Finding large Poisson polynomials using four-level variable precision

David H. Bailey
Lawrence Berkeley National Laboratory (retired)
University of California, Davis, Department of Computer Science
This talk is available at:
http://www.davidhbailey.com/dhbtalks/dhb-sc21-correct.pdf

ICERM report on reproducibility in mathematical and scientific computing

The field of high-performance computing has lagged efforts in other fields to ensure reproducibility. One key issue is numerical reproducibility:

> *Numerical reproducibility has emerged as a particularly important issue, since the scale of computations has greatly increased in recent years, particularly with computations performed on many thousands of processors and involving similarly large datasets. Large computations often greatly magnify the level of numeric error, so that numerical difficulties that were once of little import now are large enough to alter the course of the computation or to draw into question the overall validity of the results.*

- D. H. Bailey, J. M. Borwein, O. Caprotti, U. Martin, B. Salvy and M. Taufer, "Opportunities and challenges in 21st century mathematical computation: ICERM workshop report," 31 Jul 2014, `https://www.davidhbailey.com/dhbpapers/ICERM-2014.pdf`.

Commonly used formats for floating-point computing

Name	Number of bits				Digits
	Sign	Exp.	Mant.	Hidden	
IEEE half	1	5	10	1	3
ARM half	1	5	10	1	3
bfloat16	1	8	7	1	2
IEEE single	1	8	23	1	7
IEEE double	1	11	52	1	15
IEEE extended	1	15	64	0	19
IEEE quad	1	15	112	1	34
double-double	1	11	104	2	31
quad-double	1	11	208	4	62
double-quad	1	15	224	1	68
arbitrary	varies	varies	varies	varies	varies

Advantages of variable precision

Compared with using a high fixed level of precision for the entire application, employing variable precision *usually* results in:

- Faster processing.
- Better cache utilization.
- Lower run-time memory usage.
- Lower offline data storage.
- Lower energy costs.

Challenge:

- Unless variable precision is carefully managed, there is significantly greater potential for loss of numerical accuracy and reproducibility.

Finding Poisson polynomials: Application of variable precision computing

Lattice sums related to the Poisson potential function, which naturally arise in studies of gravitational and electrostatic potentials, have been studied for many years in the mathematical physics community, and recently have been applied to image sharpening:

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

In 2012 researchers discovered that when x and y are rational numbers,

$$\phi_2(x,y) = \frac{1}{\pi} \log \alpha(x,y),$$

where $\alpha(x,y)$ is an algebraic number, namely the root of a degree-m integer polynomial. What is the connection between x and y and the degree of α?

- 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.
- D. H. Bailey, J. M. Borwein, J. Kimberley and W. Ladd, "Computer discovery and analysis of large Poisson polynomials," *Experimental Mathematics*, 27 Aug 2016, vol. 26, 349–363.

Finding Poisson polynomials, cont.

By computing high-precision numerical values of $\phi_2(x, y)$ for various specific rational x and y, using a three-level variable precision code (e.g., double/500-digit/10,000-digit), and applying a variant of the PSLQ algorithm, we found the following results:

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

Kimberley's formula

Based on these preliminary results, Jason Kimberley conjectured that the degree $m(s)$ of the minimal polynomial associated with the case $x = y = 1/s$ is given by this rule:

> *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 $s = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$,*

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

Additional three-level precision computations, (e.g., double/2000-digit/64,000-digit), confirmed Kimberley's formula for all s up to 40, and for even s up to 52.

By examining the computed results, connections were found to a sequence of polynomials defined in a 2010 paper by Savin and Quarfoot. These observations ultimately led to a proof of Kimberley's formula and several other facts.

What about the more general case $x = p/s$ and $y = q/s$, for integers p, q, s?

Thread safety in arbitrary precision libraries

Most existing arbitrary precision libraries today are not thread-safe. Thus applications written to use them do not work in a multi-thread shared-memory parallel environment.

Currently the GNU MPFR package is the only arbitrary precision library that is thread-safe. It is also very fast (esp. for transcendentals), and correctly rounds results.

However:

- MPFR is only a low-level library; it does not include high-level language bindings required to support full-scale application codes.
- MPFR requires a lengthy installation procedure with administrator permission (to install MPFR and GMP on the user's computer), which discourages many users.
- If a thread-safe build option is not properly invoked during installation, the resulting software is not thread-safe.

