Mathematics **2015**, *3*, 337-367; doi:10.3390/math3020337

OPEN ACCESS

mathematics

ISSN 2227-7390

www.mdpi.com/journal/mathematics

Article

High-Precision Arithmetic in Mathematical Physics

David H. Bailey [1,*] and Jonathan M. Borwein [2]

[1] Lawrence Berkeley National Laboratory (retired) and University of California, Davis, Davis, CA 95616, USA

[2] CARMA, University of Newcastle, Callaghan, NSW 2308, Australia; E-Mail: jon.borwein@gmail.com

***** Author to whom correspondence should be addressed; E-Mail: david@davidhbailey.com.

Academic Editor: Palle E.T. Jorgensen

Received: 19 January 2015 / Accepted: 16 April 2015 / Published: 12 May 2015

Abstract: For many scientific calculations, particularly those involving empirical data, IEEE 32-bit floating-point arithmetic produces results of sufficient accuracy, while for other applications IEEE 64-bit floating-point is more appropriate. But for some very demanding applications, even higher levels of precision are often required. This article discusses the challenge of high-precision computation, in the context of mathematical physics, and highlights what facilities are required to support future computation, in light of emerging developments in computer architecture.

Keywords: high-precision arithmetic; numerical integration; PSLQ algorithm; Ising integrals; Poisson equation

---

1. Introduction

For many scientific calculations, particularly those that employ empirical data, IEEE 32-bit floating-point arithmetic is sufficiently accurate, and is preferred since it saves memory, run time and energy usage. For other applications, 64-bit floating-point arithmetic is required to produce results of sufficient accuracy, although some users find that they can obtain satisfactory results by switching between 32-bit and 64-bit, using the latter only for certain numerically sensitive sections of code. One challenge of modern computing is to develop tools that will help users determine which portions of a computation can be performed with lower precision and which must be performed with higher precision.

Moreover, some scientists and engineers running large computations have recently discovered, often to their great dismay, that with the rapidly increasing scale of their computations, numerical difficulties render the results of questionable accuracy even with 64-bit arithmetic. What often happens is that a conditional test takes the wrong branch, so that in some circumstances the results are completely wrong.

As a single example that has been reported to the present authors, the ATLAS experiment at the Large Hadron Collider (which was employed in the 2012 discovery of the Higgs boson) relies 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). The software used for these detections involves roughly five million lines of C++ and Python code, developed over a 15-year period by some 2000 physicists and engineers.

Recently, in an attempt to speed up these calculations, researchers found that merely changing the underlying math library caused some collisions to be missed and others misidentified. This suggests that their code has significant numerical sensitivities, and results may be invalid in certain cases.

How serious are these difficulties? Are they an inevitable consequence of the limited accuracy of empirical data, or are they exacerbated by numerical sensitivities in the code? How can they be tracked down in source code? And, once identified, how can they most easily be remedied or controlled?

1.1. Numerical Reproducibility

Closely related to the question of numerical error is the issue of reproducibility in scientific computing. A December 2012 workshop held at Brown University in Rhode Island, USA, noted that:

> *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.* [1,2].

Example 1.1 (Variable precision I)**.** As a simple illustration of these issues, suppose one wishes to compute the arc length of the irregular function $g(x) = x + \sum_{0 \leqslant k \leqslant 10} 2^{-k} \sin(2^k x)$, over the interval $(0, \pi)$,using 10^7 abscissa points. See Figure 1.

If this computation is performed with ordinary IEEE double arithmetic, the calculation takes 2.59 seconds and yields the result 7.073157029008510. If performed on another system, the results typically differ in the last four digits. If it is done using "double-double" arithmetic (approximately 31-digit accuracy; see Section 2), the run takes 47.39 seconds and yields the result 7.073157029007832, which is correct to 15 digits. But if only the summation is performed using double-double arithmetic, and the integrand is computed using standard double arithmetic, the result is identical to the double-double result (to 15 digits), yet the computation only takes 3.47 seconds. In other words, the judicious usage of double-double arithmetic in a critical summation completely eliminates the numerical non-reproducibility, with only a minor increase in run time.

A larger example of this sort arose in an atmospheric model (a component of large climate model). While such computations are by their fundamental nature "chaotic," so that computations will eventually

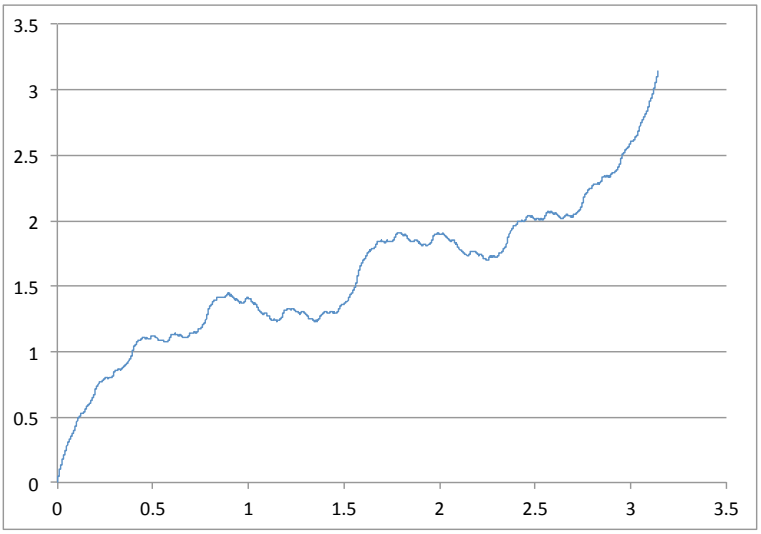

Figure 1. Graph of $g(x) = x + \sum_{0 \leqslant k \leqslant 10} 2^{-k} \sin(2^k x)$, over $(0, \pi)$.

depart from any benchmark standard case, nonetheless it is essential to distinguish avoidable numerical error from fundamental chaos.

Researchers working with this atmospheric model were perplexed by the difficulty of reproducing benchmark results. Even when their code was ported from one system to another, or when the number of processors used was changed, the computed data diverged from a benchmark run after just a few days of simulated time. As a result, they could never be sure that in the process of porting their code or changing the number of processors that they did not introduce a bug into their code.

After an in-depth analysis of this code, He and Ding found that merely by employing double-double arithmetic in two critical global summations, almost all of this numerical variability was eliminated. This permitted benchmark results to be accurately reproduced for a significantly longer time, with virtually no change in total run time [3].

Researchers working with some other large computational physics applications reported similar success, again merely by modifying their codes to employ a form of high-precision arithmetic in several critical summation loops [4].

In short, while higher-precision arithmetic is not a guarantee of absolute bit-for-bit reproducibility, nonetheless in many cases it is an excellent solution to numerical difficulties.

1.2. Dealing with Numerical Difficulties

Some suggest that the only proper way to deal with such difficulties is to carefully examine each algorithm and subroutine in a code, to ensure that only the best schemes are being used, and that they are all implemented in the most stable manner. Others suggest employing interval arithmetic in some or all sections of the code. The trouble with these choices, from a pragmatic point of view, stems in part from the regrettable fact that very few scientists and engineers who do scientific computation in their work have the requisite advanced training in numerical analysis.

For example, as we pointed out in [5], in 2010 a total of 870 students graduated in mathematics, physics, statistics, chemistry, computer science and engineering at the University of California, Berkeley. Several hundred others, in fields such as biology, geology, medicine, economics, psychology, sociology,

are also likely to do scientific computing in their work. By comparison, in 2010 only 219 students enrolled in two sections of Math 128A, a one-semester introductory numerical analysis course, and only 24 enrolled in Math 128B, a more rigorous numerical analysis course (with similar numbers for previous years). Thus of the 2010 U.C. Berkeley graduates who likely will do scientific computing in their professional work, only about 2% had advanced training in numerical analysis. This percentage does not appear to be any higher at other universities, and may well be lower.

Along this line, a 2008 report issued jointly by the Association for Computing Machinery and the IEEE Computer Society noted that "numerical methods" have been dropped from the recommended curriculum, in recognition of the fact that at most institutions, "numerical analysis is not seen to form a major part of the computer science curriculum" [6].

High-precision arithmetic is thus potentially quite useful in real-world computing, because even in cases where superior algorithms or implementation techniques are known in the literature, simply increasing the precision used for the existing algorithm, using tools such as those described below, is often both simpler and safer, because in most cases only minor changes need to be made to the existing code. And for other computations, as we shall see, there is no alternative to higher precision.

1.3. U.C. Berkeley's "Precimonious" Tool

Recently a team led by James Demmel of U. C. Berkeley has begun developing software facilities to find and ameliorate numerical anomalies in large-scale computations. These include facilities to:

- Test the level of numerical accuracy required for an application.
- Delimit the portions of code that are inaccurate.
- Search the space of possible code modifications.
- Repair numerical difficulties, including usage of high-precision arithmetic.
- Navigate through a hierarchy of precision levels (32-bit, 64-bit or higher as needed).

The current version of this tool is known as "Precimonious." Details are presented in [7].

1.4. Computations that Require Extra Precision

Some scientific computations actually require more than the standard IEEE 64-bit floating-point arithmetic. Typically these applications feature one or more of these characteristics: (a) ill-conditioned linear systems; (b) large summations; (c) long-time simulations; (d) large-scale, highly parallel simulations; (e) high-resolution computations; or (f) experimental mathematics computations.

The following example, adapted from [5], demonstrates how numerical difficulties can arise in a very innocent-looking setting, and shows how these difficulties can be ameliorated, in many cases, by the judicious usage of high-precision arithmetic.

Example 1.2 (Variable precision II)**.** Suppose one suspects that the data $(1, 32771, 262217, 885493, 2101313, 4111751, 7124761)$ are given by an integer polynomial for integer arguments $(0, 1, \ldots, 6)$.

Most scientists and engineers will employ a familiar least-squares scheme to recover this polynomial [8, pg. 44]. This can be done by constructing the $(n+1) \times (n+1)$ system

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

where (x_k) and (y_k) are the arguments and data values given above. This system is typically solved for $(a_0, a_1, \cdots, a_{n-1})$ using *LINPACK* [9] or *LAPACK* [10] software.

An implementation of this approach using standard 64-bit IEEE-754 arithmetic and the LINPACK routines for LU decomposition, with final results rounded to the nearest integer, correctly finds the vector of coefficients $(1, 0, 0, 32769, 0, 0, 1)$, which corresponds to the polynomial function $f(x) = 1 + (2^{15} + 1)x^3 + x^6$. But this scheme fails when given the 9-long sequence $(1, 1048579, 16777489, 84941299, 268501249, 655751251, 1360635409, 2523398179, 4311748609)$, which is generated by the function $f(x) = 1 + (2^{20} + 1)x^4 + x^8$. On a MacPro platform, the resulting rounded coefficient vector is $(1, 6, -16, 14, 1048570, 2, 0, 0, 1)$, which differs from the correct vector $(1, 0, 0, 0, 1048577, 0, 0, 0, 1)$ in five of the nine entries. Similar errors are seen on several other platforms.

