Vector Rational Number Reconstruction

Curtis Bright
cbright@uwaterloo.ca

Arne Storjohann
astorjoh@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

The final step of some algebraic algorithms is to reconstruct the common denominator d of a collection of rational numbers $(n_i/d)_{1\le i\le n}$ from their images $(a_i)_{1\le i\le n}$ mod M, subject to a condition such as $0 < d \le N$ and $|n_i| \le N$ for a given magnitude bound N. Applying elementwise rational number reconstruction requires that $M \in \Omega(N^2)$. Using the gradual sublattice reduction algorithm of van Hoeij and Novocin [23], we show how to perform the reconstruction efficiently even when the modulus satisfies a considerably smaller magnitude bound $M \in \Omega(N^{1+1/c})$ for c a small constant, for example $2 \le c \le 5$. Assuming $c \in O(1)$ the cost of the approach is $O(n(\log M)^3)$ bit operations using the original LLL lattice reduction algorithm, but is reduced to $O(n(\log M)^2)$ bit operations by incorporating the L^2 variant of Nguyen and Stehlé [17]. As an application, we give a robust method for reconstructing the rational solution vector of a linear system from its image, such as obtained by a solver using p-adic lifting.

Categories and Subject Descriptors

F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*Number-theoretic computations*; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms

General Terms

Algorithms

Keywords

Rational reconstruction, lattice basis reduction

1. INTRODUCTION

A rational number reconstruction of an integer $a \in \mathbb{Z}$ with respect to a positive modulus $M \in \mathbb{Z}_{>0}$ is a signed fraction $n/d \in \mathbb{Q}$ with $\gcd(n, d) = 1$ such that $a \equiv n/d \pmod M$. In general, there may be multiple possibilities, for example $a \equiv n_1/d_1 \equiv n_2/d_2 \pmod M$ with $n_1 \not\equiv n_2 \pmod M$.

Assuming a reconstruction exists, its uniqueness can be ensured by stipulating bounds for the magnitudes of the output integers n and d. In addition to a and M, the simplest version of the problem takes as input a bound $N < M$, and asks for output a pair of integers (d, n) such that

$$da \equiv n \pmod M, \qquad |n| \le N, \qquad 0 < d \le N. \tag{1}$$

Note that if (d, n) is a solution to (1) with $\gcd(n, d) = 1$ then n/d is a rational reconstruction of $a \pmod M$. Since there are $\Theta(N^2)$ coprime pairs (d, n) which satisfy the bounds of (1), a requirement for rational reconstruction uniqueness is $M \in \Omega(N^2)$ by the pigeonhole principle.

In fact, if the bound $M > 2N^2$ is satisfied then there is at most one solution of (1) with $\gcd(n, d) = 1$. Such a solution can be computed effectively using the well known approach based on the extended Euclidean algorithm and the continued fraction expansion of a/M. See for example [11, Theorem 5.1] or the books [8, 21].

Rational number reconstruction is an essential tool in many algorithms that employ a homomorphic imaging scheme to avoid intermediate expression swell, to allow for a simple coarse grain parallelization, or to facilitate an output sensitive approach; explicit examples include solving sparse rational systems [4] and computing gcds of polynomials [7]. Often, the final step of these algorithms is to reconstruct the common denominator $d \in \mathbb{Z}_{>0}$ of a collection of rational numbers $(n_i/d)_{1\le i\le n}$ from their images $(a_i)_{1\le i\le n}$ modulo M. The images modulo M are typically computed by combining multiple smaller images, either using Chinese remaindering ($M = p_1p_2\cdots p_m$) or a variation of Newton–Hensel lifting ($M = p^m$). The cost of an algorithm that uses a homomorphic imaging scheme is highly correlated to m, the number of smaller images computed, which is directly related to the bitlength of the modulus M. Ideally, just enough smaller images are computed to allow reconstruction of the common denominator d. If N is an upper bound for both d and $\max_i|n_i|$, elementwise rational reconstruction can be applied but requires that $M > 2N^2$ to ensure success.

This paper gives a deterministic algorithm for efficiently computing the common denominator d that for some applications requires about half as many image computations as the standard approach. Our specification of the vector version of the problem differs slightly from the scalar case shown in (1). The vector rational reconstruction problem takes as input a vector $\boldsymbol{a} \in \mathbb{Z}^n$ of images modulo M, and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISSAC'11, June 8–11, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0675-1/11/06 ...$10.00.

asks for a pair $(d, \boldsymbol{n}) \in (\mathbb{Z}, \mathbb{Z}^n)$ such that

$$d\boldsymbol{a} \equiv \boldsymbol{n} \pmod{M}, \qquad 0 < \left\| \left[\, d \mid \boldsymbol{n} \,\right] \right\|_2 \leq N. \qquad (2)$$

Here, we use a common bound for d and $\boldsymbol{n}$ based on the 2-norm because this is a more natural condition for the algorithm we will present, which is based on integer lattice basis reduction. In particular, the problem of computing solutions to (2) is equivalent to finding short nonzero vectors in the lattice generated by the rows of the matrix

$$\left[\begin{array}{c|c} & M\boldsymbol{I}_{n\times n} \\ \hline 1 & \boldsymbol{a} \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}. \qquad (3)$$

The lattice shown in (3) is a special case of the "knapsack-type" lattices studied by van Hoeij and Novocin [23], who give an algorithm which can be used to compute a "generating set" for (2), that is, a set $(d_i, \boldsymbol{n}_i)_{1\leq i\leq c}$ corresponding to linearly independent vectors $\left[\, d_i \mid \boldsymbol{n}_i \,\right]_{1\leq i\leq c}$ such that every solution of (2) can be expressed as a $\mathbb{Z}$-linear combination of the members of the generating set. On the one hand, from the scalar case, we know that a sufficient condition to ensure the existence of a generating set of dimension zero (no solution) or one (a unique minimal denominator solution) is that the modulus M be large enough to satisfy $M > 2N^2$. On the other hand, if c is an integer such that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied, it follows from a strengthening of [23, Theorem 2] that the generating set returned will contain at most c vectors. The generating set produced will be LLL reduced, so the 2-norm of the first vector will be at most $2^{(c-1)/2}$ times that of the shortest vector which solves (2).

To apply lattice reduction directly to a lattice with row dimension $n+1$ would be prohibitively expensive in terms of n when n is large. Instead, van Hoeij and Novocin [23] propose a gradual sublattice reduction algorithm which adds columns to the work lattice one by one, while keeping the row dimension bounded by $c+1$ by removing vectors which provably can't contribute to a solution of (2). Assuming $c \in O(1)$, this algorithm applied to bases of the form in (3) will run in $O(n^2(\log M)^3)$ bit operations when the standard LLL algorithm algorithm is used. By using properties of the special basis form (3) and incorporating the L^2 algorithm [17] we show how to reduce the cost to $O(n(\log M)^2)$ bit operations.

The approach is particularly well suited to applications where it is known *a priori* that there can exist at most one linearly independent solution to (2). In Section 5 we consider the problem of solving a nonsingular integer linear system $\boldsymbol{Ax} = \boldsymbol{b}$, which has exactly one rational solution vector with common denominator a factor of $\det \boldsymbol{A}$. As a concrete example, coming from [2], suppose $\boldsymbol{A}$ and $\boldsymbol{b}$ have dimension 10,000 and are filled randomly with single decimal digit integers. The common denominator and the numerators of a typical solution vector for such a system have about 24,044 decimal digits, or of magnitude about $\beta = 10^{24044}$. To apply elementwise rational reconstruction requires $M > 2N^2$ with $N \geq \beta$ to be satisfied, so M needs to have length about 48,088 decimal digits. But by choosing the parameter $c = 5$, the vector algorithm requires only that $M > 2^{(c+1)/2}N^{1+1/c}$ with $N \geq \sqrt{n+1}\beta$ in order to succeed, so M need have only about 28,856 decimal digits. The method we propose is robust in the sense that β need not be known beforehand; if a reconstruction is attempted with N too small, FAIL will be reported, but if N is sufficiently large the algorithm will guarantee to return the correct reconstruction.

Related work.

For the scalar version of the problem, much work has focused on decreasing the running time from $O((\log M)^2)$ to nearly linear in $\log M$ by incorporating fast integer multiplication. For a survey of work in this direction we refer to [15]. An algorithm that doesn't need a priori bounds for the numerator and denominator is described in [16].

Now consider the vector version of the problem. Two approaches are proposed in [12]. The first is a heuristic randomized algorithm to recover a solution d and $\boldsymbol{n}$ that satisfies $d\|\boldsymbol{n}\|_\infty \in O(M)$. The second, based on the good simultaneous Diophantine approximation algorithm in [13], can be applied to find a solution even when $\left\| \left[\, d \mid \boldsymbol{n} \,\right] \right\|_2 \in o(N^2)$ but the algorithm performs lattice reduction directly on $n+1$ dimensional lattices similar to (3) and seems to be expensive when n is large.

The survey [10] has an overview of using lattices with bases similar to (3) for effectively solving real and p-adic simultaneous Diophantine approximation problems. However, these results are concerned with existence, not uniqueness.

An efficient algorithm for the rational function version of the vector reconstruction problem is given in [20].

Organization.

In Section 2 we illustrate the main ideas of the algorithm with a worked example. In Section 3 we establish our notation and recall the required facts about lattices and the LLL lattice basis reduction algorithm. Section 4 presents the algorithm for vector rational reconstruction, proves its correctness, and provides a simple cost analysis. In Section 5 we show how the vector rational reconstruction algorithm can be incorporated into an algorithm for solving nonsingular linear systems to save on the number of required image computations.

2. OUTLINE OF THE ALGORITHM

As previously noted, the problem of finding solutions to (2) is identical to the problem of finding short nonzero vectors, with respect to the 2-norm, in the lattice generated by the rows of the following matrix:

$$\begin{bmatrix} & & & & M \\ & & & \iddots & \\ & & M & & \\ & M & & & \\ 1 & a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)}.$$

The first n rows of the matrix can be used to reduce modulo M the last n entries of any vector in the lattice; in particular, a vector obtained by multiplying the last row by d. The first entry of such a vector will still be d and the ith entry for $2 \leq i \leq n+1$ will be congruent to $da_{i-1} \pmod{M}$.

For example, consider the lattice basis matrix

$$\boldsymbol{L} = \begin{bmatrix} & & & & 195967 \\ & & & 195967 & \\ & & 195967 & & \\ & 195967 & & & \\ 1 & -23677 & -49539 & 74089 & -21989 \end{bmatrix}$$

where our target length is $N = 10^4$. When n is large it is infeasible to reduce the entire basis at once. Instead, the gradual sublattice reduction algorithm of [23] will keep the row dimension of the lattice constant by adding columns one by one, and removing basis vectors which are provably too big. We continue with our example, which is similar to the example given in [18, Section 2.2].

Consider reducing just the lower-left 2×2 submatrix of $\boldsymbol{L}$:

$$\begin{bmatrix} 0 & 195967 \\ 1 & -23677 \end{bmatrix} \xRightarrow{\text{LLL}} \begin{bmatrix} -389 & -96 \\ -149 & 467 \end{bmatrix}$$

Knowing the reduction of the lower-left 2×2 submatrix can help us reduce the lower-left 3×3 submatrix of $\boldsymbol{L}$. The first step is to find a basis of the lower-left 2×3 submatrix by 'adding a column'.

Though we could have kept track of the third column of $\boldsymbol{L}$ while doing the above reduction, it can also be simply computed afterwards: the third column is just a_2 times the first column, since this property holds in the the lower-left 2×3 submatrix of $\boldsymbol{L}$ and is preserved by the unimodular row operations used in lattice basis reduction.

After determining a basis of the lower-left 2×3 submatrix, the next step is simply to 'add a row' to find a basis of the lower-left 3×3 submatrix. For example, the lattice generated by the lower-left 3×3 submatrix of $\boldsymbol{L}$ has the following basis, which we again reduce:

$$\left[\begin{array}{cc|c} 0 & 0 & 195967 \\ \hline -389 & -96 & 19270671 \\ -149 & 467 & 7381311 \end{array}\right] \xRightarrow{\text{LLL}} \begin{bmatrix} -538 & 371 & 470 \\ 91 & 1030 & -808 \\ 27089 & 13738 & 20045 \end{bmatrix}$$

Now, it happens that the final vector in the Gram–Schmidt orthogonalization of this basis has norm larger than N. It follows [23, Lemma 2] that any vector in the lattice generated by this basis which includes the final basis vector must have norm at least N. Since we are only interested in vectors shorter than this, we can safely discard the last row, and repeat the same augmentation process to find a sublattice which contains all short vectors in the lattice generated by the lower-left 4×4 submatrix of $\boldsymbol{L}$.

If $M > 2^{(c+1)/2}N^{1+1/c}$ for $c \in \mathbb{Z}_{>0}$ then the process described above of adding columns and removing final rows of the lattice will keep the row dimension of the lattice bounded by $c+1$. For $c \in O(1)$ this leads to a cost estimate that is quadratic in n. To obtain a running time that is linear in n, we avoid computing the basis vectors in $\boldsymbol{L}$ as the reduction proceeds. Instead, we show how to reconstruct the entire basis from only its first column.

3. PRELIMINARIES

For a $k \times n$ matrix $\boldsymbol{L}$ we let $\boldsymbol{L}_S$ be the rows of $\boldsymbol{L}$ which have indices in $S \subseteq \{1, \ldots, k\}$, let $\boldsymbol{L}_R^{\mathrm{T}}$ be the columns of $\boldsymbol{L}$ which have indices in $R \subseteq \{1, \ldots, n\}$, and let $\boldsymbol{L}_{S,R}$ denote $(\boldsymbol{L}_S)_R^{\mathrm{T}}$. We simply write i for $\{i\}$ and $1..i$ for $\{1, \ldots, i\}$. When not used with a subscript, $\boldsymbol{L}^{\mathrm{T}}$ denotes the transpose of $\boldsymbol{L}$. A subscript on a row vector will always refer to entrywise selection, and the norm of a row vector will refer to the 2-norm, $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}}$.

Vectors are denoted by lower-case bold variables and matrices by upper-case or Greek bold variables, with the boldface dropped when referring to individual entries. The $\mathrm{rem}_M(x)$ function returns the reduction of $x \pmod M$ in the symmetric range, and applies elementwise to vectors and matrices.

3.1 Lattices

A *point lattice* is a discrete additive subgroup of $\mathbb{R}^n$. The elements of the lattice generated by the rank k matrix $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$ are given by

$$\mathcal{L}(\boldsymbol{L}) := \left\{ \sum_{i=1}^{k} r_i \boldsymbol{L}_i : r_i \in \mathbb{Z} \right\},$$

and $\boldsymbol{L}$ is a *basis* of $\mathcal{L}(\boldsymbol{L})$. If $\mathcal{L}(\boldsymbol{S}) \subseteq \mathcal{L}(\boldsymbol{L})$ then $\mathcal{L}(\boldsymbol{S})$ is known as a *sublattice* of $\mathcal{L}(\boldsymbol{L})$; this occurs if and only if there exists an integer matrix $\boldsymbol{B}$ such that $\boldsymbol{S} = \boldsymbol{B}\boldsymbol{L}$.

The set of vectors in $\mathcal{L}(\boldsymbol{L})$ shorter than some target length N is denoted

$$\mathcal{L}_N(\boldsymbol{L}) := \{\, \boldsymbol{b} \in \mathcal{L}(\boldsymbol{L}) : \|\boldsymbol{b}\| \le N \,\}.$$

We call a basis of a sublattice of $\boldsymbol{L}$ which contains all the elements of $\mathcal{L}_N(\boldsymbol{L})$ a *generating matrix* of $\mathcal{L}_N(\boldsymbol{L})$.

3.2 Linear Algebra

For a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$, let $\boldsymbol{L}^* \in \mathbb{Q}^{k\times n}$ denote its Gram–Schmidt orthogonal $\mathbb{R}$-basis and let $\boldsymbol{\mu} \in \mathbb{Q}^{k\times k}$ denote the associated change-of-basis matrix. That is,

$$\begin{bmatrix} \boldsymbol{L}_1 \\ \boldsymbol{L}_2 \\ \vdots \\ \boldsymbol{L}_k \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & & \ddots & \\ \mu_{k,1} & \mu_{k,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1^* \\ \boldsymbol{L}_2^* \\ \vdots \\ \boldsymbol{L}_k^* \end{bmatrix}$$

with $\mu_{i,j} = \boldsymbol{L}_i(\boldsymbol{L}_j^*)^{\mathrm{T}}/\|\boldsymbol{L}_j^*\|^2$. Additionally, let $\boldsymbol{G} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}} \in \mathbb{Z}^{k\times k}$ denote the Gramian matrix of $\boldsymbol{L}$.

3.3 LLL Reduction

A lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$ (or its Gramian $\boldsymbol{G} \in \mathbb{Z}^{k\times k}$) is said to be *LLL-reduced* if its Gram–Schmidt orthogonalization satisfies the conditions

1. $\|\boldsymbol{\mu} - \boldsymbol{I}_{k\times k}\|_{\max} \le \frac{1}{2}$,
2. $\|\boldsymbol{L}_i^*\|^2 \ge (\frac{3}{4} - \mu_{i,i-1}^2)\|\boldsymbol{L}_{i-1}^*\|^2$ for $1 < i \le k$.

Given a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$, the *lattice basis reduction* problem is to compute an LLL-reduced basis $\boldsymbol{L}'$ such that $\mathcal{L}(\boldsymbol{L}) = \mathcal{L}(\boldsymbol{L}')$. Assuming $k \in O(1)$, the L^2 algorithm from [17] accomplishes this in $O(n(\log B)^2)$ bit operations, where $\max_i \|\boldsymbol{L}_i\| \le B$. A well known and important feature of the LLL algorithm, that we will exploit, is that the sequence of unimodular row operations required to reduce a given lattice can be determined strictly from the $\mu_{i,j}$ and $\|\boldsymbol{L}_i^*\|^2$, or in the case of L^2, the Gramian.

Consider Algorithm 1, which only includes the specification of the input and output. If $\boldsymbol{G}$ is the Gramian for a lattice $\boldsymbol{L}$ with row dimension k and arbitrary column dimension, we could LLL reduce $\boldsymbol{L}$ in-place by calling $\mathsf{InPlaceL}^2(k, \boldsymbol{G}, \boldsymbol{L})$. Alternatively, we could also initialize a matrix $\boldsymbol{U}$ as $\boldsymbol{I}_{k\times k}$, call $\mathsf{InPlaceL}^2(k, \boldsymbol{G}, \boldsymbol{U})$ to capture all required unimodular transformations in $\boldsymbol{U}$, and then compute the reduced lattice as $\boldsymbol{U}\boldsymbol{L}$.

The following algorithm, Algorithm 2 from [23] applied to L^2, computes an LLL-reduced generating matrix (for a given target length N) of a lattice by using L^2 and discarding vectors which are too large to contribute to a vector shorter than the target length.

Algorithm 1 The $\mathsf{InPlaceL}^2(k, \boldsymbol{G}, \boldsymbol{U})$ lattice basis reduction algorithm.

Input: The Gramian $\boldsymbol{G} \in \mathbb{Z}^{k\times k}$ of some lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times *}$, and a matrix $\boldsymbol{U} \in \mathbb{Z}^{k\times *}$.
Output: Use L^2 to update $\boldsymbol{G}$ to be an LLL-reduced Gramian of $\mathcal{L}(\boldsymbol{L})$ and apply all unimodular row operations to $\boldsymbol{U}$.

Algorithm 2 The $\mathsf{L}^2\mathsf{WithRemovals}(k, \boldsymbol{G}, \boldsymbol{U}, N)$ generating matrix algorithm.

Input: The Gramian $\boldsymbol{G} \in \mathbb{Z}^{k\times k}$ of some lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times *}$, a target length $N \in \mathbb{Z}_{>0}$, and a matrix $\boldsymbol{U} \in \mathbb{Z}^{k\times *}$.
Output: Use L^2 to update $\boldsymbol{G}$ to be an LLL-reduced Gramian of a generating matrix of $\mathcal{L}_N(\boldsymbol{L})$, update k to be its number of rows, and apply all unimodular row operations to $\boldsymbol{U}$.

4. THE VECRECON ALGORITHM

In this section we present our vector rational reconstruction algorithm, which computes an LLL-reduced generating matrix for $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$, where $\boldsymbol{a} \in \mathbb{Z}^{1\times n}$, $M \in \mathbb{Z}_{>0}$, and

$$\boldsymbol{\Lambda}_{\boldsymbol{a}}^M := \begin{bmatrix} & & & & M \\ & & & \iddots & \\ & & M & & \\ & M & & & \\ 1 & a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)}.$$

The computed generating matrix contains at most c vectors, where $c \geq 1$ is a small constant such that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied. The larger c is chosen, the smaller M is allowed to be (assuming $c < \sqrt{2\log_2 N}$). However, if c is chosen too large the algorithm may be inefficient, since in the worst case it will reduce bases containing up to $c+1$ vectors.

As already outlined, the algorithm computes a generating matrix of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ gradually, by computing generating matrices of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..l}}^M)$ for $l = 1, 2, \ldots, n$. However, for efficiency these intermediate generating matrices will not be explicitly stored during the algorithm; we will only keep track of the first column $\boldsymbol{f}$. The following lemma shows how all sublattices of $\mathcal{L}(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..l}}^M)$ have bases of a special form which strongly depends on the first basis column.

LEMMA 1. *Any $\boldsymbol{L} \in \mathbb{Z}^{k\times(l+1)}$ with $\mathcal{L}(\boldsymbol{L}) \subseteq \mathcal{L}(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..l}}^M)$ is of the form*

$$\left[\, \boldsymbol{L}_1^{\mathrm{T}} \;\middle|\; \mathrm{rem}_M(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..l}) + M\boldsymbol{R} \,\right]$$

for some $\boldsymbol{R} \in \mathbb{Z}^{k\times l}$.

PROOF. Let $\boldsymbol{\Lambda}$ denote $\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..l}}^M$. Since $\mathcal{L}(\boldsymbol{L})$ is a sublattice of $\mathcal{L}(\boldsymbol{\Lambda})$ there exists a $\boldsymbol{B} \in \mathbb{Z}^{k\times(l+1)}$ such that

$$\begin{aligned} \boldsymbol{L} &= \boldsymbol{B}\boldsymbol{\Lambda} \\ &= \boldsymbol{B}_{l+1}^{\mathrm{T}}\boldsymbol{\Lambda}_{l+1} + \boldsymbol{B}_{1..l}^{\mathrm{T}}\boldsymbol{\Lambda}_{1..l} \\ &= \left[\, \boldsymbol{B}_{l+1}^{\mathrm{T}} \;\middle|\; \boldsymbol{B}_{l+1}^{\mathrm{T}}\boldsymbol{a}_{1..l} + \boldsymbol{B}_{1..l}^{\mathrm{T}}\boldsymbol{\Lambda}_{1..l,2..l+1} \,\right], \end{aligned}$$

so $\boldsymbol{B}_{l+1}^{\mathrm{T}} = \boldsymbol{L}_1^{\mathrm{T}}$. The result follows since M divides every entry of $\boldsymbol{\Lambda}_{1..l,2..l+1}$ and $\mathrm{rem}_M(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..l}) = \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..l} + M\boldsymbol{Q}$ for some $\boldsymbol{Q} \in \mathbb{Z}^{k\times l}$. ☐

The complete algorithm pseudocode is given as Algorithm 3. Were we explicitly storing the generating matrix $\boldsymbol{L}$, we would require the initialization of $L_{1,1} := 1$ in step 1, and the column/row augmentation in step 3 of

$$\boldsymbol{L} := \left[\begin{array}{c|c} \boldsymbol{0} & M \\ \hline \boldsymbol{L} & \mathrm{rem}_M(a_l\boldsymbol{f}) \end{array}\right] \text{ where } \boldsymbol{f} = \boldsymbol{L}_1^{\mathrm{T}}. \quad (4)$$

The application of rem_M to the new column entries is justified by adding suitable multiples of the first row. It is not strictly necessary, but ensures that the new entries have absolute value at most $M/2$, and allows us to give tighter bounds on the entries of $\boldsymbol{L}$. The following lemma is a strengthening of [23, Lemma 7], due to the special form of bases we are considering.

Algorithm 3 The $\mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c)$ generating matrix algorithm using L^2.

Input: $\boldsymbol{a} \in \mathbb{Z}_M^{1\times n}$ and N, $c \in \mathbb{Z}_{>0}$ with $M > 2^{(c+1)/2}N^{1+1/c}$.
Output: An LLL-reduced generating matrix $\boldsymbol{S} \in \mathbb{Z}^{k\times(n+1)}$ of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ with $k \leq c$.

```
// Note $\boldsymbol{f} \in \mathbb{Z}^{k\times 1}$, $\boldsymbol{G} \in \mathbb{Z}^{k\times k}$ after each step
1. [Initialization]
   $k := 1$; $f_1 := 1$; $G_{1,1} := 1$;
2. [Iterative lattice augmentation]
   for $l := 1$ to $n$ do
      3. [Add new vector to generating matrix]
         $k := k + 1$;
         // Update $\boldsymbol{G}$, due to addition of column $\boldsymbol{g}$ to generating matrix
         $\boldsymbol{G} := \left[\begin{array}{c|c} 0 & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{G} \end{array}\right] + \boldsymbol{g}\boldsymbol{g}^{\mathrm{T}}$ where $\boldsymbol{g} = \left[\begin{array}{c} M \\ \hline \mathrm{rem}_M(a_l\boldsymbol{f}) \end{array}\right]$;
         // Update first column of generating matrix
         $\boldsymbol{f} := \left[\begin{array}{c} 0 \\ \hline \boldsymbol{f} \end{array}\right]$;
      4. [LLL reduction with removals]
         $\mathsf{L}^2\mathsf{WithRemovals}(k, \boldsymbol{G}, \boldsymbol{f}, N)$;
         If $k = 0$, return the unique element of $\mathbb{Z}^{0\times(n+1)}$.
   assert A. $\boldsymbol{L} = \left[\, \boldsymbol{f} \mid \mathrm{rem}_M(\boldsymbol{f}\boldsymbol{a}_{1..l}) \,\right]$ is an LLL-reduced
             generating matrix of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..l}}^M)$ with Gramian $\boldsymbol{G}$
          B. $k \leq c$
5. [Complete generating matrix]
   return $\boldsymbol{S} := \left[\, \boldsymbol{f} \mid \mathrm{rem}_M(\boldsymbol{f}\boldsymbol{a}) \,\right]$;
```

LEMMA 2. *At the conclusion of the following steps during Algorithm 3 the following bounds hold:*

- *Step 3:* $\max_i\|\boldsymbol{L}_i\| \leq M$ *if* $k \leq c+1$
- *Step 4:* $\max_i\|\boldsymbol{L}_i\| < M/2$ *if* $k \leq c$

PROOF. After step 4, LLL reduction with removals (with target length N) will return a basis which satisfies $\|\boldsymbol{L}_i\| \leq 2^{(k-1)/2}N$ by [23, Lemma 3]. Since $k \leq c$, it follows

$$\|\boldsymbol{L}_i\| < 2^{(c+1)/2}N^{1+1/c}/2 < M/2.$$

After step 3, $\|\boldsymbol{L}_1\| = M$ and for $i > 1$ we have $\|\boldsymbol{L}_i\|^2 < (M/2)^2 + (M/2)^2$ by the previous bound (assuming $k \leq c$ at the start of step 3) and the fact the new entries are in the symmetric range. $\square$

We are now ready to show the assertions after step 4 of Algorithm 3 hold, from which the algorithm's correctness immediately follows.

Proposition 1. *Assertion A after step 4 of Algorithm 3 holds.*

Proof. The fact $\mathcal{L}(\boldsymbol{L})$ is a sublattice of $\mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..l}})$ follows from the fact every vector in $\boldsymbol{L}$ is a linear combination of the vectors in $\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..l}}$ by (4) and LLL reduction does not change this. The fact $\mathcal{L}(\boldsymbol{L})$ contains all vectors in $\mathcal{L}_N(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..l}})$ and that $\boldsymbol{L}$ is LLL-reduced follows from the output of $\mathsf{L}^2\mathsf{WithRemovals}$.

By how $\boldsymbol{f}$ is updated in steps 3 and 4 it is clear that $\boldsymbol{f} = \boldsymbol{L}_1^{\mathrm{T}}$. However, we still must show that $\boldsymbol{L} = \left[\, \boldsymbol{f} \mid \mathrm{rem}_M(\boldsymbol{f}\boldsymbol{a}_{1..l}) \,\right]$, i.e., when $\boldsymbol{L}$ is expressed in the form from Lemma 1, $\boldsymbol{R}$ is the zero matrix. If it were not, then some entry of $\boldsymbol{L}$ would not be in the symmetric range (mod M). In which case there would be an entry $|L_{i,j}| \geq M/2$, so $\|\boldsymbol{L}_i\| \geq M/2$, in contradiction to Lemma 2.

Finally, taking the Gramian of (4) shows that step 3 ensures $\boldsymbol{G} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$, and $\mathsf{L}^2\mathsf{WithRemovals}$ also keeps $\boldsymbol{G}$ correctly updated. $\square$

The next proposition follows from the method of proof of [23, Theorem 2] taking into account that $\|\boldsymbol{L}_i^*\| \leq M$ by Lemma 2 at the start of (and therefore during) the LLL reduction when $k \leq c+1$.

Proposition 2. *Assertion B after step 4 of Algorithm 3 holds.*

Proof. If no vector was discarded during step 4 when $k = c+1$ we would have a contradiction from bounds on the volume $\sqrt{\det \boldsymbol{G}}$ of the lattice,

$$2^{c(c+1)/2} N^{c+1} < M^c \leq \sqrt{\det \boldsymbol{G}} \leq 2^{c(c+1)/4} N^{c+1}.$$

The upper bound holds for all LLL-reduced bases with final Gram–Schmidt vector shorter than N. The lower bound is derived by noting the volume increases by a factor of M each time a vector is added to the basis, and decreases by a factor of at most M each time a vector is removed from the basis. $\square$

Finally, we analyze the bit complexity of our algorithm for $c \in O(1)$ and $\|\boldsymbol{a}\|_\infty \in O(M)$. During step 3 we have $\|\boldsymbol{f}\|_\infty < M/2$ by Lemma 2, so the bitlength of numbers involved is $O(\log M)$, and thus step 3 executes in $O((\log M)^2)$ bit operations. Step 4 executes $\mathsf{L}^2\mathsf{WithRemovals}$ on a lattice of dimension at most c, with Gramian entries of bitlength $O(\log M)$, at a cost of $O((\log M)^2)$ bit operations. Since the loop runs $O(n)$ times, the total cost is $O(n(\log M)^2)$ bit operations. Step 5 requires $O(n)$ arithmetic operations, all on integers of bitlength $O(\log M)$. This gives the following result.

Theorem 1. *Algorithm 3 returns an LLL-reduced generating matrix $\boldsymbol{S} \in \mathbb{Z}^{k \times (n+1)}$ of $\mathcal{L}_N(\boldsymbol{\Lambda}^M_{\boldsymbol{a}})$ with $k \leq c$. When $c \in O(1)$ and $\|\boldsymbol{a}\|_\infty \in O(M)$, the running time is $O(n(\log M)^2)$ bit operations.*

5. LINEAR SYSTEM SOLVING

In this section let $\boldsymbol{A} \in \mathbb{Z}^{n \times n}$ be a nonsingular matrix and $\boldsymbol{b} \in \mathbb{Z}^n$ be a vector such that $\left\| \left[\, \boldsymbol{A} \mid \boldsymbol{b} \,\right] \right\|_{\max} \leq B$. Consider the problem of computing $\boldsymbol{x} \in \mathbb{Q}^n$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, using for example Dixon's algorithm [5]. This requires reconstructing the solution $\boldsymbol{x}$ from its modular image $\boldsymbol{a} = \mathrm{rem}_M(\boldsymbol{x})$, where $M = p^m$ for some prime $p \nmid \det(\boldsymbol{A})$ and $m \in \mathbb{Z}_{>0}$ is large enough that the reconstruction is unique.

We can use p-adic lifting to recover the image vector $\boldsymbol{a}$ for $m = 2, 3, \ldots$, though it is not necessary that m increase linearly. The cost of the lifting phase of the solver is directly related to the number of lifting steps m, which dictates the precision of the image. Highly optimized implementations of p-adic lifting [3, 4, 6, 9] employ an output sensitive approach to compute the vector rational reconstruction $\boldsymbol{x}$ from $\boldsymbol{a}$ in order to avoid computing more images than required. As m increases, the algorithm periodically attempts to perform a rational reconstruction of the current image vector. The attempted rational reconstruction should either return the unique minimal denominator solution or FAIL. When FAIL is returned more lifting steps are performed before another rational reconstruction is attempted.

Suppose $(d, \boldsymbol{n}) \in (\mathbb{Z}, \mathbb{Z}^n)$ is such that $\boldsymbol{a} = \mathrm{rem}_M(\boldsymbol{n}/d)$, that is, $\boldsymbol{A}\boldsymbol{n} \equiv d\boldsymbol{b} \pmod{M}$. To check if $\boldsymbol{A}\boldsymbol{n} = d\boldsymbol{b}$, that is, if $\boldsymbol{n}/d$ is the actual solution of the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, we could directly check if $\boldsymbol{A}\boldsymbol{n} = d\boldsymbol{b}$ by performing a matrix vector product and scalar vector product. However, this direct check is too expensive. The following idea of Cabay [1] can be used to avoid the direct check, requiring us to only check some magnitude bounds.

Lemma 3. *If $\|\boldsymbol{n}\|_\infty < M/(2nB)$, $0 < |d| < M/(2B)$, and $\boldsymbol{A}\boldsymbol{n} \equiv d\boldsymbol{b} \pmod{M}$ then $\boldsymbol{x} = \boldsymbol{n}/d$ solves $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.*

Proof. Note that $\|\boldsymbol{A}\boldsymbol{n}\|_\infty \leq nB\|\boldsymbol{n}\|_\infty$ and $\|d\boldsymbol{b}\|_\infty \leq B|d|$, so by the given bounds $\|\boldsymbol{A}\boldsymbol{n}\|_\infty < M/2$ and $\|d\boldsymbol{b}\|_\infty < M/2$. Every integer absolutely bounded by $M/2$ falls into a distinct congruence class modulo M, so since the components of $\boldsymbol{A}\boldsymbol{n}$ and $d\boldsymbol{b}$ are in this range and componentwise they share the same congruence classes, $\boldsymbol{A}\boldsymbol{n} = d\boldsymbol{b}$, and the result follows. $\square$

Algorithm 4 shows how Lemma 3 can be combined with the elementwise rational reconstruction approach to get an output sensitive algorithm for the reconstruction of $\boldsymbol{x}$ from its image $\boldsymbol{a}$. Let $\mathsf{RatRecon}(a, M, N)$ be a function which returns the minimal d which solves (1), or FAIL if no solution exists.

