Two Variants of HJLS-PSLQ with Applications *

Yong Feng Jingwei Chen† Wenyuan Wu

Chongqing Key Laboratory of Automated Reasoning and Cognition,
Chongqing Institute of Green and Intelligent Technology, CAS
{yongfeng, chenjingwei, wuwenyuan}@cigit.ac.cn

ABSTRACT

The HJLS and PSLQ algorithms are the most popular algorithms for finding nontrivial integer relations for several real numbers. It has been already shown that PSLQ is essentially equivalent to HJLS under certain settings. We here call them HJLS-PSLQ.

In the present work, we provide two variants of HJLS-PSLQ. The first one is a new modification of Bailey and Broadhurst's multi-pair version. We prove the termination of our modification, while the original multi-pair version may not terminate. The second one is an incremental version of HJLS-PSLQ. For those applications requiring to call HJLS-PSLQ many times, such as finding the minimal polynomial of an algebraic number without knowing the degree, we show the incremental version is more efficient than HJLS-PSLQ, both theoretically and practically.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Algebraic algorithms*

General Terms

Algorithms

Keywords

integer relation, parallel algorithm, incremental algorithm, HJLS, PSLQ

1. INTRODUCTION

Being exact, symbolic computation is usually inefficient due to the well-known problem of intermediate swell. Being efficient, numerical computation only gives approximate results. For both reliability and efficiency, an idea that obtains exact results by using approximate computing has been of interest. We call such methods *zero-error computation*. The output of *zero-error algorithms* are exact expressions, but the intermediate process (partially) uses numerical methods, so that they are symbolic-numeric. At the end of these algorithms, errors become zero (i.e., exact results) by certain gap theorems from background knowledge, though errors appear at (almost) every step. For instance, applying the algorithm presented in [30], one can recovery the exact value of a rational number from its approximation; more generally, the algorithms in [20, 26] are for reconstructing algebraic numbers. In many zero-error algorithms, such as algebraic number reconstruction [20, 26], polynomial factorization [17, 29], etc., the problem to be solved is finally converted to finding an integer relation.

A vector $\boldsymbol{m} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ is called an *integer relation* for $\boldsymbol{x} \in \mathbb{R}^n$ if $\langle \boldsymbol{x}, \boldsymbol{m} \rangle = 0$. The problem of finding integer relations for rational or real numbers can be dated back to the time of Euclid. It is well known that the problem for $n = 2$ can be solved by applying the euclidean algorithm, or, equivalently, the continued fraction algorithm. Many famous mathematicians worked on this problem for the case $n \geq 3$, including Jacobi, Hermite, Poincaré, etc. (See [14, 10] and references therein for more history notes on this problem.) The first breakthrough presented by Ferguson and Forcade is a generalization of the euclidean algorithm [11]. It is capable of solving the integer relation finding problem for any integer n, in the sense that it either finds an relation or claims no relation with norm less than a given bound. The famous LLL algorithm also can be used to solve this problem [21, p. 525]. However, the first proved polynomial-time algorithm to solve this problem is due to Håstad, Just, Lagarias and Schnorr [14], referred to as HJLS. Later on, Ferguson and Bailey [9] presented a more frequently used integer relation finding algorithm named PSLQ (see also [10]). As Borwein indicated in [6, App. B, Th. 7], both HJLS and PSLQ are based on Ferguson and Forcade's generalization. Moreover, Meichsner [22, Sec. 2.3.1] showed that HJLS with full reduction is equivalent to PSLQ with an appropriate parameter setting. In the remainder of the present paper, we see the two algorithms HJLS and PSLQ as a unified one, and call them HJLS-PSLQ. In their 2013 paper [7], Chen, Stehlé and Villard presented a new interpretation on HJLS-PSLQ, from the lattice point of view. Up to now, the HJLS-PSLQ algorithm has been successfully used in many areas [5, 2], such as to find the coefficients of the integer minimal polynomial of an algebraic number, to give a new formula for π, to identify

*Part of this work was presented at the poster session of ISSAC 2013 [8]. This work was partly supported by NKBRPC (grant no. 2011CB302400), NSFC (grant nos. 91118001 and 11170153), and CSTC project (grant no. cstc2013jjys0002).

†Corresponding author.

SNC'14, July 28-31, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2631948.2631965

some constants that arise in quantum field theory, etc. For efficiency, Bailey and Broadhurst presented a parallel variant of HJLS-PSLQ in [3], which always swaps several pairs of rows of certain matrices. A similar technique called "all-swap reduction" was already used to design parallel lattice reduction algorithms [27, 28, 19, 15, 16].

We note that all above computational results about HJLS-PSLQ are obtained by assuming exact operation on real numbers. In this model, we present two variants of HJLS-PSLQ in this paper.

Our results. Our first contribution is to present a modified multi-pair variant of the HJLS-PSLQ algorithm with termination guaranteed. In [3], Bailey and Broadhurst presented the multi-pair variant of HJLS-PSLQ, which is suitable for parallel computing. However, the termination of that algorithm was not proven. As Bailey and Broadhurst pointed out, "for certain special problems, the multi-pair algorithm falls into a repeating cycle". By analyzing the algorithm, we determined that the reason behind this phenomenon is that the inequality in Lemma 3-(3) (see also [10, Lem. 8]) may not hold when executing the multi-pair HJLS-PSLQ algorithm. If this case happens, the algorithm may not make any progress towards finding an integer relation. Based on this analysis, we fix that by adding a judgement on choosing swap pairs (see step 2a in Algorithm 3) and present a modified multi-pair HJLS-PSLQ algorithm (Algorithm 3) whose termination is proven in Section 3.3. Furthermore, the modified algorithm is still suitable for parallel computing.

Our second contribution is to present an incremental variant of the HJLS-PSLQ algorithm, called *incremental HJLS-PSLQ* (IPSLQ for short, see Algorithm 4). Roughly speaking, given $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, HJLS-PSLQ directly detects an integer relation for $\boldsymbol{x}$, while IPSLQ presented in this paper gradually detects an integer relation for $(x_i, \ldots, x_n)$, where i ranges from $n-1$ to 1. We prove the complexity bound of IPSLQ is the same as PSLQ: it requires $\mathcal{O}(n^3 + n^2 \log \lambda(\boldsymbol{x}))$ iterations (see Algorithm 4 for the meaning of an iteration) to find an integer relation for an n-dimensional real vector $\boldsymbol{x}$, where $\lambda(\boldsymbol{x})$ denotes the smallest 2-norm among all integer relations for $\boldsymbol{x}$. Each iteration can be finished within $\mathcal{O}(n^2)$ real arithmetic operations, e.g., using the technique in [10, Sec. 8]. During our analysis, the principal difficulty is that the classical potential function for HJLS-PSLQ analysis may lead to a larger bound on the number of iterations that IPSLQ requires; see Section 4.1). To overcome this obstacle, we define a new potential function for IPSLQ. Then the complexity result for IPSLQ is proved to be the same as the original HJLS-PSLQ, but interestingly, in some practical applications, such as finding the minimal polynomial of an algebraic number, IPSLQ is more efficient than HJLS-PSLQ.

