ON THE RAPID COMPUTATION OF VARIOUS POLYLOGARITHMIC CONSTANTS

David Bailey, Peter Borwein[1] and Simon Plouffe

Abstract.

We give algorithms for the computation of the d-th digit of certain transcendental numbers in various bases. These algorithms can be easily implemented (multiple precision arithmetic is not needed), require virtually no memory, and feature run times that scale nearly linearly with the order of the digit desired. They make it feasible to compute, for example, the billionth binary digit of $\log(2)$ or π on a modest work station in a few hours run time.

We demonstrate this technique by computing the ten billionth hexadecimal digit of π, the billionth hexadecimal digits of π^2, $\log(2)$ and $\log^2(2)$, and the ten billionth decimal digit of $\log(9/10)$.

These calculations rest on the observation that very special types of identities exist for certain numbers like π, π^2, $\log(2)$ and $\log^2(2)$. These are essentially polylogarithmic ladders in an integer base. A number of these identities that we derive in this work appear to be new, for example the critical identity for π:

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right).$$

[1] Research supported in part by NSERC of Canada.

1991 *Mathematics Subject Classification.* 11A05 11Y16 68Q25.

Key words and phrases. Computation, digits, log, polylogarithms, SC, π, algorithm.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

1. Introduction.

It is widely believed that computing just the d-th digit of a number like π is really no easier than computing all of the first d digits. From a bit complexity point of view this may well be true, although it is probably very hard to prove. What we will show is that it is possible to compute just the d-th digit of many transcendentals in (essentially) linear time and logarithmic space. So while this is not of fundamentally lower complexity than the best known algorithms (for say π or $\log(2)$), this makes such calculations feasible on modest workstations without needing to implement arbitrary precision arithmetic.

We illustrate this by computing the ten billionth hexadecimal digit of π, the billionth hexadecimal digits of π^2, $\log(2)$ and $\log^2(2)$, and the ten billionth decimal digit of $\log(9/10)$. Details are given in Section 4. A previous result in this same spirit is the Rabinowitz-Wagon "spigot" algorithm for π. In that scheme, however, the computation of the digit at position n depends on all digits preceding position n.

We are interested in computing in polynomially logarithmic space and polynomial time. This class is usually denoted SC (space $= \log^{O(1)}(d)$ and time $= d^{O(1)}$ where d is the place of the "digit" to be computed). Actually we are most interested in the space we will denote by SC* of polynomially logarithmic space and (almost) linear time (here we want the time $= O(d \log^{O(1)}(d))$). There is always a possible ambiguity when computing a digit string base b in distinguishing a sequence of digits $a(b-1)(b-1)(b-1)$ from $(a+1)000$. In this particular case we consider either representation as an acceptable computation. In practice this problem does not arise.

It is not known whether division is possible in SC, similarly it is not known whether base change is possible in SC. The situation is even worse in SC*, where it is not even known whether multiplication is possible. If two numbers are in SC* (in the same base) then their product computes in time $= O(d^2 \log^{O(1)}(d))$ and is in SC but not obviously in SC*. The d^2 factor here is present because the logarithmic space requirement precludes the usage of advanced multiplication techniques, such as those based on FFTs.

We will not dwell on complexity issues except to point out that different algorithms are needed for different bases (at least given our current ignorance about base change) and very little closure exists on the class of numbers with d-th digit computable in SC. Various of the complexity related issues are discussed in [6,8,9,11,14].

As we will show in Section 3, the class of numbers we can compute in SC* in base b includes all numbers of the form

$$\sum_{k=1}^{\infty} \frac{p(k)}{b^{ck} q(k)}, \tag{1.1}$$

where p and q are polynomials with integer coefficients and c is a positive integer. Since addition is possible in SC*, integer linear combinations of such numbers are also feasible (provided the base is fixed).

The algorithm for the binary digits of π, which also shows that π is in SC* in base 2, rests on the following remarkable identity:

Theorem 1. *The following identity holds:*

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right). \tag{1.2}$$

This can also be written as:

$$\pi = \sum_{i=1}^{\infty} \frac{p_i}{16^{\lfloor \frac{i}{8} \rfloor} i}, \qquad [p_i] = [\overline{4,0,0,-2,-1,-1,0,0}] \tag{1.3}$$

where the overbar notation indicates that the sequence is periodic.

