Computer discovery and analysis of large Poisson polynomials

David H. Bailey* Jonathan M. Borwein† Jason S. Kimberley‡

with an Appendix by Watson Ladd§

May 10, 2016

Abstract

In two earlier studies of lattice sums arising from the Poisson equation of mathematical physics, it was established that the lattice sum $1/\pi \cdot \sum_{m,n \text{ odd}} \cos(m\pi x)\cos(n\pi y)/(m^2+n^2) = \log A$, where A is an algebraic number, and explicit minimal polynomials associated with A were computed for a few specific rational arguments x and y. Based on these results, one of us (Kimberley) conjectured a number-theoretic formula for the degree of A in the case $x = y = 1/s$ for some integer s.

These earlier studies were hampered by the enormous cost and complexity of the requisite computations. In this study, we address the Poisson polynomial problem with significantly more capable computational tools. As a result of this improved capability, we have confirmed that Kimberley's formula holds for all integers s up to 52 (except for $s = 41, 43, 47, 49, 51$, which are still too costly to test), and also for $s = 60$ and $s = 64$. As far as we are aware, these computations, which employed up to 64,000-digit precision, producing polynomials with degrees up to 512 and integer coefficients up to 10^{229}, constitute the largest successful integer relation computations performed to date.

By examining the computed results, we found connections to a sequence of polynomials defined in a 2010 paper by Savin and Quarfoot. These investigations subsequently led to a proof, given in the Appendix, of Kimberley's formula and the fact that when s is even, the polynomial is palindromic (i.e., coefficient $a_k = a_{m-k}$, where m is the degree).

Keywords: multiprecision computation, MPFR, PSLQ algorithm, Poisson equation, parallel computing, experimental mathematics.

*Lawrence Berkeley National Laboratory (retired), Berkeley, CA 94720, and University of California, Davis, Davis, CA 95616, USA. E-mail: david@davidhbailey.com.

†CARMA, University of Newcastle, Callaghan, NSW 2308, Australia. E-mail: jon.borwein@gmail.com.

‡CARMA, University of Newcastle, Callaghan, NSW 2308, Australia. E-mail: jskcmg@gmail.com.

§Department of Mathematics, University of California, Berkeley, CA 94720. Email: wbl@berkeley.edu

1 Introduction

Lattice sums [9] and the Poisson equation, which naturally arise in studies of gravitational and electrostatic potentials, have been studied for many years in the mathematical physics community, for example in [9, 17, 18] and also in a 2011 study of cyclotomic polylogarithms and corresponding multiple harmonic sums [1]. Recently interest in this topic has been rekindled in light of some intriguing applications to practical image processing [15, 5] (although not needed for his deblurring algorithm, Crandall observed that each pixel had the form (1) below). These developments have underscored the need to better understand the underlying theory behind both lattice sums and the associated Poisson potential functions.

In two earlier studies [5, 6] and [9, §7.2], two of the present authors, together with Richard Crandall (deceased 2012) and I. J. Zucker, analyzed the simple and accessible two-dimensional case:

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

By employing a computational approach [5, 6], these studies empirically discovered and then proved the intriguing fact that when x and y are rational numbers,

$$\phi_2(x,y) = \frac{1}{\pi} \log A, \tag{2}$$

where A is *algebraic* (i.e., A is the root of an m-th degree polynomial with integer coefficients, for some integer m). In particular, given rationals x and y, the constant $\alpha = \exp(8\pi\phi_2(x,y))$ was computed to high precision (an '8' was inserted here in light of formulas such as (48) and (49) of [5], which made it significantly easier to recover the polynomial). Then for a given m, the $(m+1)$-long vector $(1, \alpha, \alpha^2, \cdots, \alpha^m)$ was computed to high precision, and a version of the PSLQ algorithm was used to discover the coefficients of the polynomial.

In Table 1 we present a few of the results from [5]. Among other things, note that for s even, the resulting polynomial is always palindromic (i.e., coefficient $a_k = a_{m-k}$, where m is the degree). Does this pattern extend to higher cases?

s	p_s, the minimal polynomial corresponding to $x = y = 1/s$:
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	$1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8$

Table 1: Sample of polynomials produced in earlier study [5].

In another observation from this data, one of us (Kimberley) observed that the degree $m(s)$ of the minimal polynomial associated with the case $x = y = 1/s$ appears to be given

by the following formula. Set $m(2) = 1/2$. Otherwise for primes p congruent to 1 modulo 4, set $m(p) = (p-1)^2/4$, and for primes p congruent to 3 modulo 4, set $m(p) = (p^2-1)/4$. Then for any other positive integer s whose prime factorization is $s = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$,

$$m(s) \stackrel{?}{=} 4^{r-1} \prod_{i=1}^{r} p_i^{2(e_i-1)}\, m(p_i). \tag{3}$$

This sequence now appears as `http://oeis.org/A218147` in the *Online Encyclopedia of Integer Sequences*. Does Kimberley's formula hold for any or all higher s?

The earlier studies were hampered by the enormous cost and complexity of the requisite computations, which in some cases required up to 12,000-digit precision, and in numerous other cases, such as when $x = y = 1/31$, failed altogether. These computations were extremely demanding on hardware, multiprecision software and application code, since the slightest error or inaccuracy (except for a few trailing bits) at any stage of the computation almost certainly results in failure to recover the polynomial. Can these computations be done better or faster?

In this study, we address the Poisson polynomial problem with significantly more capable tools: (a) a new thread-safe, high-level arbitrary precision package, which is approximately 3X faster than the package used by the previous studies; (b) a new three-level multipair PSLQ integer relation scheme, which is approximately 4X faster than the scheme used in the previous studies; and (c) a parallel implementation on a 16-core system. These enhancements resulted in a combined speedup of approximately 156X over the software used in the previous studies (based on one typical case, running on common hardware). As a result of this improved capability, we have confirmed that Kimberley's formula holds for all integers s up to 52 (except for $s = 41, 43, 47, 49, 51$, which are still too costly to test), and also for $s = 60$ and $s = 64$. As far as we are aware, these computations, which employed up to 64,000-digit precision, producing polynomials with degrees up to 512 and integer coefficients up to 10^{229}, constitute the largest successful integer relation computations performed to date.

By examining the computed results, we found connections to a sequence of polynomials defined in a 2010 paper by Savin and Quarfoot. These investigations subsequently led to a proof, given in the Appendix, of Kimberley's formula and the fact that when s is even, the polynomial is palindromic (i.e., coefficient $a_k = a_{m-k}$, where m is the degree).

2 Computational software

Scientific computing is moving rapidly into multicore and multi-node parallel computing, because the performance of individual processors is no longer rapidly increasing [3]. Thus future improvements in performance will require aggressive exploitation of parallelism. In high-precision computing, it is difficult to achieve significant parallel speedup *within* a single high-precision arithmetic operation, but parallelization at the *application* level (e.g., parallelizing a DO loop containing multiprecision operations) is an attractive option.

On modern systems that feature multicore processors, parallel computing is more efficiently performed using a shared memory, multithreaded environment such as OpenMP, even if the Message Passing Interface (MPI) is employed between nodes [3, p. 11–33]. Furthermore, algorithms such as multipair PSLQ can only be parallelized efficiently at a rather low loop level. Computations that use a thread-parallel environment such as OpenMP must be

entirely "thread-safe," which means, among other things, that there are no read/write global variables or arrays, because otherwise there may be difficulties during parallel execution.

Many high-precision software packages employ global read/write data, such as the current working precision and data to support transcendental functions. The working precision is particularly troublesome, since in many cases it is dynamically changed by both users and the software package itself. Until recently, only one high-level arbitrary precision floating-point package was certified thread-safe, namely the MPFR C++ package [19], which is built upon the low-level MPFR package [16] (presuming MPFR is installed using the thread-safe build option). The MPFR package is very well-designed, features correct rounding, includes numerous transcendental and special functions, and achieves the the fastest overall timings among currently available arbitrary precision floating-point packages [14].

2.1 A new arbitrary precision package

The previous two studies employed the ARPREC arbitrary precision software [2], which is not thread-safe and thus cannot be used for multithreaded parallel runs. For this study (and future use), we employed a new package developed by one of us for arbitrary precision floating-point computation. The package, known as MPFUN2015 [4] is available in two versions, which are "plug-compatible" with each other: (a) a self-contained, all-Fortran version; and (b) a version that calls the MPFR package for low-level operations, which is approximately 3X faster. This package is targeted to Fortran applications (since many of our high-precision codes are in Fortran), but a C++ version is also being developed.

