The numerical evaluation of a challenging integral

Walter Gautschi*

Abstract

Standard numerical analysis tools, combined with elementary calculus, are deployed to evaluate a densely and wildly oscillatory integral proposed as a computational problem in the *SIAM 100-Digit Challenge.* Numerical results are presented to accuracies up to 64 decimal digits.

1 Introduction

The *SIAM 100-Digit Challenge* [2], launched early in 2002 by L.N. Trefethen, consists of a collection of ten challenging projects in numerical computation, one of which (the first) is computing the integral

$$(1) \qquad I = \int_0^1 t^{-1} \cos(t^{-1} \ln t) \mathrm{d}t.$$

The integrand not only is densely oscillating as $t \downarrow 0$, but at the same time the amplitudes of the oscillations tend to infinity like t^{-1}. Although the problem has been solved in [2] to as many as 10,000 digits (see the link Extreme Digit Hunting in the web site of [2]), the solution given requires integration in the complex plane along a path of (nearly) steepest descent. It thus uses techniques that are well beyond the scope of a typical course in numerical analysis. Here we attempt to provide a solution which, apart from elementary calculus, uses only tools of standard numerical analysis. Our solution indeed relies on four numerical analysis techniques: the evaluation of oscillatory integrals by Longman's method; Newton's method for solving a nonlinear equation; Gaussian quadrature; and the epsilon algorithm for accelerating the convergence of series. Our method has some points of contact with one of the solutions provided by D. Laurie in [6, §1.6], but also contains original ideas.

*Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907-2066 (`wxg@cs.purdue.edu`).

2   Some elementary calculus

A natural change of variables (also used by D. Laurie in [6]) is $t^{-1} = u$, which yields

$$(2) \qquad I = \int_1^\infty \frac{\cos(u \ln u)}{u} \, \mathrm{d}u.$$

This looks rather familiar if it weren't for $\ln u$: without the logarithm, it is precisely the negative cosine integral $-\mathrm{Ci}(z)$ evaluated at $z = 1$ (cf. [1, eqn 5.2.27]). So we can think of (2) as an "accelerated" cosine integral, the oscillations of the cosine being accelerated as $u \to \infty$. A better grip on these oscillations can be had if one makes one more change of variable,

$$(3) \qquad u \ln u = x.$$

We are interested here in the range $u \geq 1$, in which $u \ln u$ is monotonically increasing (this is so even for $u \geq e^{-1}$). There exists, therefore, a unique inverse function $u = u(x)$ on $x \geq 0$ (see Fig. 1). Intersecting the curve $x = u \ln u$ with the line $x = u$ yields $u = x = e$, and it is clear from the graph on the left of
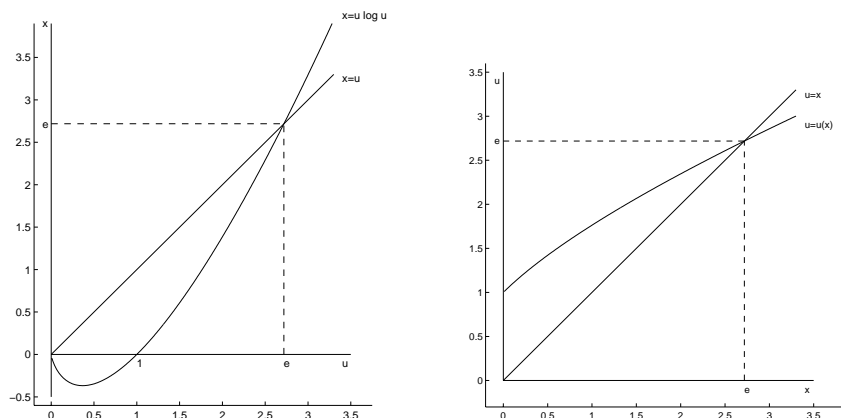

Figure 1: The function $x = u \log u$ and its inverse

Fig. 1 that $u(x) > x$ if $x < e$ and $u(x) < x$ if $x > e$. It is also elementary to show that $u(0) = 1$, $u'(0) = 1$, and that $u'(x)$ for $x > 0$ is positive and decreasing, while $u''(x)$ is negative and decreasing in absolute value. The function $u(x)$ is thus quite smooth and relatively slowly varying; see the graph on the right of Fig. 1.

After the change of variable (3), our integral assumes the form

$$(4) \qquad I = \int_0^\infty \frac{\cos x}{x + u(x)} \, \mathrm{d}x,$$

which will be our starting point for numerical evaluation.

