The two-level multipair PSLQ algorithm

David H. Bailey*

May 2, 2024

Abstract

Given a vector of real or complex numbers $\{x_1, x_2, \cdots, x_n\}$, an *integer relation algorithm* is a computational scheme to find n integers a_k, if they exist, such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$ (to within available numeric precision). Two widely used schemes are the PSLQ and the multipair PSLQ algorithms, the latter of which is moderately well-suited for parallel processing, and, as a bonus, is somewhat more efficient in terms of run time and usage of numeric precision than the original PSLQ algorithm. The computational efficiency of these algorithms can be dramatically increased by employing "multilevel" implementations, which perform most iterations in ordinary double precision, updating mulitprecision arrays only as needed. This paper presents full details of an efficient two-level multipair PSLQ implementation as a guide to future researchers.

1 Introduction

Let $\{x_1, x_2, \cdots, x_n\}$ be a vector of real or complex numbers. An *integer relation algorithm* is a computational scheme to find n integers a_k, if they exist, such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$ (to within available numeric precision). Numerous applications have been identified for integer relation algorithms, ranging from finding the minimal polynomial of some computed algebraic number to the identification of definite integrals that arise in mathematical physics [3, 1, 2]. Such computations usually require extremely high numeric precision, typically a few hundred digits but in some cases up to 100,000 digits.

One typical application of an integer relation algorithm is to recover the minimal polynomial of a degree-m algebraic number α, whose value can be computed to high precision. The procedure is to compute the $(m + 1)$-long vector $x = (1, \alpha, \alpha^2, \cdots, \alpha^m)$ and apply an integer relation algorithm. If an integer relation (a_i) is found for x that holds to the level of precision being used, then the resulting vector of integers may be the coefficients of an integer polynomial of degree m satisfied by α, subject to further verification.

As an illustration, suppose one suspects that the real constant α, whose numerical value to 40 digits is $2.1195912698291751313298483349346871106280\ldots$, is an algebraic number of degree eight. After computing the vector $(1, \alpha, \alpha^2, \cdots, \alpha^8)$ and applying an integer relation algorithm, the relation $(1, -216, 860, -744, 454, -744, 860, -216, 1)$ is produced, so that α appears to satisfy the polynomial $1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8 = 0$.

*Lawrence Berkeley National Laboratory (retired), Berkeley, CA, USA, dhbailey@lbl.gov.

Two widely used schemes are the PSLQ algorithm, due to Helaman Ferguson [4], and the multipair PSLQ algorithm, the latter of which is moderately well-suited for parallel processing, and, as a bonus, is somewhat more efficient in terms of run time and numeric precision than the original PSLQ algorithm [3].

The paper [3] briefly described "multilevel" implementations of both standard PSLQ and multipair PSLQ. These schemes dramatically reduce run times (by as much as 100X) by performing most iterations in ordinary double precision, and updating multiprecision arrays only as needed. However, orchestrating an efficient multilevel implementation is a significant challenge. Accordingly, this paper presents full details of an efficient two-level multipair PSLQ implementation to guide future researchers.

It should be added that the present author has also implemented a three-level version of the multipair PSLQ algorithm, employing ordinary double precision, medium precision (typically 200–5,000 digits), and full precision (typically 5,000–100,000 digits). The three-level scheme is significantly more complicated than the two-level scheme, but on large problems typically saves up to 30% in run time. Details of the three-level scheme will not be given here; please contact the present author for details and code.

2 The multipair PSLQ algorithm

As mentioned above, the original PSLQ algorithm is not well suited for modern parallel computer systems, so the "multipair PSLQ" algorithm was developed in [3]. It greatly reduces the number of sequential iterations that must be performed, and exhibits moderately high concurrency in the major steps of individual iterations. As a plus, it is somewhat more efficient with run time and numerical precision than the original PSLQ.

A full statement of the multipair PSLQ algorithm is as follows, taken from [3] (note that the online version and the statement below correct an error in the original published version). Here $\gamma = \sqrt{4/3}$ as before, and $\beta = 0.4$.

