Finding New Mathematical Identities via Numerical Computations

David H. Bailey
NASA Ames Research Center
December 11, 1997
Ref: *ACM SIGNUM*, vol. 33, no. 1 (Jan. 1998), pg. 17-22

Abstract

This note gives an overview some recent developments in computational mathematics, where high-precision numerical computations, together with advanced integer relation algorithms, have been employed to discover heretofore unknown mathematical identities. One of these new identities, a remarkable new formula for π, permits one to directly compute the n-th hexadecimal digit of π, without computing the first $n-1$ digits, and without the need of multiple-precision arithmetic software.

1. Introduction

In April 1993, Enrico Au-Yeung, an undergraduate at the University of Waterloo, brought to the attention of Jonathan Borwein, his professor, the curious fact that

$$\begin{aligned}\sum_{k=1}^{\infty} \left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right)^2 k^{-2} &= 4.59987\cdots \\ &\approx \frac{17}{4}\zeta(4) = \frac{17\pi^4}{360},\end{aligned}$$

based on a computation to 500,000 terms. Borwein was very skeptical of this finding — if such an identity truly existed, why hadn't it been discovered by mathematicians centuries ago? Borwein tried computing this sum to a higher level of precision in order to demonstrate to the student that this conjecture really did not precisely hold. Surprisingly, in subsequent computations by Borwein to 30 digits and by myself to over 100 decimal digits, this relation was upheld.

Intrigued by this finding, Borwein and other researchers subsequently discovered numerous other identities of this form, which have been termed "Euler sums", since Euler first studied some special cases. Many of these specific results (including the above identity), but not all, have subsequently been proven rigorously. Indeed, the methodology used in these investigations is often termed "experimental mathematics", wherein one utilizes computer programs to explore mathematical phenomena, thereby finding tentative results which afterwards are verified with rigorous proofs.

Some examples of these Euler sum identities are given in Table 1. Others are given in [2]. In the table, $\zeta(t) = \sum_{j=1}^{\infty} j^{-t}$ is the Riemann zeta function, and $\mathrm{Li}_n(x) = \sum_{j=1}^{\infty} x^j j^{-n}$ denotes the polylogarithm function.

$$\sum_{k=1}^{\infty}\left(1+\frac{1}{2}+\cdots+\frac{1}{k}\right)^2(k+1)^{-4} = \frac{37\pi^6}{22680}-\zeta^2(3)$$

$$\sum_{k=1}^{\infty}\left(1+\frac{1}{2}+\cdots+\frac{1}{k}\right)^3(k+1)^{-6} = -\frac{11\pi^4\zeta(5)}{120}+\frac{37\pi^6\zeta(3)}{7560}$$

$$\sum_{k=1}^{\infty}\left(1-\frac{1}{2}+\cdots+(-1)^{k+1}\frac{1}{k}\right)^2(k+1)^{-3} = 4\,\mathrm{Li}_5(1/2)-\frac{1}{30}\ln^5(2)-\frac{17}{32}\zeta(5)$$
$$-\frac{11}{720}\pi^4\ln(2)+\frac{7}{4}\zeta(3)\ln^2(2)+\frac{1}{18}\pi^2\ln^3(2)-\frac{3}{24}\pi^2\zeta(3)$$

Table 1: Some Euler sum identities found by computer

2. Integer Relation Algorithms

These computer searches require the values of these sums and constants to be computed to very high precision — from 100 to 1,000 decimal digits. Computing these values to such high precision requires advanced numerical techniques, and is often quite expensive. See [2] for details. Once these values are computed, an "integer relation" algorithm is utilized to see if the computed constants match an identity of a given form. An integer relation algorithm is an algorithm that, when given an n-long vector of real numbers x_i, attempts to find integer coefficients a_i, not all zero, such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$.

One such algorithm is the "PSLQ" algorithm, which was discovered by Helaman Ferguson of the Center for Computing Sciences in Maryland (better known for his mathematical sculptures). Here is a concise statement of the PSLQ algorithm. For proof and further details, see [5].

Let x be the n-long input real vector, and let nint denote the nearest integer function (for exact half-integer values, define nint to be the integer with greater absolute value). Let $\gamma := \sqrt{4/3}$. Then perform the following:

Initialize:

1. Set the $n \times n$ matrices A and B to the identity.

2. For $k := 1$ to n: compute $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor. Set $t = 1/s_1$. For $k := 1$ to n: $y_k := tx_k$; $s_k := ts_k$; endfor.