A key design goal for this package is thread safety. This is achieved by studiously avoiding any global read/write data, and by incorporating the current working precision into the data structure of each multiprecision variable and array element (this is also a feature of MPFR). When an operation or function is performed involving two or more operands, the working precision of the result is assigned to be the maximum of the precision levels of the operands. Special functions are provided to inquire the working precision assigned to a particular variable or array element, or to change the assigned level when needed.

Both versions include a full-featured Fortran interface (via custom datatypes and operator overloading), which supports multiprecision real and complex data, numerous transcendental and special functions, as well as a wide variety of mixed mode operations (e.g., double precision times multiprecision real, multiprecision complex to an integer power, etc.). Thus, for example, one can write

```
d = a + b(i) * cos(3.d0 + c(i,j))
```

where `a, b, c` and `d` are multiprecision real, and the appropriate underlying multiprecision routines are automatically called by the compiler.

Both versions also detect, and provide means to overcome, accuracy problems rooted in the usage of inexact double-precision constants and expressions, which is a problem that has plagued high-precision computation for years. For example, if a user code includes a statement such as `a = b + d / 3.d0`, where `a` and `b` are multiprecision real and `d` is double precision, then by established language conventions in Fortran, C, C++ and most other languages, the division is only performed to double precision accuracy, and this low accuracy propagates to the result `a`. Our package employs special software that detects such instances at *execution time*.

The package also provides means to correct such usage. For example, to generate the full-precision conversion of the decimal constant 1.234 and assign it to the multiprecision

real `a`, it suffices to enclose the constant in apostrophes, e.g., `a = '1.234'`.

In the present study, we employed the faster (MPFR-based) version of the package. Full details on the algorithms, design, installation and usage of this software are given in a technical report [4].

2.2 Three-level multipair PSLQ

Given an $(m+1)$-long input vector $X = (x_i)$, an integer relation algorithm attempts to find a nontrivial $(m+1)$-long vector of integers $A = (a_i)$ such that

$$a_0 x_0 + a_1 x_1 + a_2 x_2 + \cdots + a_m x_m = 0. \tag{4}$$

If $X = (1, \alpha, \alpha^2, \cdots, \alpha^m)$ for some α, and if an integer relation is found for X, then these integers are the coefficients of a polynomial of degree m satisfied by α.

The multipair PSLQ algorithm is a more efficient and moderately parallelizable variant of PSLQ, the most widely used integer relation algorithm (although some use a variant of LLL) [7, 12]. Iterations of the multipair PSLQ algorithm develop a sequence of invertible integer matrices A_n, their inverses B_n and real matrices H_n (in lower quadrature form), so that the reduced vector $Y = B_n \cdot X$ has steadily smaller entries, until one entry of Y is smaller than the epsilon specified for detection. Integer relation detection (by any algorithm) requires very high precision: at least $(m+1) \cdot \max_k \log_{10} |a_k|$ digits, or there is no chance of finding the underlying relation. Multipair PSLQ is extremely efficient with precision, compared with other integer relation algorithms, in the sense that it can typically detect a relation when the numeric precision is only a few percent higher than this minimum level [7].

The earlier studies [5, 6] employed a two-level version of the multipair PSLQ algorithm. For this study, we employed a three-level version, based on a scheme sketched (with scant details) in [7]: (a) double precision; (b) medium precision, typically 100-2000 digits; and (c) full precision, typically many thousands of digits. With this scheme, almost all iterations of the multipair PSLQ algorithm are performed in double precision. When an entry of the double precision Y vector is smaller than 10^{-14}, or when an entry in the double precision A or B arrays exceeds $2^{53} \approx 9.007 \cdot 10^{15}$ (so that several iterations must be repeated with higher precision), the medium precision arrays are updated using matrix multiplication via

$$Y := \hat{B} \cdot Y,\ B := \hat{B} \cdot B,\ A := \hat{A} \cdot A,\ H := \hat{A} \cdot H, \tag{5}$$

where the hat notation indicates the double precision arrays. When an entry of the medium precision Y vector is smaller than an epsilon corresponding to medium precision, then the full precision arrays are updated using similar formulas. *Substantial care must be taken to manage this three-level hierarchy, and to correctly handle numerous atypical scenarios.*

3 Computational algorithm

Here is the specific procedure we employed to discover the Poisson polynomials:

1. Given rationals $x = p/s$ and $y = q/s$, select a conjectured minimal polynomial degree $m(s)$ and other parameters for the run. For planning computer runs, we employed these empirically derived estimates: D is the detection level; P_1 is the medium precision level in digits; P_2 is the full precision level in digits; N is the number of multipair PSLQ

iterations; T is the total run time in core-seconds; and M is the memory requirement in Mbytes:

$$\begin{aligned}
m\,(s) &= 4^{r-1} \prod_{i=1}^{r} p_i^{2(e_i-1)}\, m(p_i) \qquad \text{i.e., the Kimberley formula (3),} \\
\log_{10} D &= -0.462\, m^2(s), \quad P_1 = 5\, m(s), \quad P_2 = 5\, m(s) + 0.462\, m^2(s), \\
\log_{10} N &= 105.278/m^2(s) - 33.1073/m(s) + 4.01963 + 0.00696322\, m(s), \\
\log_{10} T &= 374.411/m^2(s) - 79.2388/m(s) + 1.93981 + 0.0176301\, m(s), \\
M &= 1.6 \cdot 10^{-7} \cdot m^2(s)(5m(s) + 0.462\, m^2(s)).
\end{aligned} \tag{6}$$

2. Calculate $\phi_2(x, y)$ to P_2-digit precision using the following Jacobian formula from [5]:

$$\phi_2(x, y) = \frac{1}{2\pi} \log \left| \frac{\theta_2(z, q)\theta_4(z, q)}{\theta_1(z, q)\theta_3(z, q)} \right|, \tag{7}$$

where $q = e^{-\pi}$ and $z = \frac{\pi}{2}(y + ix)$. Compute the four theta functions using the following rapidly convergent definitional formulas from [8, p. 52]:

$$\begin{aligned}
\theta_1(z, q) &= 2 \sum_{k=1}^{\infty} (-1)^{k-1} q^{(2k-1)^2/4} \sin((2k-1)z), \\
\theta_2(z, q) &= 2 \sum_{k=1}^{\infty} q^{(2k-1)^2/4} \cos((2k-1)z), \\
\theta_3(z, q) &= 1 + 2 \sum_{k=1}^{\infty} q^{k^2} \cos(2kz), \\
\theta_4(z, q) &= 1 + 2 \sum_{k=1}^{\infty} (-1)^k q^{k^2} \cos(2kz).
\end{aligned} \tag{8}$$

When done, calculate $\alpha = \exp(8\pi\phi_2(x, y))$ and generate the $(m + 1)$-long vector $X = (1, \alpha, \alpha^2, \cdots, \alpha^m)$, to P_2-digit precision. Note that formulas (7) and (8) involve sines and cosines of complex arguments, since z is complex. However, our multiprecision software includes full support for the multiprecision complex datatype, so these formulas were implemented simply as stated in (7) and (8).

3. Apply the three-level multipair PSLQ algorithm to X. For larger problems, employ a parallel version of the three-level multipair PSLQ code, using the OpenMP `DO PARALLEL` construct to perform certain time-intensive loops in parallel.
4. If a numerically significant relation (i.e., a relation that holds to at least 100 digits beyond the level needed to discover it) is not found, try again with a larger degree m or higher precision P_2. If a relation is found, employ the polynomial factorization facilities in *Mathematica* or *Maple* to ensure that the resulting polynomial is irreducible.

The high-level program that includes the computation of $\phi_2(x, y)$ and the three-level multipair PSLQ scheme, as specified in steps 1 through 4 above, is approximately 2,300 lines of Fortran; it calls our multiprecision package (approximately 12,000 lines of Fortran); it calls the MPFR package (approximately 93,000 lines of C); and it calls the GMP package (approximately 83,000 lines of C); for a total of approximately 190,000 lines of code.

4 Results and analysis

Table 2 shows results for our attempts to find the minimal polynomial associated with $x = y = 1/s$, for various positive integers s. Table 3 shows results for the cases $x = 1/s,\ y = q/s$, where $q > 1$ is the smallest integer that is relatively prime to s (our experimentation indicated that all q relatively prime to s have similar behavior). In Table 3, where the resulting degree m differs from the corresponding entry in Table 2, the degree is shown in italics.