One-level multipair PSLQ algorithm:

Initialize:

1. For $j := 1$ to n: for $i := 1$ to n: if $i = j$ then set $A_{ij} := 1$ and $B_{ij} := 1$ else set $A_{ij} := 0$ and $B_{ij} := 0$; endfor; endfor.

2. For $k := 1$ to n: set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor; set $t = 1/s_1$; for $k := 1$ to n: set $y_k := tx_k$; $s_k := ts_k$; endfor.

3. Initial H: For $j := 1$ to $n-1$: for $i := 1$ to $j-1$: set $H_{ij} := 0$; endfor; set $H_{jj} := s_{j+1}/s_j$; for $i := j+1$ to n: set $H_{ij} := -y_i y_j/(s_j s_{j+1})$; endfor; endfor.

Iteration: Repeat the following steps until precision has been exhausted or a relation has been detected.

1. Sort the entries of the $(n-1)$-long vector $\{\gamma^i |H_{ii}|\}$ in decreasing order, producing the sort indices.

2. Beginning at the sort index m_1 corresponding to the largest $\gamma^i |H_{ii}|$, select pairs of indices $(m_i, m_i + 1)$, where m_i is the sort index. If at any step

either m_i or $m_i + 1$ has already been selected, pass to the next index in the list. Continue until either βn pairs have been selected, or the list is exhausted. Let p denote the number of pairs actually selected in this manner.

3. For $i := 1$ to p, exchange the entries of y indexed m_i and $m_i + 1$, and the corresponding rows of A, B and H; endfor.

4. Remove corners on H diagonal: For $j := 1$ to p: if $m_j \le n - 2$ then set $t_0 := \sqrt{H^2_{m_j,m_j} + H^2_{m_j,m_j+1}}$, $t_1 := H_{m_j,m_j}/t_0$ and $t_2 := H_{m_j,m_j+1}/t_0$; for $i := m_j$ to n: set $t_3 := H_{i,m_j}$; $t_4 := H_{i,m_j+1}$; $H_{i,m_j} := t_1 t_3 + t_2 t_4$; and $H_{i,m_j+1} := -t_2 t_3 + t_1 t_4$; endfor; endif; endfor.

5. Reduce H: For $i := 2$ to n: for $j := 1$ to $n - i + 1$: set $l := i + j - 1$; for $k := j + 1$ to $l - 1$: set $H_{lj} := H_{lj} - T_{lk} H_{kj}$; endfor; set $T_{lj} := \mathrm{nint(H_{lj}/H_{jj})}$ and $H_{lj} := H_{lj} - T_{lj} H_{jj}$; endfor; endfor. [Note that the $n \times (n-1)$ integer array T is set before it is used.]

6. Update y: For $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $y_j := y_j + T_{ij} y_i$; endfor; endfor.

7. Update A and B: For $k := 1$ to n: for $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $A_{ik} := A_{ik} - T_{ij} A_{jk}$ and $B_{jk} := B_{jk} + T_{ij} B_{ik}$; endfor; endfor; endfor.

8. Norm bound: Compute $M := 1/\max_j |H_{jj}|$. Then there can exist no relation vector whose Euclidean norm is less than M.

9. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection epsilon, a relation has been detected and is given in the corresponding row of B.

It should be added that for a basic implementation with just one level of precision, it is not necessary to compute the A matrix, which is the inverse of the B matrix. However, in two- or three-level implementations, it is necessary to compute the A matrix, as we shall see below. Along this line, it is not necessary to compute the norm bound at every iteration; typically this is computed only periodically for informational purposes.

3 LQ decomposition

Although a single-level implementation of multipair PSLQ does not require an LQ (lower-quadrature) matrix decomposition, it is required for a two- or three-level implementation. Thus before continuing we present the LQ decomposition algorithm:

LQ decomposition of input (and output) $n \times m$ matrix H:

For $l = 1$ to $\min(m, n)$: if $l = m$ then go to Label A.