3. Compute the $n \times (n-1)$ matrix H as follows:

   For $i := 1$ to n: for $j := i+1$ to $n-1$: set $H_{ij} := 0$; endfor; if $i \le n-1$ then set $H_{ii} := s_{i+1}/s_i$; for $j := 1$ to $i-1$: set $H_{ij} := -y_iy_j/(s_js_{j+1})$; endfor; endfor.

4. Perform full reduction on H, simultaneously updating y, A and B:

   For $i := 2$ to n: for $j := i-1$ to 1 step -1: $t := \mathrm{nint}(H_{ij}/H_{jj})$; $y_j := y_j + t y_i$; for $k := 1$ to j: $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to n: $A_{ik} := A_{ik} - t A_{jk}$, $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

Repeat until precision is exhausted or a relation has been detected:

1. Select m such that $\gamma^i |H_{ii}|$ is maximal when $i = m$.

2. Exchange entries m and $m+1$ of y, corresponding rows of A and H, and corresponding columns of B.

3. If $m \leq n-2$ then update H as follows:

   Set $t_0 := \sqrt{H_{mm}^2 + H_{m,m+1}^2}$, $t_1 := H_{mm}/t_0$ and $t_2 := H_{m,m+1}/t_0$. Then for $i := m$ to n: $t_3 := H_{im}$; $t_4 := H_{i,m+1}$; $H_{im} := t_1 t_3 + t_2 t_4$; $H_{i,m+1} := -t_2 t_3 + t_1 t_4$; endfor.

4. Perform block reduction on H, simultaneously updating y, A and B:

   For $i := m+1$ to n: for $j := \min(i-1, m+1)$ to 1 step -1: $t := \mathrm{nint}(H_{ij}/H_{jj})$; $y_j := y_j + t y_i$; for $k := 1$ to j: $H_{ik} := H_{ik} - t H_{jk}$; ; endfor; for $k := 1$ to n: $A_{ik} := A_{ik} - t A_{jk}$, $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

5. Norm bound: Compute $M := 1/\max_j |H_j|$, where H_j denotes the j-th row of H. Then there can exist no relation vector whose Euclidean norm is less than M.

6. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection threshold, a relation has been detected and is given in the corresponding column of B.

With regards to the termination criteria in step 6, it sometimes happens that a relation is missed at the point of potential detection because the y entry is not quite as small as the detection threshold being used (the threshold is typically set to the "epsilon" of the precision level). When this happens, however, one will note that the ratio of the smallest and largest y vector entries is suddenly very small, provided sufficient numeric precision is being used. In a normal computer run using the PSLQ algorithm, prior to the detection of a relation, this ratio is seldom smaller than 10^{-2}. Thus if this ratio suddenly decreases to an exceedingly small value, such as 10^{-20}, then almost certainly a relation has been detected — one need only adjust the detection threshold for the algorithm to terminate properly and output the relation. When detection does occur, this ratio may be thought of as a "confidence level" of the detection.

Experience with PSLQ indicates that one can expect to detect a relation of degree n, with coefficients of size 10^m, provided that the input vector is known to somewhat greater

than mn digit precision, and provided that computations are performed using at least this level of numeric precision.

3. Applications of the Integer Relation Algorithms

There are a number of applications of integer relation detection algorithms in computational mathematics. One application is to analyze whether or not a given constant α, whose value can be computed to high precision, is algebraic of some degree n or less. This can be done by first computing the vector $x = (1, \alpha, \alpha^2, \cdots, \alpha^n)$ to high precision and then applying an integer relation algorithm to the vector x. If a relation is found, this integer vector is precisely the set of coefficients of a polynomial satisfied by α. Even if a relation is not found, the resulting bound means that α cannot possibly be the root of a polynomial of degree n, with coefficients of size less than the established bound. Even negative results of this sort are often of interest.

Several computations of this type have been performed by the author. These computations have established, for example, that if Euler's constant γ satisfies an integer polynomial of degree 50 or less, then the Euclidean norm of the coefficients must exceed 7×10^{17}.

