10^{-1000} on all problems, with reasonable runtimes.

The tanh-sinh quadrature program (QUADTS) also achieved accuracies within the target tolerance in every case in both tables except for Problem 15 in Table 1. What's more, its runtimes were consistently better than QUADERF. Its initialization times were four times faster than QUADERF at 400 digits, and ten times faster at 1,000 digits.

Quadratic convergence, or in other words, the near-doubling of correct digits with each successive level, after the first few levels, was evident in all three schemes for problems in which fully accurate results were obtained. As an illustration, we include in Table 3 the actual errors of the QUADERF program at each of 13 levels, for a selection of the test problems, based on 1,000-digit runs.

The value of the secondary epsilon scheme mentioned in Section 3 is evident in the highly accurate results achieved by both QUADERF and QUADTS on problems with blow-up singularities, notably Problems 7, 10, and 12. When these three problems are run without the secondary epsilon scheme at 400 digit precision (i.e., when $p_2 = p_1 = 400$ and $\epsilon_2 = \epsilon_1 = 10^{-400}$), both the QUADERF and QUADTS programs produce results accurate to only 200 digits. At 1,000-digit precision, without the secondary epsilon scheme, the two programs achieve only 500 correct digits. But with the secondary epsilon scheme, both programs produce fully accurate results on all three problems.

None of the three programs did well on Problem 15 (see Table 1). Indeed, this problem was included, in part, to illustrate that even rather sophisticated quadrature programs can stumble on some rather innocent-looking problems. In this particular problem, none of the three quadrature programs could not handle the highly oscillatory behavior of the integrand $t^7 \sin(1/t)$ near $t = 0$, where the mth derivative increases as $1/t^{2(m-4)+1}$ for small t. So, for example, when this function is transformed to the entire real line, as described in Section 4, its high-order derivatives do not go to zero for large t, and thus the Euler-Maclaurin theorem does not imply a high-accuracy quadrature result. Note that the function itself is bounded, so the secondary epsilon scheme does not help here. For that matter, even 100-digit arithmetic is clearly wasted on this problem—a result correct to 25 digits or so can be computed in a fraction of a second by employing the double-double or quad-double software (accurate to 32 and 64 digits, respectively) available at the ARPREC website [Bailey et al. 02].

The error estimation scheme given in Section 5 performed very well in these tests. For all problems and for each of the three programs, it produced estimated error values that were within four orders of magnitude of the actual errors in the results, and in most cases were within one order. This was true both for cases where the final quadrature result was fully accurate and for those for which it was not.

We should also comment here that as far as we can tell, our implementations of these quadrature algorithms, as well as the evaluation of the functions involved in the integrands and the accumulation of sums, do not appear to suffer significantly from numerical round-off error. This is indicated not only by the highly accurate results we have achieved, but also by some separate tests that we have done. Evidently in these calculations, the 412-digit (or 1,012-digit) internal working precision is sufficient to cover the numerical error that arises in the computation of abscissa-weight pairs and in function evaluations. In cases (such as Problem 15 in Table 1) where only a few accurate digits were obtained, numerical round-off error was not a factor.

9. SUMMARY

Each of these quadrature programs has proven its value in a certain domain of quadrature problems. The Gaussian quadrature program is extremely fast and accurate for continuous, well-behaved integrands, although it requires a lengthy initialization. Both the error function and tanh-sinh programs were able to evaluate all problems except for Problem 15 in Table 1, including all problems with infinite derivatives and blow-up singularities at the endpoints, to the full target precision of 400 or 1,000 digits. We have also tried these programs at even higher precision levels, with similarly accurate results (but significantly longer runtimes, as expected).

Overall, the tanh-sinh scheme appears to be the best for integrands of the type most often encountered in experimental math research. In addition to its excellent accuracy and runtime performance, the initialization cost for this scheme is much less than the other two, particularly at 1,000-digit precision. This is not surprising, since the computational cost of this procedure has a lower-rate growth formula than the other two schemes ($p^2 \log p$ instead of $p^3 \log p$). The tanh-sinh scheme is not a "universal" quadrature scheme, since, for instance, it was not able to handle Problem 15, but as we mentioned in the introduction, it is unlikely that a truly "universal" quadrature scheme exists.

