A Polynomial Time, Numerically Stable Integer Relation Algorithm

Helaman R. P. Ferguson and David H. Bailey

RNR Technical Report RNR-91-032

July 14, 1992

Abstract

Let $x = (x_1, x_2, \cdots, x_n)$ be a vector of real numbers. x is said to possess an integer relation if there exist integers a_i not all zero such that $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0$. Beginning in 1977 several algorithms (with proofs) have been discovered to recover the a_i given x. The most efficient of these existing integer relation algorithms (in terms of run time and the precision required of the input) has the drawback of being very unstable numerically. It often requires a numeric precision level in the thousands of digits to reliably recover relations in modest-sized test problems.

We present here a new algorithm for finding integer relations, which we have named the "PSLQ" algorithm. It is proved in this paper that the PSLQ algorithm terminates with a relation in a number of iterations that is bounded by a polynomial in n. Because this algorithm employs a numerically stable matrix reduction procedure, it is free from the numerical difficulties that plague other integer relation algorithms. Furthermore, its stability admits an efficient implementation with lower run times on average than other algorithms currently in use. Finally, this stability can be used to prove that relation bounds obtained from computer runs using this algorithm are numerically accurate.

Ferguson is with the Supercomputing Research Center, 17100 Science Drive, Bowie, MD 20715. Internet: helamanf@super.org. Bailey is with NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035. Internet: dbailey@nas.nasa.gov.

1. Introduction

Let $x = (x_1, x_2, \cdots, x_n)$ be a vector of real numbers. x is said to possess an integer relation if there exist integers a_i not all zero such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$. By an integer relation algorithm, we mean an algorithm that is guaranteed (provided the computer implementation has sufficient numeric precision) to recover the integers a_i and to produce bounds within which no integer relations can exist.

One application of an integer relation algorithm is to solve "subset sum problems," wherein one determines what subset of a certain list of integers has a given sum. In other words, a subset sum problem is an integer relation problem where the relation coefficients a_i are zero or one. This application is discussed in [11] and [5].

Another application of an integer relation algorithm is to determine whether or not a certain mathematical constant, whose value can be computed to high precision, is a root of a polynomial of degree n or less. This can be done for a constant α by setting $x = (1, \alpha, \alpha^2, \cdots, \alpha^{n-1})$ and applying an integer relation algorithm to x. If any integer relation is found to hold (within the limits of the available machine precision), then the resulting a_i are precisely the coefficients of a polynomial satisfied by α (to within machine precision). Even if no relation is found, a calculation with an integer relation algorithm can establish, for example, that the constant cannot possibly satisfy any polynomial of degree n or less whose coefficients are smaller than a bound established by the algorithm. This application is discussed in [4].

The problem of finding integer relations among a set of real numbers was first studied by Euclid, who gave an iterative algorithm which, when applied to two real numbers, either terminates, yielding an exact relation, or produces an infinite sequence of approximate relations. The generalization of this problem for $n > 2$ has been attempted by Euler, Jacobi, Poincare, Minkowski, Perron, Brun, Bernstein, among others. However, none of their iterative algorithms have been proven to work for $n > 3$, and numerous counterexamples have been found.

The first integer relation algorithm with the desired properties mentioned above was discovered by one of the authors (Ferguson) and R. Forcade in 1977 [7]. In the intervening years a number of other integer relation algorithms have been discovered, including a non-recursive variant of the original algorithm [8], the "LLL" algorithm [12], the "HJLS" algorithm [10] (which is based on the LLL algorithm), and the "PSOS" [4] algorithm. These newer algorithms are significantly faster when implemented on a computer and require much less precision in the input x vector to recover the relation than the original algorithm.

2. The HJLS Algorithm

The HJLS algorithm is superior among these existing integer relation algorithms in several respects. For one thing, it has been proven that the number of iterations required for HJLS to recover a relation is bounded by a polynomial in n [10], whereas proofs of this property are lacking for the other algorithms. Further, the HJLS algorithm appears to be the most efficient of these algorithms in terms of its ability to recover the relation satisfied by an input vector known only to limited precision. Finally, based on the authors' experience, HJLS appears to require the lowest average computer time to recover a relation among previously existing algorithms.

