SIAM Review
Vol. 30, No. 4, December 1988

003

ON THE COMPLEXITY OF FAMILIAR FUNCTIONS AND NUMBERS*

J. M. BORWEIN† AND P. B. BORWEIN†

Abstract. This paper examines low-complexity approximations to familiar functions and numbers. The intent is to suggest that it is possible to base a taxonomy of such functions and numbers on their computational complexity. A central theme is that traditional methods of approximation are often very far from optimal, while good or optimal methods are often very far from obvious. For most functions, provably optimal methods are not known; however the gap between what is known and what is possible is often small. A considerable number of open problems are posed and a number of related examples are presented.

Key words. elementary functions, pi, low-complexity approximation, reduced-complexity approximation, rational approximation, algebraic approximation, computation of digits, open problems

AMS(MOS) subject classifications. 68C25, 41A30, 10A30

1. Introduction. We examine various methods for evaluating familiar functions and numbers to high precision. Primarily, we are interested in the asymptotic behavior of these methods. The kinds of questions we pose are:

(1) How much work (by various types of computational or approximation measures) is required to evaluate n digits of a given function or number?

(2) How do analytic properties of a function relate to the efficacy with which it can be approximated?

(3) To what extent are analytically simple numbers or functions also easy to compute?

(4) To what extent is it easy to compute analytically simple functions?

Even partial answers to these questions are likely to be very difficult. Some, perhaps easier, specializations of the above are:

(5) Why is the function $\sqrt{x}$ easier to compute than exp? Why is it only marginally easier?

(6) Why is the Taylor series often the wrong way to compute familiar functions?

(7) Why is the number $\sqrt{2}$ easier to compute than e or π? Why is it only marginally easier?

(8) Why is the number $.1234567891011\cdots$ computationally easier than π or e?

(9) Why is computing just the nth digit of $\exp(x)$ really no easier than computing all the first n digits?

(10) Why is computing just the nth digit of π really no easier than computing all the first n digits?

Answers to (7) and (10) are almost certainly far beyond the scope of current number-theoretic techniques. Partial answers to some of the remaining questions are available.

The traditional way to compute elementary functions, such as exp or log, is to use a partial sum of the Taylor series or a related polynomial or rational approximation. These are analytically tractable approximations, and over the class of such

* Received by the editors February 2, 1987; accepted for publication (in revised form) September 11, 1987. The second author's research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

† Department of Mathematics, Statistics, and Computing Science, Dalhousie University, Halifax, Nova Scotia, Canada B3H 3J5.

approximations are often optimal or near optimal. For example, the nth partial sums to exp are asymptotically the best polynomial approximations in the uniform norm on the unit disc in the complex plane, in the sense that if s_n is the nth partial sum of the Taylor expansion and p_n is any polynomial of degree n, then for large n,

$$\|\exp(z)-s_n(z)\|_D<\left[1+\frac{4}{n}\right]\|\exp(z)-p_n(z)\|_D.$$

Here $\| \ \|_D$ denotes the supremum norm over the unit disc in the complex plane (see [5]). If the measure of the amount of work is the degree of the approximation, as it has been from a conventional point of view, then the story for exp might end here.

Questions (1)–(3) above have a very elegant answer for polynomial approximation in the form of the Bernstein–Jackson theorems [11]. These, for example, tell us that a function is entire if and only if the error in best uniform polynomial approximation of degree n on an interval tends to zero faster than geometrically, with a similar exact differentiability classification of a function in terms of the rate of polynomial approximation.

If we wish to compute n digits of $\log(x)$ using a Taylor polynomial then we employ a polynomial of degree n and perform $O(n)$ rational operations, while for $\exp(x)$ we require $O(n/\log n)$ rational operations to compute n digits. The slight improvement for exp reflects the faster convergence rate of the Taylor series. Padé approximants, best rational approximants and best polynomial approximants all behave in roughly the same fashion, except that the constants implicit in the order symbol change [5], [8].

A startling observation is that there exist rational functions that give n digits of log, exp, or any elementary function but require only $O((\log n)^k)$ rational operations to evaluate. These approximants are of degree $O(n)$ but can be evaluated in $O((\log n)^k)$ infinite-precision arithmetic operations. The simplest example of such a function is x^n which can be evaluated in $O(\log n)$ arithmetic operations by repeated squaring. While we cannot very explicitly construct these low-rational-complexity approximations to exp or log, it is clear that much of their simplicity results from squarings of intermediate terms. The moral is that it is appropriate and useful to view x^n as having the complexity of a general polynomial of degree $\log n$, not of degree n.

The existence of such approximants is a consequence of the construction of low-bit-complexity algorithms for log and π resting on the Arithmetic-Geometric Mean (AGM) iteration of Gauss, Lagrange, and Legendre (see §2 for definitions). These algorithms were discovered and examined by Beeler, Gosper, and Schroeppel [3], Brent [9], and Salamin [21] in the 1970s. A complete exposition is available in [5]. These remarkable algorithms are both theoretically and practically faster than any of the traditional methods for extended precision evaluation of elementary functions. The exact point at which they start to outperform the usual series expansions depends critically on implementation; the switchover comes somewhere in the 100- to 1000-digit range.

