Computer discovery of large Poisson polynomials with 51,000-digit computations

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

Collaborators:
Jonathan M. Borwein and Jason Kimberley
CARMA, University of Newcastle, Australia

December 16, 2015

The PSLQ integer relation algorithm

Let $X = (x_k)$ be an $(m+1)$-long real or complex vector. An integer relation algorithm such as PSLQ finds a nontrivial integer vector $A = (a_k)$ such that

$$a_0 x_0 + a_1 x_1 + \cdots + a_m x_m \quad = \quad 0.$$

- The multipair PSLQ algorithm is a more efficient and moderately parallelizable variant of PSLQ, the most widely used integter relation algorithm (although some use a variant of LLL).
- Integer relation detection (by any algorithm) requires very high precision: at least $(m+1) \cdot \max_k \log_{10} |a_k|$ digits, both in the input data and the algorithm.
- Multipair PSLQ is extremely efficient with precision — it can usually detect a relation when the precision is only a few percent higher than this minimum level.

- D. H. Bailey and D. J. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), 1719–1736.

Application of multipair PSLQ

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

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

$$\alpha = 1.23268891306144344533147286961125564706898882454793057605763468477 8\ldots$$

What is its minimal polynomial?

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

Answer (using 250-digit arithmetic):

$$\begin{aligned} 0 = {} & 6971 - 1440\alpha - 20520\alpha^2 - 98280\alpha^3 - 102060\alpha^4 - 1458\alpha^5 + 80\alpha^6 - 43920\alpha^7 \\ & + 538380\alpha^8 - 336420\alpha^9 + 1215\alpha^{10} - 80\alpha^{12} - 56160\alpha^{13} - 135540\alpha^{14} - 540\alpha^{15} \\ & + 40\alpha^{18} - 7380\alpha^{19} + 135\alpha^{20} - 10\alpha^{24} - 18\alpha^{25} + \alpha^{30} \end{aligned}$$

The Poisson potential function problem

In a 2013 study, we numerically discovered, and then proved the intriguing fact that for rational (x, y), the 2-D Poisson potential function $\phi_2(x, y)$ satisfies

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

where α is *algebraic*, i.e., the root of a some integer polynomial of degree m.

By computing high-precision numerical values of $\phi_2(x, y)$ for various specific rational x and y, and applying a multipair PSLQ program, we were able to produce the explicit minimal polynomials for α in several specific cases.

This study was hampered by the huge complexity and computational costs involved.

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

Samples of minimal polynomials found by multipair PSLQ

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

These computations are very expensive. The case $x = y = 1/32$, for instance, required 10,000-digit arithmetic and ran for 45 hours. Other runs, using even higher precision, ultimately failed, evidently due to subtle program bugs. Help!

Kimberley's formula for the degree of the polynomial

Based on our preliminary results, Jason Kimberley of the University of Newcastle, Australia 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:

Set $m(2) = 1/2$. Otherwise for primes p congruent to 1 mod 4, set $m(p) = \mathrm{int}^2(p/2)$, where int denotes greatest integer, and for primes p congruent to 3 mod 4, set $m(p) = \mathrm{int}\,(p/2)(\mathrm{int}\,(p/2) + 1)$. Then for any other positive integer s whose prime factorization is $s = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$,

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

Does Kimberley's formula hold for larger s? Why?

What is the true mathematical connection between the pair of rationals (x, y) and the algebraic number α?

Three improvements to the Poisson polynomial computation program

1. MPFUN2015: A new thread-safe multiprecision package.
   - Speedup: 3X
2. A new 3-level multipair PSLQ program.
   - Speedup: 4.2X
3. Parallel implementation on a 16-core system.
   - Speedup: 12.2X

Overall speedup: 156X

Thread safety: A major challenge for high-precision software

- Because of greatly magnified run times (often 1000X or more) of high-precision computations, highly parallel implementations are in order. However, most high-precision packages are *not* thread-safe.
- Many packages employ global read/write variables, e.g., for transcendental function evaluation, which ruin thread safety.
- The working precision level, a global variable that is changed frequently within the package itself and often by users also, is particularly troublesome.