3   An infinite series

The zeros of the integrand in (4) are at the positive odd multiples of $\pi/2$. Following ideas of Longman [7], [8], we integrate over each half-wave (quarter-wave at the beginning) and write

$$I = \int_0^{\pi/2} \frac{\cos x}{x + u(x)}\,\mathrm{d}x + \sum_{k=1}^{\infty} \int_{(2k-1)\pi/2}^{(2k+1)\pi/2} \frac{\cos x}{x + u(x)}\,\mathrm{d}x.$$

Changing variables in the integrals of the series yields

$$(5) \qquad I = \int_0^{\pi/2} \frac{\cos x}{x + u(x)}\,\mathrm{d}x + \sum_{k=1}^{\infty} (-1)^k \int_{-\pi/2}^{\pi/2} \frac{\cos t\,\mathrm{d}t}{t + k\pi + u(t + k\pi)}\,.$$

Clearly, on the right we have an alternating series with terms decreasing in absolute value as k increases. Known as a *Leibniz series*, it has the property that if the series is truncated at some $k = n$, $n \geq 1$, the error in absolute value is bounded by the absolute value of the next term (cf., e.g., [9, §11.5]). This provides a method of computation that has a *built-in error bound*. The downside of it, of course, is very slow convergence. To get an error bound of $\frac{1}{2}10^{-5}$, for example, one needs some 116,000 terms!

A more sensible approach is to evaluate a sequence of partial sums and apply to this sequence a convergence acceleration scheme. Longman used somewhat outmoded acceleration methods (Euler's transformation in [7] and an extension thereof in [8]). More powerful methods have since been developed, one being the epsilon algorithm that works particularly well in our case. Before we discuss this, we need to concern ourselves with how best to calculate the individual terms of the series. There are two issues here: computing the inverse function $u(x)$ of $x = u \ln u$ for any $x > 0$, and doing the integrals. This is taken up in the next two sections.

4   Computing the function $u(x)$

According to (3), we have to solve the nonlinear equation

$$(6) \qquad\qquad f(u) = 0, \quad f(u) = u \ln u - x.$$

Newton's method,

$$(7) \qquad\qquad u_{\nu+1} = u_\nu - \frac{f(u_\nu)}{f'(u_\nu)}, \quad \nu = 0, 1, 2, \ldots,$$

seems to be as good a method as any, provided it is started with an appropriate initial value u_0. In the case at hand, (7) takes the form

$$(8) \qquad\qquad u_{\nu+1} = \frac{u_\nu + x}{1 + \ln u_\nu}, \quad \nu = 0, 1, 2, \ldots\,.$$

We recall the simple geometric interpretation of (7): given u_ν, one determines the zero $u_{\nu+1}$ not of f, but of the tangent line to f passing through $f(u_\nu)$. In general, Newton's method converges only locally, i.e., for u_0 sufficiently close to the zero. In our case, however, the method converges globally, i.e., for any $u_0 > 0$. Indeed, if $u_0 > u(x)$, then by the convexity of f and the geometric interpretation of Newton's method given above, the iterates u_ν converge monotonically decreasing to $u(x)$. Otherwise, when $u_0 < u(x)$, then $u_1 > u(x)$, and from then on we again have monotonic convergence. Nevertheless, an effort should be made to choose u_0 in such a manner that convergence is reasonably fast.

The problem, in another context, was already considered in [5, p. 51], where for $x \leq 10$ the starting value

$$
(9) \qquad
\begin{aligned}
u_0 = {} & 1.0125 + .8577x - .129013x^2 + .0208645x^3 \\
& - .00176148x^4 + .000057941x^5
\end{aligned}
$$

was derived from a truncated Chebyshev expansion of $u(x)$ on $[0, 10]$, and for $x > 10$

$$
(10) \qquad u_0 = \frac{x}{\ln x} \frac{1}{1 - \ln \ln x / (1 + \ln x)}
$$

from the asymptotic approximation $u(x) \sim x/\ln x$, $x \to \infty$, followed by one step of Newton's method. With these choices of u_0, and an accuracy requirement of 15 significant decimal digits, it is found that four steps in the iteration (8) suffice for all $x > 0$.

We remark that the function $u(x)$ is related to what in [6] is referred to as Lambert's W Function, the solution of $we^w = x$. Evidently, $u(x) = e^{w(x)}$. Lambert's W Function is available in the Matlab Symbolic Toolbox through the routine `lambertw.m`. Since it uses symbolic computation, it is much slower than our own Matlab implementation of (four steps in) (8).