Proof. This identity is equivalent to:

$$\pi = \int_0^{1/\sqrt{2}} \frac{4\sqrt{2} - 8x^3 - 4\sqrt{2}x^4 - 8x^5}{1-x^8} dx, \tag{1.4}$$

which on substituting $y := \sqrt{2}x$ becomes

$$\pi = \int_0^1 \frac{16\,y - 16}{y^4 - 2\,y^3 + 4\,y - 4} dy\,.$$

The equivalence of (1.2) and (1.4) is straightforward. It follows from the identity

$$\begin{aligned} \int_0^{1/\sqrt{2}} \frac{x^{k-1}}{1-x^8}\, dx &= \int_0^{1/\sqrt{2}} \sum_{i=0}^{\infty} x^{k-1+8i}\, dx \\ &= \frac{1}{\sqrt{2}^k} \sum_{i=0}^{\infty} \frac{1}{16^i(8i+k)} \end{aligned}$$

That the integral (1.4) evaluates to π is an exercise in partial fractions most easily done in Maple or Mathematica. $\square$

This proof entirely conceals the route to discovery. We found the identity (1.2) by a combination of inspired guessing and extensive searching using the PSLQ integer relation algorithm [3,12].

Shortly after the authors originally announced the result (1.2), several colleagues, including Helaman Ferguson, Tom Hales, Victor Adamchik, Stan Wagon, Donald Knuth and Robert Harley, pointed out to us other formulas for π of this type. One intriguing example is

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{2}{8i+1} + \frac{2}{4i+2} + \frac{1}{4i+3} - \frac{1/2}{4i+5} - \frac{1/2}{4i+6} - \frac{1/4}{4i+7}\right),$$

which can be written more compactly as

$$\pi = \sum_{i=0}^{\infty} \frac{(-1)^i}{4^i} \left(\frac{2}{4i+1} + \frac{2}{4i+2} + \frac{1}{4i+3} \right).$$

In [2], this and some related identities are derived using Mathematica.

As it turns out, these other formulas for π can all be written as formula (1.2) plus a rational multiple of the identity

$$0 = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{-8}{8i+1} + \frac{8}{8i+2} + \frac{4}{8i+3} + \frac{8}{8i+4} + \frac{2}{8i+5} + \frac{2}{8i+6} - \frac{1}{8i+7} \right).$$

The proof of this identity is similar to that of Theorem 1.

The identities of the next section and Section 5 show that, in base 2, π^2, $\log^2(2)$ and various other constants, including $\{\log(2), \log(3), \ldots, \log(22)\}$ are in SC*. (We don't know however if $\log(23)$ is even in SC.)

We will describe the algorithm in the Section 3. Complexity issues are discussed in [3,5,6,7,8,9,14,19,21] and algorithmic issues in [5,6,7,8,14]. The requisite special function theory may be found in [1,5,15,16,17,20].

2. Identities.

As usual, we define the m-th polylogarithm L_m by

$$L_m(z) := \sum_{i=1}^{\infty} \frac{z^i}{i^m}, \qquad |z| < 1\,. \tag{2.1}$$

The most basic identity is

$$-\log(1 - 2^{-n}) = L_1(1/2^n) \tag{2.2}$$

which shows that $\log(1 - 2^{-n})$ is in SC* base 2 for integer n. (See also section 5.)

Much less obvious are the identities

$$\pi^2 = 36L_2(1/2) - 36L_2(1/4) - 12L_2(1/8) + 6L_2(1/64) \tag{2.3}$$

and

$$\log^2(2) = 4L_2(1/2) - 6L_2(1/4) - 2L_2(1/8) + L_2(1/64)\,. \tag{2.4}$$

These can be written as

$$\pi^2 = 36 \sum_{i=1}^{\infty} \frac{a_i}{2^i i^2}, \qquad [a_i] = [\overline{1, -3, -2, -3, 1, 0}] \tag{2.5}$$

$$\text{(2.6).} \qquad \log^2(2) = 2\sum_{i=1}^{\infty} \frac{b_i}{2^i i^2}, \qquad [b_i] = [\overline{2,-10,-7,-10,2,-1}].$$

