High-precision arithmetic and PSLQ

David H. Bailey
http://www.davidhbailey.com
Lawrence Berkeley National Lab (retired)
University of California, Davis

October 2, 2017

Numerical reproducibility in scientific computing

A December 2012 workshop on reproducibility in computing, held at Brown University, USA, noted that

> *Science is built upon the foundations of theory and experiment validated and improved through open, transparent communication. ...*
>
> *Numerical round-off error and numerical differences are greatly magnified as computational simulations are scaled up to run on highly parallel systems. As a result, it is increasingly difficult to determine whether a code has been correctly ported to a new system, because computational results quickly diverge from standard benchmark cases. And it is doubly difficult for other researchers, using independently written codes and distinct computer systems, to reproduce published results.*

- V. Stodden, D. H. Bailey, J. M. Borwein, R. J. LeVeque, W. Rider and W. Stein, "Setting the default to reproducible: Reproducibility in computational and experimental mathematics," `http://www.davidhbailey.com/dhbpapers/icerm-report.pdf`.

Numerical reliability

Many applications routinely use either 32-bit or 64-bit IEEE arithmetic, and employ fairly simple algorithms, assuming that all is well. But problems can arise:

1. Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors — numerical sensitivities are greatly magnified.
2. Certain applications with highly ill-conditioned linear systems.
3. Large summations, especially those involving $+/-$ terms and cancellations.
4. Long-time, iterative simulations (such as molecular dynamics or climate models).
5. Computations to resolve small-scale phenomena.
6. Studies in computational physics and experimental mathematics.

- D. H. Bailey, R. Barrio, and J. M. Borwein, "High precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121.

Analysis of collisions at the Large Hadron Collider

- The 2012 discovery of the Higgs boson at the ATLAS experiment in the LHC relied crucially on the ability to track charged particles with exquisite precision (10 microns over a 10m length) and high reliability (over 99% of roughly 1000 charged particles per collision correctly identified).
- Software: 5 millions line of C++ and Python code, developed by roughly 2000 physicists and engineers over 15 years.
- Recently, in an attempt to speed up the calculation, researchers found that merely changing the underlying math library resulted in some collisions being missed or misidentified.

Questions:

- How serious are these numerical difficulties?
- How can they be tracked down?
- How can the library be maintained, producing numerically reliable results?

Ways to enhance numerical reproducibility and solve accuracy problems

1. Employ an expert numerical analyst to re-examine every algorithm employed in the computation to ensure that only the most stable known schemes are being used.
2. Carefully analyze every step of the computation to ascertain the level of numerical sensitivity at each step.
3. Employ interval arithmetic for large portions of the application (which greatly increases run time and code complexity).
4. Employ higher-precision arithmetic arithmetic, assisted with some “smart” tools to help determine where extra precision is needed and where it is not.

Item #4 is often the only practical solution.

Numerical analysis expertise among U.C. Berkeley graduates

Of the 2010 U.C. Berkeley graduating class, 870 were in disciplines likely to require technical computing:

- Division of Mathematical and Physical Sciences (Math, Physics, Statistics).
- College of Chemistry.
- College of Engineering (including Computer Science).

Other fields (not counted) that will likely involve significant computing:

- Biology, geology, medicine, economics, psychology, sociology.

Enrollment in numerical analysis courses:

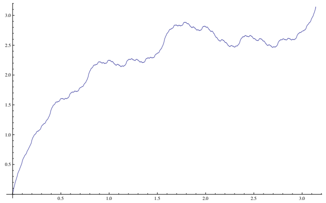
- Math 128A (Introductory numerical analysis required of math majors): 219.
- Math 128B (A more advanced course, required to do serious work): 24.

Conclusion: Fewer than 2% of U.C. Berkeley graduates who will do technical computing in their careers have had rigorous training in numerical analysis.

Enhancing reproducibility with selective usage of high-precision arithmetic

Problem: Find the arc length of the irregular function $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over the interval $(0, \pi)$ (using 10^7 abscissa points).

- If this computation is done with ordinary double precision arithmetic, the calculation takes 2.59 seconds and yields the result 7.073157029008510.
- If it is done using all double-double arithmetic (31-digit accuracy), it takes 47.39 seconds seconds and yields the result 7.073157029007832 (correct to 15 digits).
- But if only the summation is changed to double-double, the result is identical to the double-double result (to 15 digits), yet the computation only takes 3.47 seconds.



Graph of $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over $(0, \pi)$.

Aren't 64 bits enough?

There has been considerable resistance in the scientific computing community to the usage of high-precision arithmetic. Why?

- Many are persuaded that physical reality fundamentally does not require high precision, beyond, say, the limits of 64-bit IEEE arithmetic.
- Many believe that numerical sensitivity problems are always due to the usage of inferior algorithms.
- Some regard the usage of high-precision arithmetic as "cheating" or "sinful" (because it is too easy?).
- Some complain that easy-to-use high-precision arithmetic software facilities are not widely available, or the computational cost is too great.

But none of these excuses are valid.

- D. H. Bailey, R. Barrio, and J. M. Borwein, "High precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121.

Innocuous example where standard 64-bit precision is inadequate

Problem: Find a polynomial to fit the data $(1, 1048579, 16777489,$ $84941299, 268501249, 655751251, 1360635409, 2523398179,\ 4311748609)$ for arguments $0, 1, \cdots, 8$. The usual approach is to solve the linear system:

$$\begin{bmatrix} n & \sum_{k=1}^n x_k & \cdots & \sum_{k=1}^n x_k^n \\ \sum_{k=1}^n x_k & \sum_{k=1}^n x_k^2 & \cdots & \sum_{k=1}^n x_k^{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^n x_k^n & \sum_{k=1} x_k^{n+1} & \cdots & \sum_{k=1}^n x_k^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^n y_k \\ \sum_{k=1}^n x_k y_k \\ \vdots \\ \sum_{k=1}^n x_k^n y_k \end{bmatrix}$$

A 64-bit computation (e.g., using Matlab, Linpack or LAPACK) fails to find the correct polynomial in this instance, even if one rounds results to nearest integer.