Compute Householder transformation for column l: Set $N := \sqrt{\sum_{i=0}^{m-l} H_{l,l+i}}$; if $N = 0$ then go to Label A; else if $H_{l,l} \ne 0$ then set the sign of N to be the same as the sign of $H_{l,l}$; for $i = 0$ to $m - l$ set $H_{l,l+i} := H_{l,l+i}/N$; set $H_{l,l} := 1 + H_{l,l}$.

Apply the transformation to remaining H matrix: For $j = l+1$ to n; set $t = -\left(\sum_{i=0}^{m-l} H_{l,l+i}H_{j,l+i}\right)/H_{l,l}$; for $i = 0$ to $m-l$ set $H_{j,l+i} := H_{j,l+i} + tH_{l,l+i}$ end for; end for.

Set $H_{l,l} := -N$. Label A. End for.

4 Two-level implementation of multipair PSLQ

As mentioned above, the run-time performance of PSLQ or multipair PSLQ can be dramatically accelerated (up to 100X) by employing two or even three levels of precision, performing most iterations of the algorithm in ordinary IEEE double precision arithmetic.

The key challenge here is to perform as many iterations as possible in double precision, yet to stop the current set of iterations and update the multiprecision arrays before any entry of the double precision A and B arrays exceeds $2^{53} \approx 9.007 \times 10^{15}$ in absolute value (most whole numbers larger than this are not be representable exactly as IEEE double precision floats). If a larger value arises in A or B, then the integrity of the algorithm has been lost. In practice, it is necessary to set this limit at $2^{52} \approx 4.503 \times 10^{15}$, because it is possible for an intermediate value in the expression computing an entry of A or B to exceed 2^{53}, even if the resulting value stored in A or B does not exceed 2^{53}.

The present author has found that a fairly efficient rule is to stop double precision iterations when the maximum absolute value of A or B exceeds 10^{13}, or when the smallest absolute value of the Y vector is less than 10^{-14}. This works satisfactorily in most cases, but in large problems occasionally an A or B value exceeding 2^{52} may still arise. Thus a reliable two-level implementation must include code to detect and handle these precision failures: In most such cases, it is sufficient to revert to double precision arrays saved at the beginning of the current iteration, update the multiprecision arrays, reset A and B to identity matrices, and then continue with double precision iterations; but occasionally it is necessary to perform some iterations with multiprecision arithmetic before continuing in double precision. Full details are given in algorithm below.

An alternative to the test checking whether the maximum absolute value of the double precision A or B matrices exceeds 2^{52} is to utilize the IEEE_ARITHMETIC module available in several compiler systems, including, for example, the gcc/gfortran compilers. This allows one to test if the flag IEEE_INEXACT has been set during updates of the double precision A and B arrays. If true, then a precision failure has occurred and must be dealt with as described above.

Here is a detailed statement of the author's implementation (some relatively minor steps, e.g., to control output, are not included here). Int denotes integer, DP denotes double precision and MPR denotes multiprecision real. The variable names below are the same as in the author's code, which is available as part of the MPFUN2020 software package at
`https://www.davidhbailey.com/dhbsoftware`.

PSLQM2 (two-level multipair PSLQ):

Input arguments and parameters:

N	Int	Length of input vector X and output relation vector R.
NDP	Int	Precision level in digits.
$NWDS$	Int	Precision level in words (set by a formula based on NDP).
NDR	Int	$\log_{10}$ of the min acceptable dynamic range of Y vector at detection; default = 30. A smaller range is deemed unreliable.
NRB	Int	$\log_{10}$ of max size (Euclidean norm) of acceptable relation; default = 200.
NEP	Int	$\log_{10}$ of full precision epsilon; default = $30 - NDP$; for large problems replace 30 by 50 or 60.
X	MPR	N-long input multiprecision vector.
IQ	Int	Output flag: 0 (unsuccessful) or 1 (successful).
R	MPR	N-long output integer relation vector, if successful, else zeroes.
IPM	Int	Iteration check interval; default = 10.
NSQ	Int	Second dimension of $DSYQ$ and SYQ; default = 8.
$DEPS$	DP	Double precision epsilon; default = 10^{-14}.
$DREP$	DP	Dynamic range epsilon; default = 10^{-10}.