Unfortunately, the HJLS algorithm has one serious drawback: it is extremely unstable numerically. Even for modest n and small relations, enormous numeric precision is often required for the algorithm to work properly. For example, consider the 17-long vector $(1, \alpha, \alpha^2, \cdots, \alpha^{16})$, where $\alpha = 3^{1/4} - 2^{1/4}$. This vector has the relation $(1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1)$. Although only 100 digits or so of α are required as input for the relation to be recovered using HJLS, the authors have found that computations must be performed with a numeric precision level of over 10,000 digits.

One outward symptom of numerical failure in a HJLS run is that the program aborts due to the appearance of an extremely large entry in a matrix, one which cannot be rounded exactly to the nearest integer in the current working precision. Another outward symptom of numerical failure is that a computer run with HJLS produces a bound on the norm of possible relations that excludes a relation known to exist. In this second type of failure, comparison with runs using higher precision reveals that some matrix entries had become so corrupted by numerical error that all significance had been lost.

In some trials, computer runs using the HJLS algorithm succeed "by accident" in recovering a relation using only moderate levels of numeric precision, whereas the relation would not be recovered at that point in the computer run if it were performed with higher precision. But such good fortune cannot be relied on, and it is just as likely that the relation will be missed at the point where it should be recovered.

If one asks what level of precision is required for a modest-sized problem before the computer run using HJLS is identical to one using "infinite" precision, then the answer in many cases appears to be thousands of digits, based the authors' experience. "Identical" here means that all decisions are the same and all relation bounds are the same (to 8 digits or so) up to and including the point of recovery. Cases that require very high precision are infrequent for $n \leq 10$ but are quite common for larger n. There does not appear to be any *a priori* means of determining the required HJLS working precision level for a given problem.

As one would expect, these very high levels of numeric precision require large amounts of memory and processing time, although amazingly enough, HJLS is still faster on average than other previously known integer relation algorithms. However, the most significant disadvantage of the HJLS numerical instability is that one can never know with certainty that a relation of a certain size has been excluded, because the bound that is obtained after running HJLS for a while might be completely corrupted with numerical error. One's confidence in a bound result can be enhanced by increasing the numeric precision and verifying that the sequence of norm bounds is the same up to the point determined in the previous run, but one can never be certain of the result.

The root cause of this numerical instability is not known, but it is believed to derive from the fact that HJLS is based upon the Gram-Schmidt orthogonalization algorithm, which is known to be numerically unstable [9].

3. The PSLQ Algorithm

Recently one of the authors (Ferguson) discovered a new polynomial time integer relation algorithm. This algorithm has been named "PSLQ," since it is based on a partial sum of squares scheme like the PSOS algorithm, yet it can be efficiently implemented with a LQ (lower trapezoidal–orthogonal) matrix factorization.

The PSLQ algorithm exhibits the favorable features of the HJLS algorithm, including its ability to recover relations in x vectors known to only limited precision, while completely avoiding its catastrophic instability. Although a straightforward implementation of PSLQ does not appear to be faster than HJLS on average, at least for modest-sized n, the numerical stability of PSLQ admits an efficient implementation using a combination of double precision and multiprecision arithmetic that does appear to be faster than HJLS on average for a wide range of problem sizes. Most importantly, one can show that by using a working precision level that is only slightly higher than that of the input data, bound results obtained from computer runs are reliable.

Let $x \in \mathbf{R}^n$ be a nonzero n-tuple $x = (x_1, x_2, \cdots, x_{n-1}, x_n)$. Define the partial sums of squares, s_j, for x by $s_j^2 = \sum_{j \le k \le n} x_k^2$. Assume that x is a unit vector, so that $s_1 = |x| = 1$. Define the lower trapezoidal $n \times (n-1)$ matrix $H_x = (h_{i,j})$ by

$$h_{i,j} = 0 \qquad 1 \le i < j \le n-1 \tag{1}$$

$$h_{i,j} = \frac{s_{i+1}}{s_i} \qquad 1 \le i = j \le n-1 \tag{2}$$

$$h_{i,j} = -\frac{x_i x_j}{s_j s_{j+1}} \qquad 1 \le j < i \le n \tag{3}$$

Let P be the $n \times n$ matrix given by $P = HH^t$, which has rank $n-1$. By expanding this expression, it can be seen that $P = I_n - x^t \cdot x$, i.e., $P = (p_{i,j})$ where $p_{i,i} = 1 - x_i^2$ and $p_{i,j} = -x_i x_j$ for $i \ne j$. From this it follows that if $x \cdot m^t = 0$ then $Pm^t = m^t$. Let $|\cdot|$ denote the Frobenius norm, i.e., $|A|^2 = \sum a_{i,j}^2$. Then it can be seen that $|P| = |H| = \sqrt{n-1}$.