The main purpose of this paper is to catalogue the known results on complexity of familiar functions. We now appear to know enough structure to at least speculate on the existence of a reasonable taxonomy of functions based on their computational complexity. Here we have in mind something that relates computational properties of functions to their analytic or algebraic properties, something vaguely resembling the Bernstein–Jackson theorems in the polynomial case.

Likewise we would like to suggest the possibility of a taxonomy of numbers based on their computational nature. Here, we are looking for something that resembles

Mahler's classification of transcendendentals in terms of their rate of algebraic approximation [15].

It is not our intention to provide a taxonomy; this must await further progress in the field. We do hope, however, to present enough examples and pose enough interesting questions to persuade the reader that it is fruitful to pursue such an end.

2. Definitions. We consider four notions of complexity.

(1) *Rational complexity.* We say that a function f has *rational complexity* $O_{\text{rat}}(s(n))$ on a set A if there exists a sequence of rational functions R_n so that

(a) $|R_n(x)-f(x)|<10^{-n}$ for all $x\in A$;

(b) asymptotically, R_n can be evaluated using no more than $O(s(n))$ rational operations (i.e., infinite-precision additions, subtractions, multiplications, and divisions).

That exp has rational complexity $O_{\text{rat}}(\log^3 n)$ means that there is a sequence of rational functions, the nth being evaluable in roughly $\log^3 n$ arithmetic operations, giving an n-digit approximation to exp. The subscript on the order symbol is for emphasis.

We will sometimes use Ω and Ω_{rat} as the lower bound order symbols. Whenever we talk about "n-digit precision" or "computing n digits" we mean computing to an accuracy of 10^{-n}.

(2) *Algebraic complexity.* We say that a function f has *algebraic complexity* $O_{\text{alg}}(s(n))$ on a set A if there exists a sequence of algebraic functions A_n so that

(a) $|A_n(x)-f(x)|<10^{-n}$ for all $x\in A$;

(b) asymptotically, all the A_n can be evaluated using no more than $O(s(n))$ algebraic operations (i.e., infinite-precision solutions of a fixed number of prespecified algebraic equations).

This algebraic complexity measure allows us, for example, to use square root extractions in the calculation of the approximants and to count them on an equal footing with the rational operations. This is often appropriate because, from a bit-complexity point of view, root extraction is equivalent to multiplication (see §4). Note that we allow only a finite number of additional algebraic operations—so while we might allow for computing square roots, cube roots, and seventeenth roots, we would not allow an infinite number of different orders of roots.

Neither of the above measures takes account of the fact that low-precision operations are easier than high-precision operations.

(3) *Bit complexity.* We say that a function f has *bit complexity* $O_{\text{bit}}(s(n))$ on a set A if there exists a sequence of approximations B_n so that

(a) $|B_n(x)-f(x)|<10^{-n}$ for all $x\in A$;

(b) B_n is the output of an algorithm (given input n and x) that evaluates the B_n to n-digit accuracy using $O(s(n))$ *single-digit* operations ($+$, $-$, $\times$).

This is the appropriate measure of time complexity on a serial machine. (See [1] for more formal definitions.)

We wish to capture in the next definition the notion of how complex it is to compute only the nth digit of a function.

(4) *Digit complexity.* We say that a function f has *digit complexity* $O_{\text{dig}}(s(n))$ on a set A if there exists a sequence of approximations D_n so that

(a) $D_n(x)$ gives the nth digit of $f(x)$. By this we mean that $D_n(x)$ differs from the n through $(n+k)$th digits of $f(x)$ by at most 10^{-k} for any preassigned fixed k;

(b) D_n is the output of an algorithm (given input n and x) that evaluates the D_n to k digits using $O(s(n))$ *single-digit* operations.

This definition of agreement of nth digits takes account of the fact that sequences of repeated nines can occur. We really want to say that .19999··· and .2000··· agree in the first digit. As it stands, the definition above exactly computes only the nth digit to a probability dependent on k.

It is also assumed that accessing the kth through nth digit of input of x is an $O_{\text{bit}}(\max(n-k, \log k))$ operation, so that accessing the first n digits is $O_{\text{bit}}(n)$ while accessing just the nth bit is $O_{\text{bit}}(\log n)$.

Addition is $O_{\text{rat}}(1)$, $O_{\text{alg}}(1)$, $O_{\text{bit}}(n)$, and $O_{\text{dig}}(\log n)$. Here we take the set A, where we seek a uniform algorithm, to be the unit square in $\mathbb{R}^2$. The usual addition algorithm gives the upper bounds shown above. Addition is one of the very few cases where we know the exact result. Trivial uniqueness considerations show that addition is $\Omega_{\text{bit}}(n)$, and hence all the above orders are exact.