5 Gaussian quadrature

Each integral in (5) has an integrand that is quite regular and smooth on the respective interval. Gauss–Legendre quadrature, therefore, is an easy and accurate way to compute the integrals. Note that the intervals of integration have to be transformed to the canonical interval $[-1, 1]$ before Gauss–Legendre quadrature is applied. It is also worth noting that the cosine function in all the integrals of (5) has to be evaluated only once at the respective Gauss abscissae. Combined with the observed fact that the first integral requires a 20-point Gauss formula for an accuracy of 15 significant decimal digits, and all remaining integrals only a 13-point formula, the evaluation of the (partial sums of the) series is quite fast. As will be seen in the next section, only 21 terms of the series will be needed for the same accuracy.

One can speculate as to why the first integral in (5) requires more Gauss points, for the same accuracy, than the remaining integrals. One "explanation"

is that upon replacing $u(x)$ in the first integral by $u(0) = 1$ and $u(t + k\pi)$ in the kth integral of the series by $u(k\pi)$, the former exhibits a pole at -1, a distance of 1 away from the interval $[0, \pi/2]$, while the pole of the latter at $-k\pi - u(k\pi)$ is much further to the left (even when $k = 1$) to affect convergence significantly. An even better explanation is provided by the same argument with $u(x)$ resp. $u(t + k\pi)$ replaced by suitable mean values of u on the respective intervals.

6   The epsilon algorithm

The epsilon algorithm for accelerating the convergence of series has a long history (see, e.g., [3, p. 79]). It can actually be viewed from different angles—as a generalization of Aitken's Δ^2 process (*ibid.*, §2.3), as an algorithmic implementation of Stieltjes's summation procedure based on continued fractions (see [4, §4.5]), or in terms of Padé approximation [3, p. 93]. Given the sequence $s_1, s_2, \ldots, s_n$ of the first n partial sums of the series $(n \geq 2)$, it generates an array $E = \{\varepsilon_{m,k}\}$ of dimension $n \times (n + 1)$ by means of

```
for k=1,2,...,n do
    ε_{k,2} = s_k
end
for k=3,4,...,n+1 do
    for m=1,2,...,n+2-k do
```
$$\varepsilon_{m,k} = \varepsilon_{m+1,k-2} + \frac{1}{\varepsilon_{m+1,k-1} - \varepsilon_{m,k-1}}$$
```
    end
end
```

where the values of interest (the accelerated approximations) are located in the even-numbered columns of E (the first consisting of zeros). If, as we recommend, n is taken to be an odd integer, then the most accurate results are usually among the $(n + 1)/2$ entries $s_a(k) = E(n - 2k + 2, 2k)$, $k = 1, 2, \ldots, (n + 1)/2$, the last few of them often being the best.

We illustrate this in Table 1 for the series (5), using $n = 21$ and the computational simplifications noted in §§4 and 5. It can be seen from this table that 21 terms in the series indeed suffice to arrive at an accuracy of 15 significant digits. The last three entries in Table 1[1] in fact are identical with the result given in [6, Table 1.2].

The Matlab program that produced Table 1 is `challeps.m`; it calls on the routine `uofx.m` computing the function $u(x)$, the routine `epsalg.m` implementing the epsilon algorithm, and utility routines for generating the necessary Gaussian quadrature rules. The latter are contained in the package `OPQ` on the web site

---

[1] In Matlab's 16-digit arithmetic, these three entries are not exactly the same and therefore do not cause division by zero in the epsilon algorithm.

Table 1: Epsilon algorithm applied to the first 21 terms of the series (5)

k	$s_a(k)$
1	0.335055792734608
2	0.323374785249233
3	0.323367455102760
4	0.323367431889382
5	0.323367431682102
6	0.323367431677957
7	0.323367431677792
8	0.323367431677781
9	0.323367431677779
10	0.323367431677779
11	0.323367431677779

http://www.cs.purdue.edu/archives/2002/wxg/codes;

all other Matlab routines referenced in this paper can be found in `CHA` on the same web site and can be downloaded individually, including those in `OPQ`.