Given an arbitrary lower trapezoidal $n \times (n-1)$ matrix $H = (h_{i,j})$, define an associated $n \times n$ lower triangular matrix $D \in GL(n, \mathbf{Z})$ as follows. The entries of the $n \times n$ matrix $D = (d_{i,j})$ are given from $d_{i,i-1}$ back to $d_{i,1}$ by means of the recursions

$$d_{i,j} = 0 \qquad 1 \le i < j \le n \tag{4}$$

$$d_{i,j} = 1 \qquad 1 \le i = j \le n \tag{5}$$

$$d_{i,j} = \mathrm{nint}\left(-\frac{1}{h_{j,j}} \sum_{1 \le j < k \le i} d_{i,k} h_{k,j}\right) \qquad 1 \le j < i \le n \tag{6}$$

The function nint denotes the nearest integer function.

The inverse of D, namely $E = D^{-1}$, can be defined with recursions similar to the above for $E = (e_{i,j})$ by

$$e_{i,j} = 0 \qquad 1 \le i < j \le n \tag{7}$$

$$e_{i,j} = 1 \qquad 1 \le i = j \le n \tag{8}$$

$$e_{i,j} = -\sum_{1 \le j < k \le i} e_{i,k} d_{k,j} \qquad 1 \le j < i \le n \tag{9}$$

for $e_{i,i-1}$ back to $e_{i,1}$.

Given an arbitrary lower trapezoidal $n \times (n-1)$ matrix $H = (h_{i,j})$ and a *fixed* integer j, $1 \leq j \leq n-1$, define the $(n-1) \times (n-1)$ orthogonal matrix $G_j \in O(n-1)$ as follows. If $j = n-1$ set $G_{n-1} = I_{n-1}$, the matrix identity. Otherwise, for $j < n-1$, set $h_{j,j} = a$, $h_{j+1,j+1} = c$, $h_{j+1,j} = b$, and $d = \sqrt{b^2 + c^2}$. The entries of the $(n-1) \times (n-1)$ matrix $G_j = (g_{i,k})$ are given by

$$
\begin{aligned}
g_{i,i} &= 1 \qquad 1 \leq i < j \text{ or } j+1 < i \leq n && (10)\\
g_{j,j} &= b/d && (11)\\
g_{j,j+1} &= -c/d && (12)\\
g_{j+1,j} &= c/d && (13)\\
g_{j+1,j+1} &= b/d && (14)\\
g_{i,k} &= 0 \qquad \text{otherwise} && (15)
\end{aligned}
$$

Given a *fixed* integer j, $1 \leq j \leq n-1$ define the $n \times n$ permutation matrix R_j to be that matrix formed by exchanging the j-th and $j+1$-st rows of the $n \times n$ identity matrix I_n.

5. One Iteration of PSLQ

Select a constant $\gamma > 2/\sqrt{3} = 1.1547\cdots$. Suppose we are given three matrices, H, A, B, where H is a $n \times (n-1)$ lower trapezoidal matrix and A and B are $n \times n$ integral matrices, with $B = A^{-1}$. An iteration of the algorithm PSLQ is defined by the following three steps.

1. Replace H by DH.
2. Select an integer j, $1 \leq j \leq n-1$ such that $\gamma^j |h_{j,j}| \geq \gamma^i |h_{i,i}|$ for all i, $1 \leq i \leq n-1$.
3. Replace H by $R_j H G_j$, A by $R_j D A$ and B by $B E R_j$.

Theorem. Fix $\gamma > 2/\sqrt{3}$ and set $\delta^2 = 3/4 - 1/\gamma^2$. Suppose some integral linear combination of the entries of $x \in \mathbf{R}^n$ is zero, so that x has an integer relation. Let $M \geq 1$ be the least norm of any such relation m. Normalize x so that $|x| = 1$ and iterate PSLQ beginning with the following set of three matrices: $H = H_x$, $A = I_n$, $B = I_n$. Then

1. A relation for x will appear as a column of B after fewer than $\frac{2\gamma}{\delta^2} n^2 (n+1) \log(Mn^2)$ iterations of PSLQ.
2. The norm of such a relation for x appearing as a column of B is no greater than $\sqrt{n}|H||BP|M$.
3. If after a number of iterations of PSLQ no relation has yet appeared in a column of B, then there are no relations of norm less than the bound $1/|H|$.