It is clear from these tables that computational costs (more than 99% of which is due to multipair PSLQ) increase very rapidly with the polynomial degree m. Timings range from a fraction of a second for $x = y = 1/10$, to 7.74 million core-seconds for $x = y = 1/37$. For those cases that were also computed in [5], these timings represent significant speedups. For example, in the case $x = y = 1/32$, the authors of [5] reported 163,663 seconds, whereas here we report 5,126 seconds, which is 31.9X faster; for the case $x = y = 1/23$, the authors of [5] reported 212,635 seconds, whereas here we report 5,063 seconds, which is 42X faster (single core timings). However, these speedups are due in part to different software.

In Table 5, we present more carefully controlled comparative timings for the particular case $x = y = 1/35$, using (a) a 2-level multipair PSLQ scheme with the ARPREC multiprecision software, as used in [5], using code available at [2]; (b) the same 2-level code, but with our new multiprecision software; and (c) our 3-level multipair PSLQ code, with our new multiprecision package and with 1, 2, 4, 8, and 16 cores. Each run was performed on a 2.4 GHz Apple MacPro system with 16 cores (it is advertised as an 8-core system, but since it can handle 16 simultaneous threads, we consider it to be a 16-core system). The runs were executed in a typically busy environment with similar jobs running on other cores, using a common version (5.1.0) of the GNU gfortran and gcc compilers. We observe a speedup of approximately 3X for switching from ARPREC to our new arbitrary precision software, an additional speedup of approximately 4.2X for switching to our 3-level multipair PSLQ algorithm, and an additional speedup of approximately 12.1X for running in parallel on 16 cores, thus yielding an overall speedup of approximately 156X.

While a parallel speedup of 12X and an overall speedup of 156X are certainly most welcome (and we are trying to further accelerate these codes), it does not appear possible at the present time to efficiently employ hundreds or thousands of processors on these problems. It is fairly easy to achieve large parallel speedups in the multiprecision portions of the code (e.g., the multiprecision matrix multiply routine), but it is not easy to achieve large speedups within the double precision portion (e.g., the double precision multipair PSLQ iteration routine, which is called many thousands of times). Thus the double precision portions limit overall parallel scalability, as a consequence of *Amdahl's law* [3, p. 348]. It would be easy to exhibit large parallel speedups by starting, say, with the one-level multipair PSLQ code, which performs all iterations using full precision, but this would violate the principle that parallel implementations and speedup figures should be based on the most efficient single-threaded algorithm available; otherwise speedup results can be highly misleading [3, p. 1–9].

4.1 Minimal polynomials

As mentioned earlier, these computations, which employed up to 64,000-digit precision, producing polynomials with degrees up to 512 and integer coefficients up to 10^{229}, constitute the largest successful integer relation computations performed to date (as far as we are aware).

Maple 18 quickly confirmed irreducibility for each of the polynomials produced by our

program, and found that each of the polynomials splits in some small quadratic extension field (which is strong evidence that the polynomials are error-free). *Mathematica* 10.2 also confirmed irreducibility in each case, although it took an unusually large amount of time to handle the cases $x = y = 1/48$ and $x = y = 1/64$. (Wolfram Research staff informed us that this problem is resolved in their development code, and so it may be faster in the future.)

Table 4 shows one representative minimal polynomial, namely the 192-degree minimal polynomial found by our program for the case $x = y = 1/35$, in a very tiny font. It is entirely typical of Poisson minimal polynomials, in that the initial coefficient is -1, subsequent coefficients ascend to a maximum size (in this case roughly 10^{85}), and then descend back to -1. This semi-elliptical pattern, with 1 or -1 at the ends, is very strong numerical evidence that the polynomial produced by the computer program is the true mathematical minimal polynomial associated with this case, and, by implication, that all hardware, software and application code performed flawlessly, since otherwise it is *exceedingly unlikely* that the resulting coefficients would have this distinctive pattern. (In cases where the multipair PSLQ program fails to find a numerically significant relation, the resulting coefficients typically are all roughly the same size.) This neatly illustrates the role of visual output for large data.

With regards to the key objective mentioned above, namely to test Kimberley's formula for significantly higher arguments, we note that this formula was affirmed in every case $x = y = 1/s$, for s up to 52 (except for $s = 41, 43, 47, 49, 51$, which are still too costly to test), and also for $s = 60$ and $s = 64$.

5 Initial observations regarding Poisson polynomials

One observation from our computational results is that in each case where $x = y = 1/s$ for s even, the corresponding Poisson polynomial $p_s(\alpha)$ is palindromic, namely $a_k = a_{m-k}$, where a_k is the coefficient of α^k. Here, for instance, is p_{16}:

$$\begin{aligned} p_{16}(\alpha) = {} & 1 - 1376\alpha - 12560\alpha^2 - 3550496\alpha^3 + 81241720\alpha^4 - 169589984\alpha^5 \\ & + 1334964944\alpha^6 - 24307725984\alpha^7 + 238934926108\alpha^8 - 1043027124704\alpha^9 \\ & + 2328675366384\alpha^{10} - 3219896325280\alpha^{11} + 4238551472456\alpha^{12} \\ & - 10247414430048\alpha^{13} + 28552105805904\alpha^{14} - 55832851687968\alpha^{15} \\ & + 70020268309062\alpha^{16} \\ & - 55832851687968\alpha^{17} + 28552105805904\alpha^{18} - 10247414430048\alpha^{19} \\ & + 4238551472456\alpha^{20} - 3219896325280\alpha^{21} + 2328675366384\alpha^{22} \\ & - 1043027124704\alpha^{23} + 238934926108\alpha^{24} - 24307725984\alpha^{25} + 1334964944\alpha^{26} \\ & - 169589984\alpha^{27} + 81241720\alpha^{28} - 3550496\alpha^{29} - 12560\alpha^{30} - 1376\alpha^{31} + \alpha^{32}, \end{aligned} \tag{9}$$

where we have grouped the terms so that the palindromic pattern is evident.

Following one of our first presentations of these results, Nitya Mani, a student at Stanford University, reminded us that if α is a root of a palindromic polynomial such as this, then $\alpha + 1/\alpha$ is a root of a transformed polynomial of half the degree. For example, if α is a root of the degree-32 polynomial p_{16} given in (9), then $\beta = \alpha + 1/\alpha$ is a root of the degree-16

polynomial

$$\begin{aligned}
&17211171340288 - 34105850331136\beta + 287634682019 84\beta^2 - 8030085316608\beta^3 \\
&- 5013017608192\beta^4 + 3437397704704\beta^5 + 454816964608\beta^6 - 831166291968\beta^7 \\
&+ 229974967808\beta^8 - 22672640000\beta^9 + 359096832\beta^{10} - 123557376\beta^{11} \\
&+ 81417664\beta^{12} - 3529856\beta^{13} - 12576\beta^{14} - 1376\beta^{15} + \beta^{16}. \qquad (10)
\end{aligned}$$

Since the computational cost scales very rapidly as the degree of the polynomial is increased (see formula (6)), this palindromic property, which appears to hold for all even s, can be used to significantly accelerate the computation of p_s when s is large. In particular, the computational algorithm as given in Section 3 only needs to be changed in Step 2, where one computes $\beta = \alpha + 1/\alpha$ and $X = (1, \beta, \beta^2, \cdots, \beta^m)$, and in Step 3, where after recovering a polynomial in β, one must then expand the polynomial to obtain the equivalent polynomial in α. We implemented this scheme to obtain Poisson polynomials for the cases $s = 60$ and $s = 64$, and again found that the degree as predicted by Kimberley's formula matched the degrees (256 and 512, respectively) of the polynomials produced by the program. See Table 2 for details.

