Factorization Properties of Lattices over the Integers

by

A. Paz and M. Lempel

Technical Report #676
June 1991
Factorization Properties of Lattices over the Integers *

Azaria Paz
and
Mody Lempel
June 26, 1991

Abstract

Let $A$ be a nonsingular $n \times n$ matrix over the integers $L = L(A)$ denotes the lattice whose elements are combinations with integer coefficients of the rows of $A$. $L$ is cyclic if it can be defined in the modular form $L = \{ x = (x_i) : \sum a_i x_i \equiv 0 \pmod{d} \}$ where the $a_i$'s and $d$ are integers and $0 \leq a_i < d$.

Let $L, L_1, L_2(B)$ be lattices over the integers $L = L_1 L_2$ is a factorization of $L$ if every element of $L$ is a combination of the rows of $B$ such that the vector of combination coefficients is in $L_1$, and $B$ is a nonsingular $n \times n$ matrix.

The following results are proved in this paper: Every lattice can be expressed as a product of cyclic factors in polynomial time; Every cyclic lattice can be factored into 'simple' (term explained in the text) factors in polynomial time; Every simple lattice can be factored into 'prime' factors in polynomial time if a prime factorization of the determinant of its basis is given. In addition we provide polynomial algorithms for the following problems: Transform a cyclic lattice given by a basis into a modular form and vice versa; Find a basis of a finite modular lattice, given in modular form.

1 Introduction and Motivation

A lattice in $\mathbb{R}^n$ the Euclidean $m$-space over the integers, can be defined as a set of $n$-dimensional vectors with integer coordinates which is closed under addition and subtraction.

Such lattices have a rich structure and have many applications. They have been studied in connection with the general subject called "Geometry of Numbers" [5],[7]. In the past 10 years or...
so, some algorithmic aspects of lattices have been investigated in connection with various important problems, e.g. factoring polynomials with rational coefficients [9], Integer programming [10] finding the distance of codes [4],[11], [12] etc.

A basis for a lattice is defined to be a set of linearly independent elements of the lattice such that every element of the lattice is a linear combination, with integer coefficients, of those elements.

Usually a lattice is defined by a matrix with integer entries, whose rows form its basis.

Reduced bases have been defined in several ways, e.g. a base is reduced if the product of the lengths of its vectors is minimized, and base reduction algorithms have been introduced [9]. Another important algorithmic problem considered in the literature is the problem of finding the shortest vector of a lattice [6],[8].

A lattice in $n$-space is of full rank if the linear space generated by its elements is of dimension $n$. It has been shown by Paz and Schnorr [13] that, in certain cases lattices of full rank can be represented in the form $L = \{ x = (x_i) : \sum a_i x_i \equiv 0 \pmod{d}, x \in \mathbb{Z}^n \}$ where the $a_i$ and $d$ are integers, $a_i < d$ and $\gcd(a_1, ..., a_n, d) = 1$. Such a representation reduces the number of parameters needed for the representation of the lattice from $n^2$ parameters, required for specifying a basis, to $n + 1$. Lattices which can be represented in the above form are called cyclic lattices and their representation as above will be called a modular representation.

Let $L, L_1, L_2$ be lattices (over the integers). $L_1 L_2$ is a factorization of $L$ (notation $L = L_1 L_2$) iff

$L = \{ w : w = w_1 A, w_1 \in L_1, \text{ The rows of } A \text{ are a basis for } L_2 \}$.

Three polynomial factorization algorithms are given in this paper. The first algorithm factors a lattice, given by a basis $B$, into cyclic factors. The factor lattices produced by the algorithm are provided explicitly in both forms, by a basis and by a vector $a = (a_i)$ of coefficients defining the corresponding modular equation $\sum a_i x_i \equiv 0 \pmod{d}$.

The second algorithm factors a cyclic lattice, given in modular form, into simple and cyclic lattices. A simple lattice is a lattice which can be defined by a basis matrix which is equal to the unit matrix except for one column whose diagonal element is equal to a divisor $d_1$ of $d$ (the modulus of the modular equation defining the lattice at input) and whose off-diagonal elements are nonnegative and less than $d_1$. The factor lattices produced by this algorithm are also provided in both forms, by a basis- matrix and by a vector of coefficients.

Both algorithms can be applied in sequence resulting in the factorization of a given lattice into cyclic and simple lattices. If $B$ is a basis of the given lattice and $B_i$ are bases for its factors, then
the determinant of \( B \), in absolute value, is equal to the product of the determinants of the \( B_i \)'s in absolute value - as should be expected. On the other hand, the algorithms do not require, or provide, a factorization of \( |B| \) into prime factors.

The first algorithm can be used in order to get a modular representation for a cyclic lattice when it is given by a basis. The second algorithm provides a basis for a cyclic lattice when given in a modular form. The two algorithms can be used therefore in order to transform one form of representing lattices into the other (by a basis or by a modular equation).

Assume that \( B \) is a simple basis for a lattice \( L \), and assume that a factorization of \( |B| = d \) into prime factors is given. A third factorization algorithm is shown in section 8, factoring \( L \) into simple factor lattices such that if \( B_i \) is a basis for \( L_i \) then \( |B_i| \) is prime.

This algorithm requires the factorization of \( d \) into prime factors. If such a factorization is provided for \( d = |B| \) and \( B \) is a basis for a general lattice \( L \), then, by applying all three algorithms in sequence, we can factor \( L \) into a sequence of factor lattices \( L_i \) such that each \( L_i \) is cyclic, simple and the determinant of its basis is equal to a prime factor of \( d \).

A modular lattice is a finite lattice \( L_d \) defined as below

\[
L_d = \{z = (x_i) : \sum a_i x_i \equiv 0 \pmod{d}; 0 \leq a_i, x_i < d; a_i, x_i, d \in \mathbb{Z}\}
\]

A modular basis for \( L_d \) is a set of elements in \( L_d \) such that every element in \( L_d \) is a modular combination of its elements and the coefficients of combinations \( c_i \) satisfy \( 0 \leq c_i < d \).

In the last section of the paper we show how to use the second algorithm in order to find a modular basis for any given modular lattice, except for two degenerate cases (\( a = (a_i) \) has a single nonzero element or has exactly two such elements and one is the modular negative of the other).

The above algorithms provide a factorization into simple factors of any given lattice, of full rank. It is hoped that such a factorization will provide some new techniques for dealing with the problems mentioned at the beginning of this introduction (basis reduction, finding short vectors etc.).

2 Preliminaries

Lattices considered in this paper are sets of vectors with integer coordinates in \( n \)-dimensional Euclidean space, closed under addition and subtraction. A lattice is of full rank if the linear space generated by its elements is of dimension \( n \). A basis for such a lattice is a set of \( n \) vectors which belong to the lattice and such that every vector in the lattice is a combination, with integer
coefficients, of its elements. If a lattice $L$ of full rank is given by a basis then the vectors forming the basis will be given as rows of an $n \times n$ nonsingular matrix $B$ and the lattice will be denoted by $L(B)$.

Let $L_1$ and $L_2$ be lattices. $L_1$ refines $L_2$ if $L_2 \subseteq L_1$. A lattice $L$ of full rank will be called cyclic if it can be defined in the form $L(a_1, ..., a_n, d) = \{x = (x_1, ..., x_n) : \sum_{i=1}^{n} a_i x_i \equiv 0 \text{ (mod } d), x \in \mathbb{Z}^n \}$ where $a_1, ..., a_n$ and $d$ are nonnegative integers, $a_i < d$ and $\gcd(a_1, ..., a_n, d) = 1$.

Let $L_1$ and $L_2$ be lattices of full rank and let $B_1$ and $B_2$ be corresponding bases. $L_1$ is a right factor of $L_2$ if $CB_1 = B_2$ for some matrix $C$ with integer entries.

Trivially $L_1$ is a right factor of $L_2$ if and only if $L_1$ refines $L_2$.

A matrix will be called unimodular if it has integer entries and its determinant is equal to $\pm 1$.

Trivially, the inverse of a unimodular matrix is unimodular.

A lattice may have many bases but, as is easy to prove and well known, $B_1$ and $B_2$ are bases for the same lattice iff there exists a unimodular matrix $U$ such that $B_1 = UB_2$. It follows that the determinant of all bases of a given lattice are equal one to the other. Thus, the determinant of a basis of a lattice $L$ is an invariant of $L$.

In the sequel we shall consider lattices of full rank unless otherwise specified.

3 A factorization algorithm

Let $L$ be a lattice given by a basis $B$. Our first goal is to provide an algorithm for factoring $L$ into a sequence of cyclic lattices $L_1, ..., L_k$. For each $i, 1 \leq i \leq k$ the algorithm will provide a sequence of integers $a_{i1}, ..., a_{in}, d_i$ and a basis $B_i$ such that $L_i = L(a_{i1}, ..., a_{in}, d_i) = L(B_i)$ and such that $|d| = |d_1 \cdots d_k|$ where $d$ is the determinant of $B = B_k B_{k-1} \cdots B_1$.

Notice that if $|B| = \pm 1$ and $B$ has integer entries then $B^{-1}$ is also a matrix with integer entries implying that $B^{-1}B = I$ is a basis for the same lattice. Such a lattice must therefore coincide with the lattice of all vectors with integer coordinates. This lattice will be called the natural lattice.

The natural lattice is cyclic since it can be defined in the form

$$\mathbb{Z}^n = \{(x_1, ..., x_n) : \sum a_i x_i \equiv 0 \text{ (mod } 1)\}$$

where $(a_1, ..., a_n)$ is any nonzero vector with integer entries.

The factorization algorithm is described below.
Algorithm CF (Cyclic Factorization)

1. Given \( B \) find \( d := |B| \), set \( d := |d| ; U := B \);

2. If \( d = 1 \) return \((a_1, ..., a_n) = (1, ..., 1)\), \( d := 1 \), \( B = I \) \{\( L(B) = L(I) = L(1, 1, ..., 1) \), \( L(B) \) is cyclic \} \\
   else continue

3. Compute the matrix \( dU^{-1} \) then reduce its entries to nonnegative integers modulo \( d \). Denote 
   the resulting matrix by \( W \).

4. Let \( w^T = (w_1 \cdots w_n)^T \) be any nonzero column of \( W \). (Such a column exists, see lemma 1).
   Compute \( g = \gcd(w_1, ..., w_n, d) \); set \( i := 1 \);

5. While \( g > 1 \) do
   
   \begin{enumerate}
   
   \item Reset \((w'_1, ..., w'_n, d') := \frac{1}{d}(w_1, ..., w_2, d)\)
   
   \item Find a unimodular matrix \( R' \) such that
   
   \[
   \begin{bmatrix}
   w'_1 \\
   \vdots \\
   w'_n \\
   d'
   \end{bmatrix} = \begin{bmatrix}
   0 \\
   \vdots \\
   0 \\
   1
   \end{bmatrix}
   \]
   
   \{such that a matrix can be found, see appendix\}
   
   \item Denote by \( R \) the matrix derived from \( R' \) by removing its last row and its last column
   
   \item Output \((a_{i1}, ..., a_{in}, d_i) := (w'_1, ..., w'_n, d'), B_i = E \); \{\( L_i = L_i(B_i) = L_i(a_{i1}, ..., a_{in}, d_i)\)\}
   
   \item Find a matrix \( V \) with integer entries that \( VR = U \);
   
   \item Reset \( W := \frac{1}{d}RW(\mod g) \); \( U := V \); \( d := g \); \( i := i + 1 \);
   
   \item Let \( w^T = (w_1 \cdots w_n)^T \) be any nonzero column of \( w \). Compute \( g := \gcd(w_1, ..., w_n, d) \)

   \end{enumerate}

5.7 Let \( w^T = (w_1 \cdots w_n)^T \) be any nonzero column of \( w \). Compute \( g := \gcd(w_1, ..., w_n, d) \)

5.8 While \( g > 1 \) do

   \begin{enumerate}
   
   \item Reset \((w'_1, ..., w'_n, d') := \frac{1}{d}(w_1, ..., w_2, d)\)
   
   \item Find a unimodular matrix \( R' \) such that
   
   \[
   \begin{bmatrix}
   w'_1 \\
   \vdots \\
   w'_n \\
   d'
   \end{bmatrix} = \begin{bmatrix}
   0 \\
   \vdots \\
   0 \\
   1
   \end{bmatrix}
   \]
   
   \{such that a matrix can be found, see appendix\}
   
   \item Denote by \( R \) the matrix derived from \( R' \) by removing its last row and its last column
   
   \item Output \((a_{i1}, ..., a_{in}, d_i) := (w'_1, ..., w'_n, d'), B_i = E \); \{\( L_i = L_i(B_i) = L_i(a_{i1}, ..., a_{in}, d_i)\)\}
   
   \item Find a matrix \( V \) with integer entries that \( VR = U \);
   
   \item Reset \( W := \frac{1}{d}RW(\mod g) \); \( U := V \); \( d := g \); \( i := i + 1 \);
   
   \item Let \( w^T = (w_1 \cdots w_n)^T \) be any nonzero column of \( w \). Compute \( g := \gcd(w_1, ..., w_n, d) \)

   \end{enumerate}

end of algorithm.
4 Proof of Correctness and Complexity

Lemma 1 The matrix $W$ as defined in step 3 contains at least one nonzero column and all its entries are integers.

Proof: By Cramer's rule every entry in $U^{-1}$ has the form $z/d$. Given that the absolute value of the determinant of $U$ is $|d| > 1$, the absolute value of the determinant of $U^{-1}$ is less than 1 implying that at least one entry in $U^{-1}$ is not an integer. Assume this entry to be $a/b$. In $W$ this entry changes into $d \cdot (\text{mod } |d|)$. If in $W$ this corresponding entry is equal to zero then $d \cdot k = 0$ for some nonzero integer $k$. But this implies that $a/b$ is an integer contrary to our assumption. □

Lemma 1 justifies step 4 of the algorithm.

Lemma 2 The lattice $L(w_1, ..., w_n, d)$ refines the lattice $L(B)$ where $B$, and $w_1 \cdots w_n$ are as defined in step 1 to 4 of the algorithm.

Proof: Assume that the $j$-th column of $W$ is a nonzero column and let $y^T$ be the $j$-th column of $dU^{-1}$. Then $Uy^T = d\cdot e^j$ where $e^j$ is the vector whose $j$-th entry is equal to one and all other entries equal zero. Let $y = (y_1 \cdots y_n)$ and set $w_j = y_i \cdot (\text{mod } d)$. Then $w^T = (w_1 \cdots w_n)^T$ is the $j$-th column of $W$ and we have that $Uw^T = dk^T$ where $k = (k_i)$ is a vector of integers. As $U = B = [b_{ij}]$ the above equality implies that $\sum_j b_{ij}w_j = k_i d$, $1 \leq i \leq n$, so that the elements of the basis of $L(B)$ represented by the rows of $B$ and any combination with integer coefficients of those elements is in $L(w_1, ..., w_n, d)$.

Lemma 3 Let $R$ be the matrix defined in step 5.3 of the algorithm. Then

$L(R) = L(w'_1, ..., w'_n, d')$

Proof: Let $(b_1, ..., b_{n+1})$ be the $i$-th row, $i \leq n + 1$ of $R'$. Then, by the definition of $R'$, we have that $\sum_{j=1}^{n} b_{ij}w_j + b_{n+1}d = 0$ implying that $(b_1, ..., b_n)$, the $i$-th row of $R$, is in $L(w'_1, ..., w'_n, d')$. To prove the other direction notice first that $R'$ is invertible over the integers by its definition, and that the last column of $(R')^{-1}$ must be equal to $(w'_1 \cdots w'_n, d')^T$, again by definition (see step 5.2). Let $(c_1 \cdots c_n) = c$ be any vector in $L(w'_1, ..., w'_n, d')$. Then $c$ satisfies $\sum_{j=1}^{n} c_{j}w'_j + kd' = 0$ for some integer $k$. Therefore $(c_1 \cdots c_n,k)(R')^{-1} = (m_1 \cdots m_n,0)$ where $k$ and the $m_i$'s are integers, since the entries of $(R')^{-1}$ are integers and its last column is $(w'_1 \cdots w'_n, d')$. It follows that $(c_1 \cdots c_n,k) = (m_1 \cdots m_n,0)R'$ and the vector $c$ is shown to be a combination, with
integer coefficients, of the rows of $R$ (the coefficient of the last row of $R'$ in the combination is zero and the last column of $R'$ is deleted). \(\Box\)

**Remark:** Finding a matrix $R'$ as required in step 5.2 can be done via any extended Euclidean algorithm of Blankinskip type (see appendix), given that $\gcd(w_1', ..., w_n', d') = 1$, as is the case after step 5.1.

**Lemma 4** Let $R$ be the matrix defined in step 5.3 of the algorithm. Then $|R| = d' = d/g$.

**Proof:** By definition

$$R' = \begin{bmatrix} R & a^T \\ b & c \end{bmatrix}$$

where $a^T$ and $b$ are vectors of integers and $c$ is an integer.

Consider the equation $R'x = (0 \cdots 0 1)^T$.

By Cramer's rule we have that

$$x_{n+1} = \begin{bmatrix} 0 \\ \vdots \\ R \\ \vdots \\ 0 \\ b & 1 \end{bmatrix}^{+/|R|} = |R|$$

since $|R'| = 1$.

By definition (step 5.2)

$$R' \begin{bmatrix} w_1' \\ \vdots \\ w_n' \\ d' \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

so that $(x_1 \cdots x_{n+1})^T = (w_1' \cdots w_n' d')$ satisfies the equation $R'x = (0 \cdots 0 1)^T$. It follows that $x_{n+1}$ must be equal to $d'$ and it must be equal to $|R|$ (by Cramer's rule), so that $|R| = d'$. \(\Box\)

**Lemma 5** Let $V$ be the matrix defined in step 5.5 of the algorithm. Such a matrix can be found and it has the properties that $gV^{-1}(\mod g) = (\frac{1}{d}RW)(\mod g)$, where $W$ is defined in step 3 and $R$ is defined in step 5.3, and it has integer entries.

**Proof:** $U = B$ (step 1) and $VR = U$. From $|R| = d'$, $|U| = |B| = d$ we get that $|V| = d/d' = g$.

Trivially $L(w_1, ..., w_n, d) = L(w_1', ..., w_n', d')$, by the definitions and by the fact that $(w_1', ..., w_n', d') = \frac{1}{d}(w_1, ..., w_n, d)$ where $g = \gcd(w_1, ..., w_n, d)$. From lemmas 2 and 3 we get therefore that
\[ L(R) = L(w'_1, \ldots, w'_n, d') \text{ refines } L(U) = L(B). \] It follows that \( R \) is a right factor of \( U \) so that a matrix \( V \) with integer entries such that \( VR = U \) can be found.

Now, since \( RU^{-1} = V^{-1} \) it follows that \( \frac{1}{d} RdU^{-1} = gV^{-1} \). Therefore \( gV^{-1}(\text{mod } g) = (\frac{1}{d} RdU^{-1}) \) (mod \( g \)) = (\frac{1}{d} RW + gRE)(\text{mod } g) = (\frac{1}{d} RW)(\text{mod } g), \) where \( E \) is a matrix with integer entries (by the definition of \( W \)), and \( d = d'g \).

**Remark:** Since \( gV^{-1} \) is a matrix with integer entries, it follows that \( \frac{1}{d} RW \) has the same property and, by lemma 1, \( (\frac{1}{d} RW) \) (mod \( g \)) must have a nonzero column.

The correctness of the algorithm follows from the above lemmas. At iteration \( i \) of the while loop the lattices \( L(U) \) is factored into a cyclic right factor \( L(R) = L(a_{i1}, \ldots, a_{in}, d_i) \) and a left factor \( L(V) \), which is the \( L(U) \) of the next iteration, and \( |V| \) divides \( |U| \).

After at most \( \log_2 |B| \) iterations the algorithm halts with a cyclic leftmost factor (step 6).

All through the computation the algorithm produces matrices and vectors with integer entries.

**Lemma 6** The number of arithmetical operations involved in the \( CF \) algorithm is \( O((n^3 + \log d) \log d) \) and the magnitude of the intermediate results is \( O(\max(n^2M^3n, nd^n)) \), where \( M \) is the maximal entry in absolute value, of the matrix \( B \) at input.

**Proof:** Finding the determinant of \( B \) (step 1) and computing \( dU^{-1} \) (step 3) can be done in \( O(n^3) \) operations with intermediate results bounded by \( n^2M^2n \), see the algorithm of Edmonds [3].

The entries of the vector whose \( \gcd \) is to be found at step 4 are bounded by \( d \). The number of operations involved in the finding of the \( \gcd \) is therefore \( O(\log d + n) \) with the intermediate results bounded by \( d \), see Bradley [2].

The number of iterations of the while loop 5 is bounded by \( \log d \).

- Step 5.1 is \( O(n) \)
- Step 5.2 is \( O(n^3 + \log d) \) with intermediate results bounded by \( nd^n \), see appendix.
- Steps 5.5, 5.6 and 5.7 are \( O(n^3) \) with intermediate results bounded by \( nd^3 \). □

**Remark:** The vectors \( (a_{i1}, \ldots, a_{in}, d_i) \) output at step 6, have nonnegative entries and \( a_{ij} < d_i \) for all \( i \) and \( j \).

5 A secondary factorization algorithm

A matrix over the integers whose determinant is equal to \( d \) will be called \( d \)-simple if it is a unit matrix except for one column whose diagonal element equals \( d \) and the off-diagonal elements, of
that column, are all nonnegative and less than $d$.

We provide now an algorithm for factoring any given cyclic lattice $L(a_1, \ldots, a_n, d)$ into a sequence of cyclic lattices $L_1, \ldots, L_k$ such that the lattice $L_i$ in this sequence is represented by a $d$-simple matrix whose rows are its basis, and $d = d_k d_{k-1} \cdots d_1$.

We precede the algorithm by a simple procedure.

**Definition:** Given two integers $a, b > 0$ the greatest **uncommon** divisor of $a$ with regard to $b$ (notation $gud(a, b)$) is the greatest integer $a_1$ such that $a_1$ divides $a$ and such that $\gcd(a_1, b) = 1$.

Notice that this definition is not symmetric with regard to $a$ and $b$.

To find the $gud$ of $a$ with regard to $b$ we can use the following procedure

Procedure $gud(a, b)$

1. Compute $g := \gcd(a, b)$;
2. While $g > 1$ do begin
   
   $a := a/g$;
   
   $g := \gcd(a, b)$

end;

3. output $a$

**Remark:** The complexity of finding the $\gcd$ is logarithmic in the magnitude of the numbers involved and the number of iterations of the while loop is logarithmic in $a$. The complexity of this algorithm is therefore $O(\log^2 a)$.

Notice that if $\gcd(a, b) = u$ then $\gcd(u, b) = 1$.

The main algorithm is now described.

**Algorithm SF (simple factorization).**

Input $(a_1^{(0)}, \ldots, a_n^{(0)}, d^{(0)}) := (a_1, \ldots, a_n, d)$ such that $0 \leq a_l < d$ for $1 \leq l \leq n$, some $a_l$ are positive and $1 \leq l \leq n, \gcd(a_1, \ldots, a_n, d) = 1$. (These properties hold true for the vectors output by the CF algorithm at step 6).

1. Set $i := 1, j := 1$

2. While $a_l^{(j-1)} = 0$ or $\gcd(d^{(j-1)}, a_l^{(j-1)}) = 1$ set $i := i + 1$;
3. Set \( d_j := \gcd(d^{j-1}, a^{(j-1)}_i) \), output \( d_j \)
   (now \( d_j > 1 \) but \( \gcd(d_j, a^{(j-1)}_i) = 1 \), by definition);

4. Define the \( d_j \)-simple matrix \( A^{(j)} = [s^{(j)}_{ij}] \) as the unit matrix except for its \( i \)-th column whose
diagonal element is equal to \( d_j \) and whose off diagonal elements are \( s^{(j)}_{ij} = -\frac{a_i^{(j-1)}}{a_i^{(j-1)}} \mod d_j \)
   \( 1 \leq l \leq n, l \neq i \); Output \( A^{(j)} \);

5. Reset 
   \[
   \begin{align*}
   a^{(j)}_i &:= \frac{1}{d_j} (s^{(j)}_{ii} a^{(j-1)}_i + a_i^{(j-1)}) & 1 \leq i \leq n, i \neq j \\
   a^{(j)}_i &:= a_i^{(j-1)} & (s^{(j)}_{ii} = d_j) \\
   d^{(j)} &:= d^{(j-1)} / d_j
   \end{align*}
   \]
   Output \( a^{(j)} = (a^{(j)}_1, \ldots, a^{(j)}_n) \); \( (d^{(j)} = \frac{1}{d_j} A^{(j)} a^{(j-1)}_i) \).

6. If \( d^{(j)} > 1 \) then set \( j := j + 1, i := i + 1 \) and go to 2.

7. Halt.

6 Proof of correctness

The correctness of the algorithm follows from the lemmas proven below.

The number of iterations of the algorithm is finite as will be shown in the sequel.

Lemma 7 For \( a \leq j \leq k \), where \( k \) is the number of iterations of the algorithm

(a). \( \gcd(a^{(j)}_1, \ldots, a^{(j)}_n, d^{(j)}) = 1 \)

(b). For \( j \geq 1, 1 \leq i \leq n \) if \( \gcd(d^{(j-1)}, a^{(j-1)}_i) = 1 \) then \( \gcd(d^{(j)}, a^{(j)}_i) = 1 \) and

(c). If \( a^{(j-1)}_i = 0 \) then \( a^{(j)}_i = 0 \)

Proof: Part (a) of the lemma is proved by induction. It is true for \( j = 0 \) by assumption. Assume
by induction that \( \gcd(d^{(j-1)}, d^{(j-1)}) = 1 \). If \( \gcd(a^{(j)}_i, d^{(j)}) = g > 1 \), then, by definition (see step
4) \( d_j a^{(j)}_i = s^{(j)}_{ii} a^{(j-1)}_i + a_i^{(j-1)} \) and \( a^{(j)}_i = a^{(j-1)}_i \) for some \( i \). Thus \( g \) divides \( a^{(j)}_i \) and \( a^{(j-1)}_i = a^{(j)}_i \)
implying that \( g \) divides \( a^{(j-1)}_i, 1 \leq l \leq n \). Also \( d^{(j-1)} = d^{(j)} d_j \) so that if \( g \) divides \( d^{(j)} \) then it
divides \( d^{(j-1)} \). Thus \( g \) divides all the entries of \( a^{(j-1)} \) and \( g \) divides \( d^{(j-1)} \) contradicting the
assumption that \( \gcd(a^{(j-1)}, d^{(j-1)}) = 1 \).

Part (b) of the lemma follows from the following considerations:
From the definitions of \( d_j \) and \( d^{(j)} \) (steps 3 and 5) it is clear that \( d_j \) and \( d^{(j)} \) are relatively prime
and it is easy to see that iff $g_{d(j-1), af(j-1)} = 1$ then all the prime factors of $d(j-1)$ are factors of $a(i-1)$. For the given iteration $j - 1$ let $i$ be the index for which the equalities below hold:

$$d_j a_i^{(j)} = s_{ii}^{(j)} a_i^{(j-1)} + a_i^{(j-1)}, a_i^{(j)} = a_i^{(j-1)}, l \neq i.$$  

By the definition and our assumption $d_j$ contains all the prime factors of $d(j-1)$ which are not factors of $a_i^{(j-1)}$ so that all the prime factors of $d(j) = d(j-1) d_j$ are factors of $a_i^{(j-1)}$, and are not factors of $d_j$. Also all the prime factors of $d(j)$ are factors of $a_i^{(j-1)}, l \neq i$ (since $d(j)$ divides $d(j)$ and we assumed that $g_{d(j-1), a_i^{(j-1)}} = 1$).

So all the prime factors of $d(j)$ divide the right hand side of the equation defining $d_j a_i^{(j)}$. Since the prime factors considered do not divide $d_j$, they must divide $a_i^{(j)}$ in the left hand side of the equation. It follows that $g_{d(j), a_i^{(j)}} = 1$ for $l \neq i$. If $l = i$ then $g_{d(j-1), a_i^{(j-1)}} = d_j > 1$, by our assumption, and therefore the antecedent of property (b) does not hold for $l = i$. This completes the proof of property (b).

To prove part (c) of the lemma notice that if $a_i^{(j-1)} = 0$ then $s_{ii}^{(j)} = 0$ (step 4) which implies that $a_i^{(j)} = 0$ (step 5) for $i \neq i$, where $i$ is the index satisfying $a_i^{(j)} = a_i^{(j-1)}$ at the $j$-th iteration. It follows from step 2 that $a_i^{(j-1)} \neq 0$ for $l = i$ so that the premise of property c does not hold for $l = i$. The proof of lemma 7 is now complete.

Consider now the $j$-th iteration of the algorithm. Since $gcd(a(i-1), d(i-1)) = 1$ (by property a of lemma 7), it follows that every prime factor $p$ of $d(j-1)$ does not divide some entry $a_i^{(j-1)} \neq 0$ of $d(j-1)$ implying that $g_{d(j-1), a_i^{(j-1)}} > 1$ for that particular $l$.

Moreover, the smallest index $l$ satisfying $g_{d(j-1), a_i^{(j-1)}} > 1$ is bigger than the smallest index $l'$ satisfying $g_{d(j-2), a_i^{(j-2)}} > 1$, as follows from the properties (b) and (c) of lemma 7 and from the fact that $g_{d(j-1), a_i^{(j-1)}} = 1$ (since $d(j-1) = d(j-2)/d(j-1)$, $d(j-1)$ contains all the factors of $d(j-1)$ which divide $a_i^{(j-1)}$, and $a_i^{(j-1)} = a_i^{(j-2)}$ by step 5 of the algorithm). It follows that step 2 of the algorithm provides a divisor $d_j > 1$ of $d(j-1)$, provided that $d(j-1) > 1$ so that in the end $d = d(0) = d_1 d_2 \cdots d_k$ where $k$ is the number of iterations.

It is also clear from the above considerations that the number of iterations is bounded by $n$ (the running index $i$ in step 2 grows at every iteration and $i \leq n$) and is also bounded by $log_2 d(0) (d(j)$ decreases by a factor of at least 2 at every iteration). To complete the correctness proof we must show that the lattices $L(A(j))$, where $A(j)$ are the $d_j$-simple matrices output at step 4, provide a decomposition of $L(a(0), d)$ this is done in the next lemmas.

Define the lattices $L_j$ and $C_j$ as $L_j = L(A(j), d_j), j \leq 1, C_j = L(A(j), d(j)), j \geq 0$.

Where the $d_j$'s are as defined in step 3 of the algorithm.

**Lemma 8** $L_j = L(A(j))$ where $A(j)$ is the matrix defined at step 4 of the algorithm.
Proof: Extend the matrix $A^{(i)}$ into an $(n+1) \times (n+1)$ matrix $B^{(i)}$ which satisfies the equation below

$$
\begin{bmatrix}
1 & s_{1,i} & k_1 & a_1^{(j-1)} & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
1 & s_{n-1,i} & K_{i-1} & d_j & -a_i^{(j-1)} \\
& & \ddots & \ddots & \ddots \\
& & & s_{n,i} & 1 & k_{n+1} \\
& & & & & x \\
& & & & & y
\end{bmatrix}
\begin{bmatrix}
\vdots \\
1
\end{bmatrix}
= \begin{bmatrix}
\vdots \\
0
\end{bmatrix}
$$

(\ast)

The $B^{(i)}$ matrix is constructed as follows:

The upper $n \times n$ main diagonal block of $B^{(i)}$ is the matrix $A^{(i)}$; $z$ and $y$ are two integers such that

$$
za_i^{(j-1)} + yd_j = 1; k_{n+1,i} = x, b_{n+1,n+1}^{(j)} = y.
$$

Such integers exist since, by definition

$$
gcd(a_i^{(j-1)}, d_j) = 1; \\
b_{n+1,i}^{(j)} = 0 \text{ for } l \neq i \text{ and } l \neq n + 1;
$$

$k_l = -\frac{1}{a_i^{(j-1)}} s_{i,l}^{(j)} a_i^{(j-1)}, l \neq i$. By the construction of $A^{(i)}$ we have that $a_i^{(j-1)} + s_{i,l}^{(j)} a_i^{(j-1)} \equiv 0 \pmod{d_j}$ and therefore $k_l$ is an integer.

It is easy to verify now that the matrix $B^{(i)}$ constructed as above satisfies the equation (\ast).

It is also easy to verify that the determinant of $B^{(i)}$ is equal to $yd_j + za_i^{(j-1)} = 1$ (develop the determinant by its last row) and $B^{(i)}$ is therefore a unimodular matrix, implying that $(B^{(i)})^{-1}$ is a matrix with integer entries. Moreover, since $B^{(i)}(B^{(i)})^{-1} = I$ and $B^{(i)}$ satisfies the equation (\ast), the last column of $(B^{(i)})^{-1}$ must equal $(a_i^{(j-1)} \ldots a_n^{(j-1)} d_j)^T$.

Let $v = (w_1 \ldots w_n)$ be a vector in $L_j$. Then $<w, a^{(j-1)}> = 0 \pmod{d_j}$ where $<,>$ denotes scalar multiplication of vectors. There is, therefore, an integer $k$ such that
where the \( m_i \)'s are integers. Therefore \( (w_1 \cdots w_n k) B(j) \) is equal to \( \{ m_1 \cdots m_n 0 \} B(j) \) and if the last coordinate is ignored this reduces to \( (w_1 \cdots w_n) A^{(i)} \) since only the \( n \times n \) main diagonal block of \( B(j) \) contributes to the above equation.

We have thus shown that if \( w \) is in \( L_j \) then \( w \) is in \( L(\lambda) \). The other direction is trivial since all the rows of \( A^{(i)} \) are in \( L_j \) by construction (step 4 in the algorithm).

**Definition:** Let \( L, L_1, L_2 \) be lattices over the integers. \( L_1 L_2 \) is a factorization of \( L \) (notation \( L = L_1 L_2 \)) iff \( L = \{ w : w = w_1 A, w_1 \in L_1, L_2 = L(\lambda) \} \).

**Lemma 9** For \( 0 \leq j \leq k - 1 \), where \( k \) is the number of iterations of the SF algorithm

\[
C_j = C_{j+1} L_{j+1}
\]

**Proof:** By lemma 8, \( L_{j+1} = L(\lambda^{(j+1)}) \).

Assume first that \( w \in C_j \). We show that \( w A^{(j)} \in C_j \):

\[
< w A^{(j+1)}, a^{(j)} > = < w, A^{(j+1)} d^{(j)} > = < w, d_{j+1} d^{(j+1)} > \quad \text{(by step 5 in the SF algorithm)}.
\]

The above scalar product is equal to \( d_{j+1} < w, a^{(j+1)} > \) which by our assumption is equal to \( d_{j+1} k d^{(j)} = k d^{(j)} \) for some integer \( k \). Thus \( < w A^{(j+1)}, a^{(j)} > \equiv 0 \pmod{d^{(j)}} \) as required.

Assume now that \( w A^{(j+1)} \in C_j \) i.e. \( < w A^{(j+1)}, a^{(j)} > = k d^{(j)} \) for some integer \( k \). We show that \( w \in C_{j+1} \):

\[
< w, a^{(j+1)} > = \frac{1}{d_{j+1}} < w, d_{j+1} a^{(j+1)} > = \frac{1}{d_{j+1}} < w, A^{(j+1)} a^{(j)} > \quad \text{(by step 5 in the SF algorithm)}.
\]

The above scalar product is equal to \( \frac{1}{d_{j+1}} < w A^{(j+1)}, a^{(j)} > \) which by our assumption is equal to \( \frac{1}{d_{j+1}} k d^{(j)} = k d^{(j+1)} \).

This implies that \( w \in C_{j+1} \) as required.

The values \( a_j \) computed in step 5 of the algorithm are integers. This follows from the definitions:

\[
\sum a_j^{(j)} a_i^{(j-1)} = a_i^{(j-1)} + e^{(j)} a_i^{(j-1)} = a_i^{(j-1)} + \left( \frac{d^{(j)}}{d_{j+1}} \right) a_i^{(j-1)} = a_i^{(j-1)} + k d_j = k d_j
\]

for some integer \( k_j \). As \( \gcd(a^{(j-1)}, d_j) = 1 \), \( a_j^{(j)} \) is uniquely defined : \( a_j^{(j)} = k_j \).

The correctness of the algorithm is now implied by the above two lemmas 8 and 9. After at most \( k < \log_2 d \) iterations the algorithm halts. At the \( j \)-th iteration the algorithm outputs \( d_j, a^{(j)} \) and \( \lambda^{(j)} \). We have shown that : \( d = d_1 \cdots d_k \);

\[
L = L_k L_{k-1} \cdots L_1 \quad \text{(where \( L \) is the lattice at input), as follows from lemma 3; \( L_i = L(a^{(i-1)}, d_j) = L(\lambda^{(i)}) \),}
\]

\[\text{13}\]
as follows from lemma 8; If \( A = A^{(i)} \cdots A^{(i)} \) then \( L = L(A) \); The matrices \( A^{(i)} \) are \( d_i \)-simple and the lattices \( L_i \) are cyclic. □

7 The complexity of the SF algorithm

The number of iterations of the algorithm is bounded by \( \min (\log d, n) \), as mentioned in the proof of its correctness.

Steps 2 and 3 are \( O(\log^2 d) \) as determined by the \( \text{gcd} \) operation.

Step 4 is \( O(n \log d) \) since modular division is equivalent to the \( \text{gcd} \) operation.

Step 5 is \( O(n) \).

It follows that the complexity of the algorithm is \( O(\log d (\log d + n) \min (\log d, n)) \). The size of the intermediary results is bounded by \( d^2 \) as is easy to determine by considering the various steps of the algorithm.

Remark 1: Given a lattice in the form \( L(B) \) one can apply to the matrix \( B \) the factorization algorithm \( \text{CF} \) resulting in a factorization of \( L(B) \) into cyclic factors and subsequently, by using the SF algorithm, one can factor every cyclic factor into simple factors, rendering a factorization of \( L(B) \) into simple factors \( L(A_i) \).

The total number of iterations of both algorithms, when applied in sequence, is still bounded by \( \log d \) where \( d \) is the absolute value of \(|B|\). This is implied by the fact that each time a new factor is produced, by either algorithm, the \( d \) parameter for the next iteration in the CF algorithm and the \( d^{(i)} \) parameter for the next iteration in the SF algorithm is reduced by a factor of at least 2.

Remark 2: Given a factorization of a lattice \( L(B) \) into simple factors \( L(A_k) \cdots L(A_1) \) we can find a unimodular matrix \( U \) such that \( B = U A_k \cdots A_1 \). This follows from the fact that the rows of \( B \) and the rows of \( A = A_k \cdots A_1 \) span the same lattice.

8 A third factorization algorithm

The algorithm \( \text{CF} \) receives at input a matrix \( B \) whose determinant in absolute value is equal to some integer \( d \). The factorization of \( d \) into prime factors is not required in the various steps of the algorithm. Similarly, factorization into prime factors is not required for the parameter \( d \), included in the input \( (a_1 \cdots a_n, d) \), for the SF algorithm. The simple lattices output by the algorithm \( \text{SF} \) have the form \( L_j(a_1^{(j)}, \ldots, a_n^{(j)}, d_j) \) and, for some \( i, \text{gcd}(a_i^{(j)}, d_j) = 1 \).
Assume now that all the entries in $B_{j-1}$ are less than $D_{j-1}, j = 2$.

$$B_j = \begin{bmatrix}
1 & s_{1,j}^{(j)} & \cdots \\
\vdots & \ddots & \ddots \\
& & 1 & s_{j,j}^{(j)} \\
& & & d_j \\
& & & s_{j+1,j}^{(j)} \\
\end{bmatrix}$$

Let $B_l = (b_{l1} \cdot b_{ln})$ denote a typical row of $B_{j-1}$. By assumption $b_{li} \leq D_{j-1} - 1$, for $0 \leq i \leq n$.

The $j$-th row of $B_j$ is $d_j(b_{jl} \cdot b_{jn})$ with $d_j b_{jl} < d_j D_{j-1} = D_j, 1 \leq l \leq n$.

The $l$-th row of $B_j$ with $l \neq j$, has the form

$$s_{ij}^{(j)}(b_{jl} \cdot b_{jn}) + (b_{l1} \cdot b_{ln})$$

Now $s_{ij}^{(j)} b_{jn} + b_{ln} \leq (d_j - 1)(D_j - 1) + D_{j-1} - 1 = d_j(D_j - 1) < d_j D_{j-1} = D_j$

The proof is now complete.

Since all the possible cases have been considered we can add up the previous results in the theorem below.

An n-dimensional vector with nonnegative entries $a = (a_i)$ is called degenerate with regard to an integer $d$ if $a$ is the zero vector or it has a single non zero entry $a_i < d$ or it has exactly 2 non zero entries $a_i$ and $a_j$ such that $a_i \equiv -a_j \pmod{d}$.

**Theorem:** Given a modular lattice $L_d(a)$ such that $a$ is not degenerate with regard to $d$, a basis $B$ for $L_d(a)$ can be found in polynomial time.
Appendix: A Blankinship type algorithm [1]

In this appendix we present an algorithm for the following problem: Given a vector of integers $a = (a_1 \cdots a_n)$ such that $\gcd(a_1, \cdots, a_n) = 1$, find a unimodular matrix $A$ such that $Aa^T = (0 \cdots 0)^T$. Let $M$ be the maximal entry in $a$. The algorithm has the following properties: All the entries in the first $n - 1$ columns of $A$ are bounded by $M$, the intermediary results are bounded by $nM^n$, the complexity of the algorithm is $O(n^3 + \log M)$.

Algorithm BL

1. Given $a = (a_i)$ construct the $n \times (n + 1)$ matrix $B = [b_{ij}]$ whose first column is $a^T$ and the last $n$ columns are the unit matrix $I$.

2. First iteration

   2.1. Find two integers $u_1, v_1$ such that $|u_1| < a_n, |v_1| < a_1$ and such that $u_1a_1 + v_1a_n = \gcd(a_1, a_n) = g_2$.
      Denote by $b_j$ the rows of $B$.

   2.2. Reset $b_n := u_1b_1 + v_1b_n, b_1 := a_n b_1 - a_1 b_n$
      (Since initially $b_{11} = a_1$ and $b_{n1} = a_n$ we have, after the reset, that $b_{11} = 0$ and $b_{n1} = g_2$.

The transformation is a unimodular transformation since it can be described as $B := UB$ where $U$ is a matrix whose first row is $(u_1 \cdots 0v_1)$, whose last row is $(a_n \cdots 0 - a_1)$ and all other rows are unit vectors: For $i \neq 1, n$, the $i$-th row has its $i$-th coordinate equal to one - and all other coordinates are zero. $|U| = 1$ as follows from its definition.)

{ The first and last row of $B$ before step 2.1 had the form

$\begin{align*}
    b_1 &= (a_1 0 \cdots 0),
    b_n &= (a_n 0 \cdots 0 1)
\end{align*}$

After step 2.1 the same rows are changed into

$\begin{align*}
    b_1 &= (0 \cdots a_n 0 \cdots 0 - a_1),
    b_n &= (g_2 u_1 0 \cdots 0 v_1)
\end{align*}$

with $u_1 < a_n$. We have also that $b_2 = (a_2 0 \cdots 0 1)$.

22
3. Iteration $i, 2 \leq i \leq n - 1$.

Assume that before the $i$-th iteration, $2 \leq i \leq n - 1$, the $i$-th and the last row of $B$ have the form

$$b_i = (a_i 0 \cdots 0 1 0 \cdots 0), b_n = (g_i b_{n,2} \cdots b_{n,i} 0 \cdots 0 b_{n,n+1})$$

and

$$b_{n,j} < b_{jj} \text{ for } 2 \leq j \leq i, g_i = \gcd(a_i, \ldots, a_{i-1}, a_n).$$

(\text{This condition holds for } i = 2).}

3.1 (i) Find 2 integers $u_i, v_i$ such that $|u_i| < g_i, |v_i| < a_i$ and such that $u_i a_i + v_i g_i = \gcd(a_i, g_i) = \gcd(a_1, \ldots, a_i, a_n) = g_{i+1}$

3.2 (i) Reset

$$b_n := u_i b_i + v_i b_{n-1}; b_i := g_i b_i - a_i b_n$$

(After the reset we have that $b_{i+1} = 0$ and $b_{n+1} = g_{i+1}$ as explained in step 2.2. The transformation is unimodular).

3.3 (i) For $j := i$ down to 2 reset

$$b_i := b_i - \left[ \begin{array}{c} b_{ij} \\ b_{j-1,i} \end{array} \right] b_{j-1}$$

$$b_n := b_n - \left[ \begin{array}{c} b_{nj} \\ b_{j-1,n} \end{array} \right] b_{j-1}$$

4. Output the matrix consisting of the last $n$ columns of $B$, to be denoted by $A$.

End of algorithm.

Correctness:

If $B_0$ is the $B$ matrix at input and $A'$ is the matrix representing the composition of all unimodular transformations performed during the algorithm then

$$B_0 = \begin{bmatrix} a_1 & \cdot & \cdot & \cdot \\ \cdot & I & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_n & & & \cdot \end{bmatrix} \text{ and } A' B_0 = \begin{bmatrix} 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & & & 1 \end{bmatrix}$$
as follows from the definitions and from the steps of the algorithm. Therefore,

\[
\begin{bmatrix}
    a_1 \\
    \vdots \\
    a_n
\end{bmatrix}
= \begin{bmatrix}
    0 \\
    a_1 \\
    \vdots \\
    1
\end{bmatrix}
\]

as required.

**Complexity and size of intermediary results**

Let \( M \) be the maximal integer among \( a_1 \cdots a_n \). As shown in Bradley [2] steps 2.1 and 3.1 (i),

\[
2 \leq i \leq n - 1 \quad \text{are} \quad O(nM + n) \quad \text{altogether}.
\]

The \( O(n) \) iterations in steps 2.2, 3.2 (i) are \( O(n) \) each, contributing \( O(n^2) \) operations.

The \( O(n) \) iterations in steps 3.3 (i) are \( O(n^2) \) contributing \( O(n^3) \) operations.

The total number of operations is therefore \( O(n^2) \).

In the application of this algorithm to the problem considered in the paper we shall need only columns 1 to \( n - 1 \) of the matrix \( A \) output at step 4. We shall ignore therefore the last column of \( A \) (and of \( B \)) in the discussion below. In the algorithm itself the computation of the values of the entries in the last column of \( B \) can be ignored and those values, if needed can be computed at the very end: based on the requirement that \( Aa^T = (0 \cdots 01) \), the values of the last column of \( A \) can be found easily if all the other entries of \( A \) are given.

Assume that before iteration \( i \) all the entries in the first columns of \( B \) are bounded in absolute value by \( M \) (the maximal coordinate of \( a \)). We will show that this property is restored when the \( i \)-th iteration is completed while the intermediate values of those entries during the iteration are bounded by \( nM^n \).

The rows affected by the \( i \)-th iteration are the \( i \)-th row and \( n \)-th row.

Before the iteration those rows have the form:

\[
b_i = (a_i0\cdots010\cdots0), b_n = (g_i b_{n,2}\cdots b_{n,i} 0 \cdots 0 b_{n,n+1})
\]

where \( u_i, g_i, b_{n,j} \) for \( 2 \leq j \leq i \) are all bounded by \( M \) (recall that \( g_i = gcd(u_i \cdots u_{i-1}, a_n) \)) after step 3.2 \( b_i \) and \( b_n \) change to:

\[
b_i = (0 - a_ib_{n,2} \cdots - a_i b_{n,i} g_i 0 \cdots 0 - a_i b_{n,n+1})
\]

\[
b_n = (g_i + 1 u_i b_{n,2} \cdots v_i b_{n,1} u_i 0 \cdots 0 v_i b_{n,n+1})
\]
Now \(|u_i| < g_i \leq M\) and. Since \(|v_i| < |u_i|\), all the entries in the new \(b_i\) and \(b_n\) (except the last) are bounded by \(M^2\).

Consider now step 3.3 (i) for \(j = i\).

\(b_{i-1}\) has the form \(b_{i-1} = \begin{pmatrix} 0 & b_{i-1,2} & \cdots & b_{i-1,i-1} & b_{i-1,i} & 0 & \cdots & 0 \end{pmatrix} \) with \(b_{i-1,k} \leq M\) for all \(k\). Thus, \(\left| \begin{pmatrix} b_{i-1} \\ \vdots \\ b_{i-1,n} \end{pmatrix} \right| \leq M^2\) and therefore, after step 3.3(i) for \(j = 1\), \(b_{i,k} < b_{i-1,k} \leq M\) and \(b_{i,k} \leq 2M^2\) for \(k < i\).

Similarly, \(b_{n,j} < b_{i-1,j} \leq M\) and \(b_{n,k} \leq 2M^2\) for \(k < i\). Using a similar argument we get after step 3.3 (i) for \(j = i-1\) that \(b_{i-1,j}, b_{n,j-1} < b_{i-2,j-1} \leq M\) and \(b_{n,k} < 3M^2\) for \(k < i-1\)

and similarly for \(j = i-2\) etc.

The loop 3.3 (i) will therefore keep the intermediary results bounded by \(nM^n\) and when the loop is completed all the entries in \(B\) (except the last column) are bounded by \(M\).

**Corollary:** The entries in the first \(n - 1\) columns of the matrix \(A\) output by the algorithm are bounded by \(M\). Moreover if the last column and the last row of \(A\) are removed, then the remaining matrix is lower triangular and the entries in any column of the resulting matrix are nonnegative and bounded by the diagonal entry in that column.
References