6. Proof of the PSLQ Algorithm

Suppose $m \in \mathbf{Z}^n$ is a relation for x, so that $xm^t = 0$ with $m \neq 0$. For any integral invertible $C \in GL(n, \mathbf{Z})$ and orthogonal $Q \in O(n-1)$,

$$1 \;\leq\; |Cm^t| \;=\; |CPm^t| \;\leq\; |CP||m| \;=\; |CH_x||m| \;=\; |CH_xQ||m| \tag{16}$$

Note that after some fixed iteration of PSLQ, $H = AH_xG$, where G is the product of the G_j. Perform Step 1. Note that $0 \neq h_{i,i}$ if no relation has been found. Then for any relation m, including one of norm M,

$$1 \;\leq\; |m||H| \;<\; |m|\sqrt{\sum_{1\leq i\leq n-1}(n-i+1)h_{i,i}^2} \tag{17}$$

Observe that $|h_{i,i}| \leq 1$ from the beginning so that

$$\sum_{1\leq i\leq n-1}(n-i+1)h_{i,i}^2 \;\leq\; \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}| \;\leq\; \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} \tag{18}$$

Now select the integer j, $1 \leq j \leq n-1$ as in Step 2 and assume without loss of generality that $a = |h_{j,j}|, b = |h_{j+1,j}|, c = |h_{j+1,j+1}|$. Then $0 \leq b \leq a/2$ and $0 \leq c \leq a/\gamma$. Set $t = \sqrt{b^2+c^2}/a$ so that $t \leq \sqrt{1-\delta^2} = \sqrt{1/4+1/\gamma^2} < 1$ with $\gamma > 2/\sqrt{3}$. Perform Step 3 and denote the diagonal entries of R_jHG_j by $\mathbf{h}_{i,i}$. Then

$$\sum_{1\leq i\leq n-1}(n-i+1)|\mathbf{h}_{i,i}|^{1/n} \tag{19}$$

$$\begin{aligned}
&= -(n-j+1)a^{1/n}(1-t^{1/n}) \;+\; (n-j)c^{1/n}(t^{-1/n}-1) && (20)\\
&\quad + \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} && (21)\\
&= -a^{1/n}(1-t^{1/n})\left\{1+(n-j)\left[1-\left(\frac{c}{\sqrt{b^2+c^2}}\right)^{1/n}\right]\right\} && (22)\\
&\quad + \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} && (23)\\
&\leq \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} \;-\; a^{1/n}(1-t^{1/n}) && (24)\\
&< (1-\frac{\delta^2}{2\gamma n^2(n+1)})^2\sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} && (25)
\end{aligned}$$

Inequality (24) follows from the fact that the expression in braces in (22) is always at least one. Inequality (25) follows from the three approximations

$$\begin{aligned}
1-t^{1/n} &\;>\; \delta^2/(2n) \qquad \text{for } n \geq 1 && (26)\\
\gamma^n a &\;>\; |h_{i,i}| && (27)\\
\frac{1}{2}\gamma n(n+1)a^{1/n} &\;>\; \sum_{1\leq i\leq n-1}(n-i+1)|h_{i,i}|^{1/n} && (28)
\end{aligned}$$

Suppose that k iterations of PSLQ have been done and that no relation has appeared. Then from (17) and (25)

$$1 \;<\; |m| \, (1 - \frac{\delta^2}{2\gamma n^2(n+1)})^k \, n^2 \tag{29}$$

By taking logarithms, this becomes

$$k \;<\; \frac{2\gamma}{\delta^2} n^2(n+1) \log |m| n^2 \tag{30}$$

Consider the intersection of the hyperplane perpendicular to x and the convex hull of the $2n$ vertices given by $\pm$ the rows of the transpose of B. Since $|\det B| = 1$ there are no relations in the interior of this intersection. The radius of the largest $n-1$ dimensional ball in this intersection is the minimum of the 2^n quantities $1/|\delta H|$ where $\delta = (\pm 1, \cdots, \pm 1)$. This radius is bounded below by $1/(\sqrt{n}|H|)$. Consequently, as $|H|$ becomes smaller this relation-excluding ball becomes larger. The nearest candidates for relations correspond to columns of B. This concludes the proof of part 1 of the theorem. Part 2 of the theorem follows from the fact that the radius above is less than M, the norm of a smallest relation. Part 3 of the theorem follows from the general inequality $1/|CH_xQ| \leq |m|$ with which the proof began, where $C = A$ and $Q = G$ in the case of the PSLQ algorithm.