It comes as a major surprise of this side of theoretical computer science that the usual way of multiplying is far from optimal from a bit-complexity point of view. The usual multiplication algorithm has bit complexity $\Omega_{\text{bit}}(n^2)$. However, it is possible to construct a multiplication which is $O_{\text{bit}}(n \log n \log\log n)$. This is based on the Fast Fourier Transform and is due to Schönhage and Strassen (see [1], [16]). The extent to which the log terms are necessary is not known. Given a standard model of computation the best known lower bound is the trivial one, $\Omega_{\text{bit}}(n)$. We will denote the *bit complexity of multiplication* by $M(n)$.

3. A table of results. The state of our current knowledge is contained in Table 1. The orders of the various measures of complexities for computing n digits (or in the final case the nth digit) compose the columns. In each case, except addition, the only upper bound we know for the digit complexity is the same as the bit-complexity bound. When we deal with functions, we assume that we are on a compact region of the domain of the given function that is bounded away from any singularities and that contains an interval. Numbers may be considered as functions whose domain is a singleton.

For our purposes *hypergeometric functions* are functions of the form

$$f(x) := \sum a_n x^n \quad \text{where } a_n/a_{n-1} = R(n)$$

and R is a fixed rational function (with coefficients in $\mathbb{Q}$).

TABLE 1

Type of function	O_{rat}	O_{alg}	O_{bit}	Ω_{dig}
(1) Addition	1	1	n	$\log n$
(2) Multiplication	1	1	$n \log n \log\log n$	n
(3) Algebraic (nonlinear)	$\log n$	1	$M(n)$	n
(4) log (complete elliptic integrals)	$\log^2 n$	$\log n$	$(\log n)M(n)$	n
(5) exp	$\log^3 n$	$\log^2 n$	$(\log n)M(n)$	n
(6) Elementary (nonlinear)	$\log^k n$	$\log^k n$	$(\log n)M(n)$	n
(7) Hypergeometric (over $\mathbb{Q}$)	$n^{1/2+}$	$n^{1/2+}$	$(\log^2 n)M(n)$	n
(8) Gamma and zeta	$n^{1/2+}$	$n^{1/2+}$	$n^{1/2+}M(n)$	n
(9) Gamma and zeta on $\mathbb{Q}$	$n^{1/2+}$	$n^{1/2+}$	$(\log^2 n)M(n)$	$\log n$
(10) pi, log (2), $\Gamma(\frac{1}{3})$	$\log^2 n$	$\log n$	$(\log n)M(n)$	$\log n$
(11) Euler's constant (Catalan's constant)	$n^{1/2+}$	$n^{1/2+}$	$(\log^2 n)M(n)$	$\log n$

Elementary functions are functions built from rational functions (with rational coefficients) exp and log by any number of additions, multiplications, compositions, and solutions of algebraic equations.

A number of techniques are employed in deriving Table 1. Our intention is to indicate the most useful of these without going into too much detail. The next four sections outline the derivations of most of the bounds.

4. Newton's method. The calculation of algebraic functions, given that we have algorithms for addition and multiplication, is entirely an exercise in applying Newton's method to solving equations of the form $f(x)-y=0$. Newton's method for $1/x-y=0$ gives the iteration

(a) $x_{n+1}:=2x_n-yx_n^2$,

while for $x^2-y=0$ the iteration is

(b) $x_{n+1}:=(x_n+y/x_n)/2$.

These two iterations converge quadratically. Thus $O(\log n)$ iterations give n digits of $1/y$ and $\sqrt{y}$, respectively, and we have given an $O_{\text{rat}}(\log n)$ algorithm for square root extraction.

The quadratic rate of convergence is only half the story. Because Newton's method is *self-correcting*, in the sense that a small perturbation in x_n does not change the limit, it is possible to start with a single-digit estimate and double the precision with each iteration. Thus the bit complexity of root extraction is

$$O(M(1)+M(2)+M(4)+\cdots+M(n))=O(M(n)).$$

This leads to $O_{\text{bit}}(M(n))$ algorithms for root extraction and division, and a similar analysis works for any algebraic function. This explains most of (1)–(3) in Table 1. We also have the interesting result that the computation of digits of any algebraic number is asymptotically no more complicated than multiplication. (These results on the complexity of algebraic functions may be found in [5] and [9].)

The approximation in (a), x_n, is in fact the (2^n-1)st Taylor polynomial to $1/y$ at 1. In (b), x_n is in fact the $(2^n, 2^n-1)$st Padé approximant to $\sqrt{y}$ at 1. (See [5] or [11] for further material on Padé approximants.) This is one of the very few cases where Newton's method generates familiar approximants.

Newton's method is also useful for inverting functions. The inverse of f is computed from the iteration

$$x_{n+1}:=x_n-[f(x_n)-y]/f'(x_n).$$

For any reasonable f this gives the same bit complexity estimate for f^{-1} as for f. Inverting by Newton's method multiplies the rational and algebraic complexities by $\log n$.

