High-Precision Arithmetic: Progress and Challenges

David H. Bailey, *Member, IEEE* and Jonathan M. Borwein

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 and presents a sample of applications, including some new results from the past year or two. This article also discusses what facilities are required to support future high-precision computing, in light of emerging applications and changes in computer architecture, such as highly parallel systems and graphical processing units.

Index Terms—high-precision arithmetic, numerical analysis, numerical error, experimental mathematics

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 parts of a computation can be performed with lower precision and which steps 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? How can they be tracked down in source code? And, once identified, how can they most easily be remedied or controlled? For which of these applications is high-precision arithmetic required? These are questions many researchers are now asking.

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:

> *Science is built upon the foundations of theory and experiment validated and improved through open, transparent communication. With the increasingly central role of computation in scientific discovery this means communicating all details of the computations needed for others to replicate the experiment.* [70].

The workshop report further noted

> *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*

- *Bailey is with the Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA and the University of California, Davis, Department of Computer Science, Davis, CA, 95616 USA,* dhbailey@lbl.gov.
- *Borwein is with the Centre for Computer Assisted Research Mathematics and its Applications (CARMA), University of Newcastle, Callaghan, NSW 2308, Australia,* jonathan.borwein@newcastle.edu.au. *Supported in part by the Australian Research Council.*

> *cases. And it is doubly difficult for other researchers, using independently written codes and distinct computer systems, to reproduce published results.* [70]

The workshop's main recommendations are also discussed in [18], [69].

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.

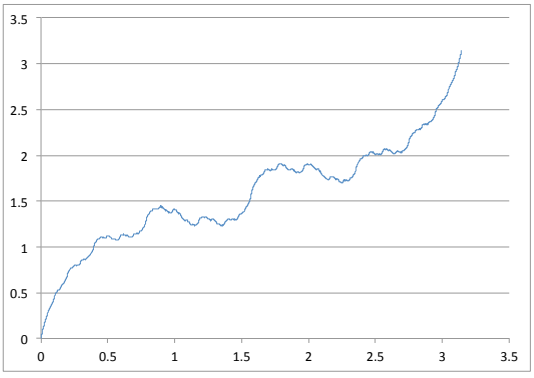

Fig. 1. Graph of $g(x) = x + \sum_{0 \leqslant k \leqslant 10} 2^{-k} \sin(2^k x)$, over $(0, \pi)$.

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 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 [4]. ◇

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 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 [51].

Researchers working with some 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 [61].

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 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 [12], 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.

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 have 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 on this tool, with some examples of its usage, are presented in [63].

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 [12], 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 [60, 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 $(a_1, a_2, \cdots, a_n)$ using *LINPACK* [45] or *LAPACK* [44] software.

An implementation of this approach using standard IEEE 64-bit floating-point arithmetic, with final results rounded to the nearest integer, correctly finds the vector of coefficients $(1, 0, 0, 32769, 0, 0, 1)$, which corresponds to $f(x) = 1 + (2^{15} + 1)x^3 + x^6$. But this scheme fails when given the 9-long sequence $(1, 1048579, 16777489, 84941299, 268501249, 655751251, 1360635409, 2523398179, 4311748609)$, which is generated by the degree-8 polynomial function $f(x) = 1 + (2^{20} + 1)x^4 + x^8$.

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 [43] (currently the state-of-the-art for such problems). A 64-bit implementation of either scheme 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 least-squares 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 floating-point arithmetic, with 112 mantissa bits, but sadly it is not yet widely implemented in hardware. A much 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.

Such data can be added by using a scheme originally proposed by Donald E. Knuth [54] (see also [64]): Let the notation `fl` mean that the indicated operations are to be performed with rounded IEEE 64-bit arithmetic. Then given 64-bit floats a and b, the sequence

$$\begin{aligned} x &= \texttt{fl}(a + b) \\ y &= \texttt{fl}(x - a) \\ z &= \texttt{fl}((a - (x - y)) + (b - y)) \end{aligned}$$

returns their sum as a double-double entity: x is the closest double value to $a + b$, and z is the low-order word, so that (x, z) jointly represents the exact sum of a and b. A simple extension of this sequence can

be used to add two double-double operands, and related schemes can be used to realize all four basic arithmetic operations on double-double data. Similar schemes also work well for quad-double arithmetic, which operates on strings of four IEEE 64-bit floats. This datatype provides roughly 62-digit accuracy [52].

For higher-levels of precision, one typically represents a high-precision datum as a string of floats or integers, where the first word contains the size of the datum, the second word contains an exponent, and each subsequent word contains B bits of the mantissa. For moderate precision levels (up to roughly 1000 digits), arithmetic on such data is typically performed using adaptations of the familiar schemes taught in grade school, operating on entire words rather than individual digits.

Above about 1000 decimal digits, advanced algorithms should be employed for maximum efficiency. For example, note that the product of two high-precision values, represented as above, may be computed by a convolution scheme 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

$$z_k \quad = \quad \sum_{k=0}^{2n-1} x_k y_{2n-k-1}. \qquad (1)$$

Finally, release carries, which is done by iterating $z_{k-1} := z_{k-1} + \mathrm{int}(z_k/2^B)$, beginning with $k = 2n-1$ 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.