- L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier and P. Zimmermann, "MPFR: A multiple precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33 (2007), `https://doi.org/10.1145/1236463.1236468`.

DHB's new MPFUN2020 arbitrary precision library

- An all-Fortran design that can be compiled in a few seconds with any Fortran-2008 compliant compiler, including the gfortran, Intel and NAG Fortran compilers, on a variety of systems, including Mac OS X and various Linux platforms.
- A 100% thread-safe design.
- A full-featured high-level Fortran language interface, so that only type statements and a few other minor changes are required to convert most double precision codes.
- Support for common transcendental functions (sin, cos, exp, etc.) and special functions.
- Support for arbitrary precision real and complex datatypes, plus double and quad.
- Support for both a full and a medium precision datatype, which in some applications (such as Poisson problems) results in lower memory usage and memory traffic.
- FFT-based multiplication for faster performance at very high precision.
- A comprehensive test suite, together with six full-scale application test codes.

A separate version calls MPFR for all lower-level operations. It is roughly 15% faster on most applications (several times faster for transcendental functions).

DHB's new four-level multipair PSLQ code for finding integer relations

Given an $(m+1)$-long input vector $X = (x_i)$ of high-precision floating-point values, an integer relation algorithm finds an $(m+1)$-long vector of integers (a_i) such that

$$a_0 x_0 + a_1 x_1 + a_2 x_2 + \cdots + a_m x_m = 0,$$

to within the tolerance of the numeric precision being used. As a simple illustration, suppose one suspects that the real constant

$$\alpha = 2.11959126982917513132984833493468711062807145783249543921443000325120624764609847\ldots$$

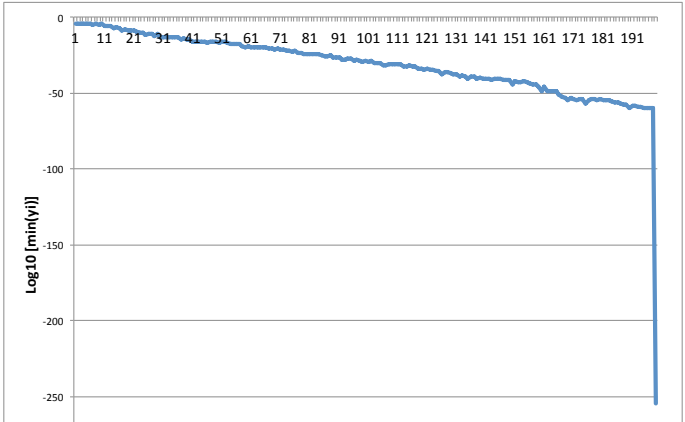
is an algebraic number of degree eight. After computing the vector $(1, \alpha, \alpha^2, \cdots, \alpha^8)$ and applying the multipair PSLQ integer relation algorithm, the relation $(1, -216, 860, -744, 454, -744, 860, -216, 1)$ is produced, so that α satisfies $1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8 = 0$.

DHB's new multipair PSLQ code employs four levels of precision: (a) IEEE double; (b) IEEE quad; (c) medium precision, typically 100-1000 digits; and (d) full precision, typically 5,000 to 50,000 digits.

How can one tell if a computed polynomial is real and not an artifact?

Iterations of the multipair PSLQ algorithm develop a sequence of invertible integer matrices A_n and inverses B_n so that $y = B_n \cdot X$ has steadily smaller entries, until one entry of y is smaller than the epsilon specified for detection.

The size of the drop in $\min(|y_i|)$ when the relation is detected can be viewed as a confidence level that the discovered relation is a real mathematical relation.