Here the overline notation indicates that the sequences repeat. Thus we see that π^2 and $\log^2(2)$ are in SC* in base 2. These two formulas can alternately be written

$$\pi^2 = \frac{9}{8}\sum_{i=0}^{\infty} \frac{1}{64^i}\Big(\frac{16}{(6i+1)^2} - \frac{24}{(6i+2)^2} - \frac{8}{(6i+3)^2} - \frac{6}{(6i+4)^2} + \frac{1}{(6i+5)^2}\Big)$$

$$\log^2(2) = \frac{1}{8}\sum_{i=0}^{\infty} \frac{1}{64^i}\Big(\frac{-16}{(6i)^2} + \frac{16}{(6i+1)^2} - \frac{40}{(6i+2)^2} - \frac{14}{(6i+3)^2} - \frac{10}{(6i+4)^2} + \frac{1}{(6i+5)^2}\Big).$$

Identities (2.3)-(2-6) are examples of polylogarithmic ladders in the base 1/2 in the sense of [16]. As with (1.2) we found them by searching for identities of this type using an integer relation algorithm. We have not found them directly in print. However (2.5) follows from equation (4.70) of [15] with $\alpha = \pi/3, \beta = \pi/2$ and $\gamma = \pi/3$. Identity (2.6) now follows from the well known identity

$$\text{(2.7)} \qquad 12L_2(1/2) = \pi^2 - 6\log^2(2).$$

A distinct but similar formula that we have found for π^2 is

$$\pi^2 = \sum_{i=0}^{\infty} \frac{1}{16^i}\Big(\frac{16}{(8i+1)^2} - \frac{16}{(8i+2)^2} - \frac{8}{(8i+3)^2} - \frac{16}{(8i+4)^2} - \frac{4}{(8i+5)^2} - \frac{4}{(8i+6)^2} + \frac{2}{(8i+7)^2}\Big),$$

which can be derived from the methods of section 1.

There are several ladder identities involving L_3:

$$\text{(2.8)} \qquad 35/2\zeta(3) - \pi^2\log(2) = 36L_3(1/2) - 18L_3(1/4) - 4L_3(1/8) + L_3(1/64),$$

$$\text{(2.9)} \qquad 2\log^3(2) - 7\zeta(3) = -24L_3(1/2) + 18L_3(1/4) + 4L_3(1/8) - L_3(1/64),$$

$$\text{(2.10)} \quad 10\log^3(2) - 2\pi^2\log(2) = -48L_3(1/2) + 54L_3(1/4) + 12L_3(1/8) - 3L_3(1/64).$$

The favored algorithms for π of the last centuries involved some variant of Machin's 1706 formula:

$$\text{(2.11)} \qquad \frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}.$$

There are many related formula [15,16,17,20] but to be useful to us all the arguments of the arctans have to be a power of a common base, and we have not discovered any such formula for π . One can however write

$$\text{(2.12)} \qquad \frac{\pi}{2} = 2\arctan\frac{1}{\sqrt{2}} + \arctan\frac{1}{\sqrt{8}}$$

This can be written as

$$\sqrt{2}\pi = 4f(1/2) + f(1/8) \quad \text{where} \quad f(x) := \sum_{i=1}^{\infty} \frac{(-1)^i x^i}{2i+1} \tag{2.13}$$

and allows for the calculation of $\sqrt{2}\pi$ in SC*.

Another two identities involving Catalan's constant G, π and log(2) are:

$$G - \frac{\pi \log(2)}{8} = \sum_{i=1}^{\infty} \frac{c_i}{2^{\lfloor \frac{i+1}{2} \rfloor} i^2}, \qquad [c_i] = [\overline{1, 1, 1, 0, -1, -1, -1, 0}] \tag{2.14}$$

and

$$\frac{5}{96}\pi^2 - \frac{\log^2(2)}{8} = \sum_{i=1}^{\infty} \frac{d_i}{2^{\lfloor \frac{i+1}{2} \rfloor} i^2}, \qquad [d_i] = [\overline{1, 0, -1, -1, -1, 0, 1, 1}] \tag{2.15}$$

These may be found in [17 p. 105, p. 151]. Thus $8G - \pi \log(2)$ is also in SC* in base 2, but it is open and interesting as to whether G is itself in SC* in base 2.

