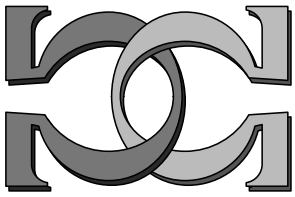
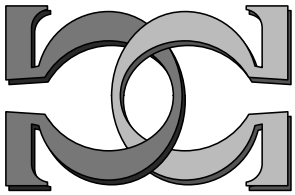
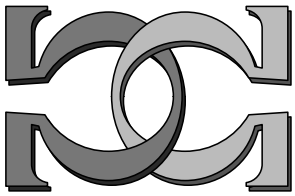
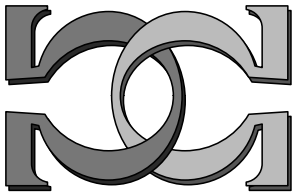


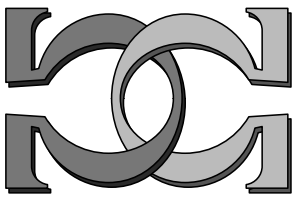




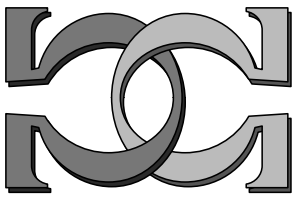
CDMTCS Research Report Series



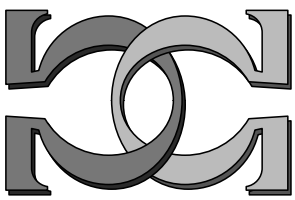
Computation with Finitely Generated Abelian Groups



Peter Huxford
University of Auckland, New Zealand



CDMTCS-517
January 2018



Centre for Discrete Mathematics and
Theoretical Computer Science

Computation with Finitely Generated Abelian Groups

Peter Huxford

0 Introduction

This aim of this report is to explain the theory of finitely generated abelian groups, and some computational methods pertaining to them. Knowledge of introductory group theory is required to understand the main ideas.

There is no algorithm to determine if a given finite presentation defines a group of finite order (or even defines the trivial group). However, such an algorithm exists if the group is also known to be abelian. We give a procedure in this report which, given a description of a finitely generated abelian group G, calculates integers $d_1, \dots, d_r, k$ such that $G \cong \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r} \oplus \mathbb{Z}^k$. The description of G is given by an integer matrix, which we transform into a diagonal matrix, known as its Smith Normal Form.

Naive algorithms inspired by Gaussian elimination often fail because of integer overflow. Intermediate matrices have entries which are very large even for relatively small inputs, making calculations in practice far too expensive to carry out. We will explore some useful techniques, which allow us to perform calculations with respect to an appropriate modulus.

1 Preliminaries

In these notes, we will always write the group operation of an abelian group additively, unless otherwise stated.

Definition 1.1 (Universal Property)**.** A group G is *free abelian* on a subset $X \subseteq G$ if every map from X to an abelian group H extends to a unique homomorphism $G \to H$. We call X a *basis* for G if G is free abelian on X.

There is a similarity with vector spaces: B is a basis of a vector space V if and only if every map from B to a vector space W extends uniquely to a linear map $V \to W$. The existence of such a linear map is equivalent to B being a linearly independent set, and the uniqueness is equivalent to B spanning V.

Theorem 1.2. The free abelian groups with finite basis, up to isomorphism, consist of the groups $\mathbb{Z}^n$ for $n \in \mathbb{N}$.

Proof. Let $e_i \in \mathbb{Z}^n$ be the ith standard basis vector. Each element of $\mathbb{Z}^n$ takes the form $m_1e_1 + \cdots m_ne_n$, for unique integers $m_i \in \mathbb{Z}$. Given a map from $\{e_1, \ldots, e_n\}$ to a group H, we can extend it to a group homomorphism $\mathbb{Z}^n \to H$ as follows. If $e_i \mapsto h_i$, then define

$$m_1e_1 + \cdots + m_ne_n \mapsto m_1h_1 + \cdots + m_nh_n.$$

It is readily seen that this defines a group homomorphism from $\mathbb{Z}^n$ to H, sending $e_i \mapsto h_i$. Moreover each group homomorphism $\mathbb{Z}^n \to H$ which maps $e_i \mapsto h_i$ must agree with this. Hence $\mathbb{Z}^n$ is free abelian on $\{e_1, \ldots, e_n\}$.

Conversely, suppose that G is free abelian on $X = \{x_1, \ldots, x_n\}$. By the universal property, we obtain a homomorphism $\phi\colon G \to \mathbb{Z}^n$ which maps $x_i \mapsto e_i$. Similarly we have a homomorphism $\psi\colon \mathbb{Z}^n \to G$ which maps $e_i \mapsto x_i$. Then $\psi \circ \phi$ is an endomorphism of G fixing X. By the uniqueness in the universal property, $\psi \circ \phi$ must be the identity on G. Similarly $\phi \circ \psi$ is the identity on $\mathbb{Z}^n$. Hence $G \cong \mathbb{Z}^n$. □

Let G be an abelian group generated by n elements. Since $\mathbb{Z}^n$ is free abelian there is an epimorphism $\mathbb{Z}^n \twoheadrightarrow G$ with kernel H. By the first isomorphism theorem, $\mathbb{Z}^n/H$ is isomorphic to G. Thus understanding the structure of subgroups of $\mathbb{Z}^n$ will give us insight into the structure of finitely generated abelian groups.

Theorem 1.3 (Dedekind)**.** A subgroup of $\mathbb{Z}^n$ can be generated by at most n elements.

Proof. We proceed by induction on n. Defining $\mathbb{Z}^0 \coloneqq \{0\}$, the case $n = 0$ becomes trivial. Let $n > 0$, and let $\varphi\colon \mathbb{Z}^n \twoheadrightarrow \mathbb{Z}^n/\langle e_n\rangle$ be the natural map. Note that $\mathbb{Z}^n/\langle e_n\rangle \cong \mathbb{Z}^{n-1}$. If $H \leq \mathbb{Z}^n$, then $\varphi(H)$ is isomorphic to a subgroup of $\mathbb{Z}^{n-1}$. Inductively, we may assume $\varphi(H) = \langle h_1 + \langle e_n\rangle, \ldots, h_{n-1} + \langle e_n\rangle\rangle$, for $h_i \in H$. Note that $H \cap \langle e_n\rangle$ is cyclic, so let $H \cap \langle e_n\rangle = \langle h_n\rangle$ for some $h_n \in H$. We claim $H = \langle h_1, \ldots, h_n\rangle$.

If $h \in H$, then $\varphi(h) = h'' + \langle e_n\rangle$ for some $h' \in \langle h_1, \ldots, h_{n-1}\rangle$. Now $h - h' \in H \cap \langle e_n\rangle = \langle h_n\rangle$. Thus $h \in \langle h_1, \ldots, h_n\rangle$, and hence $H = \langle h_1, \ldots, h_n\rangle$. By induction the theorem follows. □

Definition 1.4. The *integer row space* of an $m \times n$ integer matrix A is $S(A) \coloneqq \{xA : x \in \mathbb{Z}^m\}$, i.e. the set of all integral linear combinations of rows of A.

If $H = \langle h_1, \ldots, h_m\rangle \leq \mathbb{Z}^n$, then H consists of all integral linear combinations of the h_i. Hence each finitely generated abelian group is isomorphic to $\mathbb{Z}^n/S(A)$ for some $m \times n$ integer matrix A. For most descriptions of finitely generated abelian groups, we can explicitly find such an A.

For example, consider the abelianisation $G_{ab} \coloneqq G/[G, G]$ of a finitely presented group G. If $G = \langle X \mid R\rangle$, then the abelianisation G_{ab} is a finitely generated abelian group, with presentation $\langle X \mid R, [X, X]\rangle$. If $X = \{x_1, \ldots, x_n\}$, then we can rewrite each of the relations in R in the form $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$, $\alpha_i \in \mathbb{Z}$. The epimorphism $\mathbb{Z}^n \twoheadrightarrow G_{ab}$ sending $e_i \mapsto x_i$ has kernel generated by the $(\alpha_1, \ldots, \alpha_n)$ appearing in the relations. Using these rows we can form a "relation matrix" A, and then $G_{ab} \cong \mathbb{Z}^n/S(A)$.

2 Integer Row Reduction

A notion of row space can be defined for matrices over an arbitrary ring. When working over a field, we can apply row operations to produce a matrix in reduced row echelon form

(e.g. using Gaussian elimination). We develop a similar theory for integer matrices below.

Definition 2.1. An *integer row operation* applied to a matrix is one of the following:

1. Swap two rows.
2. Multiply a row by -1.
3. Add an integer multiple of one row to another row.

Two $m \times n$ integer matrices A and B are *row equivalent* if there is a sequence of integer row operations transforming one into the other, and in this case we write $A \sim B$.

We can reverse each integer row operation with another. The first two types are involutions. To reverse adding q times row i to row j, for $i \neq j$, we add $-q$ times row i to row j. This makes $\sim$ an equivalence relation on integer $m \times n$ matrices.

In the above definition rows may only be multiplied by -1, in contrast to row operations over a field, where rows may be multiplied by an arbitrary non-zero scalar. This is because multiplying a row by a scalar can be reversed when multiplying by a unit, and the units of a field are its non-zero elements, but the units of $\mathbb{Z}$ are just 1 and -1.

Theorem 2.2. If A and B are integer matrices with $A \sim B$, then $S(A) = S(B)$.

Proof. If B is obtained from A by a single integer row operation, then the rows of B are clearly in $S(A)$, so $S(B) \subseteq S(A)$. Moreover this row operation can be reversed, thus $S(A) \subseteq S(B)$. □

The corresponding notion of reduced row echelon form for integer matrices is row Hermite Normal Form (HNF).

Definition 2.3. An integer $m \times n$ matrix A is in *row Hermite Normal Form (HNF)* if

1. The nonzero rows of A are the first r rows of A, for some $r \leq m$.
2. If j_i is minimal with A_{i,j_i} nonzero, for $1 \leq i \leq r$, then $j_1 < j_2 < \cdots < j_r$.
3. If $1 \leq i \leq r$, then $A_{i,j_i} > 0$.
4. If $1 \leq k < i \leq r$, then $0 \leq A_{k,j_i} < A_{i,j_i}$.

In the above definition, the entries A_{i,j_i} behave similarly to the pivot entries in reduced row echelon form. Below is a sample matrix in row Hermite Normal Form.

$$A \coloneqq \begin{bmatrix} 2 & 1 & 2 & 3 \\ 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 9 \end{bmatrix}.$$

Suppose we want to determine whether a given $u \in \mathbb{Z}^4$ is in $S(A)$. For instance, let $u = (4, 2, -3, 10)$. We seek $x, y, z \in \mathbb{Z}$ with $u = xa_1 + ya_2 + za_3$, where a_1, a_2, a_3 are the rows of A. Isolating the first column, we see $4 = 2x$, so $x = 2$. Next set $v = u - 2a_1 = (0, 0, -7, 4)$. We require $v = ya_2 + za_3$. The second column holds no information. The third column tells us that $-7 = 7y$, so $y = -1$. Set $w = v + a_2 = (0, 0, 0, 9)$. We require $w = za_3$. Observing the last column shows $9 = 9z$, so $z = 1$. Hence $u = 2a_1 - a_2 + a_3$, and so $u \in S(A)$.

If at any stage we found an equation that was not solvable for integer x, y, z, then we would instead conclude that $u \notin S(A)$. Clearly, testing membership in $S(A)$ is easy when A is in HNF. One can also show the nonzero rows of A form a basis of $S(A)$ when A is in HNF.

Theorem 2.4. If A is an $m \times n$ integer matrix, then there is a unique $m \times n$ integer matrix B with $A \sim B$ and B in row Hermite Normal Form.

Proof. We prove this by induction. The result holds trivially when $m = 0$ or $n = 0$. Suppose that $m, n \geq 1$, and that the result holds for all smaller matrices. If there are two nonzero entries in the first column, say $0 < |A_{k,1}| \leq |A_{\ell.1}|$ with $k \neq \ell$, then we decrease the quantity $|A_{1,1}| + \cdots + |A_{m,1}|$ as follows.

First multiply rows k, ℓ by -1 if necessary so that $A_{k,1}, A_{\ell,1} > 0$. Next, subtract row k away from row ℓ. Since $0 \leq A_{\ell,1} - A_{k,1} < A_{\ell,1}$, this strictly decreases the quantity $|A_{1,1}| + \cdots + |A_{m,1}|$. This quantity is a non-negative integer, so it can only be decreased finitely many times. Hence we may assume that A has at most one nonzero entry in the first column. If all entries in the first column are zero, then A has the block form

$$A = \left[\begin{array}{c|c} 0 & \\ \vdots & \quad A' \quad \\ 0 & \end{array}\right]$$

By induction we can reduce A' to HNF by row operations, thus we can reduce A to HNF by row operations. Suppose that A has only one nonzero entry in the first column. By swapping rows and multiplying by -1 if necessary, we may assume that the nonzero entry is $A_{1,1}$ and $A_{1,1} > 0$. Now A has the block form

$$A = \left[\begin{array}{c|ccc} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ \hline 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{array}\right]$$

By induction we can reduce A' to HNF by row operations, so we may assume that A' is in HNF. Suppose that the nonzero rows of A are now the first r rows, and that the first nonzero entry in each row is given by A_{i,j_i}, for $1 \leq i \leq r$. Let $2 \leq k \leq r$, and suppose we have already arranged for $0 \leq A_{1,j_i} < A_{i,j_i}$ to hold, for each $i = 2, \ldots, k-1$.

Using the division algorithm we may write $A_{1,j_k} = qA_{k,j_k} + r$, where $0 \leq r < A_{k,j_k}$. We subtract q times row k away from row 1. Because A_{k,j_k} is the first nonzero entry in row k, and $1 = j_1 < j_2 < \cdots < j_k$, we still have $0 \leq A_{1,j_i} < A_{i,j_i}$ for each $i = 2, \ldots, k-1$.

Hence by induction we can reduce A to some matrix B in HNF, by integer row operations. Let B' be another matrix for which $A \sim B'$ and B' is in HNF. Let $b_1, \ldots, b_m$ and $b'_1, \ldots, b'_m$ be the rows of B and B' respectively. If $B \neq B'$, then there are entries with $B_{i,j} \neq B'_{i,j}$. Choose such i, j with j minimal, without loss of generality $B_{i,j} > B'_{i,j}$. We have $b_i, b'_i \in S(B) = S(A) = S(B')$, hence $b_i - b'_i \in S(B)$.

Suppose that only the first r rows of B are nonzero, and let B_{i,j_i} be the first nonzero entry in b_i for $1 \leq i \leq r$. The first $j-1$ entries of $b_i - b'_i$ are zero, so $b_i - b'_i$ is an integral

linear combination of the rows b_k with $j_k \geq j$. However $b_{ij} - b'_{ij} \neq 0$, and so we must have $j_k = j$ for some k and $B_{k,j} \mid B_{i,j} - B'_{i,j}$. Since $0 \leq B'_{i,j} < B_{i,j} < B_{k,j}$, we must have $|B_{i,j} - B'_{i,j}| < B_{k,j}$, so $B_{i,j} - B'_{i,j} = 0$, which is a contradiction. Therefore $B = B'$, and so each integer matrix A is row equivalent to a unique integer matrix B in HNF. □

Corollary 2.5. If A and B are $m \times n$ integer matrices with $S(A) = S(B)$, then $A \sim B$.

Proof. It suffices to prove that if A and B are in HNF, then $S(A) = S(B) \implies A = B$. We refer the reader to [Sim94, Chapter 8, Proposition 1.1] for the proof of this. □

The proof of this theorem is readily turned into an procedure, such as ROW_REDUCE given below (from [Sim94, Chapter 8, Section 1]).

Procedure : ROW_REDUCE($\sim A$);
Input: An $m \times n$ integer matrix A
Result: Integer row operations are applied to A to reach row Hermite Normal Form
$A \coloneqq B$; $i \coloneqq 1$; $j \coloneqq 1$;
while $i \leq m$ *and* $j \leq n$ **do**
  if $A_{k,j} = 0$ *for* $i \leq k \leq m$ **then**
    $j \coloneqq j + 1$
  else
    while *there exist distinct* k, ℓ *with* $i \leq k, \ell \leq m$ *and* $0 < |A_{k,j}| \leq |A_{\ell,j}|$ **do**
      $q \coloneqq A_{\ell,j}$ div $A_{k,j}$;
      Subtract q times row k of A from row ℓ
    end
    Let $A_{kj} \neq 0$ with $i \leq k \leq m$; (this k is unique)
    if $k \neq i$ **then**
      swap rows i and k of A
    end
    if $A_{i,j} < 0$ **then**
      multiply row i of A by -1
    end
    for $\ell \coloneqq 1$ *to* $i - 1$ **do**
      $q \coloneqq A_{\ell,j}$ div $A_{i,j}$;
      Subtract q times row i of A from row ℓ
    end
    $i \coloneqq i + 1$; $j \coloneqq j + 1$;
  end
end

There is some freedom when implementing the above algorithm. In the inner while loop, we must choose indices $k, \ell \geq i$ with $0 < |A_{k,j}| \leq |A_{\ell,j}|$. If k, ℓ are chosen so that $|A_{\ell,j}| - |A_{k,j}|$ is maximized, then that the quantity $|A_{1,j}| + \cdots + |A_{m,j}|$ decreases as much as possible in each iteration. As described in the second paragraph of the previous proof, the condition in the while loop fails when this quantity can no longer decrease. Thus one might expect this strategy to be the most efficient.

In practice one finds that this strategy can result in A having large entries, significantly increasing run time. This issue is discussed in [Ros52], where an alternative strategy is proposed. The *Rosser strategy* is to choose ℓ so that $|A_{\ell,j}|$ is as large as possible, and

then choose k so that $|A_{k,j}|$ is as large as possible with $k \neq \ell$. Many authors recommend this because they expect it to control the size of the entries during the procedure. See Appendix A for a MAGMA implementation of the row reduction algorithm employing this strategy.

3 Smith Normal Form

In the previous section we showed that integer row operations applied to an $m \times n$ matrix A do not change $S(A)$. We prove that column operations do not affect the isomorphism type of $\mathbb{Z}^n/S(A)$.

Definition 3.1. An *integer column operation* applied to a matrix is one of the following:

1. Swap two columns.
2. Multiply a row by -1.
3. Add an integer multiple of one column to another column.

Definition 3.2. Two $m \times n$ integer matrices A and B are *equivalent* if there is a sequence of integer row and column operations transforming one into the other, and we write $A \approx B$.

Theorem 3.3. If A, B are $m \times n$ integer matrices, then $A \approx B$ if and only if there exist matrices $P \in \mathrm{GL}_m(\mathbb{Z})$ and $Q \in \mathrm{GL}_n(\mathbb{Z})$ such that $B = PAQ$.

Proof. For each integer row operation on an $m \times n$ matrix, there is a corresponding matrix $P \in \mathrm{GL}_m(\mathbb{Z})$ such that the effect of applying the row operation is equivalent to left multiplication by P. Moreover these "elementary" matrices generate $\mathrm{GL}_m(\mathbb{Z})$. Similarly integer column operations correspond to right multiplication by matrices in $\mathrm{GL}_n(\mathbb{Z})$. □

Theorem 3.4. If A, B are equivalent integer $m \times n$ matrices, then $\mathbb{Z}^n/S(A) \cong \mathbb{Z}^n/S(B)$.

Proof. There exist $P \in \mathrm{GL}_m(\mathbb{Z})$, $Q \in \mathrm{GL}_n(\mathbb{Z})$ such that $B = PAQ$. Then $B \sim AQ$, hence $S(B) = S(AQ)$. Notice that $S(A)Q := \{uQ : u \in S(A)\} = \{xAQ : x \in \mathbb{Z}^m\} = S(AQ)$. It follows that the mapping illustrated in the diagram is a well defined homomorphism $\mathbb{Z}^n/S(A) \to \mathbb{Z}^n/S(AQ) = \mathbb{Z}^n/S(B)$. It is an isomorphism because Q is invertible.

$$\begin{array}{ccc} \mathbb{Z}^n & \xrightarrow{Q\colon x \mapsto xQ} & \mathbb{Z}^n \\ \downarrow & & \downarrow \\ \mathbb{Z}^n/S(A) & \xrightarrow[x+S(A) \mapsto xQ+S(AQ)]{} & \mathbb{Z}^n/S(AQ) \end{array}$$

□

We now define a normal form which distinguishes the isomorphism types of $\mathbb{Z}^n/S(A)$.

Definition 3.5. An $m \times n$ integer matrix A is in *Smith Normal Form (SNF)* if there is some r such that $d_i := A_{i,i} > 0$ for each $1 \leq i \leq r$, all remaining entries of A are zero, and $d_1 \mid d_2 \mid \cdots \mid d_r$. The integers $d_1, \ldots, d_r$ are the *invariant factors* of A.

Here is a sample matrix in Smith Normal Form

$$A \coloneqq \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 \end{bmatrix}.$$

Note that $S(A) = \{(2x, 4y, 12z, 0, 0) : x, y, z \in \mathbb{Z}\}$. This is the kernel of the epimorphism

$$\begin{gathered} \mathbb{Z}^5 \twoheadrightarrow \mathbb{Z}_2 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}_{12} \oplus \mathbb{Z} \oplus \mathbb{Z} \\ (a, b, c, d, e) \mapsto (a \bmod 2, b \bmod 4, c \bmod 12, d, e). \end{gathered}$$

Therefore $\mathbb{Z}^5/S(A) \cong \mathbb{Z}_2 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}_{12} \oplus \mathbb{Z} \oplus \mathbb{Z}$. Determining the isomorphism type of $\mathbb{Z}^n/S(A)$ for an $m \times n$ matrix A is easy when A is in Smith Normal Form.

Theorem 3.6. Let A be an $m \times n$ integer matrix in SNF, with invariant factors $d_1, d_2, \ldots, d_r$. Then $\mathbb{Z}^n/S(A) \cong \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r} \oplus \mathbb{Z}^{n-r}$.

Theorem 3.7. If A is an integer matrix, then there exists an integer matrix B in Smith Normal Form with $A \approx B$.

Proof. Suppose A is not the zero matrix. The first goal is to reduce A to the block form

$$\left[\begin{array}{c|ccc} d & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{array}\right] \tag{$\star$}$$

with $d > 0$. Let $A_{i,j}$ be a nonzero entry of A. Swap rows i and 1, and columns j and 1 if required, to ensure $i = j = 1$, and multiply row 1 by -1 if necessary to ensure $A_{1,1} > 0$. If $A_{1,1}$ divides all entries in row and column 1, then add multiples of row/column 1 to the other rows/columns, to reach the form ($\star$).

If $A_{1,1}$ does not divide all entries in row and column 1, then we decrease this entry as follows. Suppose that $A_{1,1} \nmid A_{i,1}$ for some i. Write $A_{i,1} = qA_{1,1} + r$ with $0 < r < A_{1,1}$. Add $-q$ times row 1 to row i, and swap rows 1 and i. Similarly, if $A_{1,1} \nmid A_{1,j}$ for some j, then write $A_{1,j} = qA_{1,1} + r$ with $0 < r < A_{1,1}$. Add $-q$ times column 1 to column j, and swap columns 1 and j.

If we repeat the operations in the previous paragraph, eventually $A_{1,1}$ will divide all entries in row and column 1, and thus we can reach the block form ($\star$). Inductively, we reduce A' to SNF by row and column operations, so we can reduce A to a matrix with block form

$$\left[\begin{array}{ccc|c} d_1 & & & \\ & \ddots & & 0 \\ & & d_r & \\ \hline & 0 & & 0 \end{array}\right]$$

where $d_i > 0$. If the divisibility condition $d_1 \mid d_2 \mid \cdots \mid d_r$ holds, then we have produced a matrix in SNF. If not, suppose that $d_i \nmid d_j$ for some $i < j$. Set $a \coloneqq d_i$ and $b \coloneqq d_j$.

The Euclidean algorithm produces integers u, v with $d := \gcd(a, b) = ua + vb$, and then $\ell := \operatorname{lcm}(a, b) = ab/d$. Now

$$\begin{bmatrix} u & v \\ -b/d & a/d \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} 1 & -vb/d \\ 1 & ua/d \end{bmatrix} = \begin{bmatrix} d & 0 \\ 0 & \ell \end{bmatrix}.$$

In this way, we can use row and column operations to replace d_i and d_j with $\gcd(d_i, d_j)$ and $\operatorname{lcm}(d_i, d_j)$ respectively. Repeating this will produce a matrix in Smith Normal Form. □

Uniqueness follows from the uniqueness of the integers in the following important theorem. Note that the previous result implies the existence of such integers. To prove the uniqueness, we refer the reader to [DF04, Chapter 5, Theorem 3].

Theorem 3.8 (Fundamental Theorem of Finitely Generated Abelian Groups)**.** Let G be a finitely generated abelian group. There exist unique integers $n, d_1, \ldots, d_r > 0$ with $d_1 \mid d_2 \mid \cdots \mid d_r$, and

$$G \cong \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r} \oplus \mathbb{Z}^{n-r}.$$

Corollary 3.9. If A is an $m \times n$ integer matrix, then there is a unique $m \times n$ integer matrix B with $A \approx B$ and B in Smith Normal Form.

Thus it makes sense to define the *invariant factors* of an integer matrix to be the invariant factors of the matrix equivalent to it which is in SNF.

Corollary 3.10. If A and B are $m \times n$ integer matrices with $\mathbb{Z}