The algorithm does not take the target length N as a parameter, but calculates an N such that there will be at most one lowest-terms reconstruction. Lemma 3 may be used following step 4 to guarantee the output in step 5 will be the unique solution vector $\boldsymbol{x}$.

Note that the elementwise approach requires us to choose N to satisfy $M > 2N^2$. If β is the maximum of the magnitudes of the denominator and numerators of the actual solution vector of the system then we need $N \geq \beta$ for the algorithm to succeed, i.e., $M \in \Omega(\beta^2)$. By Hadamard's bound and Cramer's rule we have the a priori bound $\beta \leq n^{n/2}B^n$, but in general this bound is pessimistic and to avoid needless lifting we employ a output sensitive approach, as in [22].

Algorithm 5 shows how Lemma 3 can be combined with $\mathsf{VecRecon}$ instead to get an output sensitive algorithm for the reconstruction. For this algorithm we need only $M >$

Algorithm 4 An output sensitive $\mathsf{LinSolRecon}(n, \boldsymbol{a}, M, B)$ using scalar reconstruction.

Input: The image $\boldsymbol{a} \in \mathbb{Z}_M^n$ of the solution of the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, and $B \in \mathbb{Z}_{>0}$, an upper bound on the magnitude of the entries of $\boldsymbol{A}$ and $\boldsymbol{b}$.
Output: Either the solution $\boldsymbol{x} \in \mathbb{Q}^n$ or FAIL.

```
  // Need $M > 2N^2$ and $M > 2nBN$.

  1. [Set an acceptable size bound]
     $N := \lceil \min(\sqrt{M/2}, M/(2nB)) \rceil - 1$;

  2. [Simultaneous rational reconstruction]
     $d := 1$;
     for $i := 1$ to $n$ do

       3. [Entrywise rational reconstruction]
          $d := d \cdot \mathsf{RatRecon}(\mathrm{rem}_M(da_i), M, N)$;
          If RatRecon returns FAIL then return FAIL.

  4. [Check reconstruction]
     If $|d| > N$ or $\|\mathrm{rem}_M(d\boldsymbol{a})\|_\infty > N$ then return FAIL.

  5. [Return solution]
     return $\mathrm{rem}_M(d\boldsymbol{a})/d$;
```

$2^{(c+1)/2}N^{1+1/c}$ to satisfy the precondition of VecRecon. On the one hand, Lemma 4 shows that Algorithm 5 will never return an incorrect answer. On the other hand, Lemma 5 shows that the algorithm will succeed for $M \in \Omega((\sqrt{n}\beta)^{1+1/c})$.

Algorithm 5 An output sensitive $\mathsf{LinSolRecon}(n, \boldsymbol{a}, M, B, c)$ using vector reconstruction.

Input: The image $\boldsymbol{a} \in \mathbb{Z}_M^n$ of the solution of the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, and $B \in \mathbb{Z}_{>0}$, an upper bound on the magnitude of the entries of $\boldsymbol{A}$ and $\boldsymbol{b}$. Also, a parameter $c \in \mathbb{Z}_{>0}$ controlling the maximum lattice dimension to use in VecRecon.
Output: Either the solution $\boldsymbol{x} \in \mathbb{Q}^n$ or FAIL.

```
  // Need $M > 2^{(c+1)/2}N^{1+1/c}$ and $M > 2^{(c+1)/2}nBN$.

  1. [Set an acceptable size bound]
     $N := \lceil \min(M^{c/(c+1)}/2^{c/2}, M/(2^{(c+1)/2}nB)) \rceil - 1$;

  2. [Vector rational reconstruction]
     $\boldsymbol{S} := \mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c) \in \mathbb{Z}^{k \times (n+1)}$;
     If $k = 0$ then return FAIL.

assert $k = 1$

  3. [Return solution]
     return $\boldsymbol{S}_{2..n+1}/S_1$;
```

LEMMA 4. *If Algorithm 5 does not return FAIL then the output when run on $\boldsymbol{a} = \mathrm{rem}_M(\boldsymbol{x})$ is the correct solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$.*

PROOF. First, note that every entry of $\boldsymbol{S}$ is absolutely bounded by $M/(2nB)$:

$$\begin{aligned}
\|\boldsymbol{S}\|_{\max} &\le \max_i \|\boldsymbol{S}_i\| && \text{(Norm comparison)}\\
&\le 2^{(k-1)/2}\|\boldsymbol{S}_k^*\| && \text{(Proposition (1.7) in [14])}\\
&\le 2^{(c-1)/2}N && \text{(Output of VecRecon)}\\
&< M/(2nB) && (M > 2^{(c+1)/2}nBN)
\end{aligned}$$

Then we can apply Lemma 3 on any row i of $\boldsymbol{S}$, since $\boldsymbol{A}(\boldsymbol{S}_{i,2..n+1})^{\mathrm{T}} \equiv S_{i,1}\boldsymbol{b} \pmod{M}$ by construction of $\boldsymbol{S}$. Therefore every row of $\boldsymbol{S}$ yields a solution $\boldsymbol{x} = \boldsymbol{S}_{i,2..n+1}/S_{i,1}$, but since there is only one solution and the rows of $\boldsymbol{S}$ are linearly independent, $\boldsymbol{S}$ can have at most one row. Assuming the algorithm did not return FAIL, we have $\boldsymbol{x} = \boldsymbol{S}_{2..n+1}/S_1$, as required. $\square$

n	B	Alg. 4	Alg. 5 $c=2$	Alg. 5 $c=3$	Alg. 5 $c=4$	Alg. 5 $c=5$
200	1	1061	800	712	668	642
400	1	2398	1803	1604	1504	1444
800	1	5349	4017	3571	3349	3215
1600	1	11806	8860	7876	7385	7090
Alg. 5/Alg. 4			$\approx 75\%$	$\approx 67\%$	$\approx 63\%$	$\approx 60\%$

Table 1: The value of $\log M$ required to guarantee Algorithms 4 and 5 return a solution.