After some additional examination of our computer output, we observed

Conjecture 1. *For the case $x = y = 1/s$:*

- *The algebraic number α_s is the* largest *real root of the associated polynomial $p_s(\alpha)$;*
- *That polynomial has $\varphi(s)$ real roots, where φ is the Euler totient function.*
- *α_s appears to be monotone in s.*

6 Further observations regarding Poisson polynomials

Following examination of the polynomials p_{11} and p_{13}, namely

$$\begin{aligned}
p_{11}(\alpha) = &1 + 1210\alpha - 33033\alpha^2 + 2923492\alpha^3 + 5093605\alpha^4 - 385382514\alpha^5 \\
&+ 3974726283\alpha^6 - 14323974808\alpha^7 + 57392757037\alpha^8 - 291359180310\alpha^9 \\
&+ 948497199067\alpha^{10} - 1642552094436\alpha^{11} + 1084042069649\alpha^{12} \\
&+ 1890240552750\alpha^{13} - 6610669151537\alpha^{14} + 9712525647792\alpha^{15} \\
&- 8608181312269\alpha^{16} + 5384207244702\alpha^{17} - 3223489742187\alpha^{18} \\
&+ 2175830922716\alpha^{19} - 1197743580033\alpha^{20} + 387221579866\alpha^{21} \\
&- 50897017743\alpha^{22} - 7864445336\alpha^{23} + 5391243935\alpha^{24} - 815789634\alpha^{25} \\
&+ 28366041\alpha^{26} - 5092956\alpha^{27} + 207691\alpha^{28} + 2794\alpha^{29} - 11\alpha^{30} \qquad (11)
\end{aligned}$$

$$
\begin{aligned}
p_{13}(\alpha) = &-1 - 2388\alpha + 61098\alpha^2 - 19225300\alpha^3 - 606593049\alpha^4 + 1543922656\alpha^5 \\
&- 7856476560\alpha^6 + 221753896032\alpha^7 - 1621753072244\alpha^8 \\
&+ 4542779886736\alpha^9 - 2731418674664\alpha^{10} - 36717669656304\alpha^{11} \\
&+ 200879613202428\alpha^{12} - 547249607666784\alpha^{13} + 934179604482832\alpha^{14} \\
&- 1235038888776160\alpha^{15} + 1788854212778642\alpha^{16} - 3018407750933816\alpha^{17} \\
&+ 4349780716415868\alpha^{18} - 4419228090228152\alpha^{19} + 2899766501472914\alpha^{20} \\
&- 931940880451552\alpha^{21} - 413258559018224\alpha^{22} + 857795672629664\alpha^{23} \\
&- 659989056851972\alpha^{24} + 304241349909008\alpha^{25} - 87636987790824\alpha^{26} \\
&+ 14593362219920\alpha^{27} - 1073204980340\alpha^{28} - 45138167200\alpha^{29} \\
&+ 23660433008\alpha^{30} - 2028597792\alpha^{31} + 29540327\alpha^{32} - 3238420\alpha^{33} + 73386\alpha^{34} \\
&+ 492\alpha^{35} - \alpha^{36},
\end{aligned} \tag{12}
$$

and after doing some Google searches on these coefficients, we found that the coefficient 387221579866 in p_{11} appears in a 2010 preprint by Savin and Quarfoot of the University of Utah [20], and the coefficient 221753896032 in p_{13} appears in a manuscript, also dated 2010, by Bostan, Boukraa, Hassani, Maillard, Weil, Zenine and Abarenkova [10], subsequently published as [11]. This is a dramatic illustration of how advanced computation can facilitate data mining of the literature.

Savin and Quarfoot [20], define a sequence ψ_n of polynomials in x and y, based on the curve $y^2 = x^3 + x$, as follows:

$$
\begin{aligned}
\psi_1 &:= 1 \\
\psi_2 &:= 2y \\
\psi_3 &:= 3x^4 + 6x^2 - 1 \\
\psi_4 &:= 2y(2x^6 + 10x^4 - 10x^2 - 2),
\end{aligned} \tag{13}
$$

and, recursively,

$$
\begin{aligned}
\psi_{2n+1} &:= \psi_{n+2} \cdot \psi_n^3 - \psi_{n-1} \cdot \psi_{n+1}^3 \quad \text{for } n \geq 2 \\
\psi_{2n} &:= 1/(2y) \cdot \psi_n(\psi_{n+2} \cdot \psi_{n-1}^2 - \psi_{n-2} \cdot \psi_{n+1}^2) \quad \text{for } n \geq 3.
\end{aligned} \tag{14}
$$

We construct a related sequence J_s of integer coefficient polynomials in a by setting $x = \sqrt{-a}$, and so $y^2 = x(x^2 + 1) = \sqrt{-a}(1 - a)$; we also remove the leading $2y$ from ψ_{2n}:

$$
\begin{aligned}
J_{2n+1}(a) &:= \psi_{2n+1}(x, y) \\
J_{2n}(a) &:= 1/(2y) \cdot \psi_{2n}(x, y)
\end{aligned} \tag{15}
$$

Substituting (15) into (13), the initial values of $J_s(a)$ are

$$
\begin{aligned}
J_1 &= 1 \\
J_2 &= 1 \\
J_3 &= 3a^2 - 6a - 1 \\
J_4 &= 2a^3 - 10a^2 - 10a + 2.
\end{aligned} \tag{16}
$$

Substituting (15) into (14), we obtain these three recursive formulae

$$\begin{aligned} J_{2n} &= J_n(J_{n+2}\cdot J_{n-1}^2 - J_{n-2}\cdot J_{n+1}^2);\\ J_{4k+1} &= 16a(a-1)^2\cdot J_{2k+2}\cdot J_{2k}^3 - J_{2k-1}\cdot J_{2k+1}^3;\\ J_{4k+3} &= J_{2k+3}\cdot J_{2k+1}^3 - 16a(a-1)^2\cdot J_{2k}\cdot J_{2k+2}^3. \end{aligned} \tag{17}$$

After computation in *Magma* of J_s for some further values of s, we were motivated to prove

Lemma 2. *The leading term of J_s is j_s where*

$$\begin{aligned} j_{2k} &:= ka^{(k-1)(k+1)}, \text{ and}\\ j_{2k+1} &:= (2k+1)a^{k(k+1)}. \end{aligned}$$

(The leading coefficient and degree are given by OEIS *sequences* A026741 *and* A198442 *respectively.)*

Proof. Each of $j_1,\ldots,j_4$ has the required form. Using (17) we compute by induction:

$$\begin{aligned} j_{4k} &= j_{2k}\cdot\left(j_{2k+2}\cdot j_{2k-1}^2 - j_{2k-2}\cdot j_{2k+1}^2\right)\\ &= ka^{k^2-1}\cdot\left((k+1)a^{k^2+2k}\cdot(2k-1)^2a^{2(k^2-k)} - (k-1)a^{k^2-2k}\cdot(2k+1)^2a^{2(k^2+k)}\right)\\ &= ka^{k^2-1}\cdot\left((4k^3-3k+1)a^{3k^2} - (4k^3-3k-1)a^{3k^2}\right)\\ &= ka^{k^2-1}\cdot 2a^{3k^2}\\ &= 2ka^{4k^2-1}.\\ j_{4k+1} &= 16a^3\cdot j_{2k+2}\cdot j_{2k}^3 - j_{2k-1}\cdot j_{2k+1}^3\\ &= 16a^3\cdot(k+1)a^{k^2+2k}\cdot k^3a^{3(k^2-1)} - (2k-1)a^{k^2-k}\cdot(2k+1)^3a^{3(k^2+k)}\\ &= (16k^4+16k^3)a^{4k^2+2k} - (16k^4+16k^3-4k-1)a^{4k^2+2k}\\ &= (4k+1)a^{2k(2k+1)}.\\ j_{4k+2} &= j_{2k+1}\cdot\left(j_{2k+3}\cdot j_{2k}^2 - j_{2k-1}\cdot j_{2k+2}^2\right)\\ &= (2k+1)a^{k^2+k}\cdot\left((2k+3)a^{k^2+3k+2}\cdot k^2a^{2k^2-2} - (2k-1)a^{k^2-k}\cdot(k+1)^2a^{2(k^2+2k)}\right)\\ &= (2k+1)a^{k^2+k}\cdot\left((2k^3+3k^2)a^{3k^2+3k} - (2k^3+3k^2-1)a^{3k^2+3k}\right)\\ &= (2k+1)a^{(2k+1)^2-1}.\\ j_{4k+3} &= j_{2k+3}\cdot j_{2k+1}^3 - 16a^3\cdot j_{2k}\cdot j_{2k+2}^3\\ &= (2k+3)a^{k^2+3k+2}\cdot(2k+1)^3a^{3(k^2+k)} - 16a^3\cdot ka^{k^2-1}\cdot(k+1)^3a^{3(k^2+2k)}\\ &= (16k^4+48k^3+48k^2+20k+3)a^{4k^2+6k+2} - (16k^4+48k^3+48k^2+16k)a^{4k^2+6k+2}\\ &= (4k+3)a^{(2k+1)(2k+2)}. \end{aligned}$$

Note that the first equality in each of the four cases is valid (the leading term of the difference is the difference of the leading terms) because in each case the polynomials being subtracted have matching degrees. □

Corollary 3. *For each prime $q \equiv 3 \bmod 4$ the polynomial J_q has degree $m(q) = (q^2-1)/2$.*

In fact, our computations support the following:

Conjecture 4. *For each prime $q \equiv 3 \bmod 4$, the polynomial J_q is precisely p_q as computed by PSLQ.*

More generally, for all s given in Table 2 we have confirmed more detailed structure, as we shall summarize in the next few conjectures.