One thread-safe package: The MPFR C++ package, which calls the lower-level MPFR package, is thread-safe (provided MPFR is compiled with the thread-safe build option).

MPFR is a very well-designed, features correct rounding and is the fastest low-level package currently available, but is only a low-level package.

There is no thread-safe high-level Fortran package.

MPFUN2015: A new thread-safe arbitrary precision package

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

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

Both versions include a high-level language interface, using custom datatypes and operator overloading. For most codes, only minor changes are required.

A C++ version is in the works.

The software is available at http://www.davidhbailey.com/dhbsoftware.

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

- D. H. Bailey, "MPFUN2015: A thread-safe arbitrary precision computation package," manuscript, 1 Oct 2015. http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf.

New three-level multipair PSLQ program

Employs three levels of numeric precision:

- Ordinary double precision.
- Medium precision, typically 100–2000 digits.
- Full precision, typically many thousands of digits.

When an entry of the double precision reduced vector is smaller than 10^{-14}, or when an entry of one of the integer-valued double precision arrays exceeds $2^{53} \approx 9.007 \cdot 10^{15}$, the medium precision arrays are updated by matrix multiplication.

Similarly, when an entry of the medium precision reduced vector is smaller than the medium precision "epsilon," the full-precision arrays are updated.

Substantial care must be taken to manage this three-level hierarchy, and to correctly handle numerous atypical scenarios.

A fast algorithm to compute the Poisson potential function $\phi_2(x, y)$

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

where $q = e^{-\pi}$ and $z = \frac{\pi}{2}(y + ix)$. Compute the four theta functions using the following rapidly convergent formulas involving complex variables:

$$\theta_1(z, q) = 2 \sum_{k=1}^{\infty} (-1)^{k-1} q^{(2k-1)^2/4} \sin((2k-1)z),$$

$$\theta_2(z, q) = 2 \sum_{k=1}^{\infty} q^{(2k-1)^2/4} \cos((2k-1)z),$$

$$\theta_3(z, q) = 1 + 2 \sum_{k=1}^{\infty} q^{k^2} \cos(2kz),$$

$$\theta_4(z, q) = 1 + 2 \sum_{k=1}^{\infty} (-1)^k q^{k^2} \cos(2kz).$$

High-level computational algorithm

1. Given rationals $x = p/s$ and $y = q/s$, select a conjectured minimal polynomial degree $m(s)$ (using Kimberley's formula) and other parameters for the run.
2. Calculate $\phi_2(x, y)$ to P_2-digit precision using the formulas from two viewgraphs above. 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.
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 construct `DO PARALLEL` to perform certain time-intensive loops in parallel.
4. If no numerically significant relation is 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* and *Maple* to ensure that the polynomial is irreducible.

Application program and libraries for the Poisson calculations

Description	Language	Lines of code
Poisson polynomial program*	Fortran	2,000
MPFUN-MPFR package	Fortran	12,000
MPFR package	C	93,000
GMP package	C	83,000
Total		190,000

*This includes the computation of $\phi_2(x, y)$ and the 3-level multipair PSLQ program.

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

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

Timings 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
MPFUN-MPFR	2-level	1	$5.249 \cdot 10^5$	3.05
MPFUN-MPFR	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

The run times are wall-clock run times (in seconds), measured on a 16-core 2.4 GHz MacPro, in a typically busy environment with similar jobs running on other cores.

Selected runs (degrees, precision, timings, etc.) for $x = y = 1/s$

s	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
20	32	-463.84	160	700	1967	0.81	1	$2.19 \cdot 10^0$	$2.19 \cdot 10^0$
24	64	-1883.78	320	2200	9297	11.33	1	$7.73 \cdot 10^1$	$7.73 \cdot 10^1$
30	64	-1868.01	350	2300	9064	11.33	1	$1.02 \cdot 10^2$	$1.02 \cdot 10^2$
32	128	-7577.07	650	8200	45893	168.20	1	$5.13 \cdot 10^3$	$5.13 \cdot 10^3$
34	128	-7574.93	650	8200	45914	168.20	1	$5.16 \cdot 10^3$	$5.16 \cdot 10^3$
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	$2.12 \cdot 10^4$	$1.69 \cdot 10^5$

s = denominator; m = degree; D = detection level; P_1 = medium precision; P_2 = full precision; N = number of iterations; M = Mbytes; C = cores; T = wall clock time; $C \cdot T$ = total core-seconds.