However, if Linpack routines are converted to use double-double arithmetic (31-digit accuracy), the above computation quickly produces the correct polynomial:

$$f(x) \;=\; 1 + 1048577x^4 + x^8 \;=\; 1 + (2^{20} + 1)x^4 + x^8$$

Innocuous example, cont.

The result on the previous page can be obtained with double precision using Lagrange interpolation or the Demmel-Koev algorithm. But few scientists, outside of expert numerical analysts, are aware of these schemes.

Besides, even these schemes fail for higher-degree problems.

For example: $(1, 134217731, 8589938753, 97845255883, 549772595201,$
$2097396156251, 6264239146561, 15804422886323, 35253091827713,$
$71611233653971, 135217729000001, 240913322581691,\ 409688091758593)$
is generated by:

$$f(x) \;=\; 1 + 134217729x^6 + x^{12} \;=\; 1 + (2^{27}+1)x^6 + x^{12}$$

Lagrange interpolation and Demmel-Koev fail for this problem, but a straightforward Linpack scheme, implemented with double-double arithmetic, works fine.

Some applications where high precision is essential

1. Planetary orbit calculations (32 digits).
2. Supernova simulations (32–64 digits).
3. Climate modeling (32 digits).
4. Optimization problems in biology and other fields (32 digits).
5. Coulomb n-body atomic system simulations (32–120 digits).
6. Schrodinger solutions for lithium and helium atoms (32 digits).
7. Electromagnetic scattering theory (32–100 digits).
8. Scattering amplitudes of fundamental particles (32 digits).
9. Discrete dynamical systems (32 digits).
10. The Taylor algorithm for ODEs (100–600 digits).
11. Problems in experimental mathematics (100–50,000 digits).

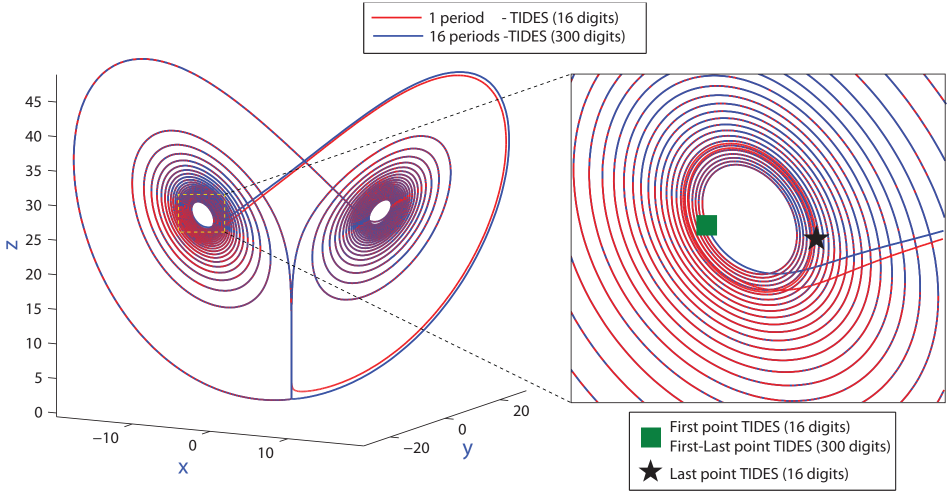
Coulomb n-body atomic system simulations

- Alexei Frolov of Queen's University in Canada has used high-precision arithmetic to solve a generalized eigenvalue problem that arises in Coulomb n-body interactions.
- Matrices are typically 5,000 x 5,000 and are very nearly singular.
- Computations typically involve massive cancellation, and high-precision arithmetic must be employed to obtain numerically reproducible results.
- Frolov has also computed elements of the Hamiltonian matrix and the overlap matrix in four- and five-body systems.
- These computations typically require 120-digit arithmetic.

Frolov: "We can consider and solve the bound state few-body problems ... beyond our imagination even four years ago."

- A. M. Frolov and D. H. Bailey, "Highly accurate evaluation of the few-body auxiliary functions and four-body integrals," *Journal of Physics B*, vol. 36, no. 9 (14 May 2003), pg. 1857–1867.

Taylor's method for ODEs with high-precision arithmetic



Numerical integration of the L25-R25 unstable periodic orbit for the Lorenz model during 16 time periods using TIDES with 300 digits, versus 1 time period using DP.

What is needed for high-precision floating-point software?

1. A rock-solid-reliable arithmetic engine, with precision scalable to 1,000,000 digits or more — 1,000 or even 10,000 digits is no longer enough.
2. Highly efficient algorithms, so single-processor performance is as fast as possible, given constraints of portability and ease of installation.
3. A thread-safe design to facilitate parallel processing, both thread-based (low-level) and message-passing (high-level) parallelism.
4. A comprehensive library of tuned transcendentals and special functions: sin, cos, exp, gamma, incomplete gamma, polylogarithms, Bessel functions, etc.
5. High-level language interfaces for C++, Fortran-90 and other languages.
6. Interoperability with *Maple* and *Mathematica*.

Free software for high-precision computation

1. ARPREC: Arbitrary precision, with numerous algebraic and transcendental functions. High-level interfaces for C++ and Fortran-90. `http://crd.lbl.gov/~dhbailey/mpdist`.
2. GFORTRAN: Now provides full REAL*16 (IEEE 128-bit, or 34-digit) support.
3. GMP: Produced by a volunteer effort and distributed under the GNU license. `http://gmplib.org`.
4. MPFR: C library for multiple-precision floating-point computations with exact rounding, based on GMP. `http://www.mpfr.org`.
5. MPFR++: High-level C++ interface to MPFR. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.
6. MPFUN90: Similar to ARPREC, but is written entirely in Fortran-90 and provides only a Fortran-90 interface. `http://crd.lbl.gov/~dhbailey/mpdist`.
7. QD: Performs "double-double" (31 digits) and "quad-double" (62 digits) arithmetic. High-level interfaces for C++ and Fortran-90. `http://crd.lbl.gov/~dhbailey/mpdist`.

How do multiprecision packages operate?