Our implementations of these three schemes (both C++ and Fortran-90 versions), as well as the associated ARPREC arbitrary precision computation software, are available from the website http://crd.lbl.gov/~dhbailey/mpdist.

We wish to add here that each of the three schemes described above are well suited for parallel computation, both for computation of the abscissa-weight pairs and for the evaluation of integrals. This is because the key parts of the computation are naturally parallel. Note, for example, that each of the individual abscissa-weight pairs can be computed independently, in all three quadrature initialization schemes. The same applies to each individual function evaluation in the accumulation of the approximation to the integral, at a given level, although some care must be taken to insure good load balance on a parallel system. Initial results indicate scalability of over 700 times, for large (2,000-digit) problems, using up to 1,024 CPUs, thus sharply reducing runtimes for these quadrature calculations [Bailey and Borwein 05].

ACKNOWLEDGMENTS

The work of the first author was supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the US Department of Energy, under contract number DE-AC03-76SF00098. This work was also supported in part by the National Science Foundation under Grant DMS-0342255.

REFERENCES

[Ahmed 02] Zafar Ahmed. "Definitely an Integral." *American Mathematical Monthly* 109:7 (2002), 670–671.

[Atkinson 93] Kendall E. Atkinson. *Elementary Numerical Analysis*. New York: John Wiley and Sons, 1993.

[Bailey 93] David H. Bailey. "Multiprecision Translation and Execution of Fortran Programs." *ACM Transactions on Mathematical Software* 19 (1993), 288–319.

[Bailey 95] David H. Bailey. "A Fortran-90 Based Multiprecision System." *ACM Transactions on Mathematical Software* 21 (1995), 379–387.

[Bailey and Borwein 05] David H. Bailey and Jonathan M. Borwein. "Highly Parallel, High-Precision Numerical Integration." Manuscript, 2005. Available from World Wide Web (http://crd.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf).

[Bailey and Broadhurst 00] David H. Bailey and David Broadhurst. "Parallel Integer Relation Detection: Techniques and Applications." *Mathematics of Computation* 70:236 (2000), 1719–1736.

[Bailey et al. 02] David H. Bailey, Yozo Hida, Xiaoye S. Li, and Brandon Thompson. "ARPREC: An Arbitrary Precision Computation Package." Technical Report LBNL-53651, 2002. Software and documentation available from World Wide Web (http://crdl.bl.gov/~dhbailey/mpdist).

[Chiarella and Reichel 68] C. Chiarella and A. Reichel. "On the Evaluation of Integrals Related to the Error Function." *Mathematics of Computation* 22 (1968), 137–143.

[Crandall 96] Richard E. Crandall. *Topics in Advanced Scientific Computation*. New York: Springer-Verlag, 1996.

[Davis and Rabinowitz 84] Philip J. Davis and Philip Rabinowitz. *Methods of Numerical Integration*. New York: Academic Press, 1984.

[Golub and Welsch 69] G. H. Golub and J. H. Welsch. "Calculation of Gauss Quadrature Rules." *Mathematics of Computation* 22 (1969), 221–230.

[Kahan 04] William Kahan. Personal communication, February 2004.

[Krommer and Ueberhuber 98] Arnold R. Krommer and Christoph W. Ueberhuber. *Computational Integration*. Philadelphia: SIAM, 1998.

[Mori 91] M. Mori. "Developments in the Double Exponential Formula for Numerical Integration." In *Proceedings of the International Congress of Mathematicians*, pp. 1585–1594. Berlin: Springer-Verlag, 1991.

[Press et al. 86] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes*. Cambridge, UK: Cambridge University Press, 1986.

[Takahasi and Mori 74] H. Takahasi and M. Mori. "Double Exponential Formulas for Numerical Integration." *Publications of RIMS, Kyoto University* 9 (1974), 721–741.

David H. Bailey, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (dhbailey@lbl.gov)

Karthik Jeyabalan, Cornell University, Ithaca, NY 14853 (kj44@cornell.edu)

Xiaoye S. Li, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (xsli@lbl.gov)

Received August 3, 2004; accepted March 2, 2005.