Conjecture 5. *For each integer $s \geq 1$, p_s is the unique degree $m(s)$ prime factor of J_s.*

What are some other factors of J_s? Notice in (17) that J_n is a factor of J_{2n}. This appears to extend to other multiples of n:

Conjecture 6. *The $\{J_n\}$ values form a divisibility sequence: $m \mid n$ implies $J_m \mid J_n$.*

Conjecture 6 has been confirmed for all $n \leq 256$ which was the largest case that the free online *Magma* calculator could handle.

Lemma 7. *For each positive integer s, $4m(s) < s^2$.*

Proof. We first consider prime cases.

$$\begin{aligned}
4m(2) &= 2 < 2^2; \\
4m(q) &= (q-1)^2 < q^2, \text{ for prime } q \equiv 1 \bmod 4; \\
4m(q) &= q^2 - 1 < q^2, \text{ for prime } q \equiv 3 \bmod 4; \\
4m(q^2 n) &= 4q^2 m(qn) < q^2 (qn)^2 = (q^2 n)^2, \text{ for any prime } q.
\end{aligned}$$

Hence for any integers j, k we have

$$4m(jk) = 4m(j) \cdot 4m(k) < j^2 k^2 = (jk)^2.$$

This completes the proof. □

Lending support to Conjecture 5 we have

Lemma 8. *For $s > 2$, $m(s) \leq \deg J_s$.*

Proof. We compute as follows. First, for even s

$$\begin{aligned}
&m(2k) < k^2, \text{ so} \\
&m(2k) \leq k^2 - 1 = \deg J_{2k}.
\end{aligned}$$

Likewise for odd s we write

$$\begin{aligned}
&4m(2k+1) < (2k+1)^2 = 4 \deg J_{2k+1} + 1, \text{ so} \\
&m(2k+1) \leq \deg J_{2k+1},
\end{aligned}$$

and we are done. □

Generalizing Conjecture 4 we empirically observe

Conjecture 9. *For each prime $q \equiv 3 \bmod 4$:*

- $J_{2q} = J_q \cdot J_q^* \cdot p_{2q} = p_q \cdot p_q^* \cdot p_{2q}$, *where* $p^*(x) := x^d p(1/x)$ *denotes the* reciprocal polynomial *of a polynomial p of degree d;*
- $J_{q^i} = J_{q^{i-1}} p_{q^i}$ *for any positive integer i;*
- $J_{qr} = J_q \cdot J_r \cdot p_{qr}$ *for any distinct prime $r \equiv 3 \bmod 4$.*

We next define another sequence of polynomials $H_s(a)$ identical to the substitution $H(s, \sqrt{a})$ where $H(n, x)$ is a sequence of polynomials defined in A154305 by Clark Kimberling. Specifically,

$$\begin{aligned} H_3 &:= a + 1 \\ H_4 &:= a^2 - 6a + 1 \\ H_n &:= 2H_{n-2}^4 - H_{n-1}^2 \end{aligned} \tag{18}$$

The sequence H_n lets us give precise results about p_{2^i}.

Lemma 10. *For each positive integer i,*

$$\deg(J_{2^i}) = \deg J_{2^{i-1}} + \deg H_{2i-1} + m(2^i).$$

Proof. We have

$$4^{i-1} - 1 = 4^{i-2} - 1 + 4^{i-2} + \frac{1}{2} 4^{i-1}$$

because $4 = 1 + 1 + 2$. □

For $i \in \{2, \ldots, 5\}$ we have determined the following

Conjecture 11. *For all integers $i \geq 2$,*

$$J_{2^i} = 2 \cdot J_{2^{i-1}} \cdot H_{2i-1} \cdot p_{2^i}.$$

This was also confirmed in our largest computation, for $s = 64$.

In Conjecture 1, at the end of the previous section, we observed that α_s is the largest real root of p_s. This also appears to be the case for J_s:

Conjecture 12. *For each integer $s > 2$, both J_s and p_s have their largest real root at α_s.*

It also seems that

Conjecture 13. *The number of real roots of J_n is given by entry* A195013$(n-2)$, *in the Online Encyclopedia of Integer Sequences, where*

$$\begin{aligned} \text{A195013}(2n-1) &= 2n, \text{ and} \\ \text{A195013}(2n) &= 3n. \end{aligned}$$

Finally, from Conjectures 1, 5, and 13, we would have

Conjecture 14. *For any odd prime q, both p_q and J_q have the same set of $\varphi(q) = q - 1$ real roots. This means for $q \equiv 1 \bmod 4$, the non-trivial quotient $\frac{J_q}{p_q}$ has no real roots.*

For example,

$$\begin{aligned} J_5 &= (5a^2 - 2a + 1) \cdot p_5 \\ &= (5a^2 - 2a + 1) \cdot (a^4 - 12a^3 - 26a^2 + 52a + 1). \end{aligned} \tag{19}$$

7 Proofs of Kimberley's formula and the palindromic property

As noted above, our computations confirmed Kimberley's formulas in every instance tested, and we concluded that a primary objective for future studies in this area was to understand and prove why these formulas hold. Our computations also confirmed that whenever s is even, the polynomial for the case $x = y = 1/s$ is palindromic.

After the first three authors first wrote up the results presented above, Watson Ladd, who attended a presentation of the results at the University of California, Berkeley, brought to our attention the fact that some of our conjectures should follow from known results in the theory of elliptic curves, Gaussian integers and ideals. After some effort, Ladd produced proofs of Kimberley's formulas and the palindromic property for s even, which proofs we present below in an Appendix.

8 Conclusions and future research

While these results substantially aid in understanding this problem, additional research remains to be done to fully understand the many other combinations, in other words $x = p/s$ and $y = q/s$, for different values of p, q and s. For example, it appears, from this and earlier studies, that Kimberley's formula also holds whenever $x = y = q/s$, where q is relatively prime to s. It also appears that for a fixed s, all the cases $0 < p < q < s/2$, where p and s, q and s, and p and q are relatively prime, share the same minimal polynomial. But, absent a proof, these conjectures need to be tested rigorously over a large set of p, q and s, which will require even more extreme amounts of computation.

In light of these challenges, research is needed in how to efficiently perform PSLQ-type integer relation computations on a highly parallel platform. As mentioned above, while a 12X parallel speedup is certainly welcome, a scheme to efficiently employ hundreds or thousands of cores is needed. A fundamentally new integer relation algorithm may well be required.

Acknowledgements. The authors wish to acknowledge computer equipment provided for our use by Apple, Inc. and by the Lawrence Berkeley National Laboratory.