- A multiprecision floating-point number is represented as a string of DP floats or long integers, typically 53 or 64 bits per word, with the first few words holding sign, exponent and working precision (see below).
- Addition and subtraction are performed using relatively straightforward extensions of familiar schemes.
- Multiplication is performed using FFTs (see below).
- Division is performed using Newton iterations, with a level of working precision that approximately doubles with each iteration.
- Square roots are performed using Newton iterations, similar to division.
- Transcendental functions (cos, sin, exp, log, etc.) are performed using series expansions, after transforming the input into a small argument range. For higher precision levels, quadratically convergent iterations are used.

How to perform high-precision multiplication using FFTs

Let $A = (a_0, a_1, a_2, \cdots, a_{n-1})$ and $B = (b_0, b_1, b_2, \cdots, b_{n-1})$, where each element stores successive bits, and extend A and B to length $2n$ by padding with zeroes. Let $C = A \times B$. Then except for releasing carries,

$$C_k = \sum_{j=0}^{2n-1} a_j b_{k-j},$$

where we add $2n$ to the b subscript if it is negative. Then

$$\begin{aligned} C &= (C_0, C_1, C_2, \cdots, C_{2n-2}) \\ &= (a_0 b_0, a_0 b_1 + a_1 b_0, a_0 b_2 + a_1 b_1 + a_2 b_0, \cdots, a_{n-1} b_{n-1}). \end{aligned}$$

This is merely the *convolution* of A and B, which can be computed using FFTs:

$$C = F^{-1}(F(A) \cdot F(B)),$$

where $(F(A) \cdot F(B))$ means the element-wise product of the sequences $F(A)$ and $F(B)$.

For sufficiently large precision levels, FFT-based multiplication is many times faster than conventional grade-school multiplication — $5n \log_2 n$ instead of n^2 operations.

Thread safety: A major challenge for arbitrary precision software

Since run times are greatly magnified by high-precision arithmetic, parallel processing is increasingly essential.

However, almost all existing arbitrary-precision packages are not thread-safe at the application user level, and this kills thread-based parallelization

- Almost all high-level packages use global variables, and these variables are deadly for thread safety.
- Most global variables can be dispensed with, but not the working precision level, which is changed frequently within the package itself and often by users also.
- The only reasonable design for user programming is via operator overloading, available in languages such as Fortran-90 and C++, but overloading does not permit the working precision level to be passed.

What can be done?

MPFUN2015: A thread-safe arbitrary precision package

DHB has written a package (approx. 50,000 lines of code) for arbitrary precision floating-point computation. It is available in two versions:

- MPFUN-Fort: Completely self-contained, all-Fortran version. Compilation is a simple one-line command, which completes in a few seconds.
- MPFUN-MPFR: Calls the MPFR package for lower-level operations. Installation is significantly more complicated (since GMP and MPFR must first be installed), but performance is roughly 3X faster. We used MPFUN-MPFR in in the Poisson polynomial study.

Both versions include a high-level language interface, using Fortran custom datatypes and operator overloading — for most applications, only a few minor changes to existing double precision code are required.

A C++ version is planned.

Full details of design, algorithms, installation and usage are given in

- D. H. Bailey, "MPFUN2015: A thread-safe arbitrary precision computation package," `http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf`.

Experimental mathematics: Discovering new mathematical results

Methodology:

1. Compute various mathematical entities (limits, infinite series sums, definite integrals, etc.) to high precision, typically 100–10,000 digits.
2. Use algorithms such as PSLQ to recognize these numerical values in terms of well-known mathematical constants.
3. When results are found experimentally, seek formal mathematical proofs of the discovered relations.

Many results have recently been found using this methodology, both in pure mathematics and in mathematical physics.

If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics.
–Ð Kurt Godel

The PSLQ integer relation algorithm

Given a vector (x_n) of real numbers, an integer relation algorithm finds integers (a_n) such that

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$$

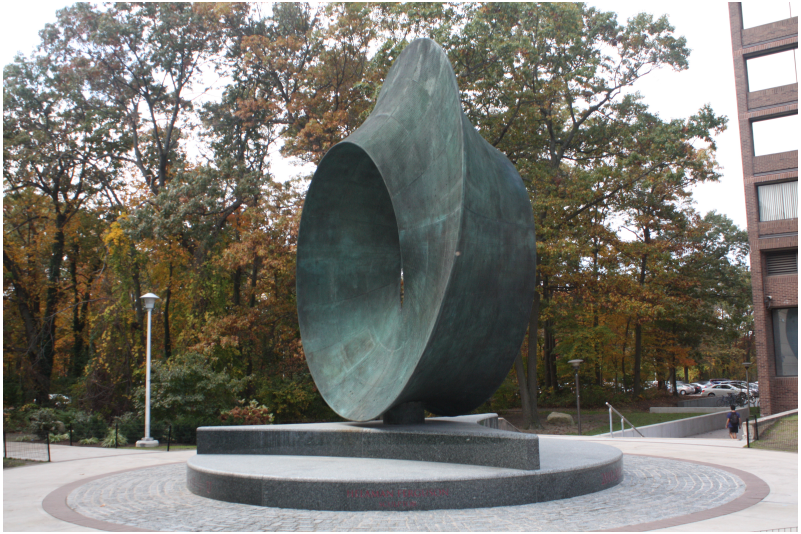
(to within the precision of the arithmetic being used), or else finds bounds within which no relation can exist.

Helaman Ferguson's PSLQ algorithm is the most widely used integer relation algorithm, although variants of the LLL algorithm can also be used.

Integer relation detection requires very high numeric precision, both in the input data and in the operation of the algorithm — at least nd digits, where d is the size in digits of the largest $|a_i|$.

1. H. R. P. Ferguson, D. H. Bailey and S. Arno, "Analysis of PSLQ, an integer relation finding algorithm," *Mathematics of Computation*, vol. 68, no. 225 (Jan 1999), 351–369.
2. D. H. Bailey and D. J. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), 1719–1736.

Helaman Ferguson's "Umbilic Torus SC" sculpture at Stony Brook Univ.