Note that the numerical expression $2\gamma/\delta^2$ has a minimum of 8 when $\gamma = 2$. Thus it would appear from this proof that $\gamma = 2$ is the best choice. However, in practice we have found that smaller values of γ, while requiring more iterations, are more effective in recovering relations when the input vector is known to only limited precision.

7. Computer Implementation

The basic PSLQ algorithm can be implemented easily using ordinary floating point arithmetic on a computer. Using double precision (i.e., 64-bit) arithmetic, relations of modest height can be recovered for n up to about 6. Beyond this level, precision is quickly exhausted. Thus a serious implementation of PSLQ or any other integer relation algorithm must employ some form of multiprecision arithmetic. The authors' implementation of PSLQ employs an automatic multiprecision translator [2] and a multiprecision arithmetic package [3], both of which were developed by one of the authors (Bailey).

A straightforward implementation of PSLQ as described in section 5 is adequate for recovering relations, but its run time is not competitive with HJLS, at least for small n. Fortunately, the stability of the PSLQ algorithm admits a multi-level implementation, i.e., one that utilizes two or even three levels of working precision, which results in greatly reduced run times. The two-level scheme can be described as follows. To initialize, perform the initialization step of the standard PSLQ algorithm (i.e., normalize x, compute H_x, and set A and B to the identity matrix) using full precision. Also set $y_0 = x$. Then perform the following three steps.

First, convert the multiprecision arrays y_0 and H as accurately as possible to the double precision arrays $\bar{y_0}$ and $\bar{H}$, scaling the results to the maximum entries in y_0 and H, respectively. Set the double precision arrays $\bar{A}$ and $\bar{B}$ to the identity matrix.

Second, perform the basic PSLQ algorithm as described in section 5 using double precision arithmetic for as many iterations as possible until precision is exhausted. The authors have found that an adequate condition to detect exhaustion of double precision is that either an entry of the A matrix exceeds 10^8 or an entry of $\bar{y} = \bar{B}\bar{y}_0$ is smaller than 10^{-14}.

Third, perform an "accurate update". This is done as follows. First, replace the multiprecision A by $\bar{A}A$ and B by $B\bar{B}$. Then compute $K = AH_x$ and perform a lower trapezoidal–orthogonal (LQ) factorization on K. This LQ factorization can be done by employing a multiprecision version of the LINPACK routine DQRDC [6], if desired. Note that when K is (uniquely) factored as LQ, where L is lower trapezoidal and Q is orthogonal, then since $H = AH_xG$ is lower trapezoidal, it follows that $L = H$ and $Q = G^{-1}$. Thus H may be set to the lower trapezoidal result of the LQ factorization, and the orthogonal result of the LQ factorization may be discarded.

Finally, set the multiprecision $y = xB$. If any entry of this updated y vector is zero (to within the multiprecision epsilon), then a relation has been detected and is contained in the corresponding column of the B matrix. The bound on possible relations may also be checked at this point using the updated H matrix. The algorithm may be continued by repeating the above procedure beginning with the conversion step, where y_0 now is set to the current updated y.

One detail has been omitted in the above procedure. In some iterations, often including the very first iteration, the entries of the y_0 vector or the H matrix have such a large dynamic range that they cannot be meaningfully converted to double precision values. In these cases it is necessary to perform the basic algorithm using multiprecision arithmetic for a number of iterations until these large dynamic ranges are eliminated.

With this implementation scheme, for $n \leq 20$ the run time is dominated by the time required for the double precision iterations and thus is very efficient. For larger n a three-level implementation can be used employing double precision, an intermediate level of perhaps 100 digits, and full multiprecision. With this scheme, the run time is dominated by the cost of double precision iterations for as large a value of n as is practical to perform on current scientific computer systems.

One major advantage of the PSLQ algorithm is that it permits a very simple analysis affirming the accuracy of computed bound results. This can be seen from the above, since the H matrix can be updated directly from the original H_x matrix. The formation of the H_x matrix requires at most two operations per entry. The matrix multiplication $H = AH_x$ requires at most $2n$ operations per entry, and the entries of A are integers. The LQ factorization of H using Householder transformations is known to be very stable [9], requiring at most $2n$ operations per matrix entry. Finally, the computation of the bound $1/|H|$ is very stable, involving only the sum of squares and a division.