Readers trained in numerical analysis may recognize that the above scheme is not the best approach for this problem, because the matrix system is known to be ill-conditioned. A better approach is to employ either a scheme based on the Lagrange interpolating polynomial, or else a scheme due to Demmel and Koev [11] (currently the state-of-the-art for such problems). An implementation of either scheme with 64-bit IEEE-754 arithmetic finds the correct polynomial in the degree-8 case. However, both fail to recover the degree-12 polynomial $1 + (2^{27} + 1)x^6 + x^{12}$ when given the input $(1, 134217731, 8589938753, 97845255883, 549772595201, 2097396156251, 6264239146561, 15804422886323, 35253091827713, 71611233653971, 135217729000001, 240913322581691, 409688091758593)$.

On the other hand, merely by modifying a simple LINPACK program to employ double-double arithmetic, using the QD software (see the next section), all three problems (degrees 6, 8 and 12) are correctly solved without incident.

2. Techniques and Software for High-Precision Arithmetic

By far the most common form of extra-precision arithmetic is roughly twice the level of standard 64-bit IEEE floating-point arithmetic. One option is the IEEE standard for 128-bit binary floating-point arithmetic, with 113 mantissa bits, but sadly it is not yet widely implemented in hardware. However, this datatype is now supported in software by the gfortran compiler, for both real and complex data.

A more common option, implemented in software, is known as "double-double" arithmetic (approximately 31-digit accuracy). This datatype consists of a pair of 64-bit IEEE floats (s, t), where s is the 64-bit floating-point value closest to the desired value, and t is the difference (positive or negative) between the true value and s. Arithmetic on such data can be performed using schemes originally

proposed by Donald E. Knuth [12] (see also [13]). Similar techniques can be used for "quad-double" (approximately 62-digit) arithmetic. These techniques have been implemented in the QD package [14].

For higher-levels of precision, one typically represents a high-precision datum as a string of floats or integers, where the first few words contain bookkeeping and exponent information, and each subsequent word contains B bits of the mantissa. For moderate precision levels (up to a few hundred digits), arithmetic on such data is typically performed using adaptations of familiar schemes.

Above several hundred decimal digits, advanced algorithms, such as Karatsuba multiplication [15, pp. 222–223] or (for larger precision levels) FFT-based multiplication [15, pp. 223–224] should be used for maximum efficiency. FFT-based convolution is done as follows. Let the two high-precision data vectors (without exponents and other "bookkeeping" words) be $x = (x_0, x_1, \cdots, x_{n-1})$ and $y = (y_0, y_1, \cdots, y_{n-1})$, where each word contains B bits, i.e., an integer from 0 to $2^B - 1$. First extend x and y to length $2n$ by appending n zeroes to each. Then compute the vector $z = (z_0, z_1, \cdots z_{2n-1})$ as

$$z_k = \sum_{k=0}^{2n-1} x_k y_{2n-k-1}. \tag{1}$$

Finally, release carries, which is done by iterating $z_k := z_k + \mathrm{int}(z_{k+1}/2^B)$, beginning with $k = 2n - 2$ and proceeding in reverse order to $k = 0$. The result is the $2n$-long mantissa of the high-precision product of x and y. After inserting appropriate sign and exponent words and rounding the result if needed to n mantissa words, the operation is complete.

The convolution operation (1) may be performed using fast Fourier transforms (FFTs):

$$z = F^{-1}[F[x] \cdot F[y]] \tag{2}$$

It should be noted in the above that it is often necessary to limit B, so that the final results of the floating-point FFT operations can be reliably rounded to the nearest integer. Another option is to use a number-theoretic FFT, which is not subject to round-off error. Either way, efficient implementations of this scheme can dramatically accelerate multiplication for very high-precision data, since in effect it reduces an $O(n^2)$ operation to an $O(n \log n \log \log n)$ operation [16, Sec. 2.3].

Square roots and n-th roots can be computed efficiently by Newton iteration-based schemes. The basic transcendental functions can be computed by means of Taylor series-based algorithms or, for higher levels of precision, quadratically convergent algorithms that approximately double the number of correct digits with each iteration, [15, pp. 215–245] and [16, Sec. 1.5].

Software for performing high-precision arithmetic has been available for quite some time, for example in the commercial packages *Mathematica* and *Maple*. However, until 10 or 15 years ago those with applications written in more conventional languages, such as C++ or Fortran-90, often found it necessary to rewrite their codes, replacing each arithmetic operation with a subroutine call, which was a very tedious and error-prone process.

Nowadays there are several freely available high-precision software packages, together with accompanying high-level language interfaces, utilizing operator overloading, that make code conversions relatively painless. In most cases, one merely changes the type statements of those variables that are to be treated as high precision and makes a handful of other modifications. Thereafter when one of these variables appears in an expression, the correct underlying routines are automatically called.

Here are a few currently available packages for high-precision floating-point arithmetic:

- ARPREC: Supports arbitrary precision real, integer and complex, with many algebraic and transcendental functions. Includes high-level interfaces for C++ and Fortran-90 [17].
- CLN: A C++ library supporting arbitrary precision integer, real and complex, with numerous algebraic and transcendental functions [18].
- GMP: Supports high-precision integer, rational and floating-point calculations. Distributed under the GNU license by the Free Software Foundation [19].
- Julia: A high-level programming environment that incorporates GMP and MPFR [20].
- MPFR: Supports multiple-precision floating-point computations with correct rounding, based on GMP [21].
- MPFR++: A high-level C++ interface to MPFR [22].
- MPFR C++: A thread-safe high-level C++ interface to MPFR [23]. See Section 9.1.
- MPFUN2015: A new thread-safe system supporting multiprecision real and complex datatypes, with transcendental functions and FFT-based arithmetic. Includes high-level Fortran interface [24]. See Section 9.1.
- mpmath: A Python library for arbitrary precision floating-point arithmetic, including numerous transcendentals [25].
- NTL: A C++ library for arbitrary precision integer and floating-point arithmetic [26].
- Pari/GP: A computer algebra system that includes facilities for high-precision arithmetic, with many transcendental functions [27].
- QD: Supports "double-double" (roughly 31 digits) and "quad-double" (roughly 62 digits) arithmetic, as well as common algebraic and transcendental functions. Includes high-level interfaces for C++ and Fortran-90 [14].
- Sage: An open-source mathematical software system that includes high-precision arithmetic facilities [28].

Obviously there is an extra cost for performing high-precision arithmetic. Quad precision or double-double computations typically run five to ten times slower than 64-bit; oct precision or quad-double computations typically run 25 to 50 times slower; and arbitrary precision arithmetic may be hundreds or thousands of times slower. Fortunately, however, high-precision arithmetic is often only needed for a small portion of code, so that the total run time may increase only modestly.

What's more, the advent of highly parallel computer systems means that high-precision computations that once were impractical can now be completed in reasonable run time. Indeed, numerous demanding high-precision computations have been done in this way, most often by invoking parallelism at the level of loops in the application rather than within individual high-precision arithmetic operations. For

example, up to 512 processors were employed to compute some two- and three-dimensional integrals in [29]. Parallel high-precision computation will be further discussed in Section 9.

3. Applications of High-Precision Arithmetic

Here we briefly summarize some of the numerous applications that have recently arisen for high-precision arithmetic in conventional scientific computing. Sections 3.4 through 4.1 are condensed from [5]; see [5] for additional details and examples.

3.1. Optimization Problems

In 2014, well-known optimization theorist Michael Saunders of Stanford University and his colleague Ding Ma explored the usage of quadruple precision, using the real*16 facility of the gfortran compiler, to address problems in optimization. They found that they were able to find numerically stable solutions to numerous problems, including several in mathematical biology, that had eluded other researchers for years. Saunders concluded, "Just as double-precision floating-point hardware revolutionized scientific computing in the 1960s, the advent of quad-precision data types (even in software) brings us to a new era of greatly improved reliability in optimization solvers." [30,31].

3.2. Computer-Assisted Solution of Smale's 14th Problem

In 2002, Warwick Tucker of Cornell University employed a combination of interval arithmetic and 80-bit IEEE extended arithmetic (approx. 17-digit precision) to prove that the Lorenz equations support a "strange" attractor. Berkeley mathematician Steven Smale had previously included this as one of the most challenging problems for the 21st century [32].

3.3. Anharmonic Oscillators

Malcolm H. Macfarlane of Indiana University has employed up to 80-digit arithmetic to obtain numerically reliable eigenvalues for a class of anharmonic oscillators, using a shifted-Lanczos method. Anharmonicity plays a key role in studies of molecular vibrations, quantum oscillations and semiconductor engineering. High precision not only handled the highly singular linear systems involved, but also prevented convergence to the wrong eigenvalue [33].

3.4. Planetary Orbit Dynamics

Planetary scientists have long debated whether the solar system is stable over many millions or billions of years, and whether our solar system is typical in this regard. Researchers running simulations of the solar system typically find that 64-bit computations are sufficiently accurate for long periods of time, but then fail at certain critical points. As a result, some researchers have employed higher precision arithmetic (typically IEEE extended or double-double) to reduce numerical error, in combination with other techniques [34,35].

3.5. Coulomb n-Body Atomic Systems

Alexei Frolov of the Western University (formerly the University of of Western Ontario) in Canada has employed high-precision software in studies of n-body Coulomb atomic systems, which require solutions of large and nearly singular linear systems. His computations typically use 120-digit arithmetic, which is sufficient to solve certain bound state problems that until recently were consider intractable [36].

3.6. Nuclear Physics Computations

Researchers have applied high-precision computations to produce accurate solutions to the Schrödinger equation (from quantum theory) for the lithium atom. In particular, they have been able to compute the non-relativistic ground state energy for lithium to nearly 12-digit accuracy, which is 1500 times better than earlier results [37]. These researchers are now targeting a computation of the fine structure constant of physics (approx. $7.2973525698 \times 10^{-3}$), hopefully to within a few parts per billion, directly from the QED theory [38].

3.7. Scattering Amplitudes

A team of physicists working on the Large Hadron Collider at CERN (distinct from the team mentioned in Section 1) is computing scattering amplitudes of collisions involving various fundamental particles (quarks, gluons, bosons). This program performs computations using 64-bit IEEE floating-point arithmetic in most cases, but then recomputes results where necessary using double-double arithmetic. These researchers are now designing a scheme that dynamically varies the precision level according to the inherent stability of the underlying computation [39–42].