5. The AGM. The two-term iteration with starting values $a_0:=x\in(0,1]$ and $b_0:=1$ given by

$$a_{n+1}:=(a_n+b_n)/2, \qquad b_{n+1}:=\sqrt{(a_n\cdot b_n)}$$

converges quadratically to $m(1,x)$, where

$$\frac{1}{m(1,x)}=\frac{2}{\pi}\int_0^{\pi/2}\frac{dt}{\sqrt{1-(1-x^2)\sin^2 t}}.$$

This is the arithmetic-geometric mean iteration of Gauss, Lagrange, and Legendre. This latter complete elliptic integral is $2K'(x)/\pi$ and is a nonelementary transcendental function with complexity

$$O_{\mathrm{alg}}(\log(n)), \quad O_{\mathrm{rat}}(\log^2(n)), \quad O_{\mathrm{bit}}(\log(n)M(n)).$$

It is also essentially the only identifiable nonelementary limit of a quadratically converging fixed iteration and as such is of central importance [5].

One way to get a low complexity algorithm for log is to use the logarithmic asymptote of K' at 0. This gives the estimate

$$|(2/\pi)\log x - 1/m(1,10^{-n}) + 1/m(1,x10^{-n})| < n10^{-2(n-1)}, \quad n>3, \quad x\in[.5,1].$$

Up to computing π, this allows for the derivation of algorithms with the complexity of entry (4) in Table 1. Algorithms for π can be derived from the same kinds of considerations (see [4], [5], [9], [18], [21]). Probably the fastest known algorithm for π is the quartic example given below [5], [2].

ALGORITHM. Let $\alpha_0 := 6 - 4\sqrt{2}$ and $y_0 := \sqrt{2} - 1$. Let

$$y_{n+1} := [1-(1-y_n^4)^{1/4}]/[1+(1-y_n^4)^{1/4}]$$

and

$$\alpha_{n+1} := (1+y_{n+1})^4\alpha_n - 2^{2n+3}y_{n+1}(1+y_{n+1}+y_{n+1}^2).$$

Then $1/\alpha_n$ tends to π quartically and

$$0 < \alpha_n - \frac{1}{\pi} < 16\cdot 4^n \exp(-2\cdot 4^n\pi).$$

The exponential function may be derived from log by inverting using Newton's method. This continues to work for appropriate complex values. The elementary functions are now built from log and exp and the solution of algebraic equations in these quantities. The constant k in the rational- and bit-complexity estimates depends on the number of these equations that require solution. This explains entries (5), (6), and (10) in Table 1, except for $\Gamma(\frac{1}{3})$. (This and a few other values of Γ arise as algebraic combinations of complete elliptic integrals and pi.) (Substantial additional material on this section is to be found in [5].)

6. FFT methods. The Fast Fourier Transform (FFT) is a way of solving the following two problems:

(a) Given the coefficients of a polynomial of degree $n-1$, evaluate the polynomial at all n of the nth roots of unity.

(b) Given the values of a polynomial of degree $n-1$ at the nth roots of unity, compute the coefficients of the polynomial.

These two problems are actually equivalent (see [1], [5], [16]). The important observation made by Cooley and Tukey in the 1960s is that both of these problems are solvable with rational complexity $O_{\mathrm{rat}}(n\log n)$, rather than the complexity of $\Omega_{\mathrm{rat}}(n^2)$ that the usual methods require (i.e., Horner's method). This is an enormously useful algorithm.

We can multiply two polynomials of degree n with complexity $O_{\mathrm{rat}}(n\log n)$ by using the FFT three times. First we compute the values of the two polynomials at $2n+1$ roots of unity. Then we work out the coefficients of the polynomial of degree $2n$ that agrees with the product at these roots.

Variations on this technique allow for the evaluation of a rational function of degree n at n points in $O_{\mathrm{rat}}(n\log^2 n)$ and $O_{\mathrm{bit}}(n\log^2 n\, M(k))$, where k is the precision to which we are working [5].

Fast multiplications are constructed by observing that multiplication of numbers is much like multiplication of polynomials whose coefficients are the digits, the additional complication being the "carries."

How does this give reduced-complexity algorithms? We illustrate with $\log(1-x)$. Let

$$s_{n^2}(x) := \sum_{k=1}^{n^2} \frac{x^k}{k}$$

and write

$$s_{n^2}(x) = \sum_{k=0}^{n-1} x^k p(kn) \quad \text{where} \, p(y) := \sum_{j=1}^{n} \frac{x^j}{j+y}.$$

Now evaluate $p(0)$, $p(n)$, $\cdots$, $pn(n-1)$ using FFT methods, and then evaluate s_{n^2}. This gives an $O_{\text{rat}}(n^{1/2}(\log n)^2)$ and $O_{\text{bit}}(n^{1/2}(\log n)^2 M(n))$ algorithm for log. At any fixed rational value r, we get an $O_{\text{bit}}((\log n)^2 M(n))$ for $\log r$. For this final estimate we must take advantage of the reduced precision possible for intermediate calculations.