A family of base 2 ladder identities exist:

$$\frac{L_m(1/64)}{6^{m-1}} - \frac{L_m(1/8)}{3^{m-1}} - \frac{2\ L_m(1/4)}{2^{m-1}} + \frac{4\ L_m(1/2)}{9} - \frac{5\ (-\log(2))^m}{9\, m!} \tag{2.16}$$

$$+ \frac{\pi^2\, (-\log(2))^{m-2}}{54\ (m-2)!} - \frac{\pi^4\, (-\log(2))^{m-4}}{486\ (m-4)!} - \frac{403\, \zeta(5)\, (-\log(2))^{m-5}}{1296\ (m-5)!} = 0.$$

The above identity holds for $1 \leq m \leq 5$; when the arguments to factorials are negative they are taken to be infinite so the corresponding terms disappear. See [16, p. 45].

As in the case of formula (1.2) for π, colleagues of the authors have subsequently pointed out several other formulas of this type for various constants. Three examples reported by Knuth, which are based on formulas in [13, p. 17, 18, 22, 47, 139], are

$$\sqrt{2}\ln(1+\sqrt{2}) = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{1}{8i+1} + \frac{1/2}{8i+3} + \frac{1/4}{8i+5} + \frac{1/8}{8i+7}\right)$$

$$\sqrt{2}\arctan(1/\sqrt{2}) = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{1}{8i+1} - \frac{1/2}{8i+3} + \frac{1/4}{8i+5} - \frac{1/8}{8i+7}\right)$$

$$\arctan(1/3) = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{1}{8i+1} - \frac{1}{8i+2} - \frac{1/2}{8i+4} - \frac{1/4}{8i+5}\right)$$

Thus these constants are also in class SC*. Some other examples can be found in [18].

3. The Algorithm.

Our algorithm to compute individual base-b digits of certain constants is based on the binary scheme for exponentiation, wherein one evaluates x^n rapidly by successive squaring and multiplication. This reduces the number of multiplications to less than $2\log_2(n)$. According to Knuth [14], where details are given, this trick goes back at least to 200 B.C. In our application, we need to perform exponentiation modulo a positive integer c, but the overall scheme is the same — one merely performs all operations modulo c. An efficient formulation of this algorithm is as follows.

To compute $r = b^n \bmod c$, first set t to be the largest power of two $\leq n$, and set $r = 1$. Then

A: if $n \geq t$ then $r \leftarrow br \bmod c$; $n \leftarrow n - t$; endif

$t \leftarrow t/2$

if $t \geq 1$ then $r \leftarrow r^2 \bmod c$; go to A; endif

Here and in what follows, "mod" is used in the binary operator sense, namely as the binary function defined by $x \bmod y := x - [x/y]y$. Note that the above algorithm is entirely performed with positive integers that do not exceed c^2 in size. Thus it can be correctly performed, without round-off error, provided a numeric precision of at least $1 + 2\log_2 c$ bits is used.

Consider now a constant defined by a series of the form

$$S = \sum_{k=0}^{\infty} \frac{1}{b^{ck} p(k)},$$

where b and c are positive integers and $p(k)$ is a polynomial with integer coefficients. First observe that the digits in the base b expansion of S beginning at position $n+1$ can be obtained from the fractional part of $b^n S$. Thus we can write

$$b^n S \mod 1 = \sum_{k=0}^{\infty} \frac{b^{n-ck}}{p(k)} \mod 1 \tag{3.4}$$

$$= \sum_{k=0}^{\lfloor n/c \rfloor} \frac{b^{n-ck} \bmod p(k)}{p(k)} \mod 1 + \sum_{k=\lfloor n/c \rfloor + 1}^{\infty} \frac{b^{n-ck}}{p(k)} \mod 1$$

For each term of the first summation, the binary exponentiation scheme is used to evaluate the numerator. Then floating-point arithmetic is used to perform the division and add the result to the sum mod 1. The second summation, where the exponent of b is negative, may be evaluated as written using floating-point arithmetic. It is only necessary to compute a few terms of this second summation, just enough to insure that the remaining terms sum to less than the "epsilon" of the floating-point arithmetic being used. The final result, a fraction between 0 and 1, is then converted to the desired base b.