7   The quest for more precision

A transcription of `challeps.m` into quadruple-precision Fortran allows us to produce results of twice the accuracy that Matlab can provide. The program, of course, has to be properly tuned for meeting the higher accuracy requirements. One finds that for 32-digit accuracy, Gauss quadrature requires 41 resp. 28 quadrature points to do the first and remaining integrals, respectively, in (5). Newton's method to compute $u(x)$, as expected, requires 5 iterations, one more than before. (Remember: Newton's method eventually, on each iteration, doubles the number of correct digits.) The Fortran program, `qchalleps.f`, that incorporates these adjustments, as well as the program `quofx.f` for calculating $u(x)$ in quadruple precision are downloadable from the web site quoted in §6. Additional quadruple-precision programs that are related to Gaussian quadrature and are used by `qchalleps.f` are from the Fortran package `ORTHPOLq` on the web site

http://www.cs.purdue.edu/archives/2001/wxg/codes

Finally, the epsilon algorithm, implemented in `qepsalg.f`, was found to require 43 partial sums of the series to provide this increased accuracy. The program `qchalleps.f` then returns the following result to 32 digits,

(11) $I = .32336743167777876139937008795217.$

It agrees with all 26 digits of the most accurate result reported in [6, p. 22].

To move on to still higher precision, we prepared symbolic/variable-precision versions `schalleps.m`, `suofx.m`, and `sepsalg.m` of the corresponding Matlab programs (without the "s" in front) allowing computation with an arbitrary number `dig` of decimal digits. Along with them, symbolic versions of some of the `OPQ` routines are also required. Run with $dig = 32$, the routine `schalleps.m` returned the same answer as given in (11) except for the last digit, which (incorrectly) is 0 instead of 7. It took about 17 minutes to run on a Sun Ultra 5 workstation (with a 360 MHz UltraSPARC-III processor and 256 MB of memory). Emboldened by this encouraging experience, we ventured to larger values of `dig`, in increments of 8 digits, starting with $dig = 40$. Each time, we determined the number i_{Newt} of Newton iterations, the numbers m_1, m_2 of Gauss quadrature points, and the number n of partial sums, required for the respective accuracies. Also monitored was the run time t. A summary of the results is given in Table 2. In view of the steadily increasing run times, we stopped at

Table 2: Performance data for `schalleps.m`

dig	i_{Newt}	m_1	m_2	n	t (in min)
40	6	50	33	53	29
48	6	61	39	63	41
56	6	70	46	73	65
64	6	80	57	83	88

$dig = 64$; the routine `schalleps.m` then gave us the result

$$I = .3233674316777787613993700879521704466510466257254696616810364434.$$
(12)

Although the author is a novice in symbolic high-precision calculations, he is fairly confident that all digits in (12) are correct. (The calculations were actually done with $dig = 68$ and $n = 87$ to ascertain the last few digits in (12).) Each increment of precision resulted in a value consistent with the lower-precision result except for one or two terminal digits. Comparing with the 10,000-digit result of [2], the result in (12) is in fact correct to all digits shown.

Acknowledgments

References

[1] ABRAMOWITZ, M. AND I.A. STEGUN, eds, *Handbook of mathematical functions*, National Bureau of Standards, Applied Mathematics Series 55, U. S. Government Printing Office, Washington, DC, 1964.

[2] BORNEMANN, F., D. LAURIE, S. WAGON, AND J. WALDVO-GEL, *The SIAM 100-digit challenge: a study in high-accuracy numerical computing*, SIAM, Philadelphia, PA, 2004. (Also, see http://www.siam.org/books/100digitchallenge.)

[3] BREZINSKI, C. AND REDIVO ZAGLIA, M., *Extrapolation methods: theory and practice*, Elsevier, Amsterdam, 1991.

[4] BULIRSCH, R., Darstellung von Funktionen in Rechenautomaten, in: *Mathematische Hilfsmittel des Ingenieurs* (R. Sauer and I. Szabó, eds), Teil III, Die Grndlagen der mathematischen Wissenschaften in Einzeldarstellungen, Band 141, Springer, Berlin, 1968, pp. 352–446.

[5] GAUTSCHI, W., Computational aspects of three-term recurrence relations, *SIAM Review* 9 (1967), 24–82.

[6] LAURIE, D., A twisted tail, In [2], pp. 17–31.

[7] LONGMAN, I.M., Note on a method for computing infinite integrals of oscillatory functions, *Proc. Cambridge Philos. Soc.* 52 (1956), 764–768.

[8] LONGMAN, I.M., A method for the numerical evaluation of finite integrals of oscillatory functions, *Math. Comp.* 14 (1960), 53–59.

[9] STEWART, J., *Calculus: early transcendentals*, 5th ed., Brooks/Cole Publ. Co., Pacific Grove, CA, 2003.