High-level computational algorithm

1. Given rationals $x = p/s$ and $y = q/s$ (typically satisfying $1 \le p, q < s/2 \le 50$), select a conjectured minimal polynomial degree m, a medium precision level P_1 digits, a full precision level P_2 digits and other parameters for the run.
2. Calculate $\phi_2(x, y)$ to P_2-digit precision using the following formula:
$$\phi_2(x, y) = \frac{1}{2\pi} \log \left| \frac{\theta_2(z, q)\theta_4(z, q)}{\theta_1(z, q)\theta_3(z, q)} \right|,$$
where $q = e^{-\pi}$ and $z = \frac{\pi}{2}(y + ix)$. Compute the four theta functions using rapidly convergent formulas. Example: $\theta_1(z, q) = 2\sum_{k=1}^{\infty}(-1)^{k-1}q^{(2k-1)^2/4}\sin((2k-1)z)$.
3. 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.
4. Apply the four-level multipair PSLQ algorithm to find an integer relation for X, if one exists, to the precision being used (P_2 digits).
5. If a numerically significant relation is not found, try again with a larger degree m or a higher full precision level P_2. If a relation is found, employ the polynomial factorization facilities in *Mathematica* or *Maple* to ensure that the resulting polynomial is irreducible.

Performance: New vs old codes

Arbitrary precision package	Multipair PSLQ	Run time (CPU seconds) $x = y = 1/29$	Run time (CPU seconds) $x = y = 1/35$
Old all-Fortran	3-level	247819.68	220202.68
New all-Fortran	3-level	55691.66	49221.37
	4-level	52920.67	47093.43
New MPFR-based	3-level	45038.87	41442.57
	4-level	45411.23	39762.23

The two application cases displayed are:

- $x = y = 1/29$: requires 19,700-digit arithmetic (full precision); produces a polynomial of degree 196 with coefficients ranging from 1 up to roughly 10^{87}.
- $x = y = 1/35$: requires 18,900-digit arithmetic (full precision), producing a polynomial of degree 192 with coefficients ranging from 1 up to roughly 10^{85}.

Results: Degrees of minimal polynomials for $x = 1/s; y = q/s$

	q: first row 1–10; second row, 11–20									
	1	2	3	4	5	6	7	8	9	10
s	11	12	13	14	15	16	17	18	19	20
10	8	16	4	16						
11	30	30	30	30	30					
12	16	32	16	32	8					
13	36	36	36	36	18	36				
14	24	48	24	48	24	48				
15	32	32	32	16	32	32	32			
16	32	64	32	64	32	64	16			
17	64	64	64	32	64	64	64	64		
18	36	72	36	72	36	72	36	72		
19	90	90	90	90	90	90	90	90	90	
20	32	64	32	64	32	64	32	64	16	
21	96	96	96	96	96	96	96	48	96	96
22	60	120	60	120	60	120	60	120	60	120
23	132	132	132	132	132	132	132	132	132	132
	132									

Results: Degrees of minimal polynomials for $x = 1/s; y = q/s$

	q: first row 1–10; second row, 11–20									
	1	2	3	4	5	6	7	8	9	10
s	11	12	13	14	15	16	17	18	19	20
24	64	128	64	128	32	128	32	128	64	128
	32									
25	100	100	100	100	100	100	50	100	100	100
	100	100								
26	72	144	72	144	36	144	72	144	72	144
	72	144								
27	162	162	162	162	162	162	162	162	162	162
	162	162	162							
28	96	192	96	192	96	192	96	192	96	192
	96	192	48							
29	196	196	196	196	196	196	196	196	196	196
	196	98	196	196						
30	64	128	64	128	64	128	64	128	64	128
	32	128	64	128						
31	240	240	240	240	240	240	240	240	240	240
	240	240	240	240	240					

Results: Degrees of minimal polynomials for $x = 1/s; y = q/s$

	q: first row 1–10; second row, 11–20									
	1	2	3	4	5	6	7	8	9	10
s	11	12	13	14	15	16	17	18	19	20
32	128	256	128	256	128	256	128	256	128	256
	128	256	128	256	64					
33	240	240	240	240	240	240	240	240	240	120
	240	240	240	240	240	240				
34	128	256	128	256	128	256	128	256	128	256
	128	256	64	256	128	256				
35	192	192	192	192	192	96	192	192	192	192
	192	192	192	192	192	192	192			
36	144	288	144	288	144	288	144	288	144	288
	144	288	144	288	144	288	72			
38	180	360	180	360	180	360	180	360	180	360
	180	360	180	360	180	360	180	360		
39	288	288	288	288	288	288	288	288	288	288
	288	288	288	144	288	288	288	288	288	
40	128	256	128	256	128	256	128	256	64	256
	64	256	128	256	128	256	128	256	64	

Research findings

Note that many of these results are not consistent with Kimberley's formula, which has been proven only for the cases $p = q = 1$ (i.e., $x = 1/s$; $y = 1/s$).