s	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
10	8	-27.62	100	200	38	0.01	1	$9.71 \cdot 10^{-3}$	$9.71 \cdot 10^{-3}$
11	30	-406.33	150	600	1865	0.64	1	$1.67 \cdot 10^{0}$	$1.67 \cdot 10^{0}$
12	16	-112.78	100	200	304	0.06	1	$8.16 \cdot 10^{-2}$	$8.16 \cdot 10^{-2}$
13	36	-586.76	180	800	2681	1.26	1	$4.02 \cdot 10^{0}$	$4.02 \cdot 10^{0}$
14	24	-253.16	120	400	998	0.28	1	$5.16 \cdot 10^{-1}$	$5.16 \cdot 10^{-1}$
15	32	-459.83	160	700	2041	0.81	1	$2.18 \cdot 10^{0}$	$2.18 \cdot 10^{0}$
16	32	-462.31	160	700	1993	0.81	1	$2.21 \cdot 10^{0}$	$2.21 \cdot 10^{0}$
17	64	-1861.15	320	2200	9411	11.33	1	$7.56 \cdot 10^{1}$	$7.56 \cdot 10^{1}$
18	36	-579.56	180	800	2550	1.26	1	$3.82 \cdot 10^{0}$	$3.82 \cdot 10^{0}$
19	90	-3732.91	450	4200	20320	42.46	1	$4.94 \cdot 10^{2}$	$4.94 \cdot 10^{2}$
20	32	-463.84	160	700	1967	0.81	1	$2.19 \cdot 10^{0}$	$2.19 \cdot 10^{0}$
21	96	-4231.61	480	4800	23269	54.60	1	$7.11 \cdot 10^{2}$	$7.11 \cdot 10^{2}$
22	60	-1634.93	300	2000	8042	8.84	1	$5.46 \cdot 10^{1}$	$5.46 \cdot 10^{1}$
23	132	-8088.30	660	8800	50768	189.79	1	$5.06 \cdot 10^{3}$	$5.06 \cdot 10^{3}$
24	64	-1883.78	320	2200	9297	11.33	1	$7.73 \cdot 10^{1}$	$7.73 \cdot 10^{1}$
25	100	-4624.71	400	5200	25744	64.03	1	$9.19 \cdot 10^{2}$	$9.19 \cdot 10^{2}$
26	72	-2374.91	360	2800	11997	17.86	1	$1.66 \cdot 10^{2}$	$1.66 \cdot 10^{2}$
27	162	-12136.17	810	13000	88525	424.52	1	$1.97 \cdot 10^{4}$	$1.97 \cdot 10^{4}$
28	96	-4253.81	480	4800	23082	54.60	1	$7.11 \cdot 10^{2}$	$7.11 \cdot 10^{2}$
29	196	-17732.44	1000	19000	160824	899.74	8	$2.03 \cdot 10^{4}$	$1.62 \cdot 10^{5}$
30	64	-1868.01	350	2300	9064	11.33	1	$1.02 \cdot 10^{2}$	$1.02 \cdot 10^{2}$
31	240	-26653.98	1200	28000	325169	2003.33	8	$8.47 \cdot 10^{4}$	$6.78 \cdot 10^{5}$
32	128	-7577.07	650	8200	45893	168.20	1	$5.13 \cdot 10^{3}$	$5.13 \cdot 10^{3}$
33	240	-26621.93	1200	28000	326616	2003.33	8	$8.51 \cdot 10^{4}$	$6.81 \cdot 10^{5}$
34	128	-7574.93	650	8200	45914	168.20	1	$5.16 \cdot 10^{3}$	$5.16 \cdot 10^{3}$
35	192	-17044.00	1000	18000	149577	829.41	8	$2.48 \cdot 10^{4}$	$1.98 \cdot 10^{5}$
36	144	-9570.86	750	10300	62282	267.10	1	$9.54 \cdot 10^{3}$	$9.54 \cdot 10^{3}$
37	324	-48431.32	1650	51000	931254	6579.66	16	$4.84 \cdot 10^{5}$	$7.74 \cdot 10^{6}$
38	180	-14951.64	900	16000	120984	642.98	1	$3.88 \cdot 10^{4}$	$3.88 \cdot 10^{4}$
39	288	-38330.14	1450	40000	667153	4124.24	16	$2.68 \cdot 10^{5}$	$4.29 \cdot 10^{6}$
40	128	-7580.00	650	8200	45655	168.20	1	$5.02 \cdot 10^{3}$	$5.02 \cdot 10^{3}$
42	192	-16993.99	1000	18000	150364	829.41	8	$1.57 \cdot 10^{4}$	$1.26 \cdot 10^{5}$
44	240	-26604.14	1200	28000	323762	2003.33	8	$7.43 \cdot 10^{4}$	$5.94 \cdot 10^{5}$
45	288	-38315.08	1450	40000	660001	4124.24	16	$2.09 \cdot 10^{5}$	$3.35 \cdot 10^{6}$
46	264	-32036.34	1350	34000	476902	2921.57	16	$1.06 \cdot 10^{5}$	$1.70 \cdot 10^{6}$
48	256	-30248.55	1350	32000	415316	2586.39	16	$8.98 \cdot 10^{4}$	$1.44 \cdot 10^{6}$
50	200	-18421.18	1000	20000	168947	974.44	8	$2.12 \cdot 10^{4}$	$1.69 \cdot 10^{5}$
52	288	-38414.49	1550	41000	655291	4124.24	16	$2.12 \cdot 10^{5}$	$3.40 \cdot 10^{6}$
*60	256	-14477.99	800	16000	90371	336.41	1	$5.28 \cdot 10^{3}$	$5.28 \cdot 10^{3}$
*64	512	-57816.90	1600	64000	802361	5172.79	16	$3.78 \cdot 10^{5}$	$2.42 \cdot 10^{6}$

Table 2: Computer runs to discover minimal polynomials for the cases $x = y = 1/s$. Here m is the degree; D is the detection level; P_1 is the medium precision, in digits; P_2 is the full precision, in digits; N is the number of multipair PSLQ iterations; M is the memory, in Mbytes; C is the number of processor cores; T is the wall-clock run time, in seconds; and the last column is the total core-seconds. In the rows labeled *, the palindromic principle of Section 5 was applied.

s	q	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
10	3	*4*	-4.94	100	200	10	0.01	1	$9.88 \cdot 10^{-3}$	$9.88 \cdot 10^{-3}$
11	2	30	-302.77	150	600	1428	0.64	1	$2.25 \cdot 10^{0}$	$2.25 \cdot 10^{0}$
12	5	*8*	-24.88	100	200	69	0.06	1	$1.53 \cdot 10^{-2}$	$1.53 \cdot 10^{-2}$
13	2	36	-443.77	180	800	2060	1.26	1	$5.56 \cdot 10^{0}$	$5.56 \cdot 10^{0}$
14	3	24	-187.79	120	400	849	0.28	1	$7.27 \cdot 10^{-1}$	$7.27 \cdot 10^{-1}$
15	2	32	-342.14	160	700	1513	0.81	1	$2.87 \cdot 10^{0}$	$2.87 \cdot 10^{0}$
16	3	32	-345.22	160	700	1636	0.81	1	$2.91 \cdot 10^{0}$	$2.91 \cdot 10^{0}$
17	2	64	-1410.53	320	2200	7146	11.33	1	$1.08 \cdot 10^{2}$	$1.08 \cdot 10^{2}$
18	5	36	-579.56	180	800	2550	1.26	1	$4.88 \cdot 10^{0}$	$4.88 \cdot 10^{0}$
19	2	90	-2743.27	450	4200	14825	42.46	1	$6.20 \cdot 10^{2}$	$6.20 \cdot 10^{2}$
20	3	32	-359.23	160	700	1637	0.81	1	$3.06 \cdot 10^{0}$	$3.06 \cdot 10^{0}$
21	2	96	-3098.89	480	4800	16837	54.60	1	$8.95 \cdot 10^{2}$	$8.95 \cdot 10^{2}$
22	3	60	-1184.89	300	2000	5991	8.84	1	$6.99 \cdot 10^{1}$	$6.99 \cdot 10^{1}$
23	2	132	-5935.65	660	8800	36880	189.79	1	$4.63 \cdot 10^{3}$	$4.63 \cdot 10^{3}$
24	5	*32*	-336.46	320	2200	1613	11.33	1	$2.83 \cdot 10^{0}$	$2.83 \cdot 10^{0}$
25	2	100	-3425.99	400	5200	18934	64.03	1	$1.16 \cdot 10^{3}$	$1.16 \cdot 10^{3}$
26	3	72	-1714.50	360	2800	8763	17.86	1	$1.93 \cdot 10^{2}$	$1.93 \cdot 10^{2}$
27	2	162	-8994.33	810	13000	64954	424.52	1	$1.34 \cdot 10^{4}$	$1.34 \cdot 10^{4}$
28	3	96	-3133.10	480	4800	17150	54.60	1	$9.39 \cdot 10^{2}$	$9.39 \cdot 10^{2}$
29	2	196	-30287.93	1000	19000	118623	899.74	8	$1.25 \cdot 10^{4}$	$1.00 \cdot 10^{5}$
30	7	64	-1291.21	350	2300	6867	11.33	1	$1.01 \cdot 10^{2}$	$1.01 \cdot 10^{2}$
31	2	240	-19498.68	1200	28000	238779	2003.33	8	$5.65 \cdot 10^{4}$	$4.52 \cdot 10^{5}$
32	3	128	-5578.06	650	8200	33829	168.20	1	$4.16 \cdot 10^{3}$	$4.16 \cdot 10^{3}$
33	2	240	-19341.77	1200	28000	234732	2003.33	8	$5.69 \cdot 10^{4}$	$4.55 \cdot 10^{5}$
34	3	128	-5411.70	650	8200	32842	168.20	1	$2.53 \cdot 10^{3}$	$2.53 \cdot 10^{3}$
35	2	192	-12551.58	1000	18000	110693	829.41	8	$1.17 \cdot 10^{4}$	$9.36 \cdot 10^{4}$
36	5	144	-6831.83	750	10300	45559	267.10	1	$6.93 \cdot 10^{3}$	$6.93 \cdot 10^{3}$
37	2	324	-35412.09	1650	51000	691277	6579.66	16	$3.42 \cdot 10^{5}$	$5.47 \cdot 10^{6}$
38	3	180	-11011.83	900	16000	89722	642.98	1	$7.42 \cdot 10^{3}$	$7.42 \cdot 10^{3}$
39	2	288	-27943.70	1450	40000	458238	4124.24	16	$1.84 \cdot 10^{5}$	$2.94 \cdot 10^{6}$
40	3	128	-5674.61	650	8200	34273	168.20	1	$4.13 \cdot 10^{3}$	$4.13 \cdot 10^{3}$
42	5	192	-12183.50	1000	18000	106770	829.41	8	$1.16 \cdot 10^{4}$	$9.27 \cdot 10^{4}$
44	3	240	-19581.93	1200	28000	232713	2003.33	8	$6.18 \cdot 10^{4}$	$4.95 \cdot 10^{5}$
45	2	288	-27857.00	1450	40000	482959	4124.24	16	$1.47 \cdot 10^{5}$	$2.35 \cdot 10^{6}$
46	3	264	-23318.37	1350	34000	346987	2921.57	16	$7.28 \cdot 10^{4}$	$1.17 \cdot 10^{6}$
48	5	256	-21480.15	1350	32000	292974	2586.39	16	$5.93 \cdot 10^{4}$	$9.50 \cdot 10^{5}$
50	3	200	-13409.44	1000	20000	122468	974.44	8	$1.63 \cdot 10^{4}$	$1.30 \cdot 10^{5}$

Table 3: Computer runs to discover minimal polynomials for the cases $x = 1/s$, $y = q/s$, where $q > 1$ is the smallest integer relatively prime to s. See caption to Table 2 for notation.