Now note that the convolution operation (1) may be performed using fast Fourier transforms (FFTs):

$$z \quad = \quad F^{-1}[F[x] \cdot F[y]] \qquad (2)$$

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 $O(n \log n)$ operation. For details, see [24] or [32, pp. 215–245].

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 [32, pp. 215–245], [36].

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 arithmetic operations with subroutine calls, 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 such packages. The ARPREC [24], QD [52] and MPFUN90 packages are available from http://crd-legacy.lbl.gov/~dhbailey/mpdist.

- ARPREC: supports arbitrary precision real, integer and complex, with many algebraic and transcendental functions. Includes high-level language interfaces for C++ and Fortran-90.
- GMP: supports high-precision integer, rational and floating-point calculations. Distributed under the GNU license by the Free Software Foundation. Available at http://gmplib.org.
- MPFR: supports multiple-precision floating-point computations with exact rounding, based on GMP. Available at http://www.mpfr.org.
- MPFR++: a high-level C++ interface to MPFR. Available at http://perso.ens-lyon.fr/nathalie.revol/software.html.
- MPFUN90: similar to ARPREC in user-level functionality, but written entirely in Fortran-90. Includes a Fortran-90 high-level interface.
- QD: includes routines to perform "double-double" (approx. 31 digits) and "quad-double" (approx. 62 digits) arithmetic, as well as many algebraic and transcendental functions. Includes high-level language interfaces for C++ and Fortran-90.

Obviously there is an extra cost for performing high-precision arithmetic. Double-double computations typically run at least five times slower than 64-bit; quad-double computations typically run at least 25 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 computa-

tions 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, 1024 processors were employed to compute a numerical integral in [7], and up to 512 processors were employed to computed some two- and three-dimensional integrals in [14]. Parallel high-precision computation will be further discussed in Section 8.

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. This material is condensed and adapted from [12]. See [12] for additional details and examples.

3.1 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 [55].

3.2 Coulomb n-body atomic systems

Alexei Frolov of the University of Western Ontario in Canada has employed high-precision software in studies of n-body Coulomb atomic systems, which require the accurate 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 a few years ago were consider intractable [22], [49].

3.3 Nuclear physics computations

Researchers have recently applied high-precision computations to produce accurate solutions to the Schrodinger 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 [78]. 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 [79].

3.4 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 [46], [29], [59], [42].

3.5 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 [68].

The Taylor method can be defined as follows [26], [28], [40]. 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

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

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 [26], [27], [28].

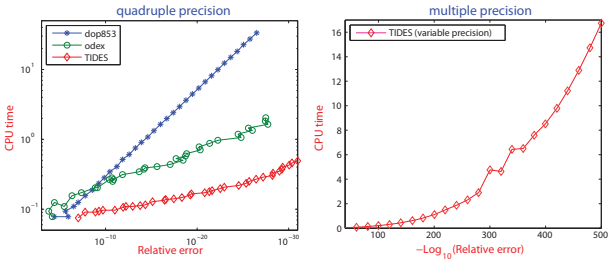

Fig. 2. Left: Error vs. CPU time for the numerical solution of the unstable periodic orbit LR for the Lorenz model (in double-double arithmetic) using a Runge-Kutta code (dop853), an extrapolation code (odex) and a Taylor series method (TIDES). Right: Error vs. CPU time for the numerical solution of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

Figure 2 (left) compares computations of the unstable periodic orbit LR in the Lorenz model [57]

with three methods: (a) the Taylor method (using the TIDES code [1]), (b) a conventional Runge-Kutta method (using the dop853 code), and (c) an extrapolation method (using the odes code) [50]. 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 [74].

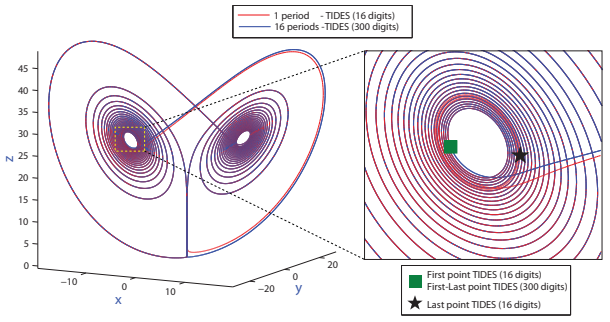

Fig. 3. Numerical solution of the $L^{25}R^{25}$ unstable periodic orbit of the Lorenz model for 16 time periods using the TIDES code with 300 digits, compared with just one time period using 64-bit IEEE arithmetic.

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

3.6 Periodic orbits

The Lindstedt-Poincaré method [73] 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 [74] 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 [74], [75].

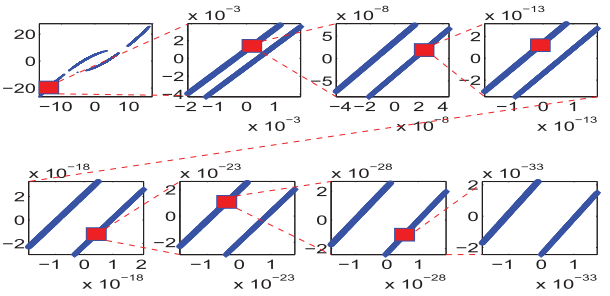

Fig. 4. Fractal property of the Lorenz attractor. The first plot shows a rectangle in the plane. All later plots zoom in on a tiny region (too small to be seen by the unaided eye) at the center of the red rectangle of the preceding plot to show that what appears to be a line is in fact not a line. (Reproduced with permission from [75]).