In short, the H matrix, from which relation bounds are computed, is not the result of a long iterative computation, but instead may be computed directly from the original H_x matrix at any point in the calculation with fewer than $4n$ operations per element. Thus it follows that the accuracy of the bound can be guaranteed provided that these operations are performed with a level of precision only a few digits higher than that required to accurately represent the integer A matrix that has been developed at a given point in the calculation. For example, suppose that $n = 25$, the entries of the A matrix do not exceed 10^{100}, and the result of each multiprecision arithmetic operation is known to be correct to within one bit in the last place. Then the norm bound result is guaranteed correct to at least eight decimal places, provided that the initial H_x and updated H matrices are computed with at least 140 digit precision. In practice, the authors have found that a working precision level of only 20 digits or so beyond the size of the largest A matrix entry is sufficient to produce reliable bounds.

Along this line, it should be mentioned that the inequality (16) above is true for any matrix norm. For example, the norm given by

$$|H| \quad = \quad \max_{1 \le i \le n} \sqrt{\sum_{1 \le j \le n-1} h_{i,j}^2} \tag{31}$$

gives better bounds than the Frobenius norm. It should also be emphasized that Part 2 of the Theorem gives a measure of how well PSLQ is doing in regard to constructing the shortest integer relation. In practice, the PSLQ algorithm almost always finds the shortest relation.

It can be seen from the above that the two- and three-level implementation schemes rely on the fact that the dynamic ranges of the y vector and the H matrix are of modest size for most iterations. It also appears that in most computer runs, after the first 1000 or so iterations, hundreds of iterations at a time may be performed in double precision between accurate updates. The reason for this fortunate behavior is not completely understood, but it may be related to a higher dimensional analog of the Kuzmin probability distribution of small partial quotients in continued fractions. We conjecture that the dynamic range of the diagonal entries of H, i.e., $\max_i |h_{i,i}| / \min_j |h_{j,j}|$, is bounded by $\gamma^n n^k$ after the initial cn^2 iterations for some fixed k and c.

8. Performance Results

Tables 1, 2, and 3 give performance results for the PSLQ algorithm. Both PSLQ and HJLS were implemented by one of the authors (Bailey) and perform high precision arithmetic using this author's MPFUN multiprecision package [3]. HJLS was "hand-coded" with direct calls to the MPFUN arithmetic routines. PSLQ was written in ordinary Fortran-77, using some Fortran subroutines from the LINPACK library [6] and a matrix multiply routine from the LAPACK library [1], and was converted to multiprecision using this author's automatic multiprecision translator [2].

The tests in Table 1 were constructed using pseudorandom number generators, and both algorithms were given the same set of ten test problems for each value of n. The tests were run on a single processor of a Silicon Graphics 380 workstation, which has a LINPACK 100 rating of 5 MFLOPS (double precision). The column headed "Dim. n" gives the dimension n of the relation vector. h is the height of the constructed relation coefficients: each a_i except a_n is chosen at random between $-h$ and h. The column headed

d gives the number of significant digits in the randomly generated x vector. The columns headed "Min. Prec." and "Max. Prec." give, in digits, the minimum and maximum working precision levels used for the runs in a given set. The column headed "Succ." gives the number of successful trials in the ten trials of a single set, i.e., the number of trials where the original constructed relation was recovered. The columns headed "Ave. Bound" give the average of the final norm bounds produced. The column headed "Ave. Time" gives the average CPU run time in seconds.

The purpose of the trials in Table 1 is to compare the effectiveness of the algorithms in recovering relations for input vectors known to only limited precision. Thus these cases were chosen with values of d just sufficient so that the constructed relation is the relation of minimum norm. The authors found that if d is set even 5 or 10 digits below the level listed in Table 1, extraneous relations are recovered by PSLQ with smaller norms than the original constructed relation. This finding underscores the fact that PSLQ is very economical in the input precision required to recover a relation and is, in fact, quite close to the information theoretical minimum in this regard.

The two-level variant of the PSLQ algorithm described in section 7 was employed for the runs in Table 1, with $\gamma = 2/\sqrt{3}$.