Integer variables:
IT, ITS, IMQ, IZD, IZM

Double precision arrays and variables (with dimensions):
$DA(N,N), DB(N,N), DH(N,N), DYSQ(N,NSQ), DY(N)$

Multiprecision real arrays and variables (with dimensions):
$B(N,N), H(N,N), SYQ(N,NSQ), Y(N), EPS, T$

Algorithm:

1. Initialize MPR arrays:
   a. Set $EPS = 10^{NEP}$.
   b. Set B to an $N \times N$ identity matrix.
   c. Compute initial H matrix as given in multipair PSLQ algorithm above.
   d. Set SYQ array to zeroes.
   e. Set IZD, IZM, IMQ, IT and ITS to zero.
2. Check if min/max absolute value of $Y < DREP$. This is often true for the first few tens of iterations. If true, go to Step 7 below.
3. Initialize DP arrays:
   a. Set T = max absolute value of Y; set $DY = Y/T$, rounded to DP.
   b. Set T = max absolute value of H; set $DH = H/T$, rounded to DP.
   c. Set DA and DB to identity matrices.
   d. Set $DYSQ$ array to zeroes.
4. Perform an LQ decomposition on DH using DP arithmetic.
5. Perform one multipair PSLQ iteration using DP arithmetic:
   a. Increment iteration count: $IT := IT + 1$; set $IZD = 0$.
   b. Save the input DA, DB, DH and DY arrays.

c. Follow the steps in multipair PSLQ algorithm above to update DY, DA, DB and DH; however, if flag $IMQ = 1$ from a previous iteration, only select one pair of entries (i.e., set $p = 1$ in Steps 2, 3 and 4 in the multipair PSLQ algorithm), then set $IMQ = 0$. It is not necessary to compute the norm bound in the DP iterations.

d. If the min absolute value of $DY < DEPS$, then set $IZD = 1$. If the max absolute value of DA or DB exceeds 10^{13}, but less than 2^{52}, then set $IZD = 1$. If the max absolute value of DA or DB exceeds 2^{52} (precision failure), then set $IZD = 2$ and restore the DA, DB, DH and DY arrays saved above in Step 5b.

e. Compare the DY vector with the DY vectors of recent iterations saved in array $DYSQ$; if a match is found, set flag $IMQ = 1$, which instructs the next iteration to be performed using only one pair of indices in the multipair scheme (in practice, this occurs only very rarely).

f. Save DY vector in row K of $DYSQ$, where $K = 1 + \mathrm{mod}(IT, NSQ)$ (i.e., DY vectors are stored a circular sequence in the $DYSQ$ array).

6. Check flags and, if needed, update MPR arrays from DP arrays:

   a. If $IZD = 0$, go to Step 5 (continue DP iterations). If $IZD = 2$, but $IT > ITS + 1$ (i.e., if the current iteration is more than one plus the previous iteration when a MPR update was performed), then set $IZD = 1$, so that after an MPR update, regular DP iterations can continue. But if $IT = ITS + 1$ (i.e., an MPR update was performed on the previous iteration), then leave $IZD = 2$.

   b. Update MPR arrays from DP arrays: Set $Y := DB \times Y$ (matrix multiplication), then find the min absolute value of the updated Y. Set $B := DB \times B$ (matrix multiplication), then find max absolute value of updated B. Set $H := DA \times H$ (matrix multiplication). Set $ITS = IT$.

   c. The max norm bound may optionally be computed here, as described in the multipair PSLQ algorithm above, and output, along with the current min and max of Y and other data for informational purposes.

   d. If the min absolute value of Y is less than $EPS\times$ max absolute value of B (tentative detection), then set $IZM = 1$; else if the min absolute value of Y is less than $EPS \times 2^{72}\times$ max absolute value of B (precision exhausted), then set $IZM = 2$; else set $IZM = 0$.

   e. Test output flag: If $IZM = 0$, then if $IZD = 2$, go to Step 7 (start MPR iterations), else go to Step 2 (start DP iterations); else if $IZM = 1$, go to Step 9 (exit); else if $IZM = 2$, go to Step 9 (exit).