As an application of IPSLQ, we consider the problem of finding minimal polynomial for an algebraic number: Given an algebraic number α with degree $\leq d$ and height $\leq M$, how to compute its minimal polynomial? Applying HJLS-PSLQ directly to solve this problem costs $\mathcal{O}(d^4 + d^3 \log M)$ iterations, while applying IPSLQ costs only $\mathcal{O}(d^3 + d^2 \log M)$ iterations; see Section 4.2 for details. We note that the gradual LLL algorithm in [18] has a good performance for this problem in practice, however, it yet requires $\mathcal{O}(d^4 + d^3 \log M)$ swaps (according to [18, Th. 4]), where the cost of a swap is similar to that of an iteration in our setting.

Notations. Throughout this paper, vectors are in row and denoted in bold. If $\boldsymbol{x}$ is a vector, $\|\boldsymbol{x}\|$ denotes its euclidean norm. For a real vector $\boldsymbol{x}$, let $\lambda(\boldsymbol{x})$ denote the smallest 2-norm among all integer relations for $\boldsymbol{x}$. Let $\mathrm{GL}_n(\mathbb{Z})$ denote the set of all $n \times n$ integral matrices with determinant ± 1.

Organization. In next section, we recall the HJLS-PSLQ algorithm and its multi-pair variant that will be the basis for us. In Section 3, we present an improved multi-pair HJLS-PSLQ whose termination is proven. We then give another variant of HJLS-PSLQ, namely IPSLQ, in Section 4 with an application to the minimal polynomial finding problem.

2. PRELIMINARIES

In this section, we recall some definitions, HJLS-PSLQ and its multi-pair variant.

Let $A \in \mathbb{R}^{n \times m}$ be a matrix of rank r. It has a unique LQ decomposition $A = L \cdot Q$, where the Q-factor $Q \in \mathbb{R}^{r \times m}$ has orthonormal rows (i.e., $QQ^T = I_r$), and the *L-factor* $L \in \mathbb{R}^{n \times r}$ satisfies the following property: there exist diagonal indices $1 \leq k_1 < \ldots < k_r \leq n$, such that $l_{i,j} = 0$ for all $i < k_j$, and $l_{k_j,j} > 0$ for all $j \leq r$. In particular, when $n = r$, the L-factor is lower-triangular with positive diagonal coefficients. The LQ decomposition of A is equivalent to the more classical QR decomposition of A^T, such as in [13, Sec. 5.2].

2.1 The HJLS-PSLQ algorithm

The HJLS-PSLQ algorithm starts with constructing a matrix H_x.

DEFINITION 1 ([10, DEF. 2]). *Given an n dimensional real vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ with $x_i \neq 0$, define the partial sums $s_j = (\sum_{k=j}^{n} x_k^2)^{1/2}$ for $j = 1, \ldots, n$, and define $H_x = (h_{ij}) \in \mathbb{R}^{n \times (n-1)}$ as follows:*

$$h_{ij} = \begin{cases} 0, & \text{if } 1 \leq i < j \leq n-1, \\ s_{i+1}/s_i, & \text{if } 1 \leq i = j \leq n-1, \\ -x_i x_j/(s_i s_{j+1}), & \text{if } 1 \leq j < i \leq n. \end{cases}$$

We call H_x the hyperplane matrix *for $\boldsymbol{x}$.*

According to this definition, $H_x \in \mathbb{R}^{n \times (n-1)}$ is a full column rank and lower trapezoidal matrix with positive diagonal elements. Moreover, one can verify that H_x is a scale invariant with respect to $\boldsymbol{x}$, i.e., if $\boldsymbol{y} = c \cdot \boldsymbol{x}$ with $c > 0$ then $H_y = H_x$.

The main part of HJLS-PSLQ is a four-step iteration, whose goal is to reduce the rows of H_x by left multiplying matrices in $\mathrm{GL}_n(\mathbb{Z})$ and right multiplying orthogonal matrices until the last diagonal element of the resulting lower trapezoidal matrix is zero. The iteration consists of *size reductions* and *Bergman swaps*, and ends up with either finding an integer relation or claiming that there does not exist an integer relation with norm $\leq M$, where M is a given bound.

DEFINITION 2. *Let $H = (h_{i,j}) \in \mathbb{R}^{n \times (n-1)}$ be a lower trapezoidal matrix with all diagonal elements positive. We say H is* size-reduced *if $|h_{i,j}| < \frac{1}{2} |h_{j,j}|$ holds for $i > j$.*

Given H, it is possible to find a unimodular matrix $U \in \mathrm{GL}_n(\mathbb{Z})$ such that $U \cdot H$ is size-reduced. It is a classical result that computing U and updating $U \cdot H$ can be completed within $\mathcal{O}(n^3)$ real arithmetic operations, e.g., using the size

reduction scheme in [21], [14], or, the *modified Hermite reduction* in [10].

DEFINITION 3 (BERGMAN SWAP). *Let $\gamma > 2/\sqrt{3}$ and $H = (h_{i,j}) \in \mathbb{R}^{n\times(n-1)}$ be a lower trapezoidal matrix with all diagonal elements nonzero. Choose the largest r such that*

$$\gamma^r h_{r,r} = \max_{j\in\{i,\ldots,n-1\}} \left\{\gamma^j \cdot h_{j,j}\right\},$$

and then swap the r-th row and the $(r+1)$-th row of H.

The swap condition in Definition 3 is called the *Bergman condition*, which was originally presented by Bergman in [4]. It considers all diagonal elements so that it is global, in contrast to the *Siegel condition* in many lattice reduction algorithms, where the swap position r is chosen to be the smallest index i such that $h_{i,i}^2 \geq \gamma^2 \cdot h_{i+1,i+1}^2$.

For convenience, we give the description of HJLS-PSLQ, as in Algorithm 1.

Algorithm 1 (HJLS-PSLQ).

Input: A vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ for $i = 1, \ldots, n$, a positive number $M \geq 1$, and a parameter $\gamma > 2/\sqrt{3}$.

Output: Either return an integer relation for $\boldsymbol{x}$, or return "$\lambda(\boldsymbol{x}) > M$".

1. Construct the hyperplane matrix $H_x \in \mathbb{R}^{n\times(n-1)}$. Set $H := H_x$, $A := I_n$ and $B := I_n$. Size-reduce H producing a unimodular matrix U, and update $A = U \cdot A$ and $B := B \cdot U^{-1}$.
2. While $h_{n-1,n-1} \neq 0$ do
   (a) Perform the Bergman swap on H producing the swap position r, and swap the r-th row (column) and the $(r+1)$-th row (column) of the matrix A (B).
   (b) Update H to the L-factor of H.
   (c) Size-reduce H producing $U \in \mathrm{GL}_n(\mathbb{Z})$, and update A and B as in step 1.
   (d) If $\max_{j\in\{1,\ldots,n-1\}} h_{j,j} < 1/M$ then return "$\lambda(\boldsymbol{x}) > M$".
3. Return the last column of B.

Note that to complete Step 2b, computing the full LQ decomposition on H is not necessary, since only four elements (i.e., the submatrix consisting of the r and $r+1$ rows of columns r and $r+1$) of the L-factor may be changed during this step; see the equation (1) for the update formula.

LEMMA 1 ([10, TH. 1]). *Let $\boldsymbol{x} \in \mathbb{R}^n \setminus \mathbf{0}$. Suppose that for any unimodular matrix U, there exists an $(n-1)\times(n-1)$ orthogonal matrix such that $H = UH_xQ$ is lower trapezoidal and all of the diagonal elements of H satisfy $h_{j,j} > 0$. Then*

$$\lambda(\boldsymbol{x}) \geq \frac{1}{\max_{1\leq j\leq n-1} h_{j,j}}.$$

In fact, all steps of HJLS-PSLQ are to produce the matrices U and Q satisfying the condition in Lemma 1, however, storing and updating Q are not necessary, as the output of the algorithm is only related to the matrices U and H.