3.8. Zeroes of the Riemann Zeta Function

In a recent study by Gleb Beliakov of Deakin University in Australia and Yuri Matiyasevich of the Steklov Institute of Mathematics in Russia, very high precision (up to 10,000-digit arithmetic) was employed to compute determinants and minors of very large and ill-conditioned matrices that arise in the analysis of the zeroes of the Riemann zeta function. Their codes employed graphics processing units (GPUs) and a parallel cluster (up to 144 CPUs) [43].

4. Dynamical Systems

High-precision arithmetic has long been employed in the study of dynamical systems, especially in the analysis of bifurcations and periodic orbit stability. Many of these computations have employed Runge-Kutta methods, but in the past few years researchers have found that the Taylor method, implemented with high-precision arithmetic, is even more effective [44].

The Taylor method can be defined as follows [45–47]. Consider the initial value problem $\dot{y} = f(t,\, y)$, where f is assumed to be a smooth function. Then the solution at t_i is approximated by y_i from the n-th degree Taylor series approximation of $y(t)$ at $t = t_i$. So, denoting $h_i = t_i - t_{i-1}$, one can write

$$\begin{aligned} y(t_0) &=: y_0, \\ y(t_i) &\approx y_{i-1} + f(t_{i-1}, y_{i-1})\, h_i + \ldots \\ &\quad + \frac{1}{n!}\, \frac{d^{n-1} f(t_{i-1}, y_{i-1})}{dt^{n-1}}\, h_i^n \;=:\; y_i. \end{aligned} \tag{3}$$

This method thus reduces the solution of the dynamical system to the determination of certain Taylor coefficients, which may be done efficiently using automatic differentiation techniques, provided one uses sufficiently accurate arithmetic [45,46,48,49].

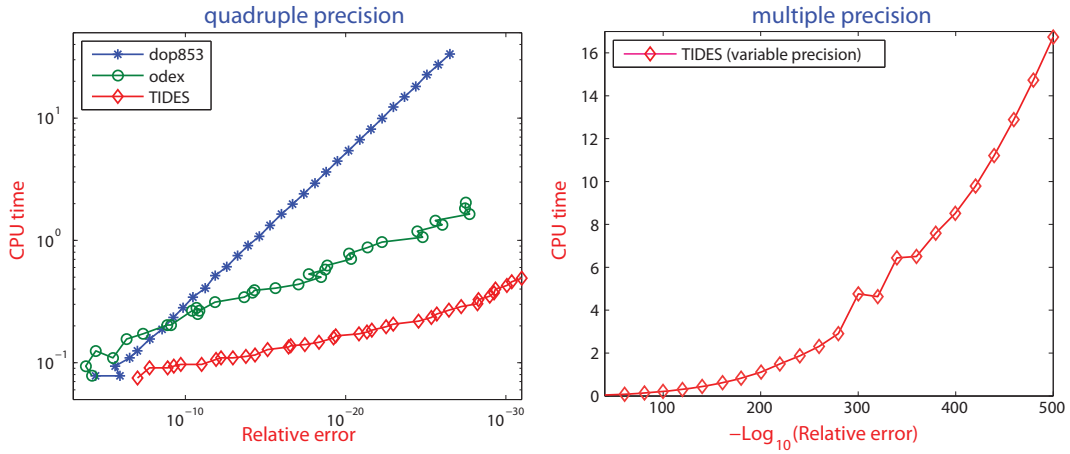

Figure 2. Left: Error vs. CPU time for the numerical solution of the unstable periodic orbit LR for the Lorenz model (in double-double arithmetic) using a Runge-Kutta code (dop853), an extrapolation code (odex) [50,51] and a Taylor series method (TIDES). Right: Error vs. CPU time for the numerical solution of an unstable periodic orbit for the Lorenz model (in 500-digit arithmetic) using the TIDES code. Taken from [52].

Figure 2 (left) compares computations of the unstable periodic orbit LR in the Lorenz model [53] with three methods: (a) the Taylor method (using the TIDES code [54]), (b) an eighth order Runge-Kutta method (using the dop853 code [55]), and (c) an extrapolation method (using the ODEX code) [50,51]. In symbolic dynamics notation, "LR" means one loop around the left equilibrium point, followed by one loop around the right equilibrium point.

When these methods are performed using double precision arithmetic, the Runge-Kutta code is most accurate, but when done in double-double or higher-precision arithmetic, the Taylor method is the fastest. In many problems, high-precision arithmetic is required to obtain accurate results, and so for such problems the Taylor scheme is the only reliable method among the standard methods. The right graph in Figure 2 shows the error for an unstable orbit using 500-digit arithmetic using the TIDES code [52].

Figure 3 demonstrates the need for high accuracy in these solutions. It shows the results for completing 16 time periods of the $\mathrm{L}^{25}\mathrm{R}^{25}$ unstable periodic orbit of the Lorenz model using the the

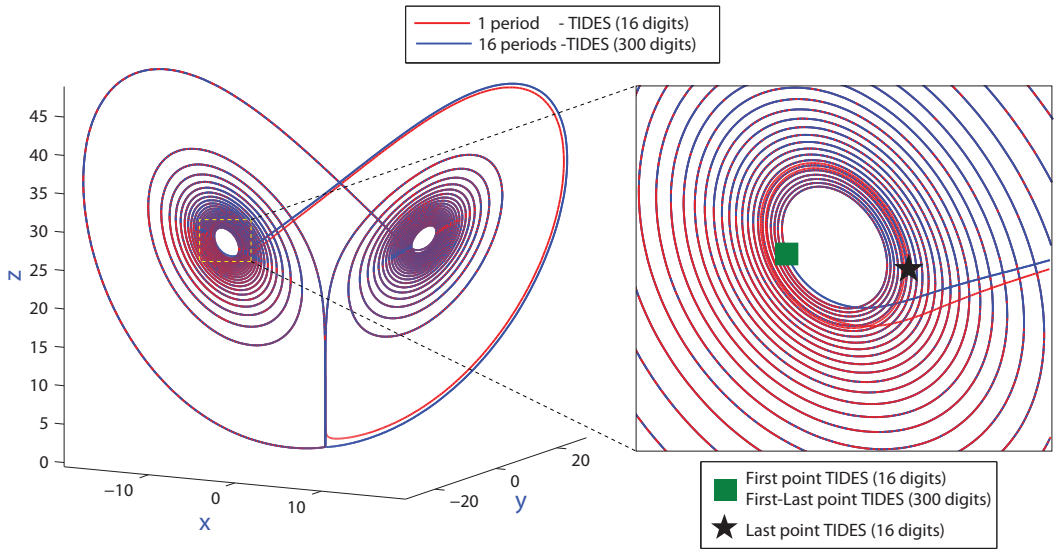

Figure 3. Numerical solution of the $L^{25}R^{25}$ unstable periodic orbit of the Lorenz model for 16 time periods using the TIDES code with 300 digits, compared with just one time period using 64-bit IEEE arithmetic.

TIDES code with 300 digits, compared with just one time period using IEEE 64-bit floating-point arithmetic. Note that since more than 16 digits are lost on each period, it is not possible to accurately compute even a single period of this orbit using only IEEE 64-bit floating-point arithmetic.

4.1. Periodic orbits

The Lindstedt-Poincaré method [56] for computing periodic orbits in dynamical systems is based on the Lindstedt-Poincaré technique of perturbation theory, together with Newton's method for solving nonlinear systems and Fourier interpolation. D. Viswanath [57] has used this method, implemented in part with high-precision arithmetic software (more than 100 digits of precision) to compute highly unstable periodic orbits. These calculations typically may lose more than 50 digits of precision in each orbit. The fractal properties of the Lorenz attractor, which require very high precision to compute, are shown in Figure 4 [57,58].

Another option currently being pursued by researchers computing these periodic orbits is to employ the Taylor series method in conjunction with Newton iterations, together with more than 1000-digit arithmetic [59].

To conclude this section, we mention a fascinating article [60], which describes how simulations must employ quantum molecular dynamics and special relativity to obtain realistic results on the element Mercury's properties that correspond to measured properties (notably its melting temperature). In arenas such as this, one must first accurately capture the physics, but one should also remember the option to parsimoniously increase precision when needed.

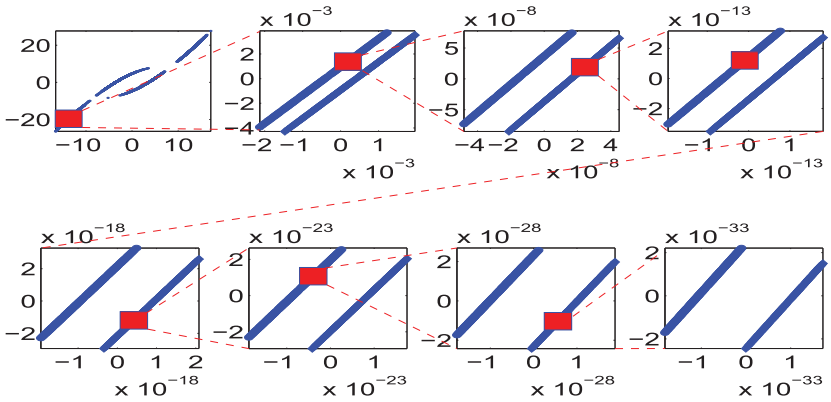

Figure 4. Fractal property of the Lorenz attractor. The first plot shows a rectangle in the plane. All later plots zoom in on a tiny region (too small to be seen by the unaided eye) at the center of the red rectangle of the preceding plot to show that what appears to be a line is in fact not a line. (Reproduced with permission from [58]).

5. The PSLQ Algorithm and Experimental Mathematics

Very high-precision floating-point arithmetic is now considered an indispensable tool in experimental mathematics, which in turn is increasingly being employed in mathematical physics [15,61].

Many of these computations involve variants of Ferguson's PSLQ integer relation detection algorithm [62,63]. Suppose one is given an n-long vector (x_i) of real or complex numbers (presented as a vector of high-precision values). The PSLQ algorithm finds the integer coefficients (a_i), not all zero, such that

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

(to available precision), or else determines that there is no such relation within a certain bound on the size of the coefficients. Integer relation detection almost always requires very high precision—at least $(n \times d)$-digit precision, where d is the size in digits of the largest a_i and n is the vector length, or else the true relation, if one exists, will be lost in a sea of spurious numerical artifacts.