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 [2].

To conclude this section, we mention a fascinating article [37], which describes how simulations must employ quantum molecular dynamics and special relativity to obtain realistic results on 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.

4 High-Precision Arithmetic in Experimental Mathematics

Very high-precision floating-point arithmetic is now considered an indispensable tool in the field of experimental mathematics [32], [6].

Many of these computations involve variants of Ferguson's PSLQ integer relation detection algorithm [47], [19]. 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. 10^{-p}, where 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 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).

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 [19]. Most of our computations have been done with a two-level multipair PSLQ program.

Figure 5, which illustrates a 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.

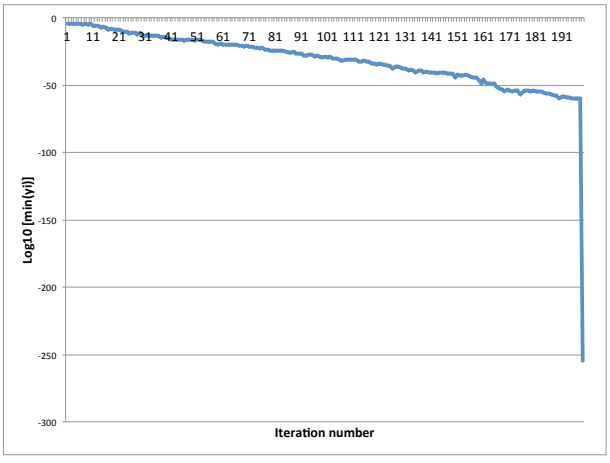

Fig. 5. Decrease of $\log_{10}(\min|y_i|)$ in a multipair PSLQ run.

4.1 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 [13], [32, 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 [3]. 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 [17].

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 [53]. 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 [17].

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}. Normality in other bases is defined similarly. 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 [20], [32, pp. 163–178]. This same line of research subsequently led to a proof that an uncountably infinite class of real numbers are 2-normal [21], [25], including, for instance,

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

(This particular constant was proven 2-normal by Stoneham in 1973; the more recent results span a much larger class.) Although this constant is provably 2-normal, interestingly it is provably *not* 6-normal [9].

5 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 [12] and some related references as shown below.

5.1 Ising integrals

In one study, the tanh-sinh quadrature scheme [23], [71], implemented using the ARPREC high-precision software [24], was employed to study the following classes of integrals [14], [15]. The D_n integrals arise in the Ising theory of mathematical physics, while the C_n are connected to quantum field theory:

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

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

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

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* [58]. 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* (available at `http://carma-lx1.newcastle.edu.au:8087`), as

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

where γ is Euler's constant. This identity was then proven [14]. 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 [14], [15].

5.2 Ramble integrals

A separate study considered n-dimensional *ramble integrals* [8]

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

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 [35], it is shown that when $s = 0$, the first derivatives of these integrals are

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

Here $J_n(x)$ is the Bessel function of the first kind [58].

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 [56], which, in turn, is based on two earlier papers by Sidi [65], [66]. While the computations were relatively expensive, we were able to compute 1000-digit values of these integrals for odd n up to 17 [8]. 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 [67], which he believes can be used for even n. We recently started to implement it in high-precision arithmetic, but do not yet have any results.

5.3 Moments of elliptic integral functions

The analysis of ramble integrals led to a study of moments of integrals involving products of elliptic integral functions [8]:

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

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 [16]. We computed over 4000 individual integrals of this form to between 1500 and 3000-digit precision using the ARPREC software

[24]. 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 [8]:

$$\begin{aligned} 81\int_0^1 x^3K^2(x)E(x)\mathrm{d}x &\stackrel{?}{=} \\ -6\int_0^1 K^3(x)\mathrm{d}x - 24\int_0^1 x^2K^3(x)\mathrm{d}x & \\ +51\int_0^1 x^3K^3(x)\mathrm{d}x + 32\int_0^1 x^4K^3(x)\mathrm{d}x, & \quad (6) \\ -243\int_0^1 x^3K(x)E(x)K'(x)\mathrm{d}x &\stackrel{?}{=} \\ -59\int_0^1 K^3(x)\mathrm{d}x + 468\int_0^1 x^2K^3(x)\mathrm{d}x & \\ +156\int_0^1 x^3K^3(x)\mathrm{d}x - 624\int_0^1 x^4K^3(x)\mathrm{d}x & \\ -135\int_0^1 xK(x)E(x)K'(x)\mathrm{d}x. & \quad (7) \end{aligned}$$

James Wan [76] has been able to prove many of these identities, including, recently, (6), but by no means all. A followup work [62] 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.

6 Lattice sums arising from a Poisson equation

In this section we describe some results, just completed [11], 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 [11]:

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

By noting some striking connections with Jacobi ϑ-function values, Richard Crandall, John Zucker and the present authors were are able to develop new closed forms for certain values of the arguments r_k [11].