LEMMA 2 ([10, TH. 2, 3]). *Given $\boldsymbol{x} \in \mathbb{R}^n$ with integer relations, HJLS-PSLQ can find an integer relation $\boldsymbol{m}$, satisfying $\|\boldsymbol{m}\| \leq \gamma^{n-2}\lambda(\boldsymbol{x})$, for $\boldsymbol{x}$ within $\mathcal{O}(n^3 + n^2 \log \lambda(\boldsymbol{x}))$ iterations.*

2.2 The multi-pair version of HJLS-PSLQ

For parallelizing HJLS-PSLQ, Bailey and Broadhurst proposed a multi-pair variant [3]. The key idea is that one can swap several pairs of rows of the matrix H on different processors within one iteration, rather than only one Bergman swap during one iteration. This technique is similar to the "all-swap reduction" in [27, 28, 19, 15, 16] for parallelizing LLL. We recall this algorithm, based on [3, p. 1728], as follows.

Algorithm 2 (multi-pair HJLS-PSLQ).

Input: A vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ for $i = 1, \ldots, n$, a positive number $M \geq 1$, and a parameter $\gamma > 2/\sqrt{3}$.

Output: Either return an integer relation for $\boldsymbol{x}$, or return "$\lambda(\boldsymbol{x}) > M$".

1. Construct $H_x \in \mathbb{R}^{n\times(n-1)}$. Set $H := H_x$, $A := I_n$ and $B := I_n$. Size-reduce H producing a unimodular matrix U, and update $A = U \cdot A$ and $B := B \cdot U^{-1}$.
2. While $h_{n-1,n-1} \neq 0$ do
   (a) Sort the $n-1$ entries $\{\gamma^j \cdot h_{j,j}\}$ and let j_1 be the index corresponding to the largest $\gamma^j \cdot h_{j,j}$. Then select pairs of indices $(j_\kappa, j_\kappa+1)$, where j_κ is the sort index. If at any step either j_κ or $j_\kappa + 1$ has already been selected, pass to the next index in the list. Continue until the list is exhausted. Let p denote the number of pairs actually selected in this manner.
   (b) For κ from 1 to p, swap the rows (columns) j_κ and $j_\kappa + 1$ of the matrix H and the matrix A (B). For κ from 1 to p, update H to the L-factor of H.
   (c) Size-reduce H, and update A and B as in step 1.
   (d) If $\max_{j\in\{1,\ldots,n-1\}} h_{j,j} < 1/M$ then return "$\lambda(\boldsymbol{x}) > M$".
3. Return the last column of B.

Let p denote the number of pairs actually selected in step 2a. Obviously, the key step, step 2b, of Algorithm 2 is suitable for parallel execution. Thus, one can use p processors to swap the j_κ-th row and the $(j_\kappa + 1)$-th row of H at the same time ($\kappa = 1, \ldots, p$). Bailey and Broadhurst showed that the selection of up to p disjoint pairs seems to have the effect of reducing the iteration count by nearly the factor p, however, they did not offer a proof for the termination of the multi-pair version of HJLS-PSLQ.

3. PARALLEL HJLS-PSLQ WITH TERMINATION

In this section, we first find out the reason why multi-pair HJLS-PSLQ may not terminate, and then present a modified version of multi-pair HJLS-PSLQ with a termination proof.

3.1 Termination of HJLS-PSLQ

To analyze the reason that may cause a repeating cycle, we first recall the termination proof for the HJLS-PSLQ algorithm, which is extracted from [10].

The idea is to prove a function $\Pi(k)$, which is called the Π function, with respect to the number of iterations k has the following properties: (1) $1 \leq \Pi(k) \leq K$ and (2) $\Pi(k)$ strictly increases by a factor τ with $\tau > 1$ when k increases by one, where K is an upper bound on $\Pi(k)$ and is related to the dimension n, $\lambda(\boldsymbol{x})$ and the parameter γ. The Π function is

defined as

$$\Pi(k) = \prod_{j=1}^{n-1} \min\{\gamma^{n-1}\lambda(\boldsymbol{x}), 1/h_{j,j}^{(k)}\}^{n-j},$$

where $h_{i,j}^{(k)}$ represents the i-th row and j-th column element of the matrix H after exactly k iterations of HJLS-PSLQ. Furthermore, the $H^{(k)}$ has the following useful properties.

LEMMA 3. *Let $H^{(k)} = (h_{i,j}^{(k)})$ denote the matrix H in HJLS-PSLQ after exactly k iterations.*

(1) *If $h_{j,j}^{(k)} = 0$ for some $1 \le j \le n-1$ and no smaller k, then $j = n-1$ and a relation for $\boldsymbol{x}$ appears as a column of the matrix B.*
(2) *During the k-th iteration of the algorithm, $h_{j,j}^{(k)} \le 1$ for $j = 1, \ldots, n-1$.*
(3) *Before doing the k-th Bergman swap, $\gamma^{n-2}\lambda(\boldsymbol{x})h_{r,r}^{(k-1)} \ge 1$.*

PROOF. This lemma is concluded from [10, Lem. 5, 6, 8]. Here, we only give the proof of (3).

By the choice of r in the step 2a, $\gamma^r h_{r,r}^{(k-1)} \ge \gamma^j h_{j,j}^{(k-1)}$ for $j = 1, \ldots, n-1$. Hence

$$\gamma^{n-1}h_{r,r}^{(k-1)} \ge \gamma^r h_{r,r}^{(k-1)} \ge \gamma^j h_{j,j}^{(k-1)} \ge \gamma h_{j,j}^{(k-1)}$$

for all j including j_0 such that $h_{j_0,j_0}^{(k-1)} = \max\{h_{j,j}^{(k-1)}\}$. It follows from Lemma 1 that $\lambda(\boldsymbol{x}) \ge 1/h_{j_0,j_0}^{(k-1)}$, which implies (3). □

When the k-th Bergman swap happened with $r < n-1$, we consider the submatrix consisting of the r-th and $(r+1)$-th rows of columns r and $r+1$ before and after Step 2a, 2b in Algorithm 1:

$$\begin{pmatrix} h_{r,r}^{(k-1)} & \\ h_{r+1,r}^{(k-1)} & h_{r+1,r+1}^{(k-1)} \end{pmatrix} \to \begin{pmatrix} h_{r,r}^{(k)} & \\ h_{r+1,r}^{(k)} & h_{r+1,r+1}^{(k)} \end{pmatrix},$$

where

$$\begin{aligned} h_{r,r}^{(k)} &= \sqrt{\left(h_{r+1,r}^{(k-1)}\right)^2 + \left(h_{r+1,r+1}^{(k-1)}\right)^2}, \\ h_{r+1,r}^{(k)} &= h_{r,r}^{(k-1)} \cdot h_{r+1,r}^{(k-1)}/h_{r,r}^{(k)}, \\ h_{r+1,r+1}^{(k)} &= h_{r,r}^{(k-1)} \cdot h_{r+1,r+1}^{(k-1)}/h_{r,r}^{(k)}. \end{aligned} \tag{1}$$