The application of finding identities for Euler sum constants is well suited to analysis with integer relation algorithms. Consider for example

$$\begin{aligned} A_{2,3} &= \sum_{k=1}^{\infty} \left(1 - \frac{1}{2} + \cdots + (-1)^{k+1}\frac{1}{k}\right)^2 (k+1)^{-3} \\ &= 0.15616693338117691588103590968798819368577670984 0\cdots \end{aligned}$$

Based on experience with other Euler sum constants, it was conjectured that $A_{2,3}$ satisfies a relation involving homogeneous combinations of degree five involving $\zeta(2), \zeta(3), \zeta(4), \zeta(5)$, $\ln(2), \mathrm{Li}_4(1/2)$ and $\mathrm{Li}_5(1/2)$, where $\mathrm{Li}_n(x) = \sum_{k=1}^{\infty} x^k k^{-n}$ denotes the polylogarithm function. The set of terms involving these constants with degree five are as follows: $\mathrm{Li}_5(1/2)$, $\mathrm{Li}_4(1/2)\ln(2)$, $\ln^5(2)$, $\zeta(5)$, $\zeta(4)\ln(2)$, $\zeta(3)\ln^2(2)$, $\zeta(2)\ln^3(2)$, $\zeta(2)\zeta(3)$. When the numerical value of $A_{2,3}$ is augmented with this set of terms, all computed to 135 decimal digits accuracy, and the resulting 9-long vector is input to the PSLQ algorithm, it detects the relation $(480, -1920, 0, 16, 255, 660, -840, -160, 360)$ at iteration 390. Solving this relation for $A_{2,3}$, one obtains the formula

$$\begin{aligned} A_{2,3} &= 4\,\mathrm{Li}_5(1/2) - \frac{1}{30}\ln^5(2) - \frac{17}{32}\zeta(5) - \frac{11}{8}\zeta(4)\ln(2) + \frac{7}{4}\zeta(3)\ln^2(2) \\ &\quad + \frac{1}{3}\zeta(2)\ln^3(2) - \frac{3}{4}\zeta(2)\zeta(3) \\ &= 4\,\mathrm{Li}_5(1/2) - \frac{1}{30}\ln^5(2) - \frac{17}{32}\zeta(5) - \frac{11}{720}\pi^4\ln(2) + \frac{7}{4}\zeta(3)\ln^2(2) \\ &\quad + \frac{1}{18}\pi^2\ln^3(2) - \frac{3}{24}\pi^2\zeta(3) \end{aligned}$$

(recall that $\zeta(2n) = (2\pi)^{2n}|B_{2n}|/\,[2(2n)!]$).

4. A New Formula for Pi

Humankind has been fascinated with the constant $\pi = 3.14159\ldots$ throughout recorded history. The Babylonians used the approximation 3.125, while the Old Testament (I Kings 7:23) assumes the value 3.0. Archimedes found the first true algorithm for computing π in about 250 BC, and used it to find π to two digits accuracy. Isaac Newton found several new formulas for π. Using these formulas, he recorded a value of π to 16 decimal places in his notebook, but later sheepishly admitted, "I am ashamed to tell you how many figures I carried these computations, having no other business at the time."

More recently, modern computer technology has made it possible to compute millions and even billions of digits, by employing fast new formulas for π and efficient arithmetic techniques, based on the fast Fourier transform (FFT). In the most recent computation of this sort, Yasumasa Kanada at the University of Tokyo computed over 51 billion decimal digits of π. Some additional details on the history of π are given in [3].

Although there are no engineering motivations for these computations, there is some mathematical interest. In particular, although mathematicians suspect that the digits of π (as well as numerous other math constants) are "random", they have never been able to prove this assertion in even a single instance. Thus there is continuing interest in the output of such calculations, to see if there are any statistical irregularities that would suggest that this conjecture is false.

Through the centuries mathematicians have assumed that the computational cost of computing just the n-th digit of π and similar constants is not significantly less than computing all of the first n digits. In other words, it has been assumed that there is no "shortcut" to computing just the n-th digit of π. Thus, it came as no small surprise when such an algorithm was recently discovered [1]. In particular, this simple scheme allows one to compute the n-th hexadecimal digit of π without computing any of the first $n-1$ digits, without using multiple-precision arithmetic software, and with very little memory. The 1,000,000-th hex digit of π can be computed in just two minutes on a personal computer.

This scheme is based on the following remarkable new formula for π:

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

Like the formulas shown above, this one was discovered using the PSLQ integer relation algorithm. This may be the first instance in history that a significant new formula for π was discovered by a computer.

The proof of this formula is not very difficult. First note that for any $k < 8$,

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

Thus we can write

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

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

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

$$\int_0^1 \frac{16\,y - 16}{y^4 - 2\,y^3 + 4\,y - 4}\, dy \;=\; \int_0^1 \frac{4y}{y^2 - 2}\, dy - \int_0^1 \frac{4y - 8}{y^2 - 2y + 2}\, dy \;=\; \pi$$

reflecting a partial fraction decomposition of the integral on the left-hand side.