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

$$\begin{aligned} \phi_2(x,y) \;=\; & \frac{1}{4\pi}\log\frac{\cosh(\pi x)+\cos(\pi y)}{\cosh(\pi x)-\cos(\pi y)} \\ & -\frac{2}{\pi}\sum_{\mathbf{m}\in\mathbb{O}^+}\frac{\cosh(\pi m x)\cos(\pi m y)}{m(1+e^{\pi m})}, \quad (8) \end{aligned}$$

which is valid for $x, y \in [-1, 1]$.

After extensive high-precision numerical experimentation using (8), we discovered (then proved) the remarkable fact that when x and y are rational,

$$\phi_2(x,y) = \frac{1}{\pi}\log A, \quad (9)$$

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

In this case we computed $\alpha = \exp(8\pi\phi_2(x,y))$ (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, 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.

Here are some examples of the minimal polynomials discovered by this process [11]:

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	$1-216\alpha+860\alpha^2-744\alpha^3+454\alpha^4-744\alpha^5$ $+860\alpha^6-216\alpha^7+\alpha^8$

Using this data, we were able to conjecture a formula that gives the degree d as a function of k [11].

These computations required prodigiously high precision—up to 20,000-digit floating-point arithmetic in some cases, such as in the computation to find 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 [5]. The norm of the largest polynomials so uncovered is large enough to cause some problems for current computer algebra systems.

7 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 [31].

Throughout this section, $Q(x) = Q(x_1, \dots, x_d)$ is a positive definite quadratic form in d variables with real coefficients and determinant $\Delta > 0$.

As proposed in [34, Chap. 7], [31] 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

$$\alpha_N(s) := \sum_{n_1=-N}^{N} \cdots \sum_{n_d=-N}^{N} \frac{1}{Q(n_1, \dots, n_d)^s}, \quad (10)$$

$$\beta_N(s) := \int_{-N-1/2}^{N+1/2} \cdots \int_{-N-1/2}^{N+1/2} \frac{\mathrm{d}x_1 \cdots \mathrm{d}x_d}{Q(x_1, \dots, x_d)^s}. \quad (11)$$

As usual, the summation in (10) is understood to avoid the term corresponding to $(n_1, \dots, n_d) = (0, \dots, 0)$. 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 therefore 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. For more on the physical background behind such sums, which motivates the interest in the limit $\sigma(s)$, we refer to [31].

In the case $d = 2$, it was shown in [30, 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 [30, 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 7.1 below extends this result to arbitrary positive definite Q in all dimensions.

7.1 Jump discontinuities in Wigner limits

As above, let $Q(x) = Q_A(x) = \sum_{1\leqslant i,j\leqslant d} a_{ij} x_i x_j$, 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

$$V(s) = V_Q(s) := \int_{\|x\|_\infty = 1} \frac{Q_{B(s)}(x)}{Q_A(x)^{s+2}} \mathrm{d}\lambda_{d-1}, \quad (12)$$

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

Theorem 7.1 (General jump discontinuity [31])**.** *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,*

$$\sigma(d/2-1) + \frac{d/2-1}{24} V_Q'(d/2-1) = \alpha(d/2-1) = \lim_{\varepsilon\to 0^+} \sigma(d/2 - 1 + \varepsilon), \quad (13)$$

with V_Q as introduced in equation (12)*.*

7.2 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 (12) we obtain that

$$V_Q'(d/2-1) = -\frac{4\operatorname{tr}(A)}{d\sqrt{\det(A)}} \frac{\pi^{d/2}}{\Gamma(d/2)} - \int_{\|x\|_\infty = 1} \frac{\operatorname{tr}(A) Q_A(x) - d Q_{A^2}(x)}{Q_A(x)^{d/2+1}} \log Q_A(x) \mathrm{d}\lambda_{d-1}. \quad (14)$$

The equality is a consequence of [31, Lemma 2.5].

Example 7.2 (Recovery of cubic jump)**.** Let us demonstrate that Theorem 7.1 reduces to the result given in [31] in the cubic lattice case. In that case, $A = I$ and $\operatorname{tr}(A) = d$, so that the integral in (14), involving the logarithm, vanishes. Hence,

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

in agreement with the value given in [31]. ◇

In [31] 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 7.3 ([31])**.** *Let Q be a positive definite quadratic form with*

$$d\, Q_{A^2}(x) \leqslant \operatorname{tr}(A) Q_A(x) \quad (15)$$

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 (15).

Example 7.4 (Some non-cubic lattices)**.** Consider the case when $A = A_p := I - pE$, where E is the matrix with all entries equal to 1. One easily checks that A_p is positive definite if and only if $p < 1/d$. Hence, we assume $p < 1/d$. We further observe that

$$Q_{A_p}(x) = \|x\|_2^2 - p\left(\sum_{j=1}^{d} x_j\right)^2,$$

as well as $A_p^2 = A_{p(2-dp)}$. Thus equipped, a brief calculation reveals that

$$dQ_{A_p^2}(x) - \operatorname{tr}(A_p) Q_{A_p}(x) = pd\|x\|_2^2 - p\,[1 - (d-1)p] \left(\sum_{j=1}^{d} x_j\right)^2.$$

Notice that, by Hölder's inequality,

$$\left(\sum_{j=1}^{d} x_j\right)^2 \leqslant \|x\|_1^2 \leqslant d\|x\|_\infty \|x\|_2^2.$$