During the iteration in HJLS-PSLQ, all factors of the Π function remain except $h_{r,r}$ and $h_{r+1,r+1}$. Therefore,

$$\frac{\Pi(k)}{\Pi(k-1)} = \left(\frac{\min\left\{\gamma^{n-1}\lambda(\boldsymbol{x}),\, 1/h_{r,r}^{(k)}\right\}}{\min\left\{\gamma^{n-1}\lambda(\boldsymbol{x}),\, 1/h_{r,r}^{(k-1)}\right\}}\right)^{n-r} \cdot \left(\frac{\min\left\{\gamma^{n-1}\lambda(\boldsymbol{x}),\, 1/h_{r+1,r+1}^{(k)}\right\}}{\min\left\{\gamma^{n-1}\lambda(\boldsymbol{x}),\, 1/h_{r+1,r+1}^{(k-1)}\right\}}\right)^{n-r-1}.$$

Set $S{=}\gamma^{n-1}\lambda(\boldsymbol{x})h_{r,r}^{(k)}$, $T{=}\gamma^{n-1}\lambda(\boldsymbol{x})h_{r+1,r+1}^{(k-1)}$, $\theta = h_{r,r}^{(k)}/h_{r,r}^{(k-1)}$ $= h_{r+1,r+1}^{(k-1)}/h_{r+1,r+1}^{(k)}$, and let

$$F(X) = \frac{\min\{X, 1\}}{\min\{X, \theta\}}.$$

Then

$$\frac{\Pi(k)}{\Pi(k-1)} = F(S)\left(\frac{F(S)}{F(T)}\right)^{n-r-1}.$$

Let $\tau = 1/\sqrt{1/4 + 1/\gamma^2}$. From equation (1) and $\gamma > 2/\sqrt{3}$, we have $0 < \theta \le 1/\tau < 1$ and $S \ge T$. For $X > 0$ and $0 < \theta < 1$, $F(X)$ does not decrease when increasing X, so that $F(S) \ge F(T)$. Furthermore, it follows from Lemma 3-(3) that $S \ge \gamma\theta \ge \theta$, so we have

$$\frac{\Pi(k)}{\Pi(k-1)} \ge \frac{\min\{S, 1\}}{\min\{S, \theta\}} \ge \tau > 1.$$

When the swap position is exactly the last row, namely $r = n-1$, then the only change is

$$h_{n-1,n-1}^{(k)} = \left|h_{n,n-1}^{(k-1)}\right| \le \frac{1}{2}h_{n-1,n-1}^{(k-1)},$$

so that

$$\frac{\Pi(k)}{\Pi(k-1)} \ge \frac{\min\{S, 1\}}{\min\{S, \theta\}} \ge \tau > 1.$$

Overall, we obtain $\Pi(k) \ge \tau\Pi(k-1)$. Then the number of iterations in Lemma 2 follows from $1 \le \Pi(k) \le \left(\gamma^{n-1}\lambda(\boldsymbol{x})\right)^{\frac{n^2-n}{2}}$.

3.2 Analysis for multi-pair HJLS-PSLQ

We now consider the same strategy to prove the termination of Algorithm 2. Suppose only two pairs are selected in step 2a before the k-th swap, namely $p = 2$, say $(j_{\kappa_1}, j_{\kappa_1}{+}1)$ and $(j_{\kappa_2}, j_{\kappa_2}{+}1)$, and assume that $\gamma^{j_{\kappa_1}} h_{j_{\kappa_1},j_{\kappa_1}}^{(k-1)} = \max \gamma^j h_{j,j}^{(k-1)}$ and $\max\{j_{\kappa_1}, j_{\kappa_2}\} < n-1$.

From the analysis in Section 3.1, we have

$$\frac{\Pi(k)}{\Pi(k-1)} \ge \tau \cdot F'(S')\left(\frac{F'(S')}{F'(T')}\right)^{n-j_{\kappa_2}-1}$$

with

$$F'(X) = \frac{\min\{X, 1\}}{\min\{X, \theta'\}},$$

where

$$\begin{aligned} S' &= \gamma^{n-1}\lambda(\boldsymbol{x})h_{j_{\kappa_2},j_{\kappa_2}}^{(k)}, \\ T' &= \gamma^{n-1}\lambda(\boldsymbol{x})h_{j_{\kappa_2}+1,j_{\kappa_2}+1}^{(k-1)}, \\ \theta' &= h_{j_{\kappa_2},j_{\kappa_2}}^{(k)}/h_{j_{\kappa_2},j_{\kappa_2}}^{(k-1)} = h_{j_{\kappa_2}+1,j_{\kappa_2}+1}^{(k-1)}/h_{j_{\kappa_2}+1,j_{\kappa_2}+1}^{(k)}. \end{aligned}$$

For the same reason, we have $F'(S') \ge F'(T')$. Therefore

$$\frac{\Pi(k)}{\Pi(k-1)} \ge \tau \cdot \frac{\min\{S', 1\}}{\min\{S', \theta'\}}.$$

Actually, if one can prove $\frac{\Pi(k)}{\Pi(k-1)} \ge \tau^2$ then Algorithm 2 really parallelizes HJLS-PSLQ. However, as stated in Theorem 1, one can only guarantee

$$\lambda(\boldsymbol{x}) \ge 1/\max_{1\le j\le n-1} h_{j,j},$$

which implies $\gamma^{n-2}\lambda(\boldsymbol{x})h_{r,r}(k-1) \ge 1$ (Lemma 3-(3)). But, unfortunately, one cannot obtain $S' \ge \gamma\theta'$, and thus cannot obtain $\frac{\min\{S',1\}}{\min\{S',\theta'\}} \ge \tau$. Even more, one cannot obtain $\frac{\min\{S',1\}}{\min\{S',\theta'\}} \ge 1/\tau$ in general, because S' might be much smaller than 1 in practice. When this case happens, the iteration of Algorithm 2 will make no progress, which leads to non-termination.

3.3 Modificatio of multi-pair HJLS-PSLQ

Based on the analysis above, we now give a modification on the pair selection strategy, which implies a modification of multi-pair HJLS-PSLQ with termination guaranteed.

Algorithm 3 (Modified multi-pair HJLS-PSLQ).

Input: Suppose $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ for $i = 1, \ldots, n$ has integer relations. Input $\boldsymbol{x}$, a positive number $M \geq 1$, a positive number N such that $1 \leq N \leq \lambda(\boldsymbol{x})$ and a parameter $\gamma > 2/\sqrt{3}$.

Output: An integer relation for $\boldsymbol{x}$.

1. Construct $H_x \in \mathbb{R}^{n\times(n-1)}$. Set $H := H_x$, $A := I_n$ and $B := I_n$. Size-reduce H producing a unimodular matrix U, and update $A = U \cdot A$ and $B := B \cdot U^{-1}$.
2. While $h_{n-1,n-1} \neq 0$ do
   (a) Sort the entries
   $$\{\gamma^j \cdot h_{j,j} : \text{ the } j\text{'s are such that } \gamma^{n-2} N h_{j,j} \geq 1\}$$
   and let j_1 be the index corresponding to the largest $\gamma^j \cdot h_{j,j}$. Then select pairs of indices $(j_\kappa, j_\kappa + 1)$, where j_κ is the sort index. If at any step either j_κ or $j_\kappa + 1$ has already been selected, pass to the next index in the list. Continue until the list is exhausted. Let p denote the number of pairs actually selected in this manner.
   (b) For κ from 1 to p, swap the rows (columns) j_κ and $j_\kappa + 1$ of the matrix H and the matrix A (B). For κ from 1 to p, update H to the L-factor of H.
   (c) Size-reduce H, and update A and B as in step 1.
3. Return the last column of B.