PSLQ constructs a sequence of integer-valued matrices B_n that reduce the size of the vector $y = x \cdot B_n$, until either the relation is found (as one of the columns of matrix B_n), or else numeric precision is exhausted. A relation is detected when the size of smallest entry of the y vector suddenly drops to zero or to less than the "epsilon" of the high-precision arithmetic system being employed (i.e. b^{-p}, where b is the number base, typically 2 or 10, and p is the number of digits of precision). The size of the drop in $\min(|y_i|)$ at the iteration when the relation is detected can be viewed as a confidence level that the relation so discovered is not a numerical artifact. A drop of 20 or more decimal orders of magnitude almost always indicates a real relation (although a numerical calculation such as this cannot be interpreted as rigorous proof that the relation is mathematically correct). See Figure 5.

Two- and three-level variants of PSLQ are known, which perform almost all iterations with only double or intermediate precision, updating full-precision arrays only as needed. These variants are hundreds of times faster than the original PSLQ. A multipair variant of PSLQ is known that dramatically reduces the number of iterations required and is thus well-suited to parallel systems. As a bonus, it runs faster even on one CPU. Finally, two- and three-level versions of multipair PSLQ are now available,

which combine the best of both worlds [63]. Most of our computations have been done with a two-level multipair PSLQ program.

Figure 5, which illustrates a typical multipair PSLQ run, shows the abrupt drop in $\min |y_i|$ (by nearly 200 orders of magnitude in this case) at iteration 199, when the relation was detected.

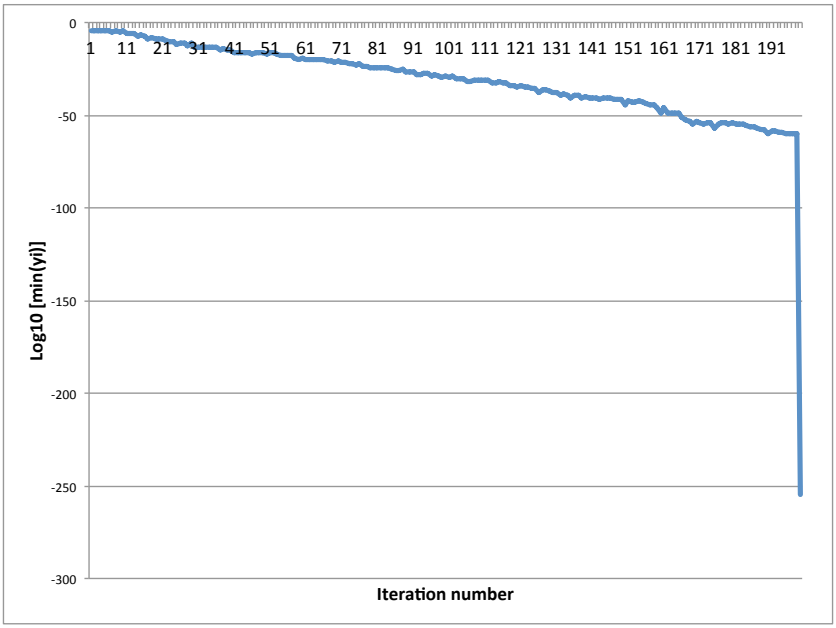

Figure 5. Plot of $\log_{10}(\min |y_i|)$ versus iteration number in a typical multipair PSLQ run. Note the sudden drop, by nearly 200 orders of magnitude, at iteration 199.

The problem of finding an integer relation in a vector of real numbers is closely related to the problem of finding a small vector in a lattice. Indeed, another widely used integer relation algorithm, the "HJLS" algorithm, is based on the Lenstra-Lenstra-Lovasz (LLL) algorithm for lattice basis reduction. Both PSLQ and HJLS can be viewed as schemes to compute the intersection between a lattice and a vector subspace. For details, see [64].

5.1. The BBP Formula for π and Normality

One of the earliest applications of PSLQ was to numerically discover what is now known as the "BBP" formula for π:

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

This remarkable formula, after a simple manipulation, can be used to calculate binary or base-16 digits of π beginning at the n-th digit, without needing to calculate any of the first $n-1$ digits. The underlying scheme requires very little memory and no multiple-precision arithmetic software [15, pp. 135–143].

Since 1996, when the BBP formula for π was discovered, numerous other formulas of this type have been found using PSLQ and then subsequently proven [65]. For example, a binary formula has been found for Catalan's constant $G = \sum_{n \geqslant 0} (-1)^n/(2n+1)^2$, and both binary and ternary (base-3) formulas have been found for π^2 [66].

These formulas have been used to compute digits that until just a decade or two ago were widely regarded as forever beyond the reach of humankind. For example, on March 14 (Pi Day), 2013, Ed Karrels of Santa Clara University computed 26 base-16 digits of π beginning at position one quadrillion [67]. His result: `8353CB3F7F0C9ACCFA9AA215F2`. In 2011, researchers at IBM Australia employed the formulas mentioned above to calculate base-64 digits of π^2, base-729 digits of π^2 and base-4096 digits of G, beginning at the ten trillionth position in each case. These results were then validated by independent computations [66].

At first, many regarded these PSLQ-discovered results to be amusing but of no deep mathematical significance. But then it was found that these BBP-type formulas have connections to the ancient (and still unanswered) question of why the digits of π and certain other related constants appear "random."

To be precise, a constant α is said to be 2-normal, or normal base-2, if it has the property that every string of m base-2 digits appears, in the limit, with frequency 2^{-m}, and similarly in other bases. Establishing normality (in any base) for various well-known mathematical constants, such as π, e or $\log 2$, is a surprisingly difficult problem. Rigorous results are known only for a few special constants, such as Champernowne's number $(0.12345678910111213141516\ldots)$ (produced by concatenating the decimal integers), certain Liouville numbers [68] and Stoneham numbers.

In 2002, Richard Crandall (deceased December 20, 2012) and one of the present authors found that the question of whether constants such as π and $\log 2$ are 2-normal reduces to a conjecture about the behavior of certain pseudorandom number generators derived from the associated BBP-type formulas [15, pp. 163–178]. This same line of research subsequently led to a proof that an uncountably infinite class of real numbers are 2-normal [15, pp. 148–154], including, for instance,

$$\alpha_{2,3} \quad = \quad \sum_{n=0}^{\infty} \frac{1}{3^n 2^{3^n}}.$$

(This particular constant was proven 2-normal by Stoneham in 1973 [69]; the more recent results span a much larger class.) Although this constant is provably 2-normal, interestingly it is provably *not* 6-normal [70]. This is the first concrete example of a number normal in one base but not normal in another.

6. High-Precision Arithmetic in Mathematical Physics

Very high-precision computations, combined with variants of the PSLQ algorithm, have been remarkably effective in resolving certain classes of definite integrals that arise in mathematical physics settings. We summarize here a few examples of this methodology in action, following [5] and some related references as shown below.

6.1. Ising Integrals

In one study, the tanh-sinh quadrature scheme [61, pp. 312–315], [71], implemented using the ARPREC high-precision software, was employed to study the following classes of integrals [29]. The

C_n are connected to quantum field theory, the D_n integrals arise in the Ising theory of mathematical physics, while the E_n integrands are derived from D_n:

$$C_n = \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$D_n = \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j}\left(\frac{u_i-u_j}{u_i+u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$E_n = 2\int_0^1 \cdots \int_0^1 \left(\prod_{1\leqslant j<k\leqslant n} \frac{u_k - u_j}{u_k + u_j}\right)^2 \mathrm{d}t_2\, \mathrm{d}t_3 \cdots dt_n.$$

In the last line $u_k = \prod_{i=1}^k t_i$.

In general, it is very difficult to compute high-precision numerical values of n-dimensional integrals such as these. But as it turn out, the C_n integrals can be converted to one-dimensional integrals, which are amenable to evaluation with the tanh-sinh scheme:

$$C_n = \frac{2^n}{n!}\int_0^\infty pK_0^n(p)\,\mathrm{d}p.$$

Here K_0 is the *modified Bessel function* [72,73] and it is in this form that C_N appears in quantum field theory. Now 1000-digit values of these sufficed to identify the first few instances of C_n in terms of well-known constants. For example, $C_4 = 7\zeta(3)/12$, where ζ denotes the Riemann zeta function. For larger n, it quickly became clear that the C_n approach the limit

$$\lim_{n\to\infty} C_n = 0.630473503374386796122040192710\ldots.$$

This numerical value was quickly identified, using the *Inverse Symbolic Calculator 2.0* [74], as

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

where γ is Euler's constant. This identity was then proven [29]. Some results were also obtained for the D_n and E_n integrals, although the numerical calculations involved there were much more expensive, requiring highly parallel computation [29]. In 2013 Erik Panzer was able to formally evaluate all the E_n in terms of the *alternating multizeta* values [75]. In particular, our purely experimental result for E_5 was confirmed, and our high precision computation of E_6 was used to confirm Panzer's evaluation.

6.2. Ramble Integrals

A separate study considered n-dimensional *ramble integrals* [76]

$$W_n(s) = \int_{[0,1]^n} \left|\sum_{k=1}^n e^{2\pi x_k i}\right|^s \mathrm{d}x, \tag{4}$$

for complex s. These integrals occur in the theory of uniform random walk integrals in the plane, where at each iteration, a step of length one is taken in a random direction. Integrals such as (4) are the s-th

moment of the distance to the origin after n steps. In [77], it is shown that when $s = 0$, the first derivatives of these integrals, which arise independently in the study of Mahler measures, are

$$\begin{aligned} W_n'(0) &= \log(2) - \gamma - \int_0^1 (J_0^n(x) - 1)\,\frac{\mathrm{d}x}{x} - \int_1^\infty J_0^n(x)\frac{\mathrm{d}x}{x} \\ &= \log(2) - \gamma - n\int_0^\infty \log(x) J_0^{n-1}(x) J_1(x)\mathrm{d}x. \end{aligned}$$

Here $J_n(x)$ is the Bessel function of the first kind [72,73].

These integrand functions, which are highly oscillatory, are very challenging to evaluate to high precision. We employed tanh-sinh quadrature and Gaussian quadrature, together with the Sidi mW extrapolation algorithm, as described in a 1994 paper by Lucas and Stone [78], which, in turn, is based on two earlier papers by Sidi [79,80]. While the computations were relatively expensive, we were able to compute 1000-digit values of these integrals for odd n up to 17 [76]. Unfortunately, this scheme does not work well for even n, but by employing a combination of symbolic and numeric computation we were able to obtain 50 to 100 good digits.

In the wake of this research, Sidi produced a refined method [81] that can be used for even n.