For example, note that when s is even, the degrees for even q are, in most cases, double the degrees for $q = 1$. But there are exceptions, when the degrees are only half as large. For instance when $s = 36$, degrees alternate between 144 and 288, yet when $q = 17$, the degree is 72.

There are also anomalies when s is odd. Note, for instance, that when $s = 35$, all degrees are 192, as given by Kimberley's formula, except the degree is 96 when $q = 6$.

Questions:

- Is there a generalization of Kimberley's formula for the much larger class $x = p/s, y = q/s,\ 1 \le p, q < s/2$?
- How can we greatly expand the scope of these computations? A new integer relation algorithm?
- How can massively parallel computing be employed? Parallel implementations to date have yielded only modest speedups, compared with the best serial code.

Degree-100 minimal polynomial found for the case $x = y = 1/25$

$$
\begin{array}{l}
-1 \\
-33300\,\alpha^{1} \\
-8029750\,\alpha^{2} \\
+1542379500\,\alpha^{3} \\
-293348778825\,\alpha^{4} \\
-28080295402080\,\alpha^{5} \\
-6629781350330800\,\alpha^{6} \\
+58722280292148000\,\alpha^{7} \\
-7197243181862115100\,\alpha^{8} \\
+182328184490475806640\,\alpha^{9} \\
+1047464861734017883208\,\alpha^{10} \\
+18061850677546574917 60\,\alpha^{11} \\
-2794588132471771782562700\,\alpha^{12} \\
+43132729144820226386860000\,\alpha^{13} \\
-256411580898396643830482000\,\alpha^{14} \\
+1396700356872030954211266400\,\alpha^{15} \\
+9088870493426588389401095750\,\alpha^{16} \\
-73359906455331763936496182500 0\,\alpha^{17} \\
+137041495749203861918095715885 00\,\alpha^{18} \\
-1578192373620900935457694576413 20\,\alpha^{19} \\
+1321853578066306850140386747623398\,\alpha^{20} \\
-8347129396578224228862993897237280\,\alpha^{21} \\
+38212880291083166937296196441746800\,\alpha^{22} \\
-104022238521780343721711917072676000\,\alpha^{23} \\
-74032065945345957753107951351295900\,\alpha^{24} \\
+2695579175102470750839898367021130608\,\alpha^{25} \\
-17580580889577528003991569683501290200\,\alpha^{26} \\
+74952710236773360685253102982788566000\,\alpha^{27} \\
-239080432366992082679933177546734267500\,\alpha^{28} \\
+594175511524312220492120757330324484000\,\alpha^{29} \\
-1200481201725892359379030176076009068400\,\alpha^{30} \\
+2370791216795342465482695956290874196000\,\alpha^{31} \\
-6958365090143220739303965994873839612375\,\alpha^{32} \\
+29340961737653182271816694472006500014100\,\alpha^{33} \\
-118465202788489603844178657796002545834090\,\alpha^{34} \\
+394388553221080980775811796973709891661972\,\alpha^{35} \\
-1059773119301328191261276484738640092975935\,\alpha^{36} \\
+2274309666161593753722296548558864618267200\,\alpha^{37} \\
-3729748087865447793480560952748424761349600\,\alpha^{38} \\
+3877725636887659946158475367090512632719680\,\alpha^{39} \\
+846723867393661736729146827990176999234824\,\alpha^{40} \\
-15960428173973401824575411855716955373304480\,\alpha^{41} \\
+47118520726980412118527981001595834084077200\,\alpha^{42} \\
-96560391964393397280402148487438432608004000\,\alpha^{43} \\
+159442357501276027171661799158377030358819240\,\alpha^{44} \\
-222707090658310894211715578612740014870494016\,\alpha^{45} \\
+268336339651868593336977017479411555696276960\,\alpha^{46} \\
-280387518545606966350071856817231380200513600\,\alpha^{47} \\
+252610498127566038453865516310244367924136500\,\alpha^{48} \\
-192275255815619622137470846245628244410759600\,\alpha^{49} \\
+117222736648159046124836188054855685036349976\,\alpha^{50} \\
-47209171248166363599083489199392291307559600\,\alpha^{51} \\