Comparing with Algorithm 2, the modified multi-pair HJLS-PSLQ algorithm features the following.

Firstly, instead of sorting $n-1$ elements $\gamma^j \cdot h_{j,j}$, Algorithm 3 only considers those subscripts j_κ such that

$$\gamma^{n-2}\lambda(\boldsymbol{x})h_{j_\kappa,j_\kappa} \geq \gamma^{n-2} N h_{j_\kappa,j_\kappa} \geq 1,$$

which guarantees that Lemma 3-(3) always holds for all selected pairs $(j_\kappa, j_\kappa + 1)$, $\kappa = 1, \ldots, p$. Namely, before doing the k-th Bergman swap, $\gamma^{n-2}\lambda(\boldsymbol{x})h^{(k-1)}_{j_\kappa,j_\kappa} \geq 1$. Thus, we have $S' \geq \gamma\theta' \geq \theta'$ so that

$$\frac{\Pi(k)}{\Pi(k-1)} \geq \tau \cdot \frac{\min\{S', 1\}}{\min\{S', \theta'\}} \geq \tau^2 > 1,$$

which implies the termination of Algorithm 3.

Secondly, it seems that Algorithm 2 has higher concurrency than Algorithm 3, as the pair count in Algorithm 3 may be larger than that in Algorithm 2. However, as showed before, more swaps do not mean more progress. According to Bailey and Broadhurst's report [3, p. 1729], a swap might not increase $\Pi(k)$ in extremely rare nontrivial problems. For modified multi-pair HJLS-PSLQ, this case will not happen. In fact, if p pairs in step 2a of Algorithm 3 are exchanged, then it follows that

$$\frac{\Pi(k)}{\Pi(k-1)} \geq \tau^p > 1.$$

This means that we make the Π function progress as doing p Bergman swaps.

Thirdly, comparing with Algorithm 2, the input of Algorithm 3 adds a positive number $N \leq \lambda(\boldsymbol{x})$. In [1], Babai, Just and Meyer auf der Heide show that it is not possible to decide whether there exists a relation for a given $\boldsymbol{x} \in \mathbb{R}^n$ under the exact real arithmetic model with the floor function. Therefore, in modified multi-pair HJLS-PSLQ we only consider the case that $\boldsymbol{x}$ has integer relations. Actually, the closer N to $\lambda(\boldsymbol{x})$, the more pairs will be selected, and hence the higher concurrency can be achieved.

THEOREM 1. *Given $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ having integer relations, the modified multi-pair HJLS-PSLQ algorithm finds an integer relation $\boldsymbol{m}$ for $\boldsymbol{x}$ within $\mathcal{O}(n^3 + n^2 \log \lambda(\boldsymbol{x}))$ iterations.*

PROOF. According to the analysis above, modified multi-pair HJLS-PSLQ performs p swaps per iteration, and each swap makes the Π function increase by a factor τ. □

4. THE IPSLQ ALGORITHM

In many applications, such as finding the minimal polynomial for an algebraic number (see Section 4.2), one needs to repeatedly call HJLS-PSLQ to try, for $i = 1, \ldots, n$, whether there exists an integer relation for $\boldsymbol{x}_i = (x_1, \ldots, x_i)$, but the intermediate process and the computed result for $\boldsymbol{x}_{i-1}$ is not useful any more for $\boldsymbol{x}_i$. This phenomenon makes that HJLS-PSLQ is not so efficient for this kind of applications. In this section, we give another variant of HJLS-PSLQ, against such a phenomenon.

4.1 Algorithm description

We now present and analyze the second variant of the HJLS-PSLQ algorithm in this paper, called *incremental HJLS-PSLQ* (IPSLQ for short).

Algorithm 4 (IPSLQ).

Input: A vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ for $i = 1, \ldots, n$, a positive number $M \geq 1$, and a parameter $\gamma > 2/\sqrt{3}$.

Output: Either return an integer relation for $\boldsymbol{x}$, or return "no relation with length smaller than M".

1. Construct $H_x \in \mathbb{R}^{n\times(n-1)}$. Set $H := H_x$, $A := I_n$ and $B := I_n$. Size-reduce H and update A and B. Set $\kappa := n - 1$.
2. While $\kappa \geq 1$ do
   (a) While $h_{n-1,n-1} \neq 0$ do
      i. Choose the largest r such that $\gamma^r h_{r,r} = \max_{j\in\{\kappa,\cdots,n-1\}} \{\gamma^j h_{j,j}\}$.
      ii. Swap the r-th and the $(r+1)$-th rows of H and update A and B.
      iii. If $r < n - 1$ then update H to the L-factor of H.
      iv. Size-reduce H and update A and B.
      v. If $\max_{j\in\{\kappa,\ldots,n-1\}} h_{j,j} < 1/M$ then do the following: If $\kappa > 1$ then update $\kappa := \kappa - 1$ and go to Step 2; Else return "no relation with length smaller than M".
   (b) Return the last column of B.

The main difference between IPSLQ and HJLS-PSLQ is that the latter one considers $x_1, \ldots, x_n$ directly, while the former one considers $x_1, \ldots, x_n$ gradually, i.e., if the vector $(x_i, \ldots, x_n)$ has no relation with 2-norm less than M then IPSLQ adds x_{i-1} to the left and tries to find an integer relation for $(x_{i-1}, x_i, \ldots, x_n)$; see Step 2(a)v.

LEMMA 4. *Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with $x_i \neq 0$ and $\boldsymbol{x}' = (x_j, \cdots, x_n)$. Then the hyperplane matrix $H_{x'}$ for $\boldsymbol{x}'$ is exactly the right-bottom-most $(n-j+1) \times (n-j)$ submatrix of the hyperplane matrix H_x for $\boldsymbol{x}$.*

PROOF. It directly follows from Definition 1. □

Note that if we let $\boldsymbol{x}' = (x_1, \cdots, x_{n-1})$, then this lemma does not follow. From Lemma 4, if one needs to repeatedly call HJLS-PSLQ for $\boldsymbol{x}_i = (x_i, \ldots, x_n)$ with $i = n-1, \ldots, 1$, one can call IPSLQ only once to solve the same problem. In fact, Lemma 4 implies that each transformation on $H_{x'}$ can be seen as transformation on H_x. Thus, all of the previous computed results for $\boldsymbol{x}_{n-1}, \ldots, \boldsymbol{x}_{i-1}$ are still able to be used for $\boldsymbol{x}_i$. Combine with this observation, one can conclude that IPSLQ has the same complexity bound as HJLS-PSLQ.

THEOREM 2. *The IPSLQ algorithm either finds an integer relation $\boldsymbol{m}$ for $\boldsymbol{x}$, or proves no relation with 2-norm smaller than M, within $\mathcal{O}(n^3 + n^2 \log \lambda(\boldsymbol{x}))$ iterations, where $M = \mathcal{O}(\lambda(\boldsymbol{x}))$. Each iteration can be finished within $\mathcal{O}(n^2)$ exact real arithmetic operations.*

PROOF. Actually, each iteration of IPSLQ can be finished within $\mathcal{O}(n^2)$ exact real operations by using the strategy in [10, p. 363].