For the HJLS runs in Table 1, the working precision level was initially set at 230 digits, which is somewhat greater than that for the PSLQ runs. In some of the HJLS runs, the calculation aborted because multiprecision numbers were encountered in the calculation that were larger than could be precisely rounded to the nearest integer in the current precision. In other runs, the HJLS algorithm terminated with the erroneous assertion that the norm bound had excluded the norm of the constructed relation. Whenever a HJLS run failed to recover the original constructed relation for one of these reasons, the working precision level was doubled and the run repeated, up to a precision level of 7400 digits (32 times the original level) if necessary.

In some of the HJLS runs, relations were recovered "by accident" — the relation would not have been recovered at that point in the run if the calculation had been performed at higher precision. In these cases the run time was taken to be the time for the run at the lower precision. In a number of other trials, HJLS recovered a relation, but this relation was an extraneous relation with a larger norm than the constructed relation. In two trials, numerical failure occurred even at 7400 digit precision, and these trials were deemed failures. Average run time and bound statistics for HJLS runs were computed only over runs where some relation was recovered, even though this results in lower average run times for HJLS than would be the case if the failures were counted.

For those HJLS runs that required a precision level of 1850 digits or more, a version of the author's HJLS program was employed that calls the advanced multiplication and division routines of the MPFUN package. The advanced multiplication routine employs a fast Fourier transform, and the advanced division routines use a Newton iteration. Usage of these routines resulted in much lower run times for those HJLS runs that required very high precision.

The results in Table 1 show that PSLQ appears to be even more effective than HJLS

	Dim. n	h	d	Min. Prec.	Max. Prec.	Succ.	Ave. Bound	Ave. Time
PSLQ								
	10	12	125	140	140	10	7.971×10^{11}	13
	15	8	130	145	145	10	9.412×10^{7}	48
	20	6	150	165	165	10	1.029×10^{6}	157
	25	5	160	175	175	10	7.471×10^{4}	462
	30	4	200	215	215	10	6.371×10^{3}	941
HJLS								
	10	12	125	230	460	10	1.265×10^{12}	38
	15	8	130	920	7400	9	1.585×10^{8}	2382
	20	6	150	920	7400	9	1.414×10^{6}	2979
	25	5	160	920	3700	4	1.164×10^{5}	641
	30	4	200	920	3700	9	1.142×10^{4}	1126

Table 1: Random Relation Tests

in recovering relations from input vectors known to only limited precision. In 9 out of a total of 50 trials, HJLS failed to recover a relation that PSLQ succeeded in recovering. In those cases where it did recover a relation, HJLS is slower on average than PSLQ. In some individual trials, HJLS actually recovered relations faster than PSLQ, but in other trials, where very high numerical precision had to be employed for HJLS runs, HJLS was many times slower than PSLQ. The PSLQ run times varied no more than 25% within a given set.

Some other results comparing PSLQ and HJLS are shown in Table 2. In these trials, the problem is to recover the polynomial of degree n satisfied by α, where $\alpha = 3^{1/r} - 2^{1/s}$. In other words, the input vector $x = (1, \alpha, \alpha^2, \cdots, \alpha^{n-1})$, where $n = rs + 1$. Here the working precision was set to the level required by PSLQ or HJLS, respectively, as determined by some preliminary test runs, and the input vector was computed with ample precision to recover the relation. In this way, these tests simply compare run times, not effectiveness in recovering relations with limited input precision.

The results in Table 2 demonstrate even more dramatically the erratic precision requirements and correspondingly erratic run times of HJLS. Whereas the first two problems required less than 100 digits working precision, and ran quite fast as a result, the last three problems required thousands of digits of precision, and the run times were hundreds of times greater than the corresponding PSLQ run times. In fact, it appears from this and other runs made by the authors that the HJLS algorithm is generally unusable for recovering non-trivial algebraic relations when n is greater than 25.

A final set of results is given in Table 3. These results do not compare PSLQ with HJLS, but instead merely explore how large a relation can be recovered in a reasonable amount of time using the PSLQ algorithm. These runs were made using the three-level variant of the PSLQ algorithm, as described in section 7, with $\gamma = 2/\sqrt{3}$. The trial problems are, as before, algebraic relations with $\alpha = 3^{1/r} - 2^{1/s}$. These runs were made on a single processor of a Silicon Graphics 380 system.

The results in Table 3 show that it is feasible at present, using a RISC workstation, to recover relations with the PSLQ algorithm for n greater than 80 and, simultaneously, for relation coefficients larger than 10^{13}. These figures are significantly larger than any with which the authors are familiar for other integer relation algorithms (compare with the results in [11], for example). It is worth noting that if a simple-minded, exhaustive search procedure had been employed to recover the polynomial satisfied by $\alpha = 3^{1/9} - 2^{1/9}$, assuming only its numerical value and the fact that it is an algebraic number of degree not exceeding 81, with coefficients bounded by 10^{14}, then it would have been necessary to examine more than 10^{1100} polynomials.