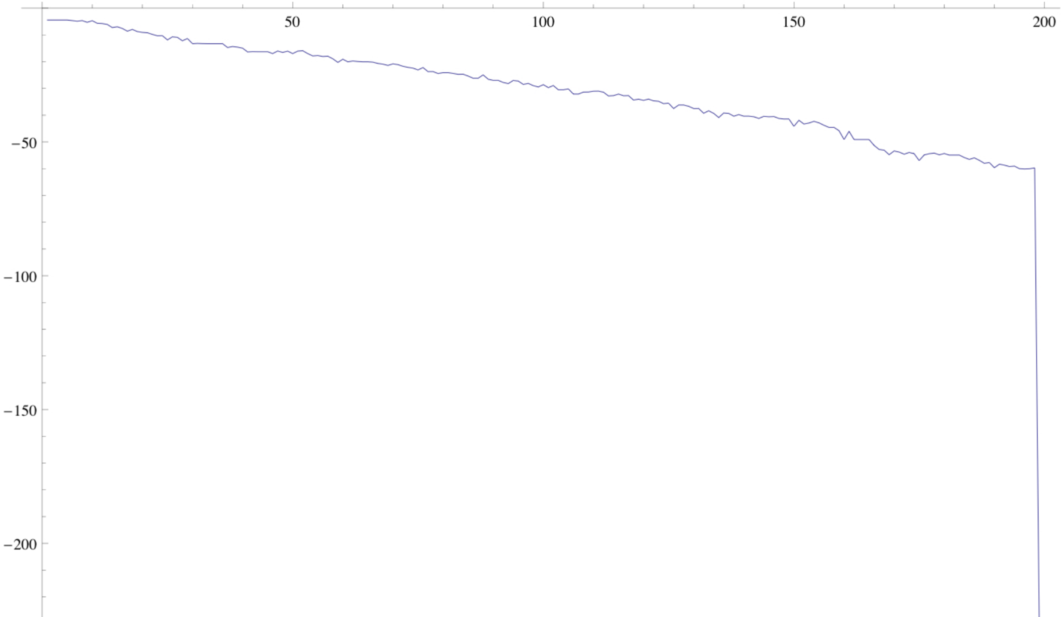
PSLQ, continued

- PSLQ constructs a sequence of integer-valued matrices B_n that reduce the vector $y = x \cdot B_n$, until either the relation is found (as one of the columns of matrix B_n), or else precision is exhausted.
- A relation is detected when the size of smallest entry of the y vector suddenly drops to roughly "epsilon" (i.e. 10^{-p}, where p precision in digits).
- The size of this drop can be viewed as a "confidence level" that the relation is not a numerical artifact: a drop of 20+ orders of magnitude almost always indicates a real relation.

Efficient variants of PSLQ:

- 2-level and 3-level PSLQ perform almost all iterations with only double precision, updating full-precision arrays as needed. They are hundreds of times faster than the original PSLQ.
- Multi-pair PSLQ dramatically reduces the number of iterations required. It was designed for parallel systems, but runs faster even on 1 CPU.

Decrease of $\log 10(\min |y_i|)$ in multipair PSLQ run



Application of multipair PSLQ

One simple but important application of multipair PSLQ is to recognize a computed numerical value as the root of an integer polynomial of degree *m*.

Example: The following constant is suspected to be an algebraic number:

$\alpha = 1.232688913061443445331472869611255647068988824547930576057634684778\ldots$

What is its minimal polynomial?

Method: Compute the vector $(1, \alpha, \alpha^2, \cdots, \alpha^m)$ for $m = 30$, then input this vector to multipair PSLQ.

Answer (using 250-digit arithmetic):

$$\begin{aligned} 0 = {} & 697 - 1440\alpha - 20520\alpha^2 - 98280\alpha^3 - 102060\alpha^4 - 1458\alpha^5 + 80\alpha^6 - 43920\alpha^7 \\ & + 538380\alpha^8 - 336420\alpha^9 + 1215\alpha^{10} - 80\alpha^{12} - 56160\alpha^{13} - 135540\alpha^{14} - 540\alpha^{15} \\ & + 40\alpha^{18} - 7380\alpha^{19} + 135\alpha^{20} - 10\alpha^{24} - 18\alpha^{25} + \alpha^{30} \end{aligned}$$

Apéry-like summations

Apéry's proof of the irrationality of $\zeta(3)$ has prompted considerable effort to extend these results to larger integer arguments. The formulas in question are

$$\zeta(2) = 3\sum_{k=1}^{\infty} \frac{1}{k^2 \binom{2k}{k}}, \tag{1}$$

$$\zeta(3) = \frac{5}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3 \binom{2k}{k}}, \tag{2}$$

$$\zeta(4) = \frac{36}{17}\sum_{k=1}^{\infty} \frac{1}{k^4 \binom{2k}{k}}. \tag{3}$$

Is there an analogous formula with a rational coefficient for $\zeta(5)$ or higher? A PSLQ computation with 10,000 digits ruled this out. What about multi-term relations?

Some multi-term relations found by PSLQ

$$\zeta(5) = 2\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^5\binom{2k}{k}} - \frac{5}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^2},$$

$$\zeta(7) = \frac{5}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^7\binom{2k}{k}} + \frac{25}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^4}$$

$$\zeta(9) = \frac{9}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^9\binom{2k}{k}} - \frac{5}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^7\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^2} + 5\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^5\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^4}$$
$$+ \frac{45}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^6} - \frac{25}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^4} \sum_{j=1}^{k-1} \frac{1}{j^2},$$

$$\zeta(11) = \frac{5}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^{11}\binom{2k}{k}} + \frac{25}{2}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^7\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^4}$$
$$- \frac{75}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^8} + \frac{125}{4}\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}} \sum_{j=1}^{k-1} \frac{1}{j^4} \sum_{i=1}^{k-1} \frac{1}{i^4}.$$

General formulas

Using bootstrapping and an application of the "Pade" function (which in both *Mathematica* and *Maple* produces Padé approximations to a rational function satisfied by a truncated power series), the following general results were discovered:

$$\sum_{k=1}^{\infty} \frac{1}{k^3(1-x^4/k^4)} = \frac{5}{2} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^3\binom{2k}{k}(1-x^4/k^4)} \prod_{m=1}^{k-1} \left(\frac{1+4x^4/m^4}{1-x^4/m^4}\right)$$

$$\sum_{k=1}^{\infty} \frac{1}{k^2-x^2} = 3 \sum_{k=1}^{\infty} \frac{1}{k^2\binom{2k}{k}(1-x^2/k^2)} \prod_{m=1}^{k-1} \left(\frac{1-4x^2/m^2}{1-x^2/m^2}\right)$$

Power series expansions for a specific x yield all the results for $\zeta(n)$.