Let $\boldsymbol{x}_\kappa = (x_\kappa, x_{\kappa+1}, \ldots, x_n)$. To analyze the worst case, assume that the algorithm terminates with $\kappa = 1$. This implies that for any $\kappa > 1$, there does not exist an integer relation for $\boldsymbol{x}_\kappa$ with length smaller than M, i.e., $\lambda(\boldsymbol{x}_\kappa) > M$ for $\kappa > 1$. For each κ, define

$$P_\kappa^{(k)} = \begin{cases} \prod_{j=\kappa}^{n-1} \min\left\{\gamma^{n-\kappa} M, 1/h_{j,j}^{(k)}\right\}^{n-j}, & \kappa = 2, \ldots, n-1, \\ \prod_{j=\kappa}^{n-1} \min\left\{\gamma^{n-\kappa} \lambda(\boldsymbol{x}), 1/h_{j,j}^{(k)}\right\}^{n-j}, & \kappa = 1 \end{cases}$$

as *the P function* of the k-th swap.

For $\kappa = 1$, the P function is the same one as the Π function for HJLS-PSLQ. For $\kappa = 2, \ldots, n-1$, because of the condition in Step 2(a)v, we have $M \geq 1/\max h_{j,j}^{(k)}$ (or else κ will be replaced by $\kappa - 1$), and thus one can obtain

$$\gamma^{n-\kappa-1} \cdot M \cdot h_{r,r}^{(k-1)} \geq 1 \tag{2}$$

for a similar reason with Lemma 3-(3), where r is the swap position and ranges from κ to $n-1$. Therefore, each swap of IPSLQ makes $P_\kappa^{(k+1)} \geq \tau P_\kappa^{(k)}$ for $\kappa = 1, \ldots, n-1$.

Let $\mathrm{Itr}(\kappa)$ be the number of iterations that IPSLQ costs for each κ. For $\kappa = n-1$, we have $1 \leq P_{n-1}^{(k)} \leq \gamma \cdot M$. The left hand side follows from $\min\{M, 1/h_{n-1,n-1}\} \geq 1$ and the right hand side follows from $\min\{M, 1/h_{n-1,n-1}\} \leq M$. Thus $\tau^{\mathrm{Itr}(n-1)} \leq \gamma \cdot M$.

We now consider for $2 \leq \kappa < n-1$. For each $\kappa = 2, \ldots, n-2$, let $P_\kappa^{(b)}$ and $P_\kappa^{(e)}$ be the P function at the beginning and at the end for each κ, respectively. Then we have $\tau^{\mathrm{Itr}(\kappa)} \leq P_\kappa^{(e)}/P_\kappa^{(b)}$. At the end,

$$P_\kappa^{(e)} \leq \prod_{j=\kappa}^{n-1} \left(\gamma^{n-\kappa} M\right)^{n-j} \leq (\gamma^{n-\kappa} M)^{(n-\kappa)(n-\kappa+1)/2}.$$

At the beginning, $h_{\kappa+1,\kappa+1}^{(b)}, \ldots, h_{n-1,n-1}^{(b)}$ for κ are the same as $h_{\kappa+1,\kappa+1}^{(e)}, \ldots, h_{n-1,n-1}^{(e)}$ for $\kappa + 1$. So we have

$$\begin{aligned} P_\kappa^{(b)} &= \min\{\gamma^{n-\kappa} M, \frac{1}{h_{\kappa,\kappa}^{(b)}}\}^{n-\kappa} \cdot \prod_{j=\kappa+1}^{n-1} \min\{\gamma^{n-\kappa} M, \frac{1}{h_{j,j}^{(b)}}\}^{n-j} \\ &\geq \prod_{j=\kappa+1}^{n-1} \min\{\gamma^{n-\kappa} M, 1/h_{j,j}^{(b)}\}^{n-j} \geq P_{\kappa+1}^{(e)} \\ &\geq P_{k+1}^{(b)} \cdot \tau^{\mathrm{Itr}(\kappa+1)} \geq P_{k+2}^{(b)} \cdot \tau^{\mathrm{Itr}(\kappa+2)} \cdot \tau^{\mathrm{Itr}(\kappa+1)} \\ &\geq P_{n-2}^{(b)} \cdot \tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(\kappa+1)} \\ &\geq \tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(\kappa+1)}. \end{aligned}$$

Therefore,

$$\tau^{\mathrm{Itr}(\kappa)} \leq \frac{P_\kappa^{(e)}}{P_\kappa^{(b)}} \leq \frac{(\gamma^{n-\kappa} M)^{(n-\kappa)(n-\kappa+1)/2}}{\tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(\kappa+1)}},$$

which means

$$\tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(2)} \leq (\gamma^{n-2} M)^{(n-2)(n-2+1)/2}.$$

Similarly, when $\kappa = 1$,

$$P_1^{(b)} \geq \tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(2)},$$

and

$$P_1^{(e)} \leq (\gamma^{n-1} \lambda(\boldsymbol{x}))^{(n-1)(n-1+1)/2}.$$

So we have

$$\tau^{\mathrm{Itr}(1)} \leq (\gamma^{n-1} \lambda(\boldsymbol{x}))^{(n-1)(n-1+1)/2} / \tau^{\mathrm{Itr}(n-2)+\cdots+\mathrm{Itr}(2)}.$$

Therefore, the total number of swaps that the IPSLQ algorithm requires is bounded by

$$\begin{aligned} \sum_{\kappa=1}^{n-1} \mathrm{Itr}(\kappa) &\leq \log_\tau (\gamma^{n-1} \lambda(\boldsymbol{x}))^{(n-1)(n-1+1)/2} + \log_\tau(\gamma \cdot M) \\ &= \mathcal{O}(n^3 + n^2 \log(\lambda(\boldsymbol{x}))), \end{aligned}$$

which completes the proof. □

REMARK 1. *If one uses the classical Π function for HJLS-PSLQ directly, then one can only obtain $\gamma^{n-\kappa} \lambda(\boldsymbol{x}_\kappa) h_{r,r}^{(k-1)} \geq \gamma$ before performing the k-th swap. From that, the function*

$$\prod_{j=\kappa}^{n-1} \min\{\gamma^{n-\kappa} \lambda(\boldsymbol{x}_\kappa), 1/h_{j,j}^{(k)}\}^{n-j}$$

implies the termination of the algorithm as well, but with a worse iteration count bound.

4.2 Application

In this section, we consider the problem of finding minimal polynomial (FMP for short) of an algebraic number: Given an algebraic number α with degree $\leq d$ and height $\leq M$, how to compute its minimal polynomial?

Note that the FMP problem is different from the problem of reconstructing the minimal polynomial from an approximation (or a floating-point approximation) of an algebraic number. We do not consider the latter one here because we restrict ourselves under the exact real arithmetic model. We refer to [20, 18, 26] for the latter problem.

Indeed, solving the FMP problem is a natural application of HJLS-PSLQ. One can call HJLS-PSLQ directly with input as $(1, \alpha, \ldots, \alpha^i)$ for $i = 1, \ldots, d$ until the algorithm returns

an integer relation for some $i = d_0 \leq d$, where d_0 is the exact degree of α. In the worst case ($d_0 = d$), one needs to call HJLS-PSLQ repeatedly d times, and the computed results of previous HJLS-PSLQ computation will not be useful for the next. So, this assignment requires totally $\mathcal{O}(d^4 + d^3 \log M)$ iterations.