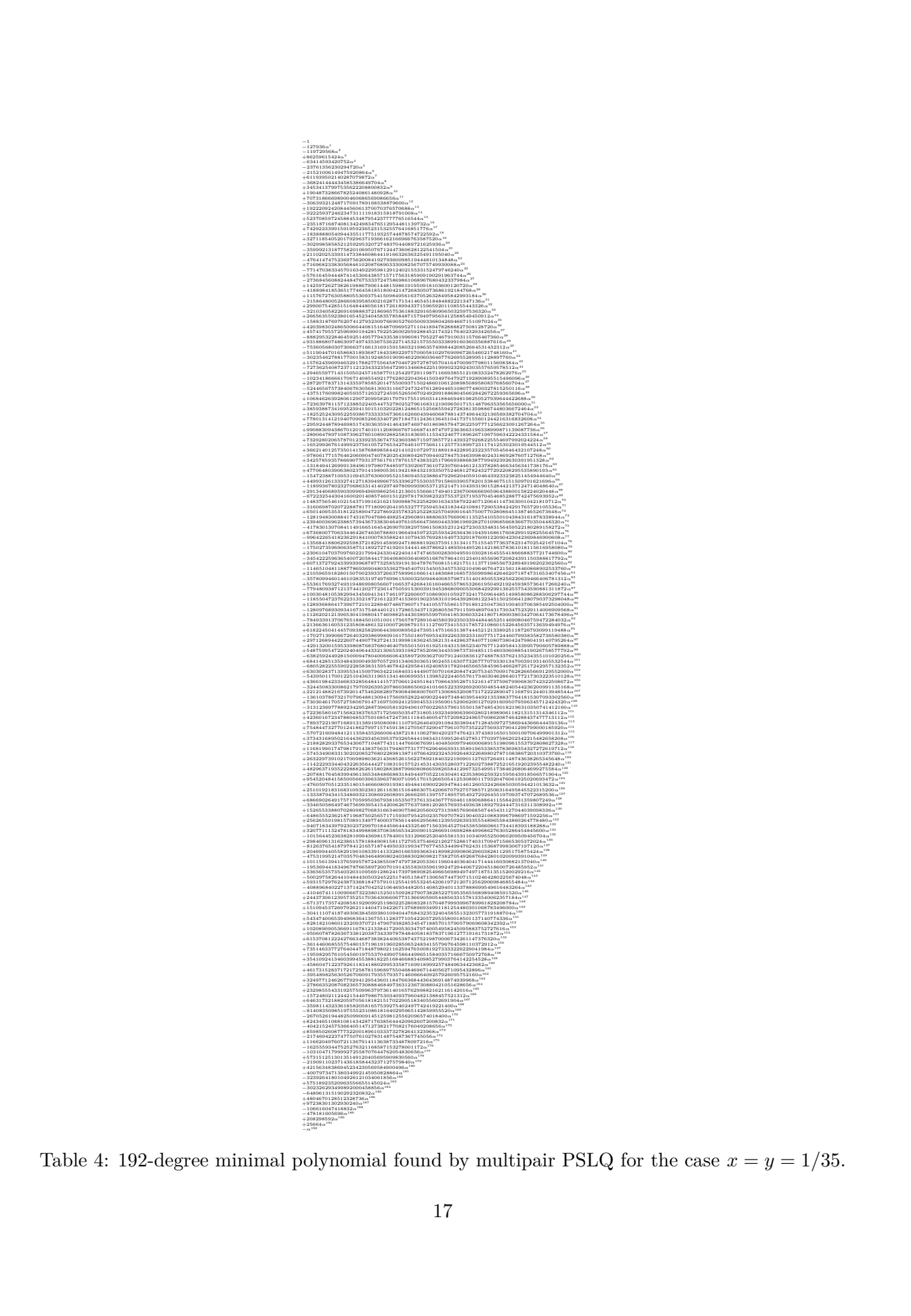

Table 4: 192-degree minimal polynomial found by multipair PSLQ for the case $x = y = 1/35$.

Multiprecision software	PSLQ code	Cores	Run time	Speedup
ARPREC	2-level	1	$1.599 \cdot 10^6$	1.00
New MPFR-based software	2-level	1	$5.249 \cdot 10^5$	3.05
New MPFR-based software	3-level	1	$1.240 \cdot 10^5$	12.90
		2	$7.585 \cdot 10^4$	21.08
		4	$4.121 \cdot 10^4$	38.80
		8	$2.476 \cdot 10^4$	64.58
		16	$1.021 \cdot 10^4$	156.61

Table 5: Wall-clock run times (in seconds) and speedup factors for the case $x = y = 1/35$, measured in a typically busy environment with similar jobs running on other cores.

References

[1] J. Ablinger, J. Blumlein and C. Schneider, "Harmonic Sums and Polylogarithms Generated by Cyclotomic Polynomials," *J. Math. Phys.* **52** (2011), 102301.

[2] "ARPREC: An arbitrary precision computation package," Sept 2002, `http://crd.lbl.gov/software/applied-mathematics-software/`.

[3] D. H. Bailey, R. F. Lucas and S. W. Williams, ed., *Performance Tuning of Scientific Applications*, CRC Press, Boca Raton, FL, 2011.

[4] D. H. Bailey, "MPFUN2015: A thread-safe arbitrary precision package (full documentation)," manuscript, 30 Apr 2015, `http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf`.

[5] D. H. Bailey, J. M. Borwein, R. E. Crandall and I. J. Zucker, "Lattice sums arising from the Poisson equation," *Journal of Physics A: Mathematical and Theoretical*, **46** (2013), p. 115201.

[6] 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.

[7] D. H. Bailey and D. J. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Mathematics of Computation*, **70**, 236 (2000), 1719–1736.

[8] J. M. Borwein and P. B. Borwein, *Pi and the AGM*, John Wiley and Sons, New York, 1987.

[9] J. Borwein, L. Glasser, R. McPhedran, J. Wan and J. Zucker, *Lattice Sums: Then and Now*, *Encyclopedia of Mathematics and its Applications*, 150, Cambridge University Press, August 2013.

[10] A. Bostan, S. Boukraa, S. Hassani, J.-M. Maillard, J.-A. Weil, N. Zenine and N. Abarenkova, "Renormalization, isogenies and rational symmetries of differential equations," January 2010, `http://arxiv.org/pdf/0911.5466.pdf`.

[11] A. Bostan, S. Boukraa, S. Hassani, J.-M. Maillard, J.-A. Weil, N. Zenine and N. Abarenkova, "Renormalization, isogenies and rational symmetries of differential equations," *Adv. Math. Phys.* (2010) Art. ID 941560, 44 pp.

[12] J. Chen, Y. Feng and W. Wu, "Two variants of HJLS-PSLQ with applications," *Proceedings of SNC'14*, 88–96, Shanghai, China, 2014.

[13] I.N. Herstein, *Topics In algebra*, 2nd. ed., Wiley, New York, 1975.

[14] "Comparison of multiple-precision floating-point software," `http://www.mpfr.org/mpfr-3.1.0/timings.html`.

[15] R. Crandall, "The Poisson equation and 'natural' Madelung constants," preprint (2012), copy in authors' possession.

[16] "The GNU MPFR library," `http://www.mpfr.org`.