	Dim. n	r	s	Working Prec.	Max. Bound	Relation Norm	Run Time
PSLQ							
	10	3	3	30	7.414×10^1	1.650×10^2	1
	11	2	5	40	3.773×10^2	5.898×10^2	3
	13	3	4	55	2.442×10^2	6.446×10^2	6
	15	2	7	70	3.027×10^3	8.248×10^3	17
	16	3	5	75	9.277×10^2	2.699×10^3	18
	17	4	4	75	1.255×10^3	3.917×10^3	22
HJLS							
	10	3	3	80	7.441×10^1	1.650×10^2	2
	11	2	5	80	1.835×10^2	5.909×10^2	3
	13	3	4	1200	1.787×10^2	6.446×10^2	131
	15	2	7	14800	4.165×10^3	8.248×10^3	10480
	16	3	5	14800	1.618×10^3	2.699×10^3	11112
	17	4	4	14800	1.390×10^3	3.917×10^3	11490

Table 2: Algebraic Relation Tests

Dim. n	r	s	Working Prec.	Max. Bound	Relation Norm	Run Time
21	4	5	110	5.770×10^3	1.473×10^4	71
26	5	5	160	3.092×10^4	1.169×10^5	218
31	5	6	230	2.380×10^5	6.686×10^5	656
37	6	6	300	1.034×10^6	5.344×10^6	1680
43	6	7	400	1.756×10^7	6.022×10^7	3981
50	7	7	500	5.871×10^7	3.308×10^8	8493
57	7	8	700	2.142×10^9	9.531×10^9	20730
65	8	8	850	2.028×10^{10}	1.205×10^{11}	46897
73	8	9	1050	4.677×10^{11}	3.120×10^{12}	93368
82	9	9	1300	5.026×10^{12}	7.973×10^{13}	185787

Table 3: Large Algebraic Relations with PSLQ

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorenson, *The LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[2] D. H. Bailey, "Automatic Translation of Fortran Programs to Multiprecision," RNR Technical Report RNR-91-025, NASA Ames Research Center, 1991.

[3] D. H. Bailey, "MPFUN: A Portable High Performance Multiprecision Package," RNR Technical Report RNR-90-022, NASA Ames Research Center, 1990.

[4] D. H. Bailey and H. R. P. Ferguson, "Numerical Results on Relations Between Numerical Constants Using a New Algorithm," *Mathematics of Computation*, vol. 53 (October 1989), p. 649 - 656.

[5] M. J. Coster, B. A. LaMacchia, A. M. Odlyzko and C. P. Schnorr, "An Improved Low-Density Subset Sum Algorithm" in D. W. Davies, ed., *Advances in Cryptology: Eurocrypt '91*, Springer Verlag, New York, to appear 1992.

[6] J. J. Dongarra, C. B. Moler, J. R. Bunch and G. W. Stewart, *The LINPACK Users' Guide*, SIAM, Philadelphia, 1979.

[7] H. R. P. Ferguson and R. W. Forcade, "Generalization of the Euclidean Algorithm for Real Numbers to All Dimensions Higher Than Two," *Bulletin of the American Mathematical Society*, 1 (1979), p. 912 - 914.

[8] H. R. P. Ferguson, "A Non-Inductive GL(n,Z) Algorithm That Constructs Linear Relations for n Z-Linearly Dependent Real Numbers," *Journal of Algorithms*, Vol. 8 (1987), p. 131 - 145.

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins, Baltimore, 1989.

[10] J. Hastad, B. Just, J. C. Lagarias and C. P. Schnorr, "Polynomial Time Algorithms for Finding Integer Relations Among Real Numbers," *SIAM Journal on Computing*, vol. 18 (1988), p. 859 - 881.

[11] J. C. Lagarias and A. M. Odlyzko, "Solving Low-Density Subset Sum Problems," *Journal of the ACM*, vol. 32 (January 1985), p. 229 - 246.

[12] A. K. Lenstra, H. W. Lenstra and L. Lovasz, "Factoring Polynomials with Rational Coefficients," *Math. Annalen*, vol. 261 (1982), p. 515 - 534.