-4942922820170238254884354265559936285783500\,\alpha^{52} \\
+36400498321158716446352704351511072445758400\,\alpha^{53} \\
-50714975271028177138621429774290438585502240\,\alpha^{54} \\
+52508039392118222466312798018961036691290304\,\alpha^{55} \\
-45426960236840080028112470330711413632425560\,\alpha^{56} \\
+33189561099972574095810629259221045183420000\,\alpha^{57} \\
-20184727758357355309386195853692979100127600\,\alpha^{58} \\
+10078674543222169818704320601337376637695840\,\alpha^{59} \\
-4216672803412456875435463721793522689401592\,\alpha^{60} \\
+1722566374653535529797841998864588575882560\,\alpha^{61} \\
-940122938034032272236892859372976755103200\,\alpha^{62} \\
+672500761678739947489218957049737685556800\,\alpha^{63} \\
-461584164001204318877039495371276336862015\,\alpha^{64} \\
+265582157437208924810452188771019969018068\,\alpha^{65} \\
-127220613278091048562581392274127224367210\,\alpha^{66} \\
+52420552698515377367407956646216340562900\,\alpha^{67} \\
-19432168360605247012293398573854487548375\,\alpha^{68} \\
+6715976041225158674019447203640258687520\,\alpha^{69} \\
-2160843064282037086551775761827735549808\,\alpha^{70} \\
+617282737944009515075102146542202738080\,\alpha^{71} \\
-145142540389934636970079472194894728300\,\alpha^{72} \\
+25729074826332880513711838275787958000\,\alpha^{73} \\
-3264527861366792560817221584368055000\,\alpha^{74} \\
+473260992168563618404463109451174000\,\alpha^{75} \\
-178858482702935977606858126791577500\,\alpha^{76} \\
+66028976498454634546379541782684000\,\alpha^{77} \\
-15761469487294858226633454090400400\,\alpha^{78} \\
+2480913858883135298303773583710384 0\,\alpha^{79} \\
-283939423573690705553213818042394\,\alpha^{80} \\
+27924016290268330781557752871960\,\alpha^{81} \\
-18902649357383087571408191955 00\,\alpha^{82} \\
-1976460509542059894968712490 00\,\alpha^{83} \\
+92058025400867206020845473350\,\alpha^{84} \\
-15898833972285544673912261280\,\alpha^{85} \\
+1776153714893883182147476400\,\alpha^{86} \\
-134561774900933516621588000\,\alpha^{87} \\
+5713502521319481241590900\,\alpha^{88} \\
-280130362788084262599 20\,\alpha^{89} \\
-1074061784968018208453 6\,\alpha^{90} \\
+416466536564927187120\,\alpha^{91} \\
-5335454032157718300\,\alpha^{92} \\
-61275002107276000\,\alpha^{93} \\
-662049081315760\,\alpha^{94} \\
+123800803998624\,\alpha^{95} \\
-330633806665\,\alpha^{96} \\
-8699478100\,\alpha^{97} \\
+13986250\,\alpha^{98} \\
+6700\,\alpha^{99}
\end{array}
$$

Suggestions for future research in variable precision computing

1. Seamlessly incorporating an arbitrary precision facility, say based on the MPFR package, in future Fortran and C language standards.
2. Checking all available software supporting variable and arbitrary precision for thread safety, and fixing or deprecating libraries that are not strictly thread-safe.
3. Rethinking automatic type conversions and mixed-mode operations, which hinder the development of correct and reproducible variable precision computing, in DHB's opinion. At the least, could some of these features be optional?
4. Investigating novel numerical algorithms, error control and error analysis techniques, appropriate not only for single and double precision on a single processor system, but also for the realm of very high precision and very highly parallel computing.
5. Developing fundamentally new paradigms for variable precision computing, perhaps along the lines of Gustafson's "unum" framework.

This talk is available at:
http://www.davidhbailey.com/dhbtalks/dhb-sc21-correct.pdf