Assume further that $p \geqslant 0$ so that $p\,[1-(d-1)p] > 0$. We then find that, for all x with $\|x\|_\infty = 1$,

$$dQ_{A_p^2}(x) - \mathrm{tr}(A_p) Q_{A_p}(x) \geqslant p^2 d(d-1)\|x\|_2^2 \geqslant 0.$$

By Proposition 7.3, we have thus shown that $V_Q'(d/2-1) < 0$, with $Q = Q_{A_p}$, for all $0 \leqslant p < 1/d$. ◇

Proposition 7.3 can fail comprehensively if its conditions are not fully met:

Example 7.5 (Some scaled cubic lattices)**.** Consider the case when $A = A_p := I + pD(a)$, where $D(a) = D(a_1, \ldots, a_d)$ is a diagonal matrix with $t := \mathrm{tr}(D) = \sum_{k=1}^{d} a_k$ and without loss $p \geq 0$. Now A_p is positive definite if and only if $pa_k + 1 > 0$ for all $1 \leq k \leq d$. Suppose that $t = \mathrm{tr}(D) = 0$. Then $\mathrm{tr}(A_p) = d$, Also $A_p^2 = I + 2pD(a) + p^2 D(a_1^2, \ldots, a_k^2)$. Thence,

$$dQ_{A_p^2}(x) - \mathrm{tr}(A_p) Q_{A_p}(x) = pd \sum_{k=1}^{d} a_k(1+pa_k)x_k^2,$$

which must change signs on the sphere, since a_k does, and so Proposition 7.3 does not apply. ◇

On the basis of our analysis, motivated by cases such as Examples 7.4 and 7.5 where Proposition 7.3 does not apply, we were led to the conjecture below.

Conjecture 7.6 (Negative jumps)**.** *For all dimensions d and all positive definite forms Q, one has $V'(d/2-1) < 0$.*

7.3 Numerical exploration of $V_Q'(d/2-1)$

In an attempt to better understand Conjecture 7.6, we observed that the integral in (14) 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

$$\sum_{i=1}^{d} v_i(+) + \sum_{i=1}^{d} v_i(-) < \frac{4\,\mathrm{tr}(A)}{d\sqrt{\det(A)}} \frac{\pi^{d/2}}{\Gamma(d/2)} \qquad (16)$$

can ever fail. As the dimension grows, the cost of the required numerical integration increases substantially, as noted above in Section 5.1. Thus, we decided to computationally examine whether the ratio $L/R = \mathrm{LHS}/\mathrm{RHS}$ in (16) is always less than 1 for $d = 3$, which is the physically most meaningful case of Conjecture 7.6.

7.4 Numerical evidence for the Conjecture

In our tests of Conjecture 7.6, we programmed both sides of (16) using double-double arithmetic with the QD software package [52]. 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 thus also the eigenvalues of the test matrix $A = ODO^*$. The resulting 2-D integrals implicit in (16) were computed using the tanh-sinh quadrature scheme [23], [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 the table below, 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 7.6.

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

In future studies, we will attempt to test Conjecture 7.6 in four dimensions. These studies will likely require highly parallel computation in addition to high-precision arithmetic.

8 Requirements for future high-precision arithmetic software

It is clear from the preceding survey of applications that high-precision arithmetic facilities are indispensable for a growing body of numerically demanding applications spanning numerous disciplines, ranging from dynamical systems and experimental mathematics to computational physics.

Software tools for performing such computations, and tools for converting scientific codes to use high-precision arithmetic, are in general more efficient, useable and robust than they were in the past. Yet it is clear, from the examples we have presented and other experiences, that the presently available tools still have a long ways to go.

Even commercial packages, which in general are significantly more complete and robust 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 [10].

Here are some specific areas of needed improvement.

8.1 High-precision and emerging architectures

The scientific computing world is moving very rapidly into parallel computing, including multicore computing [77]. 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 5.1 and a numerical integration reported in [7].

But on modern multicore systems, parallel processing is more efficiently performed using a shared memory, threaded environment such as OpenMP [77] 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.

As it turns out, few if any high-precision software packages currently available are entirely thread-safe (or, at the least, it is not clear from the available documentation that they are thread-safe). Thus it is not possible to execute parallel programs using these high-precision packages in the most efficient parallel environments on multicore systems.

Another important development here is the recent emergence of graphics processing units (GPUs) and their usage for high-performance computing applications [77]. At the time of this writing, four of the top ten computer systems on the Top 500 list of the world's most powerful supercomputers incorporate GPUs [72]. Thus it is increasingly clear that future high-precision packages, whether they be integrated into commercial software or as augmentations to conventional programming environments, must be able to run, as an option, on GPU hardware.