7. Perform an LQ decomposition on H using MPR arithmetic.

8. Perform one multipair PSLQ iteration using MPR arithmetic:

   a. Increment iteration count: $IT := IT + 1$; set $IZM = 0$.

   b. Perform one iteration of the multipair PSLQ algorithm using MPR arithmetic, as described above (except no need to compute norm bound).

   c. If the min absolute value of Y is less than $EPS\times$ max absolute value of B (tentative detection), then set $IZM = 1$; else if the min absolute value of Y is less than $EPS \times 2^{72}\times$ max absolute value of B (precision exhausted), then set $IZM = 2$.

d. Test output flag: If $IZM = 0$, then periodically (every IPM iterations since the last MPR update) check if the min/max dynamic range of $Y < DREP$; if true, go to Step 8 (continue MPR iterations); else go to Step 3 (start DP iterations); else if $IZM = 1$, go to Step 9 (exit); else if $IZM = 2$, go to Step 9 (exit).

9. Exit:

   a. If $IZM = 1$ find the index of Y with the min absolute value; set $R =$ row of B corresponding to that index. If the Euclidean norm of the relation is less than 10^{NRB}, and the dynamic range of the Y vector is at least 10^{NDR}, set $IQ = 1$ (success) and exit; otherwise set $R =$ zeroes and set $IQ = 0$ (failure), then exit.

   b. If $IZM = 2$, then set $R =$ zeroes and set $IQ = 0$ (failure), then exit.

5 Sample test results

The algorithm above has been tested on several problems, including these two:

1. Recover the degree-56 minimal polynomial of the algebraic number $\alpha = 3^{1/7} - 2^{1/8} = 0.079423080093429\ldots$. This run employed 750-digit arithmetic and ran 3.4 seconds on the author's Apple Mac Studio system, with the detection at iteration 2,893. The resulting polynomial is:

$$\begin{aligned}0 = {} & 6433 - 10752\alpha - 330624\alpha^2 - 4523904\alpha^3 - 26535600\alpha^4 \\ & - 52744608\alpha^5 - 17513496\alpha^6 - 17496\alpha^7 + 448\alpha^8 - 3806208\alpha^9 \\ & + 337256640\alpha^{10} - 3329569152\alpha^{11} + 3802034376\alpha^{12} - 217020384\alpha^{13} \\ & + 20412\alpha^{14} - 672\alpha^{16} - 25366656\alpha^{17} - 2748602304\alpha^{18} \\ & - 7518801024\alpha^{19} - 358251012\alpha^{20} - 13608\alpha^{21} + 560\alpha^{24} - 25826304\alpha^{25} \\ & + 944957664\alpha^{26} - 132239520\alpha^{27} + 5670\alpha^{28} - 280\alpha^{32} - 5146848\alpha^{33} \\ & - 11195352\alpha^{34} - 1512\alpha^{35} + 84\alpha^{40} - 143808\alpha^{41} + 252\alpha^{42} - 14\alpha^{48} \\ & - 24\alpha^{49} + \alpha^{56}\end{aligned}$$

2. Recover the degree-64 minimal polynomial of the algebraic number $\alpha = \exp(8\pi\phi_2(x,y) = 1767.3603891331088833\ldots$, where

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

in the specific case $x = y = 1/17$. The $\phi_2(x,y)$ function can be efficiently computed as [2]:

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

where $q = e^{-\pi}$ and $z = \pi/2 \cdot (y+ix)$. The theta functions can be computed by these rapidly convergent formulas:

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

This run employed 2,500-digit arithmetic and ran 52.5 seconds, with the detection at iteration 9,495. The resulting polynomial is:

$$\begin{aligned}
0 = {} & 1 + 6912\alpha - 1023008\alpha^2 + 535196800\alpha^3 + 7742027760\alpha^4 - 2451239864832\alpha^5 \\
& + 140264665723552\alpha^6 - 2494265652888704\alpha^7 + 18453445522215032\alpha^8 \\
& + 21614293158955264\alpha^9 - 1840469978381611680\alpha^{10} + 26560170568288794240\alpha^{11} \\
& - 219265475764921569840\alpha^{12} + 1143759465759937297408\alpha^{13} \\
& - 4563932639248948435424\alpha^{14} + 21048406812137688311168\alpha^{15} \\
& - 123756069205191278016740\alpha^{16} + 662708878348907477250816\alpha^{17} \\
& - 2671051287612630032421280\alpha^{18} + 7693234584556635821267584\alpha^{19} \\
& - 14862548097474240887146768\alpha^{20} + 11985439092809681992002048\alpha^{21} \\
& + 44351668349396581870408736\alpha^{22} - 259625664937972467300807296\alpha^{23} \\
& + 803186115899676703948238664\alpha^{24} - 1789602095389051533149533952\alpha^{25} \\
& + 3055552833334608777606289376\alpha^{26} - 4156271487999506323835036544\alpha^{27} \\
& + 4903963676671959157531751248\alpha^{28} - 6019517253583536219231909888\alpha^{29} \\
& + 8780067564346216307831284640\alpha^{30} - 13334548483907481046238812288\alpha^{31} \\
& + 17362857489419448630866293318\alpha^{32} - 17345855629600599241800189696\alpha^{33} \\
& + 11966489230110362129440701856\alpha^{34} - 3898119322387426442055756416\alpha^{35} \\
& - 2451983939727870545406928048\alpha^{36} + 4743446591055878746050587136\alpha^{37} \\
& - 3881818694457698660972764704\alpha^{38} + 2101492937309911776817793664\alpha^{39} \\
& - 830074840813669608610951352\alpha^{40} + 269366792757186303037874944\alpha^{41} \\
& - 96596567511508184274883040\alpha^{42} + 46311532722057913438161792\alpha^{43} \\
& - 22155672572673873192657168\alpha^{44} + 8153783303351403692882944\alpha^{45} \\
& - 2079969173966458011379616\alpha^{46} + 331427117746835861477504\alpha^{47} \\
& - 18856552838875733014756\alpha^{48} - 7235322856083561662208\alpha^{49} \\
& + 329260920507960885856\alpha^{50} - 738833647673944491136\alpha^{51} \\
& + 76552613117134517712\alpha^{52} - 1424154241008650752\alpha^{53} + 342676113911934816\alpha^{54} \\
& - 8982528472719040 0\alpha^{55} + 3891480748650616\alpha^{56} - 154854254425344\alpha^{57} \\
& - 3704022727520\alpha^{58} + 404224147840\alpha^{59} - 125943824\alpha^{60} + 62013440\alpha^{61} \\
& - 670240\alpha^{62} - 1408\alpha^{63} + \alpha^{64}
\end{aligned}$$

References

[1] David H. Bailey and Jonathan M. Borwein, “Experimental mathematics: Examples, methods and implications,” *Notices of the American Mathematical Society*, vol. 52, no. 5 (May 2005), 502–514, available at `https://www.ams.org/notices/200505/fea-borwein.pdf`.

[2] David H. Bailey, Jonathan M. Borwein, Jason Kimberley and Watson Ladd, “Computer discovery and analysis of large Poisson polynomials,” *Experimental Mathematics*, 27 Aug 2016, vol. 26, 349–363, available at `https://www.davidhbailey.com/dhbpapers/poisson-res.pdf`.

[3] David H. Bailey and David J. Broadhurst, “Parallel integer relation detection: Techniques and applications,” *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), 1719-1736, preprint draft available at `https://www.davidhbailey.com/dhbpapers/ppslq.pdf`.

[4] Helaman R. P. Ferguson, David H. Bailey and Stephen Arno, “Analysis of PSLQ, an integer relation finding algorithm,” *Mathematics of Computation*, vol. 68, no. 225 (Jan 1999), 351–369, preprint draft available at `https://www.davidhbailey.com/dhbpapers/cpslq.pdf`.