6.3. Moments of Elliptic Integral Functions

The analysis of ramble integrals led to a study of moments of integrals involving products of elliptic integral functions [76]:

$$I(n_0, n_1, n_2, n_3, n_4) = \int_0^1 x^{n_0} K^{n_1}(x) K'^{n_2}(x) E^{n_3}(x) E'^{n_4}(x)\mathrm{d}x. \tag{5}$$

Here the elliptic functions K, E and their complementary versions are given by:

$$\begin{aligned} K(x) &= \int_0^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-x^2t^2)}} \\ K'(x) &= K(\sqrt{1-x^2}) \\ E(x) &= \int_0^1 \frac{\sqrt{1-x^2t^2}}{\sqrt{1-t^2}}\,\mathrm{d}t \\ E'(x) &= E(\sqrt{1-x^2}). \end{aligned}$$

These and related expressions arise in a variety of physical contexts [82]. We computed over 4000 individual integrals of this form to between 1500 and 3000-digit precision using the ARPREC software. In most cases we were *not* able to identify these integrals in closed form. But we did find many interesting

relations *among* these integrals, using a high-precision multipair PSLQ program. Two of the conjectured identities found by the multipair PSLQ program are [76]:

$$81\int_0^1 x^3K^2(x)E(x)\mathrm{d}x \stackrel{?}{=} -6\int_0^1 K^3(x)\mathrm{d}x - 24\int_0^1 x^2K^3(x)\mathrm{d}x + 51\int_0^1 x^3K^3(x)\mathrm{d}x + 32\int_0^1 x^4K^3(x)\mathrm{d}x, \tag{6}$$

$$-243\int_0^1 x^3K(x)E(x)K'(x)\mathrm{d}x \stackrel{?}{=} -59\int_0^1 K^3(x)\mathrm{d}x + 468\int_0^1 x^2K^3(x)\mathrm{d}x + 156\int_0^1 x^3K^3(x)\mathrm{d}x - 624\int_0^1 x^4K^3(x)\mathrm{d}x - 135\int_0^1 xK(x)E(x)K'(x)\mathrm{d}x. \tag{7}$$

James Wan [83] has been able to prove many of these identities, including (6), but by no means all. A followup work [84] contains remarkable identities, such as

$$\int_0^1 E(k)K'(k)^2\mathrm{d}k = \frac{\pi^3}{12} + \frac{\Gamma^8(\frac{1}{4})}{384\pi^2} = \frac{\pi^3}{12} + \frac{2}{3}K^4\left(\frac{1}{\sqrt{2}}\right),$$

for integrands which are three-fold products of elliptic integrals.

7. Lattice Sums Arising from a Poisson Equation

In this section we describe some results, just completed [85], that arose out of attempts to solve the Poisson equation, which arises in various contexts such as engineering applications, the analysis of crystal structures, and even the sharpening of photographic images.

The following lattice sums arise as values of basis functions in the Poisson solutions [85]:

$$\phi_n(r_1,\ldots,r_n) = \frac{1}{\pi^2}\sum_{m_1,\ldots,m_n \text{ odd}} \frac{e^{i\pi(m_1r_1+\cdots+m_nr_n)}}{m_1^2+\cdots+m_n^2}. \tag{8}$$

By noting some striking connections with Jacobi ϑ-function values, Crandall, Zucker and the present authors were able to develop new closed forms for certain values of the arguments r_k [85].

Computing high-precision values of such sums was facilitated by deriving formulas such as

$$\phi_2(x,y) = \frac{1}{4\pi}\log\frac{\cosh(\pi x)+\cos(\pi y)}{\cosh(\pi x)-\cos(\pi y)} - \frac{2}{\pi}\sum_{m\in\mathbb{O}^+}\frac{\cosh(\pi m x)\cos(\pi m y)}{m(1+e^{\pi m})}, \tag{9}$$

where $\mathbb{O}^+$ means positive odd integers, which is valid for $x, y \in [-1, 1]$.

After extensive high-precision numerical experimentation using (9), we discovered (then proved) the remarkable fact that when x and y are rational,

$$\phi_2(x,y) = \frac{1}{\pi}\log A, \tag{10}$$

where A is an *algebraic number*, namely the root of an algebraic equation with integer coefficients.

In this case we computed $\alpha = A^8 = \exp(8\pi\phi_2(x,y))$ (as it turned out, the '8' substantially reduces the degree of polynomials and so computational cost), and then generated the vector $(1, \alpha, \alpha^2, \cdots, \alpha^d)$, which, for various conjectured values of d, was input to the multipair PSLQ program. When successful,

Table 1. Minimal polynomials found by PSLQ computations

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 program returned the vector of integer coefficients $(a_0, a_1, a_2, \cdots, a_d)$ of a polynomial satisfied by α as output. With some experimentation on the degree d, and after symbolic verification using *Mathematica*, we were able to ensure that the resulting polynomial is in fact the minimal polynomial satisfied by α. Table 1 shows some examples of the minimal polynomials discovered by this process [85].

Using this data, we were able to conjecture a formula that gives the degree d as a function of k [85]. These computations required prodigiously high precision: up to 20,000-digit floating-point arithmetic in some cases, such as in finding the degree-128 polynomial satisfied by $\alpha = \exp(8\pi\phi_2(1/32, 1/32))$. Related computations in a follow-up study required up to 50,000-digit arithmetic.

8. Integrals Arising in the Study of Wigner Electron Sums

In this section, we report some new results on "Wigner electron sums," which are discussed in [86]. Throughout this section, $Q(x) = Q(x_1, \ldots, x_d)$ is a positive definite quadratic form in d variables with real coefficients and determinant $\Delta > 0$.

As proposed in [87, Chap. 7], the paper [86] examined the behavior of

$$\sigma_N(s) := \alpha_N(s) - \beta_N(s)$$

as $N \to \infty$, where α_N and β_N are given by

$$\alpha_N(s) := \sum_{n_1=-N}^{N} \cdots \sum_{n_d=-N}^{N} \frac{1}{Q(n_1, \ldots, n_d)^s}, \tag{11}$$

$$\beta_N(s) := \int_{-N-1/2}^{N+1/2} \cdots \int_{-N-1/2}^{N+1/2} \frac{\mathrm{d}x_1 \cdots \mathrm{d}x_d}{Q(x_1, \ldots, x_d)^s}. \tag{12}$$

As usual, the summation in (11) is understood to avoid the term corresponding to $(n_1, \ldots, n_d) = (0, \ldots, 0)$. If $\operatorname{Re} s > d/2$, then $\alpha_N(s)$ converges to the Epstein zeta function $\alpha(s) = Z_Q(s)$ as $N \to \infty$. On the other hand, each integral $\beta_N(s)$ is only defined for $\operatorname{Re} s < d/2$.

A priori it is unclear, for any s, whether the limit $\sigma(s) := \lim_{N\to\infty} \sigma_N(s)$ should exist. In what follows, we will write $\sigma(s)$ for the limit. The infinite sums and integrals (which never both converge) were introduced by Wigner eighty years ago to offer an alternative way of viewing crystal sums within quantum theory, but rigorous mathematical analysis is much more recent see [86].

In the case $d = 2$, it was shown in [88, Theorem 1] that the limit $\sigma(s)$ exists in the strip $0 < \operatorname{Re} s < 1$ and that it coincides therein with the analytic continuation of $\alpha(s)$. Further, in the case $d = 3$ with $Q(x) = x_1^2 + x_2^2 + x_3^2$, it was shown in [88, Theorem 3] that the limit $\sigma(s) := \lim_{N\to\infty} \sigma_N(s)$ exists for $1/2 < \operatorname{Re} s < 3/2$ as well as for $s = 1/2$. However, it was proven that $\sigma(1/2) - \pi/6 = \lim_{\varepsilon\to 0^+} \sigma(1/2 + \varepsilon)$. In other words, the limit $\sigma(s)$ exhibits a jump discontinuity at $s = 1/2$.

It is therefore natural to ask in what senses the phenomenon observed for the cubic lattice when $d = 3$ extends both to higher dimensions and to more general quadratic forms. Theorem 8.1 below extends this result to arbitrary positive definite Q in all dimensions.

8.1. Jump Discontinuities in Wigner Limits

As above, let $Q(x) = Q_A(x) = \sum_{1\leqslant i,j\leqslant d} a_{ij}x_ix_j$, with $A = (a_{ij})_{1\leqslant i,j\leqslant d}$ symmetric and positive definite. Set also $B(s) = \operatorname{tr}(A)A - 2(s+1)A^2$. Finally, define

$$V_Q(s) := \int_{\|x\|_\infty=1} \frac{Q_{B(s)}(x)}{Q_A(x)^{s+2}} \mathrm{d}\lambda_{d-1}, \tag{13}$$

with λ_{d-1} the induced $(d-1)$-dimensional measure on the faces.

Theorem 8.1 (General jump discontinuity [86])**.** *Let Q be an arbitrary positive definite quadratic form. Then the limit $\sigma(s) := \lim_{N\to\infty} \sigma_N(s)$ exists in the strip $d/2 - 1 < \operatorname{Re} s < d/2$ and for $s = d/2 - 1$. In the strip, $\sigma(s)$ coincides with the analytic continuation of $\alpha(s)$. On the other hand,*

$$\begin{aligned} &\sigma(d/2-1) + \frac{d/2-1}{24} V_Q'(d/2-1) \\ &\quad = \alpha(d/2-1) = \lim_{\varepsilon\to 0^+} \sigma(d/2-1+\varepsilon), \end{aligned} \tag{14}$$

with V_Q as introduced in equation (13).

8.2. The Behavior of $V_Q'(d/2-1)$

We now examine the nature of $V_Q'(d/2-1)$ in somewhat more detail. From the definition (13) we obtain that

$$\begin{aligned} V_Q'(d/2-1) &= -\frac{4\operatorname{tr}(A)}{d\sqrt{\det(A)}} \frac{\pi^{d/2}}{\Gamma(d/2)} \\ &- \int_{\|x\|_\infty=1} \frac{\operatorname{tr}(A)Q_A(x) - dQ_{A^2}(x)}{Q_A(x)^{d/2+1}} \log Q_A(x) \mathrm{d}\lambda_{d-1}. \end{aligned} \tag{15}$$

The equality is a consequence of [86, Lemma 2.5].

Theorem 8.1 reduces to the result given in [86] in the cubic lattice case. In that case, $A = I$ and $\mathrm{tr}(A) = d$, so the integral in (15), involving the logarithm, vanishes. Hence,

$$V'(d/2-1) = -4\frac{\pi^{d/2}}{\Gamma(d/2)},$$

in agreement with the value given in [86].

In [86] the question was *not* settled of whether $V'_Q(d/2-1)$ can possibly vanish for some positive definite quadratic form Q. However, a simple criterion for $V'_Q(d/2-1) < 0$ was given.