8.2 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. This is one area where commercial packages such as *Maple* and *Mathematica* generally shine, but others generally fall short. Modern high-precision packages should support the following:

1) Basic transcendentals—exp, log, sin, cos, tan, hyperbolic functions—and the corresponding inverse functions [58, Sec. 4].
2) Gamma, digamma, polygamma, incomplete gamma, beta and incomplete beta functions [58, Sec. 5, 8].
3) Riemann zeta function, polylogarithms and Dirichlet L-functions [58, Sec. 25].
4) Bessel functions (first, second and third kinds, modified, etc.) [58, Sec. 10].
5) Hypergeometric functions [58, Sec. 15].
6) Airy functions [58, Sec. 9].
7) Elliptic integral functions [58, Sec. 19].
8) Jacobian elliptic functions and Weierstrass elliptic/modular functions [58, Sec. 22, 23].
9) Theta functions [58, 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 [32, pp. 215–245], [38], [41], [58]—but these published schemes need to be implemented in publicly available high-precision computing environments.

8.3 Reproducibility

As we noted above in Section 1, reproducibility is increasingly important in scientific computing. 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, and round-off errors inevitably accumulate. Many 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 parallel environment, where deterministic order of execution often cannot be guaranteed.

One benefit of high-precision arithmetic is to enhance reproducibility of results, since it can dramatically reduce floating-point roundoff error. Some examples were mentioned in Section 1. In most cases, by appropriately adjusting the level of precision, high levels of reproducibility can be achieved.

Some researchers are investigating facilities to guarantee bit-for-bit reproducibility in IEEE floating-point arithmetic across different platforms, although for the time they feature only partial reproducibility [39]. Thus at some point similar facilities may be provided in high-precision computing systems as well.

However, bit-for-bit reproducibility, in addition to possibly slowing down execution, has the potential downside of masking serious numerical difficulties. Often numerical difficulties in a code only come to light when a minor code change or a run on a different number of processors produces a surprisingly large change in the results. Ensuring bit-for-bit reproducibility might hide such problems from the user.

In any event, while bit-for-bit reproducibility may eventually be realized in high-precision computing, it is not likely to appear anytime soon. Thus for the foreseeable future, users of these libraries must be willing to accept some level of non-reproducibility and manage their computations accordingly.

REFERENCES

[1] A. Abad, R. Barrio, F. Blesa and M. Rodriguez, "TIDES: a Taylor series Integrator for Differential EquationS," *ACM Trans. on Math. Software*, to appear (2012). Software available online at http:gme.unizar.es/software/tides.

[2] A. Abad, R. Barrio, and A. Dena, "Computing periodic orbits with arbitrary precision," *Phys. Rev. E*, 84 (2011), 016701.

[3] D. H. Bailey, "A compendium of BBP-type formulas," Apr. 2011, available at http://www.davidhbailey.com/dhbpapers/bbp-formulas.pdf. An interactive database is online at http://bbp.carma.newcastle.edu.au.

[4] D. H. Bailey, "Resolving numerical anomalies in scientific computation," Apr. 2015, available at http://www.davidhbailey.com/dhbpapers/numerical-bugs.pdf.

[5] D. H. Bailey and J. M. Borwein, "Compressed lattice sums arising from the Poisson equation: Dedicated to Professor Hari Sirvastava," *Boundary Value Problems*, 75 (2013), DOI: 10.1186/1687-2770-2013-75, http://www.boundaryvalueproblems.com/content/2013/1/75.

[6] D. H. Bailey and J. M. Borwein, "Experimental mathematics: Examples, methods and implications," *Notices of the AMS*, 52 (May 2005), 502-514.

[7] D. H. Bailey and J. M. Borwein, "Highly parallel, high-precision numerical integration," Apr. 2008, available at http://www.davidhbailey.com/dhbpapers/quadparallel.pdf.

[8] D. H. Bailey and J. M. Borwein, "Hand-to-hand combat with thousand-digit integrals," *J. of Computational Science*, 3 (2012), 77–86.

[9] D. H. Bailey and J. M. Borwein, "Nonnormality of Stoneham constants," Ramanujan J., 29 (2012), 409-422; DOI 10.1007/s11139-012-9417-3.

[10] D. H. Bailey, J. M. Borwein and R. E. Crandall, "Computation and theory of extended Mordell-Tornheim-Witten sums," *Mathematics of Computation*, to appear, Jul 2012, http://www.davidhbailey.com/dhbpapers/BBC.pdf.

[11] D. H. Bailey, J. M. Borwein, R. E. Crandall and J. Zucker, "Lattice sums arising from the Poisson equation," *J. Physics A: Math. and Theor.*, 46 (2013), 115201.

[12] D. H. Bailey, R. Barrio and J. M. Borwein, "High-precision computation: Mathematical physics and dynamics," *Appl. Math. and Computation*, 218 (2012), 10106–10121.

[13] D. H. Bailey, P. B. Borwein and S. Plouffe, "On the rapid computation of various polylogarithmic constants," *Math. of Computation*, 66 (Apr 1997), 903–913.

[14] D. H. Bailey, J. M. Borwein and R. E. Crandall, "Integrals of the Ising class," *J. Physics A: Math. and Gen.*, 39 (2006), 12271–12302.

[15] D. H. Bailey, D. Borwein, J. M. Borwein and R. E. Crandall, "Hypergeometric forms for Ising-class integrals," *Exp. Mathematics*, 16 (2007), 257–276.

[16] D. H. Bailey, J. M. Borwein, D. M. Broadhurst and L. Glasser, "Elliptic integral representation of Bessel moments," *J. Phys. A: Math. and Theor.*, **41** (2008), 5203–5231. DOI 205203 (IoP Select).