This is not as good an estimate as the AGM estimates for log. It is, however, a much more generally applicable method. We can orchestrate the calculation, much as above, for any hypergeometric function. This is how the estimates in line (7) in Table 1 are deduced. Schroeppel [3], [22] shows how a similar circle of ideas can be used to give $O_{\text{bit}}(\log^k n\, M(n))$ algorithms for the solutions of linear differential equations whose coefficients are rational functions with coefficients in $\mathbb{Q}$.[1]

The gamma function, Γ, can be computed from the estimate

$$\left| \Gamma(x) - N^x \sum_{k=0}^{6N} \frac{(-1)^k N^k}{k!(x+k)} \right| < 2Ne^{-N}, \qquad x \in [1,2]$$

(see [5] for details). The zeta function, ζ, is then computable from Riemann's integral [24]:

$$\zeta(x)\Gamma\left(\frac{x}{2}\right)\pi^{-x/2} - \frac{1}{x(x-1)} = \int_1^\infty \frac{t^{(1-x)/2} + t^{x/2}}{t} \sum_{n=1}^{\infty} e^{-n^2 \pi t}\, dt.$$

We truncate both the integral and the sum. These two formulae explain lines (8) and (9) of Table 1.

Catalan's constant

$$G := \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^2}$$

can be computed from Ramanujan's sum

$$\frac{8}{3} G = \frac{\pi}{3} \log(2+\sqrt{3}) + \sum_{m=0}^{\infty} \frac{m!\, m!}{(2m+1)^2 (2m)!},$$

while Euler's constant, γ, can be computed from the asymptotic expansion

$$\gamma = -\log x - \sum_{k=1}^{\infty} \frac{(-x)^k}{k \cdot k!} + O(\exp(-x)), \qquad x > 1.$$

[1] Chudnovsky and Chudnovsky [26] provide a low-bit complexity approach to solutions of linear differential equations in [26].

This gives line (11) of Table 1. Some of the details may be found in [5] and [6].

A variation of the above method for computing γ has been used by Brent and McMillan [10] to compute over 29,000 partial quotients of the continued fraction of γ. From this computation it follows that if γ is rational its denominator exceeds $10^{15,000}$.

7. Digit complexity. The aim of this section is to explain the last column in Table 1. The main observation is that the digit complexity of computing the mth digit ($m \leqq n$) of the product of two n-digit numbers is $\Omega_{\text{dig}}(m)$. This is essentially just a uniqueness argument the details of which may be pursued in [7].

Now suppose that f is analytic around zero (C^3 suffices). Then

$$f(x)=a+bx+cx^2+O(x^3)$$

or equivalently

$$cx^2=f(x)-a-bx+O(x^3).$$

If f is of low-digit complexity then, as above, truncating after one term gives a low-complexity algorithm for $a + bx$. Recall that addition is $O_{\text{dig}}(\log n)$. This in turn gives a low-digit complexity evaluation of cx^2 in a neighborhood of zero, but evaluation of cx^2 is essentially equivalent to multiplication. Once again, the details are available in [7]. Thus, if f is any nonlinear C^3 function it is $\Omega_{\text{dig}}(n)$, or we would have too good an algorithm for calculating the mth digit of multiplication.

We now have the following type of theorem.

THEOREM. *If f is a nonlinear elementary function (on an interval) then f is*

$$O_{\text{bit}}(n(\log n)^k) \quad \textit{and} \quad \Omega_{\text{dig}}(n).$$

This is now close to an exact result. Actually we can say considerably more. For example, we have the following theorem.

THEOREM. *If f is a nonlinear C^3 function (on an interval) then the set of x for which the digit complexity of $f(x)$ is $o(n)$ by <u>any</u> algorithm is of the first Baire category.*

A set of first Baire category is small in a topological sense (see [25]).

We define the class of *sublinear numbers* by calling a number x sublinear if the digit complexity of x is $O_{\text{dig}}(n^{1^-})$. Call α a *sublinear multiplier* if the function αx is sublinear for all $x \in [0, 1]$ (given both α and x as input).

THEOREM. *The set of sublinear multipliers is a nonempty set of the first Baire category.*

Two more definitions are useful in relation to numbers of very low digit complexity. We say that x is *sparse* if x has digit complexity $O_{\text{dig}}(n^\delta)$ for all $\delta > 0$, and we say that α is a *sparse multiplier* if αx is sparse for all $x \in [0, 1]$. Sparse multipliers have sparse digits. Indeed, let $S := \{x \mid \#(\text{nonzero digits of } x \text{ among the first } n \text{ digits}) = O(n^\delta) \text{ for all } \delta > 0\}$.

THEOREM. *The set of sparse multipliers is exactly the set S.*

Thus there are uncountably many sparse multipliers and hence also uncountably many sublinear multipliers.