Proposition 8.2 ([86]). *Let Q be a positive definite quadratic form with*

$$d\,Q_{A^2}(x) \leqslant \mathrm{tr}(A) Q_A(x) \tag{16}$$

for all x with $\|x\|_\infty = 1$. Then $V'_Q(d/2-1) < 0$. The same conclusion holds if '$\leqslant$' is replaced with '$\geqslant$' in (16).

Proposition 8.2 can fail comprehensively if its conditions are not fully met [86]. Nonetheless, on the basis of our analysis, motivated by cases where Proposition 8.2 does not apply, we were led to the physically attractive conjecture below.

Conjecture 8.3 (Negative jumps). *For all dimensions d and all positive definite forms Q, one has $V'_Q(d/2-1) < 0$.*

Truth of the conjecture would provide confidence to try to resolve it by proving that $V_Q(s)$ is sufficiently strongly decreasing for all $s > 0$.

8.3. Numerical Exploration of $V'_Q(d/2-1)$

In an attempt to better understand Conjecture 8.3, we employed double-double computations to test the inequality $V'(d/2-1) < 0$ (what follows are new results for this paper). We first observed that the integral in (15) decomposes as $2d$ integrals of the form

$$v_i(\pm) := \int_{\substack{\|x\|_\infty \leq 1,\\ x_i = \pm 1}} \frac{dQ_{A^2}(x) - \mathrm{tr}(A) Q_A(x)}{Q_A(x)^{d/2+1}} \log Q_A(x)\, \mathrm{d}\lambda_{d-1}$$

for $1 \leq i \leq d$ over $(d-1)$-dimensional hypercubes. The task at hand is to explore whether the inequality

$$\sum_{i=1}^{d} v_i(+) + \sum_{i=1}^{d} v_i(-) < \frac{4\,\mathrm{tr}(A)}{d\sqrt{\det(A)}} \frac{\pi^{d/2}}{\Gamma(d/2)} \tag{17}$$

can ever fail.

As the dimension grows, the cost of the required numerical integration increases substantially, as noted above in Section 6.1. Thus, we opted only to computationally examine whether t*he ratio* $L/R = \mathrm{LHS}/\mathrm{RHS}$ *in* (17) *is always less than one for* $d = 3$, which is the physically most meaningful case of Conjecture 8.3.

Table 2. Computational results and run times for tests of Conjecture 8.3.

n	L/R ratio	quad. level	run time
1	0.50270699	6	3.28
2	0.90761214	6	3.28
3	0.98835424	7	13.11
4	0.99877007	8	52.30
5	0.99987615	10	528.98
6	0.99998760	12	8360.91

8.4. Numerical Evidence for the Conjecture

In our tests of Conjecture 8.3, we programmed both sides of (17) using double-double arithmetic with the QD software package [14]. This permitted highly accurate analysis, which is essential since, as we discovered, the most interesting regions of the space also correspond to nearly singular matrices and correspondingly difficult integrands.

In particular, we generated a set of 1000 symmetric positive definite 3×3 test matrices, each of which was constructed as $A = ODO^*$, where O is the orthonormalization of a pseudorandom 3×3 matrix with integer entries chosen in $[-1000, 1000]$, and D is a diagonal matrix with integer entries in $[1, 1000]$. Note that the diagonal entries of D are also the eigenvalues of the test matrix $A = ODO^*$. The resulting 2-D integrals implicit in (17) were computed using the tanh-sinh quadrature scheme ([61, pp. 312–315], [71]) with sufficiently large numbers of quadrature points to ensure more than 10-digit accuracy in the results.

We found that the ratio L/R is indeed less than one in all cases, but that it is close to one for those matrices whose eigenvalues have two very small entries and one large entry. For example, a test matrix A with eigenvalues $(1, 5, 987)$ produced the ratio $0.987901\ldots$, and a test matrix A with eigenvalues $(1, 1, 999)$ produced the ratio $0.993626\ldots$.

Finally, we explored the case where the O matrix is generated pseudorandomly as above, but the eigenvalues in D (and thus the eigenvalues of A) are set to $(1, 1, 10^n)$, for $n = 1, 2, \cdots, 6$. The resulting L/R ratios are shown in Table 2, truncated to 8 digits, with quadrature levels and run times in seconds. Here the column labeled "quad. level" indicates the quadrature level Q. The number of quadrature points, approximately 8×2^Q in each of the two dimensions, is a index of the computational effort required.

All of these tests confirm that indeed the ratio $L/R < 1$, although evidently it can be arbitrarily close to one with nearly singular matrices. Thus, we see no reason to reject Conjecture 8.3.

In October 2014, mathematical physicists Mathieu Lewin and Elliott H. Lieb published an analysis consistent with our result, together with an argument concluding that the derivative never vanishes [89].

9. Requirements for Future High-Precision Arithmetic Software

As we have seen, high-precision arithmetic facilities are indispensable for a growing body of numerically demanding applications spanning numerous disciplines, especially applications in

mathematical physics. Software for performing such computations, and tools for converting scientific codes to use this software, are in general more efficient, robust and useable than they were in the past. Yet it is clear, from these examples and other experiences, that the presently available tools still have a long way to go.

Even commercial packages, which in general are significantly more complete than open software packages, could use some improvements. For example, a recent study by the present authors and Richard Crandall of Mordell-Tornheim-Witten sums, which arise in mathematical physics, required numerical values of derivatives of polylogarithms with respect to the order. Our version of *Maple* did not offer this functionality, and while our version of *Mathematica* attempted to evaluate these derivatives, the execution was rather slow and did not return the expected number of correct digits [90].

Here are some specific areas of needed improvement.

9.1. High-Precision and Emerging Architectures

The scientific computing world is moving into multicore and multinode parallel computing, because the frequency and performance of individual processors is no longer rapidly increasing [91]. It is possible to perform high-precision computations in parallel by utilizing message passing interface (MPI) software at the application level (rather than attempting to parallelize individual high-precision operations). MPI employs a "shared none" environment that avoids many difficulties. Indeed, numerous high-precision applications have been performed on highly parallel systems using MPI, including the Ising study mentioned in Section 6.1 [29].

But on modern systems that feature multicore processors, parallel computing is more efficiently performed using a shared memory, threaded environment such as OpenMP [91] within a single node, even if MPI is employed for parallelism between nodes. Computations that use an environment such as OpenMP must be entirely "thread-safe," which means, among other things, that no shared variables are actively written to, or otherwise there may be difficulties with processors stepping on each other during parallel execution. Employing "locks" and the like may remedy such difficulties, but only by reducing parallel efficiency.

From a brief survey performed by the present authors, only a few of the currently available high-precision software packages are certified thread-safe (in most cases, the available documentation makes no statement one way or the other). One high-level package that is thread-safe is the MPFR / MPFR C++ library [21,23], which has a "thread-safe" build option. It is is based on the GNU multiprecision library, which in turn is thread-safe. Also, one of the present authors has recently completed the MPFUN2015 package, a new Fortran-90-based system that is thread-safe [24]. Both of these two packages employ data structures for multiprecision data that incorporate the working precision level and other key information, enabling the precision level to be dynamically changed during execution without loss of thread safety.

Another important development here is the recent emergence of graphics processing units (GPUs) and on-chip accelerators (such as Intel's MIC design), which are now being employed for large-scale high-performance computing applications [91]. At the time of this writing, five of the top ten computer

systems on the Top 500 list of the world's most powerful supercomputers incorporate GPUs or accelerators, and this trend is expected to continue [92].

The experience of early implementations that have been attempted is mixed. Beliakov reported that GPU hardware did not perform as well as conventional CPUs for high-precision applications [43]. But Khanna reported "an order of magnitude" speedup compared with a conventional multicore processor on a simulation of Kerr black hole tails [93], and Mukunoki and Takahashi reported "up to 30 times faster" performance of double-double precision arithmetic on basic linear algebra software (BLAS) operations, on an Nvidia Tesla C1060 GPU, compared with an unaccelerated CPU [94]. Other than, say, the software produced by Khanna, Mukunoki and Takahashi, the present authors are not aware of any software for GPUs or accelerators specifically for high-precision computation, but this situation may soon change. In any event, it is clear that future work in high-precision software, either open-source or commercial, should investigate the potential usage of GPU and accelerator hardware.

9.2. Precision Level and Transcendental Support

As we noted above, some emerging applications require prodigious levels of numeric precision—10,000, 50,000 or more digits. Thus future facilities for high-precision arithmetic must employ advanced data structures and algorithms, such as FFT-based multiplication, that are efficient for extremely high-precision computation.

Along this line, it is no longer sufficient to simply provide basic arithmetic operations, at any precision level, since a surprisingly wide range of transcendental and special functions have arisen in recent studies, particularly in mathematical physics applications. Modern high-precision packages should support the following:

1. Basic transcendentals—exp, log, sin, cos, tan, hyperbolic functions—and the corresponding inverse functions [72, Sec. 4].
2. Gamma, digamma, polygamma, incomplete gamma, beta and incomplete beta functions [72, Sec. 5, 8].
3. Riemann zeta function, polylogarithms and Dirichlet L-functions [72, Sec. 25].
4. Bessel functions (first, second and third kinds, modified, etc.) [72,73, Sec. 10].
5. Hypergeometric functions [72, Sec. 15].
6. Airy functions [72, Sec. 9].
7. Elliptic integral functions [72, Sec. 19].
8. Jacobian elliptic functions and Weierstrass elliptic/modular functions [72, Sec. 22, 23].
9. Theta functions [72, Sec. 20, 21].

These functions should be implemented with the best available algorithms for different argument ranges and precision levels, and should also support both real and complex arguments where possible.

Recent research gives hope in this arena—see, for instance [15, pp. 215–245], [72,95–97]—but these published schemes need to be implemented in publicly available high-precision computing environments.

Along this line, our experience with high precision implementation of special functions in *Mathematica* has been rather disappointing. This is not only our opinion. For example, Johansson [97] writes, "The series expansion of the gamma function in *Mathematica*, included for scale, seems to be poorly implemented."

9.3. Reproducibility

As we noted above in Section 1, reproducibility is increasingly important in scientific computing, at least in the general sense of producing numerically reliable results consistent across different platforms and equivalent implementations [1,2]. For example, computations involved in highly regulated industries, such as the financial, pharmaceutical and aircraft industries, are increasingly required to certify their numerical reliability. Required or not, engineers often deploy reproducibility checks to forestall legal challenges.

As is well known, architectural differences, subtle changes in the order in which compilers generate instructions and even changes to the processor count can alter results. What's more, round-off errors inevitably accumulate in large calculations, which may render results invalid. Some of these difficulties stem from the fact that floating-point arithmetic operations (standard or high-precision) are not associative: $(a+b)+c$ is not guaranteed to be the same as $a+(b+c)$. They are further exacerbated in a highly parallel environment, where operations are performed disjointly on many different processors, and where deterministic, sequential order of execution typically cannot be guaranteed.