Since floating-point arithmetic is used here in divisions and in addition modulo 1, the result is of course subject to round-off error. If the floating-point arithmetic system being used has the property that the result of each individual floating-point operation is in error by at most one bit (as in systems implementing the IEEE arithmetic standard), then no more than $\log_2(2n)$ bits of the final result will be corrupted. This is actually a generous estimate, since it does not assume any cancelation of errors, which would yield a lower estimate. In any event, it is clear that ordinary IEEE 64-bit arithmetic is sufficient to obtain a numerically significant result for even a large computation, and "quad precision" (i.e. 128-bit) arithmetic, if available, can insure that the final result is accurate to several digits beyond the one desired. One can check the significance of a computed result beginning at position n by also performing a computation at position $n+1$ or $n-1$ and comparing the trailing digits produced.

The most basic interesting constant whose digits can be computed using this scheme is

$$\log(2) = \sum_{k=1}^{\infty} \frac{1}{k2^k}$$

in base 2. Using this scheme to compute hexademical digits of π from identity (1.2) is only marginally more complicated, since one can rewrite formula (1.2) using four sums of the required form. Details are given in the next section. In both cases, in order to compute the n-th binary digit (or a fixed number of binary digits at the n-th place) we must sum $O(n)$ terms of the series. Each term requires $O(\log(n))$ arithmetic operations and the required precision is $O(\log(n))$ digits. This gives a total bit complexity of $O(n\log(n)M(\log(n)))$ where $M(j)$ is the complexity of multiplying j bit integers. So even with ordinary multiplication the bit complexity is $O(n\log^3(n))$.

This algorithm is, by a factor of $\log(\log(\log(n)))$, asymptotically slower than the fastest known algorithms for generating the n-th digit by generating all of the first n digits of $\log(2)$ or π [7]. The asymptotically fastest algorithms for all the first n digits known requires a Strassen-Schönhage multiplication [19]; the algorithms actually employed use an FFT based multiplication and are marginally slower than our algorithm, from a complexity point of view, for computing just the n-th digit. Of course this complexity analysis is totally misleading: the strength of our algorithm rests mostly on its easy implementation in standard precision without requiring FFT methods to accelerate the computation.

It is clear that the above methods can easily be extended to evaluate digits of contstants defined by a formula of the form

$$S = \sum_{k=0}^{\infty} \frac{p(k)}{b^{ck}q(k)},$$

where p and q are polynomials with integer coefficients and c is a positive integer. Similarly if p and q are slowly growing analytic functions of various types the method extends.

4. Computations.

We report here computations of π, $\log(2)$, $\log^2(2)$, π^2 and $\log(9/10)$, based on the formulas (1.1), (2.2), (2.5), (2.6) and the identity $\log(9/10) = -L_1(1/10)$, respectively.

Each of our computations employed quad precision floating-point arithmetic for division and sum mod 1 operations. Quad precision is supported from Fortran on the IBM RS6000/590 and the SGI Power Challenge (R8000), which were employed by the authors in these computations. We were able to avoid the usage of explicit quad precision in the exponentiation scheme by exploiting a hardware feature common to these two systems, namely the 106-bit internal registers in the multiply-add operation. This saved considerable time, because quad precision operations are significantly more expensive than 64-bit operations.

Computation of π^2 and $\log^2(2)$ presented a special challenge, because one must perform the exponentiation algorithm modulo k^2 instead of k. When n is larger than only 2^{13}, some terms of the series (2.5) and (2.6) must be computed with a modulus k^2 that is greater than 2^{26}. Squares that appear in the exponentiation algorithm will then exceed 2^{52}, which is the nearly the maximum precision of IEEE 64-bit floating-point numbers. When n is larger than 2^{26}, then squares in the exponentiation algorithm will exceed 2^{104}, which is nearly the limit of quad precision.