LEMMA 5. *Let d, $\boldsymbol{n}$ be the minimal denominator and numerators of the system's unique rational solution vector $\boldsymbol{x}$, and suppose $|d|$, $\|\boldsymbol{n}\|_\infty \le \beta$. If $M \ge 2^{(c+1)/2}(\sqrt{n+1}\beta)^{1+1/c}$ then Algorithm 5 run on $\boldsymbol{a} = \mathrm{rem}_M(\boldsymbol{x})$ will return $\boldsymbol{x}$, not FAIL.*

PROOF. From the given bounds, $\left\|\left[\, d \mid \boldsymbol{n} \,\right]\right\|^2 \le (n+1)\,\beta^2$, so when $\sqrt{n+1}\beta \le N$ we have that $\left[\, d \mid \boldsymbol{n} \,\right] \in \mathcal{L}_N(\boldsymbol{\Lambda}_a^M)$, which is guaranteed to be in the lattice $\mathcal{L}(\boldsymbol{S})$ found by VecRecon in step 2.

It is straightforward to check that if M satisfies the given bound then in fact after step 1 we have $N \ge \sqrt{n+1}\beta$, so the algorithm will not return FAIL in this case. By Lemma 4 the output will be the correct solution $\boldsymbol{x}$. $\square$

The running time of Algorithm 5 is simply that of VecRecon, which by Theorem 1 is $O(n(\log M)^2)$ bit operations. Table 1 shows the reduction in required bitlength of $\log M$ by comparing some minimal bounds on $\log M$ for Algorithms 4 and 5 to succeed.

6. FUTURE WORK

The recent article [19] gives an LLL-like algorithm for lattice basis reduction which is quasi-linear in the bitlength M of the input basis entries, that is, for a lattice with vectors of constant dimension it runs in $O((\log M)^{1+\epsilon})$ bit operations. It would be interesting to see if this quasi-linear algorithm could be employed with VecRecon.

The VecRecon algorithm, based on the L^2WithRemovals algorithm of [23], requires that $M \in \Omega(N^{1+1/c})$ for some positive integer c. But according to Lemma 3, when the input $\boldsymbol{a}$ to the vector reconstruction problem is the image of a rational linear system of dimension n with size of entries bounded by B, we only need $M > 2nBN$ to ensure that at most one linearly independent reconstruction $d\boldsymbol{a} \equiv \boldsymbol{n}$

(mod M) with $|d|$, $\|\boldsymbol{n}\|_\infty \leq N$ exists. This raises the following open question: is there a fast algorithm to find d, if it exists, that only requires $M \in O(nBN)$?

7. REFERENCES

[1] S. Cabay. Exact solution of linear systems. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, pages 248–253, 1971.

[2] Z. Chen. A BLAS based C library for exact linear algebra on integer matrices. Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2005.

[3] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '05*, pages 92–99. ACM Press, New York, 2005.

[4] W. Cook and D. E. Steffy. Solving very sparse rational systems of equations. *ACM Trans. Math. Softw.*, 37:39:1–39:21, February 2011.

[5] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40:137–141, 1982.

[6] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. LinBox: A generic library for exact linear algebra. In A. J. Cohen and N. Gao, X.-S. andl Takayama, editors, *Proc. First Internat. Congress Math. Software ICMS 2002, Beijing, China*, pages 40–50, Singapore, 2002. World Scientific.

[7] M. J. Encarnación. Computing gcds of polynomials over algebraic number fields. *J. Symb. Comput.*, 20:299–313, September 1995.

[8] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2 edition, 2003.

[9] P. Giorgi. *Arithmetic and algorithmic in exact linear algebra for the LinBox library*. PhD thesis, École normale supérieure de Lyon, LIP, Lyon, France, December 2004.

[10] G. Hanrot. LLL: a tool for effective diophantine approximation. In *Conference in honour of the 25th birthday of the LLL algorithm - LLL+25*, Caen France, 2007.

[11] E. Kaltofen and H. Rolletschek. Computing greatest common divisors and factorizations in quadratic number fields. *Math. Comput.*, 53(188), 1989.

[12] E. Kaltofen and Z. Yang. On exact and approximate interpolation of sparse rational functions. In J. P. May, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '07*, pages 203–210. ACM Press, New York, 2007.

[13] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal of Computing*, 14(1):196–209, 1985.

[14] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[15] D. Lichtblau. Half-GCD and fast rational recovery. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '05*, pages 231–236. ACM Press, New York, 2005. Extended version available at `http://library.wolfram.com/infocenter/Conferences/7534/HGCD_and_planar_lattices.pdf`.

[16] M. Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In J. Gutierrez, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '04*, pages 243–249. ACM Press, New York, 2004.

[17] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[18] A. Novocin. *Factoring Univariate Polynomials over the Rationals*. PhD thesis, Florida State University, 2008.

[19] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. *STOC 2011: 43rd ACM Symposium on Theory of Computing*, to appear.

[20] Z. Olesh and A. Storjohann. The vector rational function reconstruction problem. In I. Kotsireas and E. Zima, editors, *Proc. of the Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov (WWCA-2006)*, pages 137–149. World Scientific, 2006.

[21] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2 edition, 2005.

[22] D. E. Steffy. Exact solutions to linear systems of equations using output sensitive lifting. *SIGSAM Bull.*, 44:160–182, January 2011.

[23] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. In A. López-Ortiz, editor, *LATIN 2010: Theoretical Informatics*, volume 6034 of *Lecture Notes in Computer Science*, pages 539–553. Springer Berlin / Heidelberg, 2010.