The second formula was proven by the Wilf-Zeilberger method, a computer-based scheme that yields a rigorous proof. It was thus discovered and proven by computer.

- D. H. Bailey and J. M. Borwein, "Computer-assisted discovery and proof," T. Amdeberhan and V. H. Moll, ed., *Tapas in Experimental Mathematics*, AMS, May 2008, pg. 21–52.

The tanh-sinh numerical integration algorithm

Given $f(x)$ defined on $(-1,1)$, define $g(t) = \tanh(\pi/2 \cdot \sinh t)$. Then setting $x = g(t)$ yields

$$\int_{-1}^{1} f(x)\,dx \;=\; \int_{-\infty}^{\infty} f(g(t))g'(t)\,dt \;\approx\; h \sum_{j=-N}^{N} w_j f(x_j),$$

where $x_j = g(h_j)$ and $w_j = g'(h_j)$ can be precomputed for a given h. Since $g'(t)$ goes to zero very rapidly for large t, the integrand $f(g(t))g'(t)$ typically is a nice bell-shaped function, so that the simple summation above converges very rapidly. Reducing h by half typically doubles the number of correct digits.

We have found that tanh-sinh is the best general-purpose integration scheme for functions with vertical derivatives or singularities at endpoints, or for any function at very high precision ($>$ 1000 digits).

1. D. H. Bailey, X. S. Li and K. Jeyabalan, "A comparison of three high-precision quadrature schemes," *Experimental Mathematics*, vol. 14 (2005), no. 3, pg. 317–329.
2. H. Takahasi and M. Mori, "Double exponential formulas for numerical integration," *Publications of RIMS*, Kyoto University, vol. 9 (1974), pg. 721-Ð741.

Ising integrals from mathematical physics

We applied our methods to study three classes of integrals: C_n are connected to quantum field theory, D_n arise in the Ising theory of mathematical physics, while the E_n integrands are derived from Dn:

$$\begin{aligned}
C_n &:= \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n} \\
D_n &:= \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j}\left(\frac{u_i-u_j}{u_i+u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n} \\
E_n &= 2\int_0^1 \cdots \int_0^1 \left(\prod_{1\le j<k\le n} \frac{u_k - u_j}{u_k + u_j}\right)^2 \mathrm{d}t_2\, dt_3 \cdots \mathrm{d}t_n
\end{aligned}$$

where in the last line $u_k = t_1 t_2 \cdots t_k$.

D. H. Bailey, J. M. Borwein and R. E. Crandall, “Integrals of the Ising class,” *Journal of Physics A: Mathematical and General*, vol. 39 (2006), pg. 12271–12302.

Limiting value of C_n: What is this number?

Key observation: The C_n integrals can be converted to one- dimensional integrals involving the modified Bessel function $K_0(t)$:

$$C_n = \frac{2^n}{n!} \int_0^\infty t K_0^n(t) \, \mathrm{d}t$$

High-precision numerical values, computed using this formula and tanh-sinh quadrature, approach a limit. For example:

$$C_{1024} = 0.6304735033743867961220401927108789043545870787\ldots$$

What is this number? We copied the first 50 digits into the online Inverse Symbolic Calculator (ISC) at https://isc.carma.newcastle.edu.au. The result was:

$$\lim_{n\to\infty} C_n = 2e^{-2\gamma}.$$

where γ denotes Euler's constant. This is now proven.

Other Ising integral evaluations found using PSLQ

$$
\begin{aligned}
D_3 &= 8 + 4\pi^2/3 - 27\,\mathrm{L}_{-3}(2) \\
D_4 &= 4\pi^2/9 - 1/6 - 7\zeta(3)/2 \\
E_2 &= 6 - 8\log 2 \\
E_3 &= 10 - 2\pi^2 - 8\log 2 + 32\log^2 2 \\
E_4 &= 22 - 82\zeta(3) - 24\log 2 + 176\log^2 2 - 256(\log^3 2)/3 \\
&\quad + 16\pi^2\log 2 - 22\pi^2/3 \\
E_5 &= 42 - 1984\,\mathrm{Li}_4(1/2) + 189\pi^4/10 - 74\zeta(3) - 1272\zeta(3)\log 2 + 40\pi^2\log^2 2 \\
&\quad - 62\pi^2/3 + 40(\pi^2\log 2)/3 + 88\log^4 2 + 464\log^2 2 - 40\log 2
\end{aligned}
$$

where ζ is the Riemann zeta function and $Li_n(x)$ is the polylog function. E_5 remained a "numerical conjecture" for several years, but was proven in March 2014 by Erik Panzer.

E_5 was reduced to a 3-D integral of a 60-line integrand, which was evaluated using tanh-sinh quadrature to 250-digit arithmetic using over 1000 CPU-hours on a highly parallel system. The PSLQ calculation required only seconds.

Algebraic numbers in Poisson potential functions

Lattice sums arising from the Poisson equation have been studied widely in mathematical physics and image processing. We numerically discovered, and then proved, that for rational (x, y), the 2-D Poisson potential function satisfies

$$\phi_2(x, y) \quad = \quad \frac{1}{\pi^2} \sum_{m,n \text{ odd}} \frac{\cos(m\pi x)\cos(n\pi y)}{m^2 + n^2} \quad = \quad \frac{1}{\pi} \log \alpha$$

where α is *algebraic*, i.e., the root of an integer polynomial

$$0 \quad = \quad a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_n\alpha^n$$

The minimal polynomials for these α were found by PSLQ calculations, with the $(n+1)$-long vector $(1, \alpha, \alpha^2, \cdots, \alpha^n)$ as input, where $\alpha = \exp(\pi\phi_2(x, y))$. PSLQ returns the vector of integer coefficients $(a_0, a_1, a_2, \cdots, a_n)$ as output.

- D. H. Bailey, J. M. Borwein, R. E. Crandall and J. Zucker, "Lattice sums arising from the Poisson equation," *Journal of Physics A: Mathematical and Theoretical*, vol. 46 (2013), pg. 115201, `http://www.davidhbailey.com/dhbpapers/PoissonLattice.pdf`.