[17] D. H. Bailey, J. M. Borwein, A. Mattingly and G. Wightwick, "The computation of previously inaccessible digits of π^2 and Catalan's constant," *Notices of the AMS*, 60 (2013), no. 7, 844–854.

[18] D. H. Bailey, J. M. Borwein and V. Stodden, "Set the default to 'open'," *Notices of the AMS*, 60 (6) (2013), 679–680.

[19] D. H. Bailey and D. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Math. of Computation*, 70 (2000), 1719–1736.

[20] D. H. Bailey and R. E. Crandall, "On the random character of fundamental constant expansions," *Exp. Mathematics*, 10 (2001), 175–190.

[21] D. H. Bailey and R. E. Crandall, "Random generators and normal numbers," *Exp. Mathematics*, 11 (2004), 527–546.

[22] D. H. Bailey and A. M. Frolov, "Universal variational expansion for high-precision bound-state calculations in three-body systems. Applications to weakly-bound, adiabatic and two-shell cluster systems," *J. Physics B*, 35 (2002), 42870–4298.

[23] D. H. Bailey, X. S. Li and K. Jeyabalan, "A comparison of three high-precision quadrature schemes," *Exp. Mathematics*, 14 (2005), 317–329.

[24] D. H. Bailey, X. S. Li and B. Thompson, "ARPREC: An arbitrary precision computation package," Sep 2002, http://www.davidhbailey.com/dhbpapers/arprec.pdf.

[25] D. H. Bailey and M. Misiurewicz, "A strong hot spot theorem," *Proc. of the AMS*, 134 (2006), 2495–2501.

[26] R. Barrio, "Performance of the Taylor series method for ODEs/DAEs," *Appl. Math. Comput.*, 163 (2005), 525–545.

[27] R. Barrio, "Sensitivity analysis of ODEs/DAEs using the Taylor series method," *SIAM J. Sci. Computing*, 27 (2006), 1929–1947.

[28] R. Barrio, F. Blesa, M. Lara, "VSVO formulation of the Taylor method for the numerical solution of ODEs," *Comput. Math. Appl.*, 50 (2005), 93–111.

[29] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, H. Ita, D. A. Kosower and D. Maitre, "An automated implementation of on-shell methods for one-loop amplitudes," *Phys. Rev. D*, 78 (2008), 036003, http://arxiv.org/abs/0803.4180.

[30] D. Borwein, J. M. Borwein, and R. Shail, "Analysis of certain lattice sums," *J. Math. Anal. and Appl.*, 143 (1989), 126–137.

[31] D. Borwein, J. M. Borwein, and A. Straub, "On lattice sums and Wigner limits," preprint, July 2013.

[32] J. M. Borwein and D. H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, A.K. Peters, Natick, MA, second edition, 2008.

[33] J. M. Borwein and D. H. Bailey, *Experimentation in Mathematics: Computational Paths to Discovery*, A.K. Peters, Natick, MA, 2004.

[34] J. M. Borwein, L. Glasser, R. McPhedran, J. G. Wan, and I. J. Zucker, *Lattice Sums: Then and Now*, Cambridge University Press, 2013.

[35] J. M. Borwein, A. Straub, and J. Wan, "Three-step and four-step random walk integrals," *Exp. Mathematics*, 22 (2013), 1–14.

[36] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge Univ. Press, 2010.

[37] F. Calvo et al., "Evidence for Low-Temperature Melting of Mercury owing to Relativity", *Angew. Chem. Intl. Ed. Engl.* (2013) doi: 0.1002/anie.201302742.

[38] S. Chevillard and M. Mezzarobba, "Multiple precision evaluation of the Airy Ai function with reduced cancellation," *Proceedings of the 21st IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2013.

[39] M. J. Corden and D. Kreitzer, "Consistency of floating-point results using the Intel compiler, or why doesn't my application always give the same answer," Intel Corporation, 2010, http://software.intel.com/sites/default/files/article/164389/fp-consistency-102511.pdf.

[40] G. Corliss and Y. F. Chang, "Solving ordinary differential equations using Taylor series," *ACM Trans. Math. Software*, 8 (1982), 114–144.

[41] R. E. Crandall, "Unified algorithms for polylogarithm, L-series and zeta variants," Mar. 2012, http://www.perfscipress.com/papers/UniversalTOC25.pdf.

[42] M. Czakon, "Tops from light quarks: Full mass dependence at two-Loops in QCD," *Phys. Lett. B*, vol. 664 (2008), 307, http://arxiv.org/abs/0803.1400.

[43] J. Demmel and P. Koev, "The accurate and efficient solution of a totally positive generalized Vandermonde linear system," *SIAM J. of Matrix Analysis Appl.*, 27 (2005), 145–152.

[44] J. Dongarra, "LAPACK," http://www.netlib.org/lapack.

[45] J. Dongarra, "LINPACK," http://www.netlib.org/linpack.

[46] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov and G. Zanderighi, "One-loop amplitudes for W+3 jet production in hadron collisions," manuscript, 15 Oct 2008, http://arXiv.org/abs/0810.2762.

[47] H. R. P. Ferguson, D. H. Bailey and S. Arno, "Analysis of PSLQ, An Integer Relation Finding Algorithm," *Math. of Computation*, 68, no. 225 (Jan 1999), 351–369.