This difficulty can be remedied using a method which has been employed for example in searches for Wieferich primes [10]. Represent the running value r in the exponentiation algorithm by the ordered pair (r_1, r_2), where $r = r_1 + kr_2$, and where r_1 and r_2 are positive integers less than k. Then one can write

$$r^2 = (r_1 + kr_2)^2 = r_1^2 + 2r_1r_2k + r_2^2k^2$$

When this is reduced mod k^2, the last term disappears. The remaining expression is of the required ordered pair form, provided that r_1^2 is first reduced mod k, the carry from this reduction is added to $2r_1r_2$, and this sum is also reduced mod k. Note that this scheme can be implemented with integers of size not exceeding $2k^2$. Since the computation of r^2 mod k^2 is the key operation of the binary exponentiation algorithm, this means that ordinary IEEE 64-bit floating-point arithmetic can be used to compute the n-th hexadecimal digit of π^2 or $\log^2(2)$ for n up to about 2^{24}. For larger n, we still used this basic scheme, but we employed the multiply-add "trick" mentioned above to avoid the need for explicit quad precision in this section of code.

Our results are given below. The first entry, for example, gives the 10^6-th through 10^6+13-th hexadecimal digits of π after the "decimal" point. In all cases we did the calculations twice — the second calculation was similar to the first, except shifted back one position. Since this changes all the arithmetic performed, it is a highly rigorous validity check. Thus we believe that all the digits shown below are correct.

Constant:	Base:	Position:	Digits from Position:
π	16	10^6	26C65E52CB4593
		10^7	17AF5863EFED8D
		10^8	ECB840E21926EC
		10^9	85895585A0428B
		10^{10}	921C73C6838FB2
$\log(2)$	16	10^6	418489A9406EC9
		10^7	815F479E2B9102
		10^8	E648F40940E13E
		10^9	B1EEF1252297EC
π^2	16	10^6	685554E1228505
		10^7	9862837AD8AABF
		10^8	4861AAF8F861BE
		10^9	437A2BA4A13591
$\log^2(2)$	16	10^6	2EC7EDB82B2DF7
		10^7	33374B47882B32
		10^8	3F55150F1AB3DC
		10^9	8BA7C885CEFCE8
$\log(9/10)$	10	10^6	80174212190900
		10^7	21093001236414
		10^8	01309302330968
		10^9	44066397959215
		10^{10}	82528693381274

These computations were done at NASA Ames Research Center, using workstation cycles that otherwise would have been idle.

5. Logs in base 2.

It is easy to compute, in base 2, the d-th binary digit of

$$\log(1-2^{-n}) = L_1(1/2^n)\,. \tag{5.1}$$

So it is easy to compute $\log(m)$ for any integer m that can be written as

$$m := \frac{(2^{a_1}-1)(2^{a_2}-1)\cdots(2^{a_h}-1)}{(2^{b_1}-1)(2^{b_2}-1)\cdots(2^{b_j}-1)}. \tag{5.2}$$

In particular the n-th cyclotomic polynomial evaluated at 2 is so computable. A check shows that all primes less than 19 are of this form. The beginning of this list is:

$$\{2,3,5,7,11,13,17,31,43,57,73,127,151,205,257\}\,.$$

Since
$$2^{18} - 1 = 7 \cdot 9 \cdot 19 \cdot 73,$$
and since 7, $\sqrt{9}$ and 73 are all on the above list we can compute $\log(19)$ in SC* from
$$\log(19) = \log(2^{18} - 1) - \log(7) - \log(9) - \log(73).$$
Note that $2^{11} - 1 = 23 \cdot 89$ so either both $\log(23)$ and $\log(89)$ are in SC* or neither is.

We would like to thank Carl Pomerance for showing that an identity of type (5.2) does not exist for 23. This is a consequence of the fact that each cyclotomic polynomial evaluated at two has a new distinct prime factor. We would also like to thank Robert Harley for pointing out that 29 and 37 are in SC^* in base 2 via consideration of the Aurefeuillian factors $2^{2n-1} + 2^n + 1$ and $2^{2n-1} - 2^n + 1$.