Samples of minimal polynomials found by PSLQ

k	Minimal polynomial for $\exp(8\pi\phi_2(1/k, 1/k))$
5	$1 + 52\alpha - 26\alpha^2 - 12\alpha^3 + \alpha^4$
6	$1 - 28\alpha + 6\alpha^2 - 28\alpha^3 + \alpha^4$
7	$-1 - 196\alpha + 1302\alpha^2 - 14756\alpha^3 + 15673\alpha^4 + 42168\alpha^5 - 111916\alpha^6 + 82264\alpha^7$ $-35231\alpha^8 + 19852\alpha^9 - 2954\alpha^{10} - 308\alpha^{11} + 7\alpha^{12}$
8	$1 - 88\alpha + 92\alpha^2 - 872\alpha^3 + 1990\alpha^4 - 872\alpha^5 + 92\alpha^6 - 88\alpha^7 + \alpha^8$
9	$-1 - 534\alpha + 10923\alpha^2 - 342864\alpha^3 + 2304684\alpha^4 - 7820712\alpha^5 + 13729068\alpha^6$ $-22321584\alpha^7 + 39775986\alpha^8 - 44431044\alpha^9 + 19899882\alpha^{10} + 3546576\alpha^{11}$ $-8458020\alpha^{12} + 4009176\alpha^{13} - 273348\alpha^{14} + 121392\alpha^{15}$ $-11385\alpha^{16} - 342\alpha^{17} + 3\alpha^{18}$
10	$1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8$

The minimal polynomial for $\exp(8\pi\phi_2(1/32, 1/32))$ has degree 128, with individual coefficients ranging from 1 to over 10^{56} (see next viewgraph). This computation required 10,000-digit precision. Most of the time was spent on PSLQ, not quadrature.

Other polynomials required up to 50,000-digit precision, with runs extending to several days. The ARPREC software failed on at least one of these problems.

192-degree minimal polynomial found by multipair PSLQ for $x = y = 1/35$

$$\begin{array}{l}
-1 \\
-127936\alpha^{1} \\
-119729568\alpha^{2} \\
+86259615424\alpha^{3} \\
-63414593420752\alpha^{4} \\
-23761356230294720\alpha^{5} \\
-21521006149475920864\alpha^{6} \\
+611939502140287079872\alpha^{7} \\
-368241444434585386649704\alpha^{8} \\
+34534137997535622208800832\alpha^{9} \\
+190487328667825240861480928\alpha^{10} \\
+70731866698900460686569086656\alpha^{11} \\
-30639321248717091789168538879600\alpha^{12} \\
+1922209242084456061370070376570688\alpha^{13} \\
-92225937246234731111918315818791008\alpha^{14} \\
+523708597245884534879542377776516544\alpha^{15} \\
-235187168740813424983476512954481139732\alpha^{16} \\
+7429223399159195923652315325576416851776\alpha^{17} \\
-183888805409443551177519325744878574722592\alpha^{18} \\
+3271185405201792963719366162166966763587520\alpha^{19} \\
-30299858585212592953207274837044089721625936\alpha^{20} \\
-359992131877582010695076712447360628122541504\alpha^{21} \\
+21102025339314733846086441916632636325491195040\alpha^{22} \\
-476414747523697562008419279360098519444810134848\alpha^{23} \\
+7169682338305684610208768903330082567075749930088\alpha^{24} \\
-77147038334570163492295981291240215533152479746240\alpha^{25} \\
+576164594448741453064385715717563185909190291963744\alpha^{26} \\
-2736845608824484767533372475869861068967680432337984\alpha^{27} \\
+14259726273826198867906144815986191950918103600120720\alpha^{28} \\
-418898418536517746458185180042147268305073686192184768\alpha^{29} \\
+11576727630588055309375415098495616370526328495842993184\alpha^{30} \\
-215864800528660839585002162871715414654518484882221347136\alpha^{31} \\
+2990075428515164844805618172618994337159659201108555443326\alpha^{32} \\
-32103405822691698883721869657536188329165809065032597536320\alpha^{33} \\
+266563559238616545234045835785848715794979563412588549450912\alpha^{34} \\
-1588318769762074127932309766905276050093368042694667151097024\alpha^{35} \\
+4203983024865006644081516487096952711041894782888827508128720\alpha^{36} \\
+45741795572596990194281792252600295928845217432176402329342656\alpha^{37} \\
-888295322846459251495779433538199608179522746791003115766467360\alpha^{38} \\
+9318868074863097497435367536227145321575550333899160360356887616\alpha^{39} \\
-75360568030730663716613169159158032198635749984420852664531432312\alpha^{40} \\
+511904470165868318936871843389229757000581029769096726546021748160\alpha^{41} \\
-3023546278817700158319248501909046229060364677626955289951128897760\alpha^{42} \\
+15762439699463291788277556458704672972787957041647009977980115608384\alpha^{43} \\
-72736254087237112123433235647299134668422519990232924303557659578512\alpha^{44} \\
+294655977143150502457165877012542972911987116693855121083332478262976\alpha^{45} \\
-1023418666617067140855492177628022043641503497647927192890895515496096\alpha^{46} \\
+2872077837131433597858520147550093715024860106120898508958083768560704\alpha^{47} \\
-5244656757384067630568130031166724732476128944651080774800327815250116\alpha^{48} \\
-4375176099824059357126327245955265067024929918868045662842672259365696\alpha^{49} \\