Needless to say, this proof is not terribly difficult or obscure. One wonders why Euler, for example, didn't discover this formula. A similar formula for π^2 (which also was first discovered using the PSLQ algorithm) is as follows:

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

Formulas of this type for a few other mathematical constants are given in [1].

5. Computing the n-th Hex Digit of Pi

Computing individual hexadecimal digits of π using the above formula relies on the binary algorithm for exponentiation, wherein one efficiently evaluates x^n by successive squaring and multiplication. According to Knuth, this technique dates back at least to 200 B.C [6]. In this application, it is necessary to obtain the exponentiation result modulo a positive integer c, but this can be efficiently done with the following variant of the binary exponentiation algorithm, wherein the result of each multiplication is reduced modulo c:

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

A: if $n \ge t$ then $r \leftarrow br \bmod c$; $\quad n \leftarrow n - t$; $\quad$ endif
$t \leftarrow t/2$
if $t \ge 1$ then $r \leftarrow r^2 \bmod c$; $\quad$ go to A; $\quad$ endif

Here "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.

Consider now the first of the four sums in the formula above for π:

$$S_1 \;=\; \sum_{k=0}^{\infty} \frac{1}{16^k(8k+1)}$$

First observe that the hexadecimal digits of S_1 beginning at position $d+1$ can be obtained from the fractional part of $16^d S_1$. Then we can write

$$\mathrm{frac}(16^d S_1) \;=\; \sum_{k=0}^{\infty} \frac{16^{d-k}}{8k+1} \mod 1$$
$$= \sum_{k=0}^{d} \frac{16^{d-k} \bmod 8k+1}{8k+1} \mod 1 \;+\; \sum_{k=d+1}^{\infty} \frac{16^{d-k}}{8k+1} \mod 1$$

Position	Hex Digits Beginning At This Position
10^6	26C65E52CB4593
10^7	17AF5863EFED8D
10^8	ECB840E21926EC
10^9	85895585A0428B
10^{10}	921C73C6838FB2
2.5×10^{11}	87F72B1DC97869

Table 2: Hexadecimal Digits of π

For each term of the first summation of this last line, the binary exponentiation algorithm can be used to rapidly evaluate the numerator. This can be done using either integer or 64-bit floating-point arithmetic. Then floating-point arithmetic can be used to perform the division and add the quotient to the sum mod 1 (i.e. retaining only the fractional part). The second summation, where the exponent of 16 is negative, may be evaluated in a straightforward fashion using floating-point arithmetic. Only a few terms of this second summation need be evaluated, 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 base 16, yielding the $(d+1)$-th hexadecimal digit, plus several additional digits. Full details of this scheme, including some numerical considerations, as well as analogous formulas for a number of other basic mathematical constants, can be found in [1]. Sample implementations of this scheme in both Fortran and C are available from the web site `http://www.cecm.sfu.ca/personal/pborwein/`.

Along this line, Table 2 gives some hexadecimal digits of π computed using the above scheme. The last of these was recently obtained by Fabrice Bellard in France.

One question that immediately arises in the wake of this discovery is whether or not there is a formula of this type and an associated computational scheme to compute individual *decimal* digits of π. Alas, no decimal scheme for π is known at this time, although there is for certain constants such as $\log(9/10)$ — see [1]. On the other hand, there is not yet any proof that a decimal scheme for π cannot exist. This question is currently being actively pursued by researchers.

References

[1] D. H. Bailey, P. B. Borwein and S. Plouffe, "On The Rapid Computation of Various Polylogarithmic Constants", *Mathematics of Computation*, vol. 66, no. 218 (April 1997), pg. 903–913.

[2] David H. Bailey, Jonathan M. Borwein and Roland Girgensohn, "Experimental Evaluation of Euler Sums", *Experimental Mathematics*, vol. 3, no. 1 (1994), pg. 17–30.

[3] David H. Bailey, Jonathan M. Borwein, Peter B. Borwein and Simon Plouffe, "The Quest for Pi", Mathematical Intelligencer, vol. 19, no. 1 (January 1997), pg. 50–57.

[4] Jonathan M. Borwein and Peter B. Borwein, *Pi and the AGM*, Wiley, New York, NY, 1987.

[5] Helaman R. P. Ferguson and David H. Bailey, "Analysis of PSLQ, An Integer Relation Finding Algorithm", *Mathematics of Computation*, to appear.

[6] Donald E. Knuth, *The Art of Computer Programming*, vol. 2, Addison-Wesley, Reading, MA, 1981.