6. Relation Bounds.

One of the first questions that arises in the wake of the above study is whether there exists a scheme of this type to compute decimal digits of π. At present we know of no identity like (1.2) in base 10. The chances that there is such an identity are dimmed by some numerical results that we have obtained using the PSLQ integer relation algorithm [3, 12]. These computations establish (with the usual provisos of computer "proofs") that there are no identities (except for the case $n = 16$) of the form
$$\pi = \frac{a_1}{a_0} + \frac{1}{a_0} \sum_{k=0}^{\infty} \frac{1}{n^k} \left[\frac{a_2}{mk+1} + \frac{a_3}{mk+2} + \cdots + \frac{a_{m+1}}{mk+m} \right],$$
where n ranges from 2 to 128, where m ranges from 1 to $\min(n, 32)$, and where the Euclidean norm of the integer vector $(a_0, a_1, \cdots, a_{m+1})$ is 10^{12} or less. These results of course do not have any bearing on the possibility that there is a formula not of this form which permits computation of π in some non-binary base.

In fact, J. P. Buhler has reported a proof that any identity for π of the above form must have $n = 2^K$ or $n = \sqrt{2}^K$. This also does not exclude more complicated formulae for the computation of π base 10.

7. Questions.

As mentioned in the previous section, we cannot at present compute decimal digits of π by our methods because we know of no identity like (1.2) in base 10. But it seems unlikely that it is fundamentally impossible to do so. This raises the following obvious problem:

1] Find an algorithm for the n-th decimal digit of π in SC*. It is not even clear that π is in SC in base 10 but it ought to be possible to show this.

2] Show that π is in SC in all bases.

3] Are e and $\sqrt{2}$ in SC (SC*) in any base?

Similarly the treatment of log is incomplete:

4] Is log(2) in SC* in base 10?

5] Is log(23) in SC* in base 2?

8. Acknowledgments.

The authors wish to acknowledge the following for their helpful comments: V. Adamchik, J. Borwein, J. Buhler, R. Crandall, H. Ferguson, T. Hales, R. Harley, D. Knuth, C. Pomerance and S. Wagon.

References

1. M. Abramowitz & I.A. Stegun, *Handbook of Mathematical Functions*, Dover, New York, NY, 1965.

2. V. Adamchik & S. Wagon, *Pi: A 2000-year search changes direction (preprint).*

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

4. D. H. Bailey, J. Borwein and R. Girgensohn, *Experimental evaluation of Euler sums*, Experimental Mathematics **3** (1994), 17–30.

5. J. Borwein, & P Borwein, *Pi and the AGM – A Study in Analytic Number Theory and Computational Complexity*, Wiley, New York, NY, 1987.

6. J. Borwein & P. Borwein, *On the complexity of familiar functions and numbers*, SIAM Review **30** (1988), 589–601.

7. J. Borwein, P. Borwein & D. H. Bailey, *Ramanujan, modular equations and approximations to pi*, Amer. Math. Monthly **96** (1989), 201–219.

8. R. Brent, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach. **21** (1974), 201–206.

9. S. Cook, *A taxonomy of problems with fast parallel algorithms*, Information and Control **64** (1985), 2–22.

10. R. Crandall, K. Dilcher, and C. Pomerance, *A search for Wieferich and Wilson primes (preprint).*

11. R. Crandall and J. Buhler, *On the evaluation of Euler sums*, Experimental Mathematics **3,** (1995), 275–285.

12. H. R. P. Ferguson & D. H. Bailey, *Analysis of PSLQ, an integer relation algorithm (preprint).*

13. E. R. Hansen, *A Table of Series and Products*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

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

15. L. Lewin, *Polylogarithms and Associated Functions*, North Holland, New York, 1981.

16. L. Lewin, *Structural Properties of Polylogarithms*, Amer. Math. Soc., RI., 1991.

17. N. Nielsen, *Der Eulersche Dilogarithmus*, Halle, Leipzig, 1909.

18. S. D. Rabinowitz and S. Wagon, *A spigot algorithm for pi*, Amer. Math. Monthly **103** (1995), 195-203.

19. A. Schönhage, *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients*, in: EUROCAM (1982) Marseille, Springer Lecture Notes in Computer Science, vol. 144, 1982, pp. 3–15.

20. J. Todd, *A problem on arc tangent relations*, MAA Monthly **56** (1940), 517–528.

21. H. S. Wilf, *Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1986.

Bailey: NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA, USA 94035-1000 dbailey@nas.nasa.gov

Borwein: Department of Mathematics and Statistics, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 pborwein@cecm.sfu.ca

Plouffe: Department of Mathematics and Statistics, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 plouffe@cecm.sfu.ca