We now consider how to use IPSLQ to solve the FMP problem. If we feed IPSLQ with $\boldsymbol{x} = (1, \alpha, \ldots, \alpha^d)$ as in classical methods, then IPSLQ firstly computes the integer relation for (α^{d-1}, α^d), which does not make sense. Naturally, we can feed the input as $\boldsymbol{x} = (\alpha^d, \alpha^{d-1}, \ldots, \alpha, 1)$. Lemma 4 guarantees that, for $i = 2, \ldots, d$, the hyperplane matrix for $(\alpha^{i-1}, \ldots, \alpha, 1)$ is exactly the right-bottom most submatrix of the hyperplane matrix for $(\alpha^i, \alpha^{i-1}, \ldots, \alpha, 1)$. Then we have the following algorithm.

Algorithm 5 (FMP).

Input: An algebraic number α with two upper bounds, d on the degree and M on the height of α, and a parameter $\gamma > 2/\sqrt{3}$.

Output: The exact minimal polynomial of α.

1. Initialize $\boldsymbol{x} = (\alpha^d, \alpha^{d-1}, \ldots, \alpha, 1)$.
2. Calling Algorithm 4 with input as $\boldsymbol{x}$, M and γ returns $\boldsymbol{m} = (m_d, m_{d-1}, \cdots, m_1, m_0)$ as an integer relation for $\boldsymbol{x}$.
3. Return $P(X) = m_d X^d + m_{d-1} X^{d-1} + \cdots + m_1 X + m_0$.

The application of IPSLQ to efficiently solve the FMP problem depends on the following two key points: (1) We use $(x_1, \ldots, x_n) = (\alpha^{d-1}, \ldots, \alpha, 1)$, which is the reverse order of the traditional version, to construct the hyperplane matrix H. (2) The important observation is that the matrix H_x for $(x_i, \ldots, x_n)$ is exactly the right-bottom most submatrix of H_x for $(x_{i-1}, x_i, \ldots, x_n)$. When we add x_{i-1} to $(x_i, \ldots, x_n)$ from the left, the new hyperplane matrix adds a new column to the old one from the left. Thus, the information produced by the previous iterations is still useful for the new matrix H. Traditional methods based on HJLS-PSLQ do not possess this property, and thus are less efficient, i.e., using more iterations.

Based on the analysis above, the complexity bound on solving FMP with IPSLQ, given merely an upper bound on degree, can be the same as with PSLQ, given the exact degree.

COROLLARY 1. *Algorithm 5 correctly computes the minimal polynomial of α. It costs $\mathcal{O}(d^3 + d^2 \log M)$ IPSLQ iterations.*

PROOF. Obviously, the most costly step is the IPSLQ call, whose complexity result is given in Theorem 2. □

4.3 Experimental results

The aim of the following experiments is only to show the difference between the traditional HJLS-PSLQ (Algorithm 1) and IPSLQ (Algorithm 4) when solving the FMP problem. Since we have neither the theoretical bit-complexity of HJLS-PSLQ nor its variant (see Section 5), all experimental results presented in this section are preliminary, and do not aim to compare with many other mature software packages with similar function but using other techniques. Thus, we only count the number of iterations.

All the implementations are coded in Maple 15, using multi-precision floating-point arithmetic. For fairness, the same function uses the same technique in our implementations of HJLS-PSLQ and IPSLQ.

Consider $\alpha = 3^{1/s} + 2^{1/t}$. Running these experiments in Maple 15 with `Digits := 500` gives preliminary experimental results in Table 1. Note that here `Digits := 500` may not be necessary. It is for fairness and it guarantees that both HJLS-PSLQ and IPSLQ return the correct answer for each test example. (To some extent, it simulates the exact real arithmetic behavior with multi-precision floating point arithmetic.) The reason why we choose such type of α is that the degree of α is easy to determine (this is convenient for our purpose) and that a similar type of α was used to test HJLS-PSLQ in [9].

In Table 1, the input degree bound and height bound in these tests are d and $M + 1$; the exact degree and height of α are $d - 1$ and M, respectively. All these experimental results are obtained by using a Windows 7 (32 bits mode) PC with AMD Athlon II X4 645 processor (3.10 GHz) and 4 GB memory. Note that there exists a built-in function `IntegerRelations:-PSLQ` in Maple 15, but for the comparison in Table 1, we implement the PSLQ algorithm by ourselves. The reasons we do not use the built-in function is that there does not exist a height parameter M in the built-in function. This may cause that the built-in function will go on performing the iterations even though the height has been greater than M.

Table 1: Algorithm 1 versus Algorithm 4 for FMP

No.	s	t	d	M	Itr_I	Itr_P
1	2	2	5	10	12	33
2	2	3	7	36	39	160
3	3	3	10	125	173	1,119
4	3	4	13	540	504	3,663
5	2	7	15	5,103	990	7,803
6	3	6	19	10,278	2,034	18,784
7	4	5	21	11,160	2,542	26,135
8	5	5	26	57,500	5,225	59,681
9	5	6	31	538,380	9,471	124,541
10	6	6	37	4,281,690	16,560	246,798

According to Table 1, the IPSLQ algorithm is faster than the HJLS-PSLQ algorithm when used to solve the FMP problem in the sense that IPSLQ uses less iterations (Itr_I) than that of HJLS-PSLQ (Itr_P).

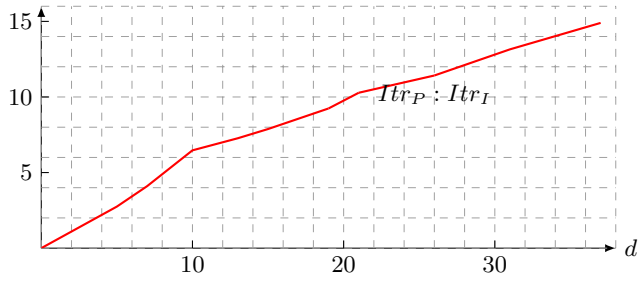

Figure 1: The relation between $Itr_P : Itr_I$ and the dimension d

Moreover, the ratio between Itr_P and Itr_I appears to be linear with respect to d (see Figure 1) and seems to get larger and larger with increasing d but always smaller than d. This result supports the theoretical analysis in the above

section that claims IPSLQ saves a factor d (up to a constant) comparing with HJLS-PSLQ when solving FMP. The above experiments indicate that the constant is smaller than 1, because for each $1 \le \kappa \le n-1$, IPSLQ performs the four step iteration until $\max_{j \in \{\kappa,\ldots,n-1\}} h_{j,j} < 1/M$, while HJLS-PSLQ may not make $h_{j,j}$ so small due to the global Bergman swap strategy.

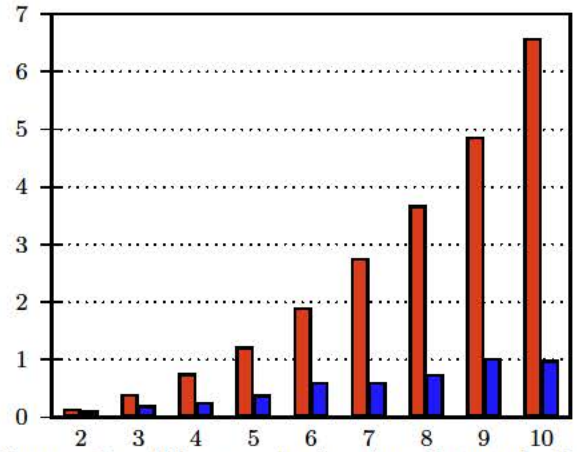

Figure 2: The x-axis is the detected degree d', i.e., current detecting an integer relation for $(\alpha^{d'}, \alpha^{d'-1}, \ldots, \alpha, 1)$; the y-axis is the number of iterations divided by 100; the red bar is for HJLS-PSLQ and the blue bar is for IPSLQ.