+106846263928061290720995820179791755195031418846948198250527939644422688\alpha^{50} \\
-723639781157123885224054475278025279616831219096501715148796353565656000\alpha^{51} \\
+3859388734169523941501510320228124865152568559427283813598867448036672464\alpha^{52} \\
-18252524309522593867333335673661626604394606878814374964432136569382704704\alpha^{53} \\
+77801314121940709085266334072671847312436136451041737155601244216316832608\alpha^{54} \\
-295924487894698517430363594146438746974018698578472622597771256623091267264\alpha^{55} \\
+990883094386701201740101120896676716687418747972363663196338999871139087736\alpha^{56} \\
-2890647897108739627601089028825831836951153432467718962671967596342224331584\alpha^{57} \\
+7329280206578701233923536747523603867159738577214393279268225554697992024224\alpha^{58} \\
-16529926761499923756105727653427646107756611125773189972311741253023019544512\alpha^{59} \\
+36621401257350141587688985844214102107297318891842289523223570545646432107248\alpha^{60} \\
-97806177157646206090474078202543089426709440278475346399840243186928760712768\alpha^{61} \\
+342578593578669077931375617617876157438332517966938868387799492392630391951328\alpha^{62} \\
-1318494126999138496197980784859753020673610723076044612133782854663456341738176\alpha^{63} \\
+4770648039063802379141989053619421884321933507524681278243277292208295535890193\alpha^{64} \\
-15472388710953109453763060955215809455238864792962040591046439233238251454944640\alpha^{65} \\
+44993126133327412718394996675533962755303579158603905782013384675151509701621696\alpha^{66} \\
-118993678023270686331414029749780909390537125214711043931901528442137124714048640\alpha^{67} \\
+291344068959030996949609862561213601556661749401236700666696596438800158224020448\alpha^{68} \\
-672232544304160020140857460151229781783982323755372371953704546852887742475693952\alpha^{69} \\
+1483756546102154371991621621590988762258290163435879224071206411473630010421819712\alpha^{70} \\
-3160698702072288781771809020419553277725945343183442108817290538424291765729105536\alpha^{71} \\
+6501409535318122589047227869235783252522832570490016457500770280864513874652673648\alpha^{72} \\
-12819483008841743167047686498254296089188806357669061135254105501043843161878338944\alpha^{73} \\
+23940036962388573943673383046497610566473660443396196928270109685668366770350448320\alpha^{74} \\
-41783013070841149166516454269070382975961508352312427230333483156459522180289158272\alpha^{75} \\
+67368007706334464267463678880196649459723255934263643619439168617608299192825564576\alpha^{76} \\
-99642265418236291841000783588241107943576928164973329187609122090423042369846900608\alpha^{77} \\
+135684188062925983721829145899247186881926375911313411751554577363782314702542167104\alpha^{78} \\
-175027359690635875118927274192015444148378662148930449526142186378361018115616958080\alpha^{79} \\
+230610470370976022179942433042240414747465002830049591030281645554186668837721744600\alpha^{80} \\
-345422259636540072058441736406800364089516876786410123401855696720824391150388817792\alpha^{81} \\
+607137279243399399687877325853919130478767608151821751113771985567328949196202362560\alpha^{82} \\
-1146510481188778693690480353627945407015450534575302104964676472156118460868902533760\alpha^{83} \\
+2105965918280150700239337266375899616661414836881685735099986426462071874731653407456\alpha^{84} \\
-3578099460146102835319749769961500032509484008579871514018505538256220639466406781312\alpha^{85} \\
+5536176932749319486998056607166537426841616046655786532661950492192459385736417266240\alpha^{86} \\
-7794809387121374412027723614750591530039194538680900530684292991362557543590881311872\alpha^{87} \\
+10030481053829943456941341746197226060710869001059273241750964485149858086288306297744\alpha^{88} \\
-11855047237622313521872161223741536919023583101964392808122345150250641280790373298048\alpha^{89} \\
+12893688641739677210122894074867960717441055755861579189125047363100403706385492504000\alpha^{90} \\
-12809768930934167317548440121172865343713268055679115994897043173034752329114006909568\alpha^{91} \\
+11262021213965304198804174698825443038955997004185306033241807189003803427064173678496\alpha^{92} \\
-7849339137067651884501051001175657872891640580392350339448446525146908046759472284032\alpha^{93} \\
+2136636160531235808486132100072698791511127607341553178572108601522645635713639494976\alpha^{94} \\
+6182245041445709382582906443600895624739514751663138744452121338925118726793099119488\alpha^{95} \\
-17027139906672640329386998091617550180769534392263392331607751724460799383582736580380\alpha^{96}
\end{array}$$