These are base-dependent notions. The previous theorem shows that $\frac{1}{2}$ is a sparse multiplier base 2 but not base 3. We can prove directly that irrational sparse multipliers must be transcendental. Various questions concerning these matters will be raised in the next sections.

8. Questions on the complexity of functions. The hardest problems associated with Table 1 of §3 concern the almost complete lack of nontrivial lower bound estimates. This reflects the current state of affairs in theoretical computer science. Not only is the question of whether $P = NP$ still open, it is still not resolved that any NP problems are nonlinear. Friedman [13], for example, shows that we can take maxima over the class of polynomially computable functions if and only if $P = NP$ and that we can integrate over this class if and only if $P = \#P$. While these notions are somewhat tangential to our concerns they do indicate that some of our problems are likely to be hard.

One of the reasons for looking at the rational complexity is that it is likely to be a little more amenable to analysis. We can show that exp and log *cannot* have rational complexity $o(\log n)$. This is a consequence of the known estimates in approximating exp and log by rational functions of degree n [5], [8]. Note that n rational operations can generate a rational function of at most degree 2^n. Thus there is only a small gap between the known and best possible rational complexity estimates for log.

Question 1. Does log have rational complexity $O_{\text{rat}}(\log n)$?

The extra power of log in the rational complexity of exp over that of log is almost certainly an artifact of the method. So at least one power of log ought to be removable.

Question 2. Show that exp has rational complexity $O_{\text{rat}}(\log^2 n)$. Does exp have rational complexity $O_{\text{rat}}(\log n)$?

The low-complexity approximants to exp and log are constructed indirectly. It would be valuable to have a direct construction.

Question 3. Construct, as explicitly as possible, approximants to exp and log with complexity $O_{\text{rat}}(\log^k n)$.

There is a big difference in the rational complexity of exp and of Γ. It is tempting to speculate that this is artificial.

Question 4. Does Γ have rational complexity $O_{\text{rat}}(\log^k n)$?

Ideally we would like to identify those functions with this complexity.

Question 5. Classify (analytic) functions with rational complexity $O_{\text{rat}}(\log^k n)$.

This last question is almost certainly very hard.

We would expect there to be little difference between rational complexity and algebraic complexity.

Question 6. Does any of exp, log, or K have rational complexity essentially slower than its algebraic complexity?

In the case of bit complexity, there are no nontrivial lower bounds. At best we can say that the bit complexity is always at least that of multiplication. Thus a crucial first step is the content of the next question.

Question 7. Show that exp, log, or any of the functions we have considered is *not* $O_{\text{bit}}(M(n))$.

It is easy to construct entire functions with very low bit complexity; we simply use very rapidly converging power series. Thus there exist nonalgebraic analytic functions with bit complexity $O_{\text{bit}}(a_n M(n))$, where a_n is any sequence tending to infinity. However, the following question appears to be open.

Question 8. Does there exist a nonalgebraic analytic function with bit complexity $O_{\text{bit}}(M(n))$?

A negative answer to this question would also resolve the question preceding it.

A very natural class to examine is the class of functions that satisfy algebraic differential equations (not necessarily linear). Almost all familiar functions arise in this context. Even an unlikely example like the theta function

$$\theta_3(q) := \sum_{n \in \mathbb{Z}} q^{n^2},$$

satisfies a nonlinear algebraic differential equation, as Jacobi showed (see [20]).

Question 9. How do solutions of algebraic differential equations fit into the complexity table?

We end with a question on digit complexity.

Question 10. Does there exist an analytic function whose digit complexity is essentially faster than its bit complexity? Does there exist an analytic function with digit complexity $O_{\text{dig}}(n)$?

There exist functions of the form

$$\sum a_n|x-b_n|$$

with low-digit complexity, where a_n and b_n are low-digit complexity numbers. Possibly we can construct nowhere differentiable functions that are sublinear, in the sense of digit complexity.

9. Questions on the complexity of numbers. Questions concerning the transcendence of functions tend to be easier than questions on the transcendence of individual numbers. In much the same way, questions on the complexity of functions tend to be easier than those on the complexity of specific numbers. The intent of this section is to pose various problems that suggest the link between complexity and transcendence. Such questions, while raised before, tend to have been concerned just with the notion of computability rather than also considering the rate of the computation (see [14]).

The class of sublinear numbers, defined in §7, contains all rational numbers; it also contains known transcendents such as

$$\alpha := .123456789101111213\cdots.$$

However, while the rationals are in this class in a base-independent fashion, it is not at all clear that the above number α is sublinear in bases relatively prime to 10. The 10^{10}th digit, base 10, is 1. What is it in base 2?

Question 11. Are there any irrational numbers that are sublinear in every base?