Selected runs (degrees, precision, timings, etc.) for $x = 1/s, y = q/s$

s	q	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
20	3	32	-359.23	160	700	1637	0.81	1	$3.06 \cdot 10^0$	$3.06 \cdot 10^0$
24	5	*32*	-336.46	320	2200	1613	11.33	1	$2.83 \cdot 10^0$	$2.83 \cdot 10^0$
30	7	64	-1291.21	350	2300	6867	11.33	1	$1.01 \cdot 10^2$	$1.01 \cdot 10^2$
32	3	128	-5578.06	650	8200	33829	168.20	1	$4.16 \cdot 10^3$	$4.16 \cdot 10^3$
34	3	128	-5411.70	650	8200	32842	168.20	1	$2.53 \cdot 10^3$	$2.53 \cdot 10^3$
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	$1.63 \cdot 10^4$	$1.30 \cdot 10^5$

s = denominator; m = degree; D = detection level; P_1 = medium precision; P_2 = full precision; N = number of iterations; M = Mbytes; C = cores; T = wall clock time; $C \cdot T$ = total core-seconds.

Analysis

From our results, in the case $(1/s, 1/s)$ where s is even, the resulting polynomial is always palindromic ($a_k = a_{m-k}$). For instance, when $s = 16$, it is:

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

The eight real roots of this polynomial are symmetric (if α is root, so is $1/\alpha$):

$$
\begin{array}{ll}
7.21047139876944319282\ldots \times 10^{-4} & 1.38687187660249572981\ldots \times 10^{3} \\
5.70748499825567769711\ldots \times 10^{-2} & 1.75208520093459751406\ldots \times 10^{1} \\
3.77402362524897556853\ldots \times 10^{-1} & 2.64969194498359597353\ldots \times 10^{0} \\
8.98134799939575125663\ldots \times 10^{-1} & 1.11341860939725103576\ldots \times 10^{0}
\end{array}
$$

Jason Kimberley's new observations for the case $(1/s, 1/s)$

By examining some of the polynomials produced by our program, Jason Kimberley observed the following:

- The algebraic number α_s is the *largest* real root of the associated polynomial.
- The polynomial has $\varphi(s)$ real roots, where φ is the Euler totient function.
- $\sqrt{-\alpha_s}$ is the largest purely imaginary root of $\psi_s(x, 1)$, where ψ is a sequence of polynomials defined in a 2010 paper by Savin and Quarfoot of the Univ. of Utah.

Kimberley found the Savin-Quarfoot paper by doing an Internet search for "387221579866," which is a coefficient of the polynomial for the case $(1/11, 1/11)$.

He found another relevant paper by Zudilin by searching for "221753896032," which is a coefficient for the case $(1/13, 1/13)$.

Latest news: Kimberley reports that he has found a recursion that generates the polynomials in the $(1/s, 1/s)$ case. Details will follow.

Conclusions

- As far as we are aware, these computations, which employed up to 51,000-digit precision, producing polynomials with degrees up to 324 and integer coefficients up to 10^{145}, constitute the largest successful integer relation computations performed to date.
- Kimberley's 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).
- The resulting polynomial coefficients have yielded clues as to why Kimberley's formula holds.
- Additional research is needed to understand many other combinations, e.g., $x = p/s$ and $y = q/s$, for different values of p, q and s, which will require even more extreme computation.
- While a 12X parallel speedup, with a 156X overall speedup, are certainly welcome, a scheme to efficiently employ hundreds or thousands of processor cores is needed. A fundamentally new integer relation algorithm may well be required.

Thanks!

- This talk is available at http://www.davidhbailey.com/dhbtalks/dhb-wcnt2015.pdf.
- A technical report with more details, results and analysis is available at http://www.davidhbailey.com/dhbpapers/poisson-res.pdf.
- The MPFUN-Fort and MPFUN-MPFR software packages are available at http://www.davidhbailey.com/dhbsoftware.
- A technical report giving full details of MPFUN-Fort and MPFUN-MPFR is available at http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf.