Numerical Results on the Transcendence of Constants Involving π, e, and Euler's Constant

David H. Bailey

February 27, 1987

Ref: *Mathematics of Computation*, vol. 50, no. 181 (Jan. 1988), pg. 275–281

Abstract

Let $x = (x_1, x_2, \cdots, x_n)$ be a vector of real numbers. x is said to possess an integer relation if there exist integers a_i such that $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0$. Recently Ferguson and Forcade discovered practical algorithms [7, 8, 9] which, for any n, either find a relation if one exists or else establish bounds within which no relation can exist. One obvious application of these algorithms is to determine whether or not a given computed real number satisfies any algebraic polynomial with integer coefficients (where the sizes of the coefficients are within some bound).

The recursive form of the Ferguson-Forcade algorithm has been implemented with multiprecision arithmetic on the Cray-2 supercomputer at NASA Ames Research Center. The resulting computer program has been used to probe the question of whether or not certain constants involving π, e, and γ satisfy any simple polynomials. These computations established that the following constants cannot satisfy any algebraic equation of degree eight or less with integer coefficients whose Euclidean norm is 10^9 or less: e/π, $e + \pi$, $\log_e \pi$, γ, e^γ, γ/e, γ/π, and $\log_e \gamma$. Stronger results were obtained in several cases. These computations thus lend credence to the conjecture that each of the above mathematical constants is transcendental.

The author is with the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center, Moffett Field, CA 94035.

Introduction

The problem of finding integer relations among a set of real numbers goes back to Euler, who showed that the Euclidean algorithm when applied to two real numbers either terminates, yielding an exact relation, or else produces an infinite sequence of approximate relations. Hermite asked for generalizations for $n > 2$. Jacobi responded with an algorithm that was subsequently modified and developed by Perron, Bernstein, and others. Poincare suggested an algorithm that was later refined by Brun. In the case where a relation does exist, Brun's algorithm has been proven to terminate and produce a relation when $n = 3$. However, none of these algorithms has been proven to work for $n > 3$, and numerous counterexamples have been found. In the case where the entries of a vector x have no exact integer relations, some of these algorithms provide a sequence of lattice approximations that converges to the line between the origin and x in the angular sense, but none produces a sequence that converges to the line in the absolute distance sense.

A breakthrough in this area occurred in 1979, when Ferguson and Forcade [7, 9] discovered a recursive algorithm that is guaranteed to find an integer relation for any vector x of any length n (if a relation exists). If the vector x does not satisfy an exact relation, then this algorithm produces a sequence of lattice approximations that converges to the line in the absolute distance sense (not just in the angular sense). Further, their algorithm provides a means of establishing firm lower bounds on the size of any possible relation. More recently Ferguson [8] found non-recursive algorithms that also have these properties, although these non-recursive algorithms are significantly more difficult to state and to implement on a computer.

These new algorithms have numerous possible applications, including factorization of polynomials, study of "Gauss sums", analysis of possible relationships between the fundamental constants of physics, and the analysis of the cosmological stability of the solar system. The most obvious application, however, is to determine whether or not a real number α whose value can be calculated on a computer is the root of any algebraic polynomial with integer coefficients. For this application it suffices to apply one of these algorithms to the $(n+1)$-long vector $x = (1, \alpha, \alpha^2, \cdots, \alpha^n)$. If a relation is found, then these integers are the coefficients of a polynomial satisfied by the number α. Conversely, if a computation establishes a bound within which no relations exist, then α cannot satisfy any algebraic polynomial whose coefficients are within this class. This method thus provides a computational technique for grasping the property of a number being algebraic.

A recursive form of the Ferguson-Forcade algorithm has been implemented by the author on the Cray-2 supercomputer operated by the Numerical Aerodynamic Simulation System at NASA Ames Research Center. This program employs a package of high-performance multiprecision arithmetic routines. It is necessary to use multiprecision arithmetic because the Ferguson-Forcade algorithm requires an extraordinarily high level of numeric precision to probe for integer relations of higher degree. The computer run time requirement is correspondingly high for seeking these high degree relations, but a number of useful computations of this sort can be performed on a supercomputer such as the Cray-2.

The following constants were selected for analysis by the above procedure: e/π, $e +$

π, $\log_e \pi$, γ, e^γ, γ/e, γ/π, and $\log_e \gamma$. Note that π, e, and e^π were not included because these are known to be transcendental [3]. There are of course many other interesting constants that could have been selected. It is hoped that some of these others can be analyzed in the future.

The Ferguson-Forcade Algorithm

The following is a precise statement of the particular version of the algorithm that was implemented for these applications. A full discussion of the mathematical theory behind this algorithm may be found in [9]. Lower case symbols will be used to denote vectors of real numbers and upper case symbols will be used to denote matrices of real numbers. The norms of the vector x and the matrix A are defined as the Euclidean norms:

$$|x| = \sqrt{\sum_i x_i^2}$$
$$|A| = \sqrt{\sum_{i,j} a_{ij}^2}$$

The transpose of the row vector x and the matrix A will be denoted by x^t and A^t, respectively. Finally, I_n will be used to denote an identity matrix of size $n \times n$.

Let x denote an n-long input vector of real numbers. To initialize the calculation set $P = xx^t I_n - x^t x$. In other words, $P_{ij} = -x_i x_j$ if $i \neq j$, and $P_{jj} = \sum_{i \neq j} x_i^2$. Now set $x' = x$, $P' = P$, and $A = I_n$. Then perform the procedure ALG (n, x', P', A), which is defined below.

If ALG (n, x', P', A) terminates, then the original row vector x, when multiplied on the right by the inverse of the current A matrix, should yield one entry that is within a reasonable tolerance of machine zero. The column of A^{-1} that produced this zero is then the desired relation. If ALG (n, x', P', A) completes but does not terminate, then any possible integer relation r must satisfy $|r| \geq |x|^2/|AP|$, where x and P are the initial arrays and A is the current A matrix. This fact is proved in [9]. At this point the process may be continued by performing ALG (n, x', P', A) again, and repeating until either a legitimate relation is discovered or else precision is exhausted. One can tell that precision has been exhausted if the algorithm terminates with a matrix A, but no entry of xA^{-1} is sufficiently close to machine zero.

The function of the procedure ALG (n, x, P, A) will now be described.

ALG $(1, x, P, A)$: If $x = 0$ then terminate; otherwise set $P = 0$ and $A = 1$ and exit.

ALG (n, x, P, A) for $n \geq 2$: If some entry of x is zero (or within a reasonable tolerance of machine zero), then terminate. Otherwise, perform the following steps:

1. Find a row of P with the smallest norm. Exchange this row with the last row of P, and also exchange the corresponding rows of A and entries of x.

2. Construct the $(n-1) \times (n-1)$ matrix $Q = uu^t I_{n-1} - u^t u$, where the vector $u =$

$(x_1/x_n, x_2/x_n, \cdots, x_{n-1}/x_n)$. Set $u' = u$, $Q' = Q$, and $B = I_{n-1}$. Then perform ALG $(n-1, u', Q', B)$. If it terminates, then set $c = 0$. Otherwise repeat it until the condition

$$|BQW| \ < \ \frac{|u|^2|v|}{2\sqrt{n+1}}$$

is satisfied. Here W denotes the $(n-1) \times n$ matrix consisting of all rows of P except the last, and v is the last row of P. When it is satisfied, set c to the integer vector closest to $Bu^t/|u|^2$.

3. Set

$$C \ = \ \begin{bmatrix} B & c \\ 0 & 1 \end{bmatrix}$$

and replace x by xC^{-1}, P by CP, and A by CA.

Actually, it is not necessary ever to invert the matrices A and C. The author's program carries both A and A^{-1} through all steps of the above procedure. For initialization, both A and A^{-1} are set to the identity. In the first step above, the columns of A^{-1} are exchanged instead of the rows. In the third step, the matrix B^{-1} and the negative of c are used to construct C^{-1}, and finally A^{-1} is replaced by $A^{-1}C^{-1}$.

There is something of a numeric difficulty in being able to clearly recognize a zero entry in the first step above. The author found that it was satisfactory to examine the entries for either being within twelve orders of magnitude of the "machine epsilon" (i.e., 10^{-6w}, where w is the number of words of precision used), or else being twenty orders of magnitude smaller than the other nonzero entries. It is necessary to allow this last condition because repeated constructions of the u vector from the x vector in the second step above can renormalize these numbers far above machine epsilon.

Multiprecision Techniques

Unfortunately, a very high level of numeric precision is required to perform the Ferguson-Forcade algorithm for values of n higher than three. In fact, the calculations reported here employed either 6,144 or 12,288 digit precision. For this purpose a package of high-performance multiprecision arithmetic routines was employed. These routines are similar to the ones previously used by the author in a high-precision computation of π [1]. Several improvements have been made in these routines since that computation, and these differences will be summarized here.

The main difference between these computations and those described in [1] is that an ordinary complex fast Fourier transform (FFT) is used here for multiplication instead of dual prime modulus transforms. Although the complex FFT technique fails due to numeric difficulties for very high precision (millions of digits), it runs approximately five times faster than the prime modulus technique on the Cray-2 and thus is preferable for this application. Another difference is that the radix of the multiprecision number representation is 10^6

instead of 10^7 as in [1]. This allows data to be split into two words containing three digits each upon entry to the FFT multiply routine.

The FFT routine used in this program is currently the fastest software available to perform a one-dimensional FFT on the Cray-2. Details of this FFT algorithm may be found in [2]. Multiprecision multiplication is performed using this FFT as follows. Let $x = (x_0, x_1, \cdots, x_{n-1})$ and $y = (y_0, y_1, \cdots, y_{n-1})$ denote the radix-b representations of two multiprecision numbers. Extend x and y to length $N = 2n$ by appending n zeroes to each. Then the product z of x and y (except for releasing carrys) is merely the convolution

$$z_k = C_k(x, y) = \sum_{j=0}^{N-1} x_j y_{k-j}$$

where the subscript $k - j$ is to be interpreted as $k - j + N$ if negative. This convolution is not evaluated directly but as

$$C(x, y) = F^{-1}[F(x)F(y)]$$

where F and F^{-1} denote the discrete Fourier transform and its inverse:

$$F_k(x) = \sum_{j=0}^{N-1} x_j e^{-2\pi ijk/N}$$

$$F_k^{-1}(x) = \frac{1}{N} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N}$$

Since the input data x and y and the output data z are all purely real, a technique described in [6] is used to reduce both the forward and reverse transforms to complex transforms of one lower order, which dramatically reduces the run time.

Multiprecision division and square root extraction are performed using forms of Newton's iteration that require only multiplications, and thus they piggyback off of the multiply procedure described above.

It should be noted that it is not necessary to perform all operations of the Ferguson-Forcade algorithm using high precision. For instance, the computation of matrix norms can always be done in single precision, although the author found it necessary to manually maintain the exponent, since otherwise even the very high dynamic range of the Cray floating-point format ($10^{2,466}$) occasionally overflows. Also, in the early stages of the algorithm, the A matrix in particular contains integers of only modest size, and a simple "schoolboy" multiplication procedure suffices for operations involving these numbers. The author's multiprecision multiply routine thus checks the number of nonzero words of precision in the arguments and performs the FFT multiply algorithm only if the actual precision of both arguments is above a certain level.

Algorithms for Computing the Constants

The constant π was computed using Borweins' quartically convergent algorithm, which was discovered in 1985 [5]. This algorithm is as follows: Let $a_0 = 6 - 4\sqrt{2}$ and $y_0 = \sqrt{2} - 1$. Iterate

$$\begin{aligned} y_{k+1} &= \frac{1 - (1 - y_k^4)^{1/4}}{1 + (1 - y_k^4)^{1/4}} \\ a_{k+1} &= a_k(1 + y_{k+1})^4 - 2^{2k+3} y_{k+1}(1 + y_{k+1} + y_{k+1}^2) \end{aligned}$$

Then a_k converges quartically to $1/\pi$: each successive iteration approximately *quadruples* the number of correct digits in the result.

Euler's constant γ was calculated using the following formulas, which are an improvement of a technique previously used by Sweeney [10].

$$\begin{aligned} \gamma &= \frac{2^n}{e^{2^n}} \sum_{m=0}^{\infty} \frac{2^{nm}}{(m+1)!} \sum_{t=0}^{m} \frac{1}{t+1} \; - \; n \log 2 \; + \; O(\frac{1}{2^n e^{2^n}}) \\ \log 2 &= \sum_{k=1}^{\infty} \frac{1}{(2k-1)3^{2k-1}} \end{aligned}$$

Unfortunately, this procedure exhibits only linear convergence. No quadratically convergent algorithm is yet known for γ. Nonetheless, it is feasible to compute γ to the precision required for these calculations without expending too much computer time.

Exponentials and logarithms (and e itself) were computed using quadratically convergent algorithms, which are also due to the Borweins [4]. The algorithm for computing e^t is as follows.