One benefit of high-precision arithmetic is to enhance reproducibility of results, since it can dramatically reduce floating-point roundoff error. Extra precision is not a cure-all, though. One frequently-cited example is due to Siegfried Rump [98, pg. 12]: compute

$$f(a,b) \;=\; 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b) \tag{18}$$

where a = 77617 and b = 33096. On a present-day IEEE-754 system, the results for 32-bit, 64-bit, double-double and quad-double are $-6.338253 \times 10^{29}, -1.1805916207 \times 10^{21}, 1.1726039400$ and -0.8273960599, respectively. Only the last value, computed with quad-double (62-digit) arithmetic, returned a correct answer to 10-digit accuracy. On at least one other system, 32-bit and 64-bit results were nearly the same (but wrong), potentially leading a user to think that 64-bit was sufficient. In most cases, including Rump's polynomial, by appropriately adjusting the level of precision, a high level of reproducibility can be achieved. But there is no foolproof approach; some analysis and experimentation is always required.

Some researchers are investigating facilities to guarantee bit-for-bit reproducibility in standard IEEE floating-point arithmetic, not just at the single operation level, but also at the application level, across different platforms, although for the time being they feature only partial reproducibility [99]. Similar work is also underway to provide this functionality in a high-precision environment. For example, in a significant step forward, the GMP, MPFR and MPFR C++ packages now provide correctly rounded arbitrary precision arithmetic, in analogy to the correctly rounded modes of IEEE-754 arithmetic.

At the application level, however, high-precision bit-for-bit reproducibility is still a challenge, since high-precision operations, like their IEEE-754 equivalents, are not associative, and high-level compilers may alter the order of operations that are sent to the lower-level multiprecision routines. In *Maple*, for example, according to a Maplesoft researcher whom we consulted, "all bets are off" once more than one operation is involved. What's more, even if bit-for-bit reproducibility is achieved for high precision on a single-processor computer system, parallel systems present additional challenges, for the reasons mentioned above.

Thus for the foreseeable future, scientific researchers and engineers who use high-precision libraries must design their application-level codes to be robust in the presence of some level of numerical error and non-reproducibility.

Additionally, bit-for-bit reproducibility, although it may be helpful for some high-precision applications, has the potential downside of masking serious numerical difficulties. Even with IEEE 32- or 64-bit floating-point codes, numerical problems often come to light only when a minor code change or a run on a different number of processors produces a surprisingly large change in the results (recall the experience of the ATLAS code, mentioned in the Introduction). Thus ensuring bit-for-bit reproducibility in high-precision computation might hide such problems from the user. At the least, bit-for-bit reproducibility should be an optional feature, and rounding modes should be controllable by the user, just as they are with the IEEE-754 standard.

Acknowledgements

Jonathan M. Borwein was supported in part by the Australian Research Council.

Author Contributions

Each of the authors made major contributions both to the research and the writing of this article.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Stodden, V.; Borwein, J. M.; Bailey, D. H. Publishing Standards for Computational Science: 'Setting the Default to Reproducible.' *SIAM News* **2013**, *46*, 4–6.
2. Stodden, V. ; Bailey, D. H.; Borwein, J. M.; LeVeque, R. J.; Rider, W.; Stein, W. Setting the default to reproducible: Reproducibility in computational and experimental mathematics, 2013. Available online: http://www.davidhbailey.com/dhbpapers/icerm-report.pdf (accessed on 30 April 2015).
3. He, Y.; Ding, C.; Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications. *J. Supercomput.* **2001**, *18*, 259–277.
4. Robey, R. W.; Robey, J. M.; Aulwes, R. In search of numerical consistency in parallel programming. *Parallel Comput.* **2011**, *37*, 217–219.
5. Bailey, D. H.; Barrio, R.; Borwein, J. M. High-precision computation: Mathematical physics and dynamics. *Appl. Math. Computat.* **2012**, *218*, 10106–10121.

6. Computer Science Curriculum 2008: An interim revision of CS 2001. Association for Computing Machinery and IEEE Computer Society: New York, NY, USA 2008. Available online: http://www.acm.org/education/curricula/ComputerScience2008.pdf (accessed on 30 April 2015).
7. Rubio-Gonzalez, C.; Nguyen, C.; Nguyen, H. D.; Demmel, J.; Kahan, W.; Sen, K.; Bailey, D. H.; Iancu, C. Precimonious: Tuning assistant for floating-point precision. *Proc. of SC13*, Salt Lake City, UT, November 2013. Available online: http://www.davidhbailey.com/dhbpapers/precimonious.pdf (accessed on 30 April 2015).
8. Press, W. H.; Eukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, 3rd ed.; Cambridge University Press: Cambridge, UK, 2007.
9. LINPACK. Available online: http://www.netlib.org/linpack (accessed on 30 April 2015).
10. LAPACK. Available online: http://www.netlib.org/lapack (accessed on 30 April 2015).
11. Demmel, J.; Koev, P. The accurate and efficient solution of a totally positive generalized Vandermonde linear system. *SIAM J. Matrix Anal. Appl.* **2005**, *27*, 145–152.
12. Knuth, D. E. *The Art of Computer Programming: Seminumerical Algorithms* 3rd ed.; Addison-Wesley: New York, NY, USA, 1998.
13. Shewchuk, J. R. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discr. and Comp. Geometry*, **1997**, *18*, 305–363.
14. Hida, Y.; Li, X. S.; Bailey, D. H. Algorithms for Quad-Double Precision Floating Point Arithmetic. Proc. of the 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society, Vail, CO, USA, June 2001. Available online: http://crd-legacy.lbl.gov/~dhbailey/mpdist (accessed on 30 April 2015).
15. Borwein, J. M.; Bailey, D. H. *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, 2nd ed.; A.K. Peters: Natick, MA, USA, 2008.
16. Brent, R. P.; Zimmermann, P. *Modern Computer Arithmetic*; Cambridge Univ. Press: Cambridge, UK, 2010.
17. Bailey, D. H.; Hida, Y.; Li, Y.; Thompson, B. ARPREC: An arbitrary precision computation package, 2002. Available online: http://www.davidhbailey.com/dhbpapers/arprec.pdf (accessed on 30 April 2015). Software available online: http://crd-legacy.lbl.gov/~dhbailey/mpdist (accessed on 30 April 2015).
18. CLN — Class Library for Numbers. Available online: http://www.ginac.de/CLN (accessed on 30 April 2015).
19. The GNU multiple precision library. Available online: https://gmplib.org (accessed on 30 April 2015).
20. Julia. Available online: http://julialang.org (accessed on 30 April 2015).
21. The GNU MPFR library. Available online: http://www.mpfr.org (accessed on 30 April 2015).
22. MPFR++. Available online: http://perso.ens-lyon.fr/nathalie.revol/software.html (accessed on 30 April 2015).
23. MPFR C++. Available online: http://www.holoborodko.com/pavel/mpfr (accessed on 30 April 2015).

24. Bailey, D. H. MPFUN2015: A thread-safe arbitrary precision computation package, 2015. Available online: http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf (accessed on 30 April 2015).
25. mpmath: Python library for arbitrary-precision floating-point arithmetic. Available online: https://code.google.com/p/mpmath/ (accessed on 30 April 2015).
26. NTL: A library for doing number theory. Available online: http://www.shoup.net/ntl/ (accessed on 30 April 2015).
27. PARI/GP. Available online: http://pari.math.u-bordeaux.fr (accessed on 30 April 2015).
28. SageMath. Available online: http://sagemath.org (accessed on 30 April 2015).
29. Bailey, D. H.; Borwein, J. M.; Crandall, R. E. Integrals of the Ising class. *J. Physics A: Math. Gen.* **2006**, *39*, 12271–12302.
30. Ma, D.; Saunders, M. Solving multiscale linear programs using the simplex method in quadruple precision, 2014. Available online: http://stanford.edu/group/SOL/multiscale/papers/quadLP3.pdf (accessed on 30 April 2015).
31. Ma, D.; Saunders, M. Experiments with linear and nonlinear optimization using quad precision, 2014. Available online: http://stanford.edu/group/SOL/multiscale/talks/14informsQuadMINOS.pdf (accessed on 30 April 2015).
32. Tucker, W. A rigorous ODE solver and Smale's 14th problem. *Found. Comput. Math.* **2002**, *2*, 53–117.
33. Macfarlane, M. H. A high-precision study of anharmonic-oscillator spectra. *Ann. Phys.* **1999**, *271*, 159–202.
34. Lake, G.; Quinn, T.; Richardson, D. C. From Sir Isaac to the Sloan survey: Calculating the structure and chaos due to gravity in the universe. Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, November 1997; pp. 1–10.
35. Farrés, A.;l Laskar, J.; Blanes, S.; Casas, F.; Makazaga, J.; Murua, A. High precision symplectic integrators for the Solar System. *Celest. Mech. Dyn. Astron.* **2013**, *116*, 141–174.
36. Frolov, A. M.; Bailey, D. H. Highly accurate evaluation of the few-body auxiliary functions and four-body integrals. *J. Phys. B* **2003**, *36*, 1857–1867.
37. Yan, Z.-C.; Drake, G. W. F. Bethe logarithm and QED shift for Lithium. *Phys. Rev. Lett.* **2003**, *81*, 774–777.
38. Zhang, T.; Yan, Z.-C.; Drake, G. W. F. QED corrections of $O(mc^2\alpha^7 \ln \alpha)$ to the fine structure splittings of Helium and He-Like ions. *Phys. Rev. Lett.* **1994**, *77*, 1715–1718.
39. Ellis, R. K.; Giele, W. T.; Kunszt, Z.; Melnikov, K.; Zanderighi, G. One-loop amplitudes for W+3 jet production in hadron collisions, 2008. Available online: http://arXiv.org/abs/0810.2762 (accessed on 30 April 2015).
40. Berger, C. F.; Bern, Z.; Dixon, L. J.; Febres Cordero, F.; Forde, D.; Ita, H.; Kosower, D. A.; Maitre, D. An automated implementation of on-shell methods for one-loop amplitudes. *Phys. Rev. D* **2008**, *78*, 036003.
41. Ossola, G.; Papadopoulos, C. G.; Pittau, R. CutTools: A program implementing the OPP reduction method to compute one-loop amplitudes. *J. High-Energy Phys.* **2008**, *0803*, 042.