$$\begin{array}{l}
+29712689442226074490778272413199981836245382131442863784077108073804247980419140795264\alpha^{97} \\
-42913200159533980876837680464079550150161925164315385234076771249584133995790605789888\alpha^{98} \\
+54875995472202404064433213065393198278520963443598737304851154803060885419026758577792\alpha^{99} \\
-63825924492815000947804006660643589720936270079124038361274887833762135234355101020256\alpha^{100} \\
+68414285135348430004939705729313406303651902455163077326777079330134700391931405532544\alpha^{101} \\
-68052822555902228583831595467842429584162408591782046566558459654662872517242957132352\alpha^{102} \\
+63030283713395534150979634221684031444907507016820847420753457009176282665669129552000\alpha^{103} \\
-54395011700122510436311965134146069935113985222440557617340304628640177217303223510128\alpha^{104} \\
+43661984233468332856484141573706612495184170864395287152161473759679906830742322508672\alpha^{105} \\
-32445083300862179709263952078603686506241016652233926920050485448240544236200991135168\alpha^{106} \\
+22121488216739201475462682897890849680076071306863200873172222890471168791244013948544\alpha^{107} \\
-13610378673217079648813094175609528224090224497348403954492135388377641815307093302560\alpha^{108} \\
+7303046170572758067914716975092412590455319569015290620012702916095079596345712424320\alpha^{109} \\
-3131236977889234295288739605819294961076022655796155501587485430182196310350741412160\alpha^{110} \\
+722365801671568238376537172560503547318051932349906390028021898906118213151314346112\alpha^{111} \\
+423601672347880485375916854724736111845460547572098224965700862087464288437477713112\alpha^{112} \\
-789372219071689131389195080081110795264640291084303894471284509727586944366644459136\alpha^{113} \\
+754844732770124186279971574591381270567329047796107073522275693379041299799000185920\alpha^{114} \\
-570721609484121135843526600643872181106278042023747642137438316501500109706499901312\alpha^{115} \\
+373431689502164436293456395379326584419834315995264527851770397586202342215482658208\alpha^{116} \\
-218828293376534306771048774311447660676991404850097946000689151980961553792808627328\alpha^{117} \\
+116819901747981791438375631794807731777629646639313589196533853783698354327272619712\alpha^{118} \\
-57453490833130202085276802289813871676642923245392648322689802787108386720310373760\alpha^{119} \\
+26322973910217009898036214368526156227892184032219090112763726491148743638265345648\alpha^{120} \\
-11422293344043226356444271083191575214531430352803712260273887252165192023955482240\alpha^{121} \\
+4829637193522288826261580288388799608086659826584129673254995173846268064699275584\alpha^{122} \\
-2078817645839949613653484866883184944970522163048142353896259321595643918566571904\alpha^{123} \\
+954520484158500566039633963780071095170152665054125308901179320476061925020693472\alpha^{124} \\
-476059705123351801546660809193814948416900226947841461260532426685030594421013632\alpha^{125} \\
+251019218316831093023612611636151648630754206670792757985712506316495845522315200\alpha^{126} \\
-133387943415348693213086926089912666295139757189579549272926455197093747072689536\alpha^{127} \\
+68669026491757170599503679381653507376133436777604611890688641155842031359807249\alpha^{128} \\
-33465058649746756993954154200626776375881202657693549363818927924447316311308992\alpha^{129} \\
+15265533880702809827068316634690758620560027313985769068567445431127044039098336\alpha^{130} \\
-6486555236218719687502565717159307954250235769707821904032108839967986971592256\alpha^{131} \\
+2562655019815708913497740003785614466295686123950263933555489655843860264778480\alpha^{132} \\
-940718343979230237299701844566444332546715633645270453853660861734418393188288\alpha^{133} \\
+320771113247818349988983708385653420090152866910698288499686276305286454845600\alpha^{134} \\
-101564452363828199943698157849015312966252040558153110340955250960209509456704\alpha^{135} \\
+29840961316238615781884908158117270537546621262752881740317094715665305372024\alpha^{136} \\
-8126376541879784121657187449503319934776774553449947624311536879983067197120\alpha^{137} \\
+2046994405582919610833914133280166599368341899820908062960382811295175875424\alpha^{138} \\
-475319952147035704834648908024038830280982173827054926876842801020099391040\alpha^{139} \\
+101156139413765995787243855087479738205336119604403640417144416030882137040\alpha^{140} \\
-19536944183496787665897200701914355830359619924729440672204518600726485952\alpha^{141} \\
+3365653573540320310956912862417397989082549665698949749718751351520029216\alpha^{142} \\
-500297582644104844305032452251740515847130656744730715102464280225674048\alpha^{143} \\
+59315729762438733681847579101255419553245420619721207125629069846855484\alpha^{144} \\
-4088968402271371424704252106469344820514085294011337888699549616483264\alpha^{145} \\
-410467411100906673223801525015092827907382852275953565568989408591520\alpha^{146} \\
+244373061239573525170364306606773136690590544856331578133540062357184\alpha^{147} \\
-67137173574208581929099251980225280832815704879993996789961828208784\alpha^{148} \\
+15109453726979262114404719422671376896934991181254480301068783496000\alpha^{149} \\
-3041110741874930638456938010940447684323532404585513230577319188704\alpha^{150} \\
+543474006539496836413675511283771054220572953580018501137140774336\alpha^{151} \\
-82818210860123209370721479679382853454718857015790579069608342392\alpha^{152} \\
+10208909053669116781213384172905303479740054958245095883752727616\alpha^{153} \\
-950607878263673381203873433978784840581837837196127719161731872\alpha^{154} \\
+61537081222427663468738382440653874375219870006734261147376320\alpha^{155} \\
-3614460685557548015719619196028506524834155796764598110372912\alpha^{156} \\
+735146337727640447184879802116259476500819273333226229041984\alpha^{157} \\
-195982957610545601975537049907586449965158403571666750972768\alpha^{158} \\
+35410924134603994553881822516846688340985279903764142254528\alpha^{159} \\
-4586047122379261183418802995335871699189992574849634423682\alpha^{160} \\
+461731528371721725878159689755046846967144056271095432896\alpha^{161} \\
-39548982563052670609179355793571460666409257926095752160\alpha^{162} \\
+3249771246267792941295436011847663684436436914874939968\alpha^{163} \\
-278663520870823657308884684973631236730880421051628656\alpha^{164} \\
+23298555433192575099637973614016576299882162116142016\alpha^{165} \\
-1572480211244215449798675303409379604821388457521312\alpha^{166} \\
+64631732188205970561818215170229051834055602691904\alpha^{167} \\
-359811432336185820581657539275402497742419221400\alpha^{168} \\
-81408350985197555231086181640295965142859955520\alpha^{169} \\
-2670526194482509900914512598125562096574018400\alpha^{170} \\
+824346510881081434287176385644420962607200832\alpha^{171} \\
-40421524575366405147127382177082176049208656\alpha^{172} \\
+859850260877732200189610333732782641323968\alpha^{173} \\
-21746942237477507610278314875487367745056\alpha^{174} \\
+1166204976072113679141136387334878097216\alpha^{175} \\
-16255593447525276321168587153278001172\alpha^{176} \\
-1031047179999272558707644762054830656\alpha^{177} \\
+57315125130135149120405695909830560\alpha^{178} \\
-2190911023714361858443237127579840\alpha^{179} \\
+42156348386945234230569584900496\alpha^{180} \\
-400797347138034992145950828864\alpha^{181} \\
-3239264180104926121034061856\alpha^{182} \\
+57518923520963556655145024\alpha^{183} \\
-302326293499892000458856\alpha^{184} \\
-648961315190292320832\alpha^{185} \\
+4804670128512328736\alpha^{186} \\
+97238301302930240\alpha^{187} \\
-106616047418832\alpha^{188} \\
-478181605696\alpha^{189} \\
+208298592\alpha^{190} \\
+25664\alpha^{191} \\
-\alpha^{192}
\end{array}$$

This distinctive semi-elliptical pattern is strong evidence that the result is correct.

Summary

- High-precision arithmetic can now be performed rapidly by means of widely available, easy-to-use software.
- High-precision arithmetic is required by the PSLQ algorithm, which finds integer relations among a set of computed floating-point real numbers.
- This combined capability (HP arithmetic and PSLQ) is a tool of enormous power for experimental mathematics.

Thanks!

This talk is available at
`http://www.davidhbailey.com/dhbtalks/dhb-hiprec-2017.pdf`