[48] T. Ferris, *Coming of Age in the Milky Way*, HarperCollins, New York, 2003.

[49] A. M. Frolov and D. H. Bailey, "Highly accurate evaluation of the few-body auxiliary functions and four-body integrals," *J. Physics B*, 36 (2003), 1857–1867.

[50] E. Hairer, S. Nørsett and G. Wanner, *Solving ordinary differential equations. I. Nonstiff problems*, second edition, Springer Series in Computational Mathematics, vol. 8, Springer-Verlag, Berlin, 1993.

[51] Y. He and C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *J. Supercomputing*, 18 (Mar 2001), 259–277.

[52] Y. Hida, X. S. Li and D. H. Bailey, "Algorithms for Quad-Double Precision Floating Point Arithmetic," *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2001.

[53] E. Karrels, "Computing digits of pi with CUDA," 14 Mar 2013, available at http://www.karrels.org/pi.

[54] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, third edition, 1998.

[55] G. Lake, T. Quinn and D. C. Richardson, "From Sir Isaac to the Sloan survey: Calculating the structure and chaos due to gravity in the universe," *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 1997, 1–10.

[56] S. K. Lucas and H. A. Stone, "Evaluating infinite integrals involving Bessel functions of arbitrary order," *J. Comp. and Appl. Math.*, 64 (1995), 217–231.

[57] E. Lorenz, "Deterministic nonperiodic flow," *J. Atmospheric Sci.*, 20 (1963), 130–141.

[58] *NIST Digital Library of Mathematical Functions*, version 1.0.6 (May 2013), http://dlmf.nist.gov.

[59] G. Ossola, C. G. Papadopoulos and R. Pittau, "CutTools: A program implementing the OPP reduction method to compute one-loop amplitudes," *J. High-Energy Phys.*, 0803 (2008), 042, http://arxiv.org/abs/0711.3596.

[60] W. H. Press, S. A. Eukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd edition, Cambridge University Press, 2007.

[61] R. W. Robey, J. M. Robey and R. Aulwes, "In search of numerical consistency in parallel programming," *Parallel Computing*, 37 (2011), 217–219.

[62] M. Rogers, J. G. Wan, and I. J. Zucker, "Moments of elliptic integrals and critical L-values," 2013, submitted, 15 pages.

[63] C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey and C. Iancu, "Precimonious: Tuning assistant for floating-point precision," *Proc. of SC13*, to appear, May 2013, http://www.davidhbailey.com/dhbpapers/precimonious.pdf.

[64] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discr. and Comp. Geometry*, 18 (1997), 305–363.

[65] A. Sidi, "The numerical evaluation of very oscillatory infinite integrals by extrapolation," *Math. of Computation*, 38 (1982), 517–529.

[66] A. Sidi, "A user-friendly extrapolation method for oscillatory infinite integrals," *Math. of Computation*, 51 (1988), 249–266.

[67] A. Sidi, "A user-friendly extrapolation method for computing infinite range integrals of products of oscillatory functions," *IMA J. Numer. Anal.*, 32 (2012), 6020–631.

[68] C. Simó, "Global dynamics and fast indicators," in *Global Analysis of Dynamical Systems*, 373–389, Inst. Phys., Bristol, 2001.

[69] V. Stodden, J. M. Borwein and D. H. Bailey, "Publishing Standards for Computational Science:'Setting the Default to Reproducible'," *SIAM News*, 46 no. 5 (June 2013), 4–6, available at http://www.siam.org/news/news.php?id=2078.

[70] 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," Jan. 2013, available at http://www.davidhbailey.com/dhbpapers/icerm-report.pdf.

[71] H. Takahasi and M. Mori, "Double exponential formulas for numerical integration," *Pub. RIMS*, Kyoto University, 9 (1974), 721–741.

[72] "June 2013 Top 500 list," June 2013, available at http://top500.org/lists/2013/06.

[73] D. Viswanath, "The Lindstedt-Poincaré technique as an algorithm for computing periodic orbits," *SIAM Review*, 43 (2001), 478–495.

[74] D. Viswanath, "The fractal property of the Lorenz attractor," *J. Phys. D*, 190 (2004), 115–128.

[75] D. Viswanath and S. Şahutŏglu, "Complex singularities and the Lorenz attractor," *SIAM Review*, 52 (2010), 294–314.

[76] J. Wan, "Moments of products of elliptic integrals," *Adv. in Appl. Math.* 48 (2012), no. 1, 121-141.

[77] S. W. Williams and D. H. Bailey, "Parallel computer architecture," in David H. Bailey, Robert F. Lucas and Samuel W. Williams, ed., *Performance Tuning of Scientific Applications*, CRC Press, Boca Raton, FL, 2011, 11–33.

[78] Z.-C. Yan and G. W. F. Drake, "Bethe logarithm and QED shift for Lithium," *Phys. Rev. Letters*, 81 (2003), 774–777.

[79] T. Zhang, Z.-C. Yan and G. W. F. Drake, "QED corrections of $O(mc^2\alpha^7 \ln \alpha)$ to the fine structure splittings of Helium and He-Like ions," *Physical Review Letters*, 77 (1994), 1715–1718.