42. Czakon, M. Tops from light quarks: Full mass dependence at two-Loops in QCD. *Phys. Lett. B* **2008**, *664*, 307.
43. Beliakov, G.; Matiyasevich, Y. A parallel algorithm for calculation of large determinants with high accuracy for GPUs and MPI clusters, 2013. Available online: http://arxiv.org/abs/1308.1536 (accessed on 30 April 2015).
44. Simó, C. Global dynamics and fast indicators. *Global Analysis of Dynamical Systems*, Krauskopf, B., Broer, H. W., Vegtr, G., Eds.; Taylor and Francis: New York, USA, 2001; pp. 373–389.
45. Barrio, R. Performance of the Taylor series method for ODEs/DAEs. *Appl. Math. and Comput.* **2005**, *163*, 525–545.
46. Barrio, R. ; Blesa, F.; Lara, M. VSVO formulation of the Taylor method for the numerical solution of ODEs. *Comput. Math. Appl.* **2005**, *50*, 93–111.
47. Corliss, G.; Chang, Y. F. Solving ordinary differential equations using Taylor series. *ACM Trans. Math. Software* **1982**, *8*, 114–144.
48. Barrio, R. Sensitivity analysis of ODEs/DAEs using the Taylor series method. *SIAM J. Sci. Comput.* **2006**, *27*, 1929–1947.
49. Dena, A.; Rodriguez, M.; Serrano, S.; Barrio, R. High-precision continuation of periodic orbits. *Abstr. Appl. Anal.* **2012**, 716024.
50. Hairer, E.; Nørsett, S.; Wanner, G. *Solving ordinary differential equations. I. Nonstiff problems*; 2nd ed.; Springer-Verlag, Berlin, Germany, 1993.
51. Hairerm E.; Ostermann, A. Dense output for extrapolation methods. *Numer. Math.* **1990**, *58*, 419–439.
52. Barrio, R.; Rodriguez, M.; Abad, A.; Blesa, F. Breaking the limits: The Taylor series method. *Appl. Math. and Comput.* **2011**, *217*, 7940–7954.
53. Lorenz, E. Deterministic nonperiodic flow. *J. Atmospheric Sci.* **1963**, *20*, 130–141.
54. Abad, A.; Barrio, R.; Blesa, F.; Rodriguez, M. TIDES: a Taylor series Integrator for Differential EquationS. *ACM Trans. Math. Software* **2012**, *39*, 1.
55. Dormand, J. R.; Prince, P. J. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.* **1980**, *6*, 19–26.
56. Viswanath, D. The Lindstedt-Poincaré technique as an algorithm for computing periodic orbits. *SIAM Rev.* **2001**, *43*, 478–495.
57. Viswanath, D. The fractal property of the Lorenz attractor. *J. Phys. D* **2004**, *190*, 115–128.
58. Viswanath, D.; Şahutŏglu, S. Complex singularities and the Lorenz attractor. *SIAM Rev.* **2010**, *52*, 294–314.
59. Abad, A.; Barrio, R.; Dena, A. Computing periodic orbits with arbitrary precision. *Phys. Rev. E* **2011**, *84*, 016701.
60. Calvo, F.; Pahl, E.; Wormit, M.; Schwerdtfeger, P. Evidence for low-temperature melting of Mercury owing to relativity. *Angew. Chem. Intl. Ed. Engl*, **2013**, doi: 0.1002/anie.201302742.
61. Borwein, J. M.; Bailey, D. H.; Girgensohn, R. *Experimentation in Mathematics: Computational Paths to Discovery*, A.K. Peters: Natick, MA, USA, 2004.
62. Ferguson, H. R. P.; Bailey, D. H.; Arno, S. Analysis of PSLQ, an integer relation finding algorithm. *Math. Comput.* **1999**, *68*, 351–369.

63. Bailey, D. H.; Broadhurst, D. Parallel integer relation detection: Techniques and applications. *Math. Comput.* **2000**, *70*, 1719–1736.
64. Chen, J.; Stehle, D.; Villard, G. A new view on HJLS and PSLQ: Sums and projections of lattices. Proceedings of ISSAC'13, Boston, MA, USA, June 2013; pp. 149–156.
65. Bailey, D. H. A compendium of BBP-type formulas. **2011**. Available online: http://www.davidhbailey.com/dhbpapers/bbp-formulas.pdf (accessed on 30 April 2015).
66. Bailey, D. H.; Borwein, J. M.; Mattingly, A.; Wightwick, G. The computation of previously inaccessible digits of π^2 and Catalan's constant. *Not. AMS* **2013**, *60*, 844–854.
67. Karrels, E. Computing digits of pi with CUDA, 2013. Available online: http://www.karrels.org/pi (accessed on 30 April 2015).
68. Becher, V.; Heiber, P. A.; Slaman, T. A. A computable absolutely normal Liouville number, 2014. Available online: http://math.berkeley.edu/~slaman/papers/liouville.pdf (accessed on 30 April 2015).
69. Stoneham, R. On absolute (j, ε)-normality in the rational fractions with applications to normal numbers. *Acta Arithmetica* **1973**, *22*, 277–286.
70. Bailey, D. H.; Borwein, J. M. Nonnormality of Stoneham constants. *Ramanujan J.* **2012**, *29*, 409-422; DOI:10.1007/s11139-012-9417-3.
71. Takahasi, H.; Mori, M. Double exponential formulas for numerical integration. *Pub. RIMS* **1974**, *9*, 721–741.
72. NIST Digital Library of Mathematical Functions, version 1.0.6, May 2013. Available online: http://dlmf.nist.gov (accessed on 30 April 2015).
73. *NIST Handbook of Mathematical Functions*; Olver, F. W. J., Lozier, D. W., Boisvert, R. F., Clark, C. W., Eds.; Cambridge University Press: New York, NY, USA, 2010.
74. The Inverse Symbolic Calculator 2.0. Available online: https:/isc.carma.newcastle.edu.au (accessed on 30 April 2014).
75. Panzer, E. Algorithms for the symbolic integration of hyperlogarithms with applications to Feynman integrals, 2014. Available online: http://arxiv.org/abs/1403.3385 (accessed on 30 April 2015).
76. Bailey, D. H.; Borwein, J. M. Hand-to-hand combat with thousand-digit integrals. *J. Comput. Sci.* **2012**, *3*, 77–86.
77. Borwein, J. M.; Straub, A.; Wan, J. Three-step and four-step random walk integrals. *Exp. Math.* **2013**, *22*, 1–14.
78. Lucas, S. K.; Stone, H. A. Evaluating infinite integrals involving Bessel functions of arbitrary order. *J. Comp. Appl. Math.* **1995**, *64*, 217–231.
79. Sidi, A. The numerical evaluation of very oscillatory infinite integrals by extrapolation. *Math. Comput.* **1982**, *38* 517–529.
80. Sidi, A.; A user-friendly extrapolation method for oscillatory infinite integrals. *Math. Comput.* **1988**, *51*, 249–266.
81. Sidi, A. A user-friendly extrapolation method for computing infinite range integrals of products of oscillatory functions. *IMA J. Numer. Anal.* **2012**, *32*, 602–631.

82. Bailey, D. H.; Borwein, J. M.; Broadhurst, D. M.; Glasser, L. Elliptic integral representation of Bessel moments. *J. Phys. A: Math. Theor.* **2008**, *41*, 5203–5231.
83. Wan, J. Moments of products of elliptic integrals. *Adv. Appl. Math* **2012**, *48*, 121–141.
84. Rogers, M.; Wan, J. G.; Zucker, I. J. Moments of elliptic integrals and critical L-values. **2013**. Available online: http://arxiv.org/abs/1303.2259 (accessed on 30 April 2015).
85. Bailey, D. H.; Borwein, J. M.; Crandall, R. E.; Zucker, J. Lattice sums arising from the Poisson equation. *J. Physics A: Math. Theor.* **2013**, *46*, 115201.
86. Borwein, D.; Borwein, J. M.; Straub, A. On lattice sums and Wigner limits. *J. Math. Anal. Appl.* **2014**, *414*, 489–513.
87. Borwein, J. M.; Glasser, L.; McPhedran, R.; Wan, J. G.; Zucker, I. J. *Lattice Sums: Then Now* Cambridge University Press: Cambridge, UK, 2013.
88. Borwein, D.; Borwein, J. M.; Shail, R. Analysis of certain lattice sums. *J. Math. Anal. Appl.* **1989**, *143*, 126–137.
89. Lewin, M.; Lieb, E. H. Improved Lieb-Oxford exchange-correlation inequality with gradient correction. **2014**. Available online: http://arxiv.org/abs/1408.3358 (accessed 30 on April 2015).
90. Bailey, D. H.; Borwein, J. M.; Crandall, R. E. Computation and theory of extended Mordell-Tornheim-Witten sums. *Math. Comput.* **2015**. Available online: http://www.davidhbailey.com/dhbpapers/BBC.pdf (accessed on 30 April 2015).
91. Williams, S. W.; Bailey, D. H. Parallel computer architecture. *Performance Tuning of Scientific Applications*, Bailey, D. H., Lucas, R. F., Williams, S. W., Eds; CRC Press: Boca Raton, FL, USA, 2011; pp. 11–33.
92. Top 500 list. Nov. 2014. Available online: http://top500.org (accessed on 30 April 2015).
93. Khanna, G. High-precision numerical simulations on a CUDA GPU: Kerr black hole tails. *J. Sci. Comput.* **2013**, *56*, 366–380.
94. Mukunoki, D.; Takahashi, D. Implementation and evaluation of quadruple precision BLAS functions on GPUs. *Appl. Parallel Sci. Comput.* **2012**, *7133*, 249–259.
95. Chevillard, S.; Mezzarobba, M. Multiple precision evaluation of the Airy Ai function with reduced cancellation. *Proceedings of the 21st IEEE Symposium on Computer Arithmetic*, IEEE Computer Society: Austin, TX, USA, June 2013.
96. Crandall, R. E. Unified algorithms for polylogarithm, L-series and zeta variants. *Algorithmic Reflections: Selected Works*; PSIPress: Portland, OR, USA, 2012.
97. Johansson, F. Fast and rigorous computation of special functions to high precision; PhD thesis, RSC Institute: Linz, Austria, 2014.
98. Muller, J.-M.; Brisebarre, N.; de Dinechin, F.; Jeannerod, C.-P.; Lefevre, V.; Melquiond, G.; Revol, N.; Stehle, D.; Torres, S. *Handbook of Floating-Point Arithmetic*; Birkhauser: Boston, MA, USA, 2010.

99. Corden, M. J.; Kreitzer, D. Consistency of floating-point results using the Intel compiler, or why doesn't my application always give the same answer. Intel Corporation, 2012. Available online: http://software.intel.com/sites/default/files/article/164389/fp-consistency-102511.pdf (accessed on 30 April 2015).