Figure 2 illustrates how many iterations one can gain from the incremental modification. We choose the fourth row in Table 1, i.e., $\alpha = \sqrt[4]{2} + \sqrt[3]{3}$. Other rows behave similarly. The reason why only considering $d' \le 10$ is that for those $d' \le 10$, the sum of iteration counts for IPSLQ is already 501, out of the total count 504. As the figure showed, for each d', the iteration count of IPSLQ is fairly smaller than that of HJLS-PSLQ. Moreover, the difference between them turns to be larger and larger when d' increases. The reason is that IPSLQ reuses the information computed by previous steps. In contrast, HJLS-PSLQ does not share this property and has to compute the previous information repeatedly. When d' increases, the amount of repeated computation becomes large.

5. FUTURE WORK

In this paper, we discuss all problems and algorithms under the exact real arithmetic model. A natural question is how to analyze the bit-complexity under the floating-point arithmetic model, like the successful analyses of floating-point LLL algorithms [24, 23, 25, 12]. However, to the best of the authors' knowledge, there exists no theoretical bit-complexity analysis for HJLS-PSLQ, nor its variants. Another interesting project is to find more applications for which the IPSLQ algorithm can be used to replace many repeated HJLS-PSLQ calls. Also, it is worth investigating the practical efficiency of the parallel implementation of the modified multi-pair HJLS-PSLQ algorithm.

Acknowledgements. The authors would like to thank the three anonymous referees for their helpful remarks and suggestions, especially the one who pointed out us references about parallel lattice reduction algorithms.

6. REFERENCES

[1] Babai, L., Just, B., and Meyer auf der Heide, F. On the limits of computations with the floor function. *Information and Computation 78*, 2 (1988), 99–107.

[2] Bailey, D. H., and Borwein, J. M. PSLQ: An algorithm to discover integer relations. *Computeralgebra Rundbrief*, 45 (2009), 8–11.

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

[4] Bergman, G. M. Notes on Ferguson and Forcade's generalized euclidean algorithm. *University of California at Berkeley, unpublished notes* (1980). Available from: `math.berkeley.edu/~gbergman/papers/unpub/FF_Euc.pdf`.

[5] Borwein, J. M., and Lisoněk, P. Applications of integer relation algorithms. *Discrete Mathematics 217*, 1–3 (2000), 65–82.

[6] Borwein, P. *Computational Excursions in Analysis and Number Theory*. Springer, New York, 2002.

[7] Chen, J., Stehlé, D., and Villard, G. A new view on HJLS and PSLQ: Sums and projections of lattices. In *Proceedings of the 2013 international symposium on Symbolic and algebraic computation* (Boston, USA, 2013), pp. 149–156.

[8] Feng, Y., Chen, J., and Wu, W. Incremental PSLQ with application to algebraic number reconstruction. *ACM Communications in Computer Algebra 47*, 3 (2013), 112–113.

[9] Ferguson, H. R. P., and Bailey, D. H. A polynomial time, numerically stable integer relation algorithm. Tech. Rep. RNR-91-032, NAS Applied Research Branch, NASA Ames Research Center, July 1992.

[10] Ferguson, H. R. P., Bailey, D. H., and Arno, S. Analysis of PSLQ, an integer relation finding algorithm. *Mathematics of Computation 68*, 225 (1999), 351–369.

[11] Ferguson, H. R. P., and Forcade, R. W. Generalization of the euclidean algorithm for real numbers to all dimensions higher than two. *Bulletin of the American Mathematical Society 1*, 6 (1979), 912–914.

[12] Goel, S., Morel, I., Stehlé D., and Villard, G. LLL reducing with the most significant bits. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation* (Kobe, Japan, 2014). To appear.

[13] Golub, G. H., and van Loan, C. *Matrix Computations*, 4th ed. The John Hopkins University Press, Baltimore, 2013.

[14] Håstad, J., Just, B., Lagarias, J. C., and Schnorr, C.-P. Polynomial time algorithms for finding integer relations among real numbers. *SIAM Journal on Computing 18*, 5 (1989), 859–881. Preliminary version: In Proceedings of STACS'86, pp. 105–118, 1986.

[15] Heckler, C., and Thiele, L. A parallel lattice basis reduction for mesh-connected processor arrays and parallel complexity. In *Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing*

(Dallas, USA, 1993), pp. 400–407.

[16] Heckler, C., and Thiele, L. Complexity analysis of a parallel lattice basis reduction algorithm. *SIAM Journal on Computing 27*, 5 (1998), 1295–1302.

[17] van Hoeij, M. Factoring polynomials and the knapsack problem. *Journal of Number Theory 95*, 2 (2002), 167–189.

[18] van Hoeij, M., and Novocin, A. Gradual sub-lattice reduction and a new complexity for factoring polynomials. *Algorithmica 63*, 3 (2012), 616–633. Preliminary version: In Proceedings of LATIN '10, pp. 539–553, 2010.

[19] Joux, A. A fast parallel lattice reduction algorithm. In *Proceedings of the 2nd Gauss Symposium* (Munich, Germany, 1993). Available at `www-polsys.lip6.fr/~joux/pages/papers/FastParallel.pdf`.

[20] Kannan, R., Lenstra, A. K., and Lovász, L. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. *Mathematics of Computation 50*, 181 (1988), 235–250. Preliminary version: In Proceedings of STOC '84, pp. 191–200, 1984.

[21] Lenstra, A. K., Lenstra, H. W., and Lovász, L. Factoring polynomials with rational coefficients. *Mathematische Annalen 261*, 4 (1982), 515–534.

[22] Meichsner, A. Integer Relation Algorithms and the Recognition of Numerical Constants. Master's thesis, Simon Fraser University, 2001.

[23] Morel, I., Stehlé, D., and Villard, G. H-LLL: Using householder inside LLL. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation* (Seoul, Republic of Korea, 2009), pp. 271–278.

[24] Nguyen, P. Q., and Stehlé, D. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing 39*, 3 (2009), 874–903. Preliminary version: In Proceedings of EUROCRYPT '05, LNCS 3494, pp. 215–233, 2005.

[25] Novocin, A., Stehlé, D., and Villard, G. An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In *Proceedings of the 43rd annual ACM symposium on Theory of computing* (San Jose, USA, 2011), pp. 403–412.

[26] Qin, X., Feng, Y., Chen, J., and Zhang, J. A complete algorithm to find exact minimal polynomial by approximations. *International Journal of Computer Mathematics 89*, 17 (2012), 2333–2344. Preliminary version: In Proceedings of SNC '09, pp. 125–131, 2009.

[27] Roch, J.-L., and Villard, G. Parallel gcd and lattice basis reduction. In *Parallel Processing: CONPAR 92–VAPP V*, L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, Eds., vol. 634 of *Lecture Notes in Computer Science*. Springer, 1992, pp. 557–564.

[28] Villard, G. Parallel lattice basis reduction. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation* (Berkeley, USA, 1992), pp. 269–277.

[29] Wu, W., Chen, J., and Feng, Y. Sparse bivariate polynomial factorization. *Science China Mathematics* (2014). To appear.

[30] Zhang, J., and Feng, Y. Obtaining exact value by approximate computations. *Science in China Series A: Mathematics 50*, 9 (2007), 1361–1368.