It is easy to generate numbers that are sublinear in particular bases. Numbers such as

$$a := .d_1 d_2 \cdots \qquad \begin{array}{ll} d_i := 1 & \text{if } i \text{ is a square,} \\ d_i := 0 & \text{otherwise,} \end{array}$$

or

$$b := .d_1 d_2 \cdots \qquad \begin{array}{ll} d_i := 1 & \text{if } i \text{ is a power of 2,} \\ d_i := 0 & \text{otherwise,} \end{array}$$

are sublinear in whatever base is specified. It is tempting to conjecture that the next question has a positive answer.

Question 12. Must an irrational number that is sublinear (in all bases) be transcendental?

Loxton and van der Poorten [17] show that a particular very special class of sublinear numbers, namely those generated by finite automata, are either rational or transcendental. These are numbers for which computation of the nth digit essentially requires no memory of the preceding digits. The base dependence of these numbers is discussed in [12].

Question 13. Is either of π or e sublinear (in any base)?

Almost certainly the answer to this question is no. There is an interesting observation relating to this. Consider the series

$$\frac{1}{\pi}=\sum_{n=0}^{\infty}\binom{2n}{n}^3\frac{42n+5}{2^{12n+4}}.$$

This series due to Ramanujan [5], [19] has numerators that grow roughly, e.g., 2^{6n}, while the denominators are powers of 2. Thus, as has been observed, we can compute the second length n block of binary digits of $1/\pi$ without computing the first block. Likewise, in base 10, we can compute the second block of length n of decimal digits of $\sqrt{(\frac{5}{3})}$ from the series

$$\frac{1}{\sqrt{1-4x}}=\sum_{n=0}^{\infty}\binom{2n}{n}x^n.$$

In neither case, however, is there any reduction in the order of complexity.

It seems likely that computing the nth digit of π is an $\Omega_{\text{dig}}(n)$ calculation. Thus, we might make the strong conjecture that no one will ever compute the 10^{1000}th digit of π. This number arises from an (over)estimate of the number of electrons in the known universe and as such almost certainly overestimates the amount of storage that will ever be available for such a calculation.

The set of sparse multipliers is a subset of the sublinear numbers that can be shown directly to contain no irrational algebraics. We do not know this about sparse numbers, though we strongly suspect it to be true.

Recall that a sparse multiplier has mostly zero digits and observe that a nonintegral rational cannot possess a terminating expansion in two relatively prime bases. This suggests the following question.

Question 14. Do there exist irrationals that are sparse multipliers in two relatively prime bases? Do there exist irrationals whose digits are asymptotically mostly zeros in two relatively prime bases?

Many of these questions are at least partly related to questions on normality [23]. Virtually nothing is known about the normality of familiar numbers. The following is a somewhat related question by Mahler.

Question 15 (Mahler [15]). Does there exist a nonrational function

$$f(x):=\sum_{n=0}^{\infty}a_nx^n$$

where the a_n are a bounded sequence of positive integers, that maps algebraic numbers in the unit disc to algebraic numbers?

Suppose that such an example exists, and suppose the a_n are bounded by 9. Then

$$f(1/1000)=a_0.00a_100a_2\cdots$$

is a thoroughly nonnormal irrational algebraic. Thus, in some sense, Mahler's question is a very weak conjecture concerning normality. Note also that, if in such an example the a_n were sublinearly computable, we would have produced sublinearly computable algebraic irrationalities.

Perhaps we will be able to distinguish rational numbers by their digit complexity. What can we hope to say about algebraic numbers? A natural class to look at is the class of numbers that are *linear* (*in multiplication*), that is, numbers with bit complexity $O_{\text{bit}}(M(n))$. This class contains all algebraic numbers in a base-independent fashion. It also contains numbers such as

$$a := \sum_{n=0}^{\infty} \frac{1}{3^{3^n}} \quad\text{and}\quad b := \prod_{n=0}^{\infty} (1+3^{-3^n}),$$

also in a base-independent fashion.

Question 16. Can we identify the class of numbers that are linear in multiplication?

This is almost certainly hard. As is the following question.

Question 17. Are either e or π linear in multiplication?

A negative answer to the above would include a proof of the transcendence of π. The place to start might be with the following.

Question 18. Can we construct any natural nonlinear number?

Our current state of knowledge is that γ and G have bit complexity $O_{\text{bit}}(\log^2 nM(n))$.

Question 19. Are γ and G both $O_{\text{bit}}(\log nM(n))$?

We might expect that elementary functions cannot take sublinear numbers to sublinear numbers.

Question 20. Does there exist a number $a \neq 0$ so that both a and $\exp(a)$ are sublinear (in some base)? Can a and $\exp(a)$ both be linear in multiplication?

It seems likely that the answer is no. Question 20 should also be asked about other elementary transcendental functions.