[17] S. Kanemitsu, Y. Tanigawa, H. Tsukada and M. Yoshimoto, "Crystal symmetry viewed as zeta symmetry," p. 91–130 in T. Aoki, S. Kanemitsu, M. Nakahara and Y. Ohno, eds., *Zeta Functions, Topology and Quantum Physics*, Springer, New York, 2005 (reprinted 2010).

[18] S. Kanemitsu and H. Tsukada, "Crystal symmetry viewed as zeta symmetry II," p. 275–292 in K. Alladi, J. R. Klauder and C. R. Rao, eds., *The Legacy of Alladi Ramakrishnan in the Mathematical Sciences*, Springer, New York, 2010.

[19] "MPFR C++," `http://www.holoborodko.com/pavel/mpfr`.

[20] G. Savin and D. Quarfoot, "On attaching coordinates of Gaussian prime torsion points of $y^2 = x^3 + x$ to $Q(i)$," March 2010, `http://www.math.utah.edu/~savin/EllipticCurvesPaper.pdf`.

[21] J.H. Silverman, *Advanced Topics in the Arithmetic of Elliptic Curves*, GTM 151, Springer-Verlag, New York, 1994.

[22] E.T. Whittaker and G.N. Watson, *A Course of Modern Analysis*, 4th ed., Cambridge University Press, Cambridge, 1927 (reprinted 1962).

A Proofs of Kimberley's formula and the palindromic property

In this section we prove Kimberley's formula and the reciprocal nature of $p_{2s}(x)$. We begin by considering

$$\phi_2(x,y) = \frac{1}{2\pi}\log\left|\frac{\theta_2(z,q)\theta_4(z,q)}{\theta_1(z,q)\theta_3(z,q)}\right|$$

and

$$\alpha = e^{8\pi\phi_2(x,y)},$$

which after basic algebra gives

$$\alpha = \left|\frac{\theta_2(z,q)\theta_4(z,q)}{\theta_1(z,q)\theta_3(z,q)}\right|^4.$$

The definitions of the theta functions earlier in this paper agree with those in Whittaker and Watson in the chapter "Theta Functions" [22] and therefore the identities presented there can be used with $\tau = i$. We now introduce the function

$$f(z) = \frac{\theta_2(z)^2\theta_4^2(z)}{\theta_1(z)^2\theta_3^2(z)}.$$

The function $f(z)$ is doubly periodic with periods $\pi/2 + i/2$ and $\pi/2 - i/2$, and has a double pole at 0 and a double zero at $\pi i/2$. The double periodicity follows from the quasiperiodicity of the theta functions. The poles and zeros follow from examining the poles and zeros of the theta functions, as recorded in Whittaker and Watson.

Every holomorphic doubly-periodic function is a constant, by the corollary to Liouville's theorem found in [22, Chap. 2]. Consider $g(z) = f(\pi/2(1+i)z)/\wp(z)$, where $\wp(z)$ is the Weierstrass $\wp$ function on the lattice spanned by 1 and i. Then $g(z)$ is doubly periodic with periods 1 and i, and has no poles or zeros. It is therefore a constant.

Now $\alpha = f(\pi/2(1/s+i/s))f(\pi/2(1/s-i/s))$, which is $a^2\wp(1/s)\wp(-i/s)$. But $\wp(-i/s) = -\wp(1/s)$,so $\alpha = -a^2\wp(1/s)^2$, for a constant a independent of s. From any of our examples we conclude $\alpha = -\wp(1/s)^2$.

The Weierstrass $\wp$ function parametrizes the x-coordinates of points on an elliptic curve. For the lattice spanned by 1 and i that elliptic curve is $y^2 = x^3 + x$. For any z, w in the lattice, $(\wp(z), \wp'(z))$ and $(\wp(w), \wp'(w))$ are points on the elliptic curve, and $(\wp(z+w), \wp'(z+w)) = (\wp(z), \wp'(z)) + (\wp(w), \wp'(z))$ where the second sum is the group law on the elliptic curve.

The curve $y^2 = x^3 + x$ has complex multiplication defined over $\mathbb{Q}(i)$. The map i sending (x, y) to $(-x, iy)$ is a group homomorphism, and when iterated twice is negation. Thefore the endomorphism ring is the ring of integers in $\mathbb{Q}(i)$, and we can define multiplication on the curve not just for integers, but for Gaussian integers.

The Gaussian integers are a unique factorization domain, in fact a principal ideal domain. The nonzero prime ideals are maximal, and are either (p) for p a prime 3 mod 4 in the integers, or $(a + ib)$ where $a^2 + b^2 = q$, a prime 1 mod 4 in the integers, or $(1 + i)$.

In Silverman's discussion of the theory of complex multiplication in [21, Chap. 2, Thm. 5.6], he constructs the ray class field for a divisor d in $\mathbb{Q}(i)$ by adjoining the squares of the x-coordinates of points which are d torsion points on $y^2 = x^3 + x$ to $\mathbb{Q}(i)$. The ray class field has degree equal to the order of the ray class group. We know that α falls in the field

generated by the s torsion points and not any of the smaller ray class fields. In fact, the action of the Galois group of the ray class field is given by multiplication by the ray class group. This action is transitive, and so the minimal polynomial of α over $\mathbb{Q}(i)$ has degree equal to the order of the ray class group.

The ray class group is defined as the set of all ideals relatively prime to s modulo the set of principal ideals generated by β such that $\beta - 1$ is divisible by s. This is isomorphic to the set of units in $\mathbb{Z}(i)/s\mathbb{Z}(i)$ modulo the units of $\mathbb{Z}(i)$. The size of this group is exactly $m(s)$ by use of inclusion-exclusion: for a number d in $\mathbb{Z}(i)$ there are $N(d)$ elements of $\mathbb{Z}(i)$, and we wish to count the ones which are not divisible by any divisor of d, exactly analgous to the computation of $\phi(n)$ for a regular integer.

More specifically $4m(s)$ is the number of units in $\mathbb{Z}(i)/s\mathbb{Z}(i)$. Extend the Mö'bius function μ to $\mathbb{Z}(i)$ with the same definition: 0 for non-squarefree arguments, and $(-1)^k$ for numbers with k prime factors. Define $\rho(s) = 4m(s)$ to be the number of units in $\mathbb{Z}(i)/s\mathbb{Z}(i)$.

Then $\rho(s) = \sum_{k|d} \mu(d/k)N(k)$. This is a multiplicative function, and so $4m(s) = \prod_{p^{e_i}|d} \rho(p^{e_i})$. The primes in $\mathbb{Z}(i)$ are either $a+bi$ and $a-bi$ with a^2+b^2 a prime $1 \mod 4$ or ordinary primes which are $3 \mod 4$, or $(1+i)$.

If s is divisible by $a+bi$ with b nonzero it is also divisible by $a-bi$ by the same amount as it is real. So we can rewrite the product decomposition as a product over ordinary primes, and then have to evaluate $\rho(p^{e_i})$ for an ordinary prime. If the prime is $3 \mod 4$ this is just $p^{2e_i} - p^{2(e_i-1)}$. If the prime is $1 \mod 4$ this is $(p^{e_i} - p^{e_i-1})^2$. If the prime is 2 this is 2. Some algebra now shows that this is equvalent to formula 3.

When we consider $K/\mathbb{Q}$ we obtain an extension where the subgroup of the Galois group fixing $\mathbb{Q}(i)$ is normal, as it has index 2 in the Galois group, by [13, Thm. 5.5.6]. Because it is normal it is fixed under conjugation.

Now, α is real, and so writing $q_s(x) = \prod_\tau (x - \tau\alpha)$ where τ ranges over the elements of the Galois group of $K/\mathbb{Q}$ fixing $\mathbb{Q}(i)$ we see that

$$\sigma q_s(x) = \prod_\tau (x - \sigma\tau\alpha) = \prod_\tau (x - \tau\sigma\alpha) = \prod_\tau (x - \tau\alpha) = q_s(x)$$

by the normality of the set of τ. Therefore $q_s(x)$ has real coefficients.

Our $q_s(x)$ does not have integral coefficients, but merely rational. However, $p_s(x)$ is $q_s(-x)$ times some constant, and so Kimberley's formula is true. Note that if $P = (x, y)$ on the elliptic curve $y^2 = x^3 + x$ is a $2s$ torsion point, so is $P + (0, 0)$ which has x-coordinate $1/x$ by applying the addition formulas. Furthermore P and $P + (0, 0)$ have the same order. Therefore $q_{2s}(x)$ is a reciprocal polynomial as conjectured, and so is $p_{2s}(x)$.

We finish by observing that the same building blocks were used in the orginal more primitive proof [5] of algebraicness of α.