First we need to define the functions $P(s)$ and $Q(s)$. To define $P(s)$, set $x_0 = s$ and $y_0 = 16/(1 - s^2)$. Then iterate the following until convergence:

$$\begin{aligned} x_{k+1} &= \frac{2x_k}{x_k + 1} \\ y_{k+1} &= y_k \left(\frac{x_k + 1}{2} \right)^{2^{1-k}} \end{aligned}$$

The extraction of 2^k-th roots in the last line is performed using Newton's iteration with a level of precision that doubles at each step. $P(s)$ is then defined as the limiting value of y_k. To define $Q(s)$, set $a_0 = 1$, $b_0 = s$, $a'_0 = 1$, and $b'_0 = \sqrt{1 - s^2}$. Then iterate the following until convergence:

$$\begin{aligned} a_{k+1} &= \frac{a_k + b_k}{2} \\ b_{k+1} &= \sqrt{a_k b_k} \\ a'_{k+1} &= \frac{a'_k + b'_k}{2} \\ b'_{k+1} &= \sqrt{a'_k b'_k} \end{aligned}$$

$Q(s)$ is defined as the ratio of the limits of a and a'. With $P(s)$ and $Q(s)$ defined, the exponential function of t may be evaluated by using Newton iterations (with a variable level of precision as before) to solve the equation $Q(s) = t/\pi$ for s, and then evaluating $P(s)$. As a starting value for these Newton iterations, the author has found that a single precision calculation of the following is satisfactory:

$$s_0 = \begin{cases} 0.028762^{1/p} & \text{when} \quad p \leq 2.5 \\ 1 - e^{2.08-p} & \text{when} \quad 2.5 < p \leq 30 \\ 10^{0.434(2-p)} & \text{when} \quad p > 30 \end{cases}$$

where $p = t/\pi$. The natural logarithm of z can be obtained by using Newton iterations to solve $P(s) = z$ for s, and then evaluating $\pi Q(s)$.

Numerical Results

Computer programs employing the above algorithms, including the multiprecision routines, have been implemented in the ANSI Fortran-77 language. The Fortran compiler on the Cray-2 was able to automatically vectorize almost all loops in these codes. In the few cases where loops are vectorizable but not automatically vectorized by the compiler, vectorization was forced with directives. As a result, these programs run at nearly 100 million floating-point operations per second on one processor of the four-processor Cray-2. No attempt was made to utilize more than one processor. Most of these eight runs required on the order of two hours of processing time. Normally it would have been very difficult to obtain this much computer time for such an application. However, in early 1987 the Cray-2 and auxiliary equipment were moved to a new building, and before full production usage resumed some extra time was available.

The results of these calculations are listed in the table 1. The precision figures listed are the number of decimal digits of precision used. The bounds listed are the minimum Euclidean norm of the coefficients of any possible degree eight polynomial that the given constant could satisfy.

Acknowledgement

The author wishes to acknowledge the patient assistance of Prof. Helaman Ferguson in suggesting this work and in explicating the details of his algorithm.

Constant	Precision	Bound
e/π	12,288	6.1030×10^{14}
$e + \pi$	12,288	2.2753×10^{18}
$\log \pi$	6,144	8.7697×10^{09}
γ	6,144	3.5739×10^{09}
e^{γ}	12,288	1.6176×10^{17}
γ/e	6,144	1.8440×10^{11}
γ/π	6,144	6.5403×10^{09}
$\log \gamma$	6,144	2.6881×10^{10}

Table 1: Lower Bounds for the Euclidean Norms of Degree Eight Polynomials

References

1. Bailey, D. H., "The Computation of π to 29,360,000 Decimal Digits Using Borweins' Quartically Convergent Algorithm", preprint.

2. Bailey, D. H., "A High Performance Fast Fourier Transform Algorithm for the Cray-2", *Journal of Supercomputing*, to appear March 1987.

3. Baker, A., *Transcendental Number Theory*, Cambridge University Press, London, 1975.

4. Borwein, J. M., and Borwein, P. B., "The Arithmetic-Geometric Mean and Fast Computation of Elementary Functions", *SIAM Review* 26 (1984), p. 351-365.

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

6. Brigham, E. O., *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

7. Ferguson, H. R. P., and Forcade, R. W., "Generalization of the Euclidean Algorithm for Real Numbers to All Dimensions Higher Than Two", *Bulletin of the American Mathematical Society*, 1 (1979), p. 912-914.

8. Ferguson, H. R. P., "A Non-Inductive GL(n,Z) Algorithm That Constructs Linear Relations for n Z-Linearly Dependent Real Numbers", *Journal of Algorithms*, to appear.

9. Ferguson, H. R. P., "A Short Proof of the Existence of Vector Euclidean Algorithms", *Proceedings of the American Mathematical Society*, Vol. 97, No. 1 (May 1986), p. 8-10.

10. Sweeney, D. W., "On the Computation of Euler's Constant", *Mathematics of Computation*, 17 (1963), p. 170-178.