For simple nonelementary functions Question 20 has a positive answer. Consider the function $F := (2/\pi)K$, where K is the complete elliptic integral of the first kind. Then F satisfies a linear differential equation of order 2 and is a nonelementary transcendental function. However, if

$$k(q) := q^{1/2}\left(\sum_{n\in\mathbb{Z}} q^{n^2+n}\right)^2 \bigg/ \left(\sum_{n\in\mathbb{Z}} q^{n^2}\right)^2,$$

then

$$F(k(q)) = \left(\sum_{n\in\mathbb{Z}} q^{n^2}\right)^2,$$

and when $q := 1/10^{2k}$ both $F(k(q))$ and $k(q)$ are linear in multiplication, at least in base 10. (This is because the series above have particularly low complexity for $q := 1/10^{2k}$.)

Note also that the function

$$\theta_3(q) := \sum_{n\in\mathbb{Z}} q^{n^2},$$

which satisfies a nonlinear algebraic differential equation, takes sublinear numbers of the form $q := 1/10^n$ to sublinear numbers (base 10).

10. Conclusion. Many issues have not been touched upon at all. One such issue is the overhead costs of these low-complexity algorithms. This amounts to a discussion of the constants buried in the asymptotic estimates. Sometimes the theoretically low-complexity algorithms are also of low complexity practically. This is the case for AGM-related algorithms for complete elliptic integrals. These are probably the algorithms of choice in any precision. The AGM-related algorithms for log and exp will certainly not outperform more traditional methods in the usual ranges in which we compute (less than 100 digits). Some of the FFT-related algorithms are probably of only theoretical interest, even for computing millions of digits, because the overhead constants are so large. In other cases, such as multiplication or the computation of π, an FFT-related method is vital for very high precision computations.

We have not succeeded in completely answering any of the questions in the Introduction. In large part, this is because we have virtually no methods for handling lower bounds for such problems. The questions raised in this paper seem to be fundamental. The partial answers have provided a number of substantial surprises. For these reasons we believe these questions are deserving of study.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] D. H. Bailey, *The computation of π to 29,360,000 decimal digits using Borweins' quartically convergent algorithm*, Math. Comp., 50 (1988), pp. 283–296.

[3] M. Beeler, R. W. Gosper, and R. Schroeppel, *Hakmem*, MIT Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, 1972.

[4] J. M. Borwein and P. B. Borwein, *The arithmetic-geometric mean and fast computation of elementary functions*, SIAM Rev., 26 (1984), pp. 351–365.

[5] ———, *Pi and the* AGM—*A Study in Analytic Number Theory and Computational Complexity*, John Wiley, New York, 1987.

[6] P. B. Borwein, *Reduced complexity evaluation of hypergeometric functions*, J. Approx. Theory, 50 (1987), pp. 193–199.

[7] ———, *Digit complexity*, in preparation.

[8] D. Braess, *Nonlinear Approximation Theory*, Springer-Verlag, Berlin, 1986.

[9] R. P. Brent, *Fast multiple-precision evaluation of elementary functions*, J. Assoc. Comput. Mach., 23 (1976), pp. 242–251.

[10] R. P. Brent and E. M. McMillan, *Some new algorithms for high-precision calculation of Euler's constant*, Math. Comput., 34 (1980), pp. 305–312.

[11] E. W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[12] A. Cobham, *On the base-dependence of sets of numbers recognizable by finite automata*, Math. Systems Theory, 3 (1969), pp. 186–192.

[13] H. Friedman, *The computational complexity of maximization and integration*, Adv. in Math., 53 (1984), pp. 80–98.

[14] J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 265–306.

[15] K. Mahler, *Lectures on Transcendental Numbers*, Lecture Notes in Math. 546, Springer-Verlag, Berlin, New York, 1976.

[16] D. Knuth, *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1981.

[17] J. H. Loxton and A. J. van der Poorten, *Arithmetic properties of the solution of a class of functional equations*, J. Reine Angew. Math., 330 (1982), pp. 159–172.

[18] D. J. Newman, *Rational approximation versus fast computer methods*, in Lectures on Approximation and Value Distribution, Presses de l'Université de Montreal, Montreal, Canada, 1982, pp. 149–174.

[19] S. Ramanujan, *Modular equations and approximations to π*, Quart. J. Math., 45 (1914), 350–372.

[20] L. A. Rubel, *Some research problems about algebraic differential equations*, Trans. Amer. Math. Soc., 280 (1983), pp. 43–52.

[21] E. Salamin, *Computation of π using arithmetic-geometric mean*, Math. Comput. 30 (1976), pp. 565–570.

[22] R. Schroeppel, unpublished manuscript.

[23] S. Wagon, *Is π normal*? Math. Intelligencer, 7 (1985), pp. 65–67.

[24] E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis*, Fourth edition, Cambridge University Press, London, 1927.

[25] A. Wilansky, *Modern Methods in Topological Vector Spaces*, McGraw-Hill, New York, 1978.

[26] D. V. Chudnovsky and G. V. Chudnovsky, *Approximations and complex multiplication according to Ramanujan*, in Ramanujan Revisited, G. Andrews, R. Ashey, B. Berndt, K. Ramanathan, R. Rankin, eds., Academic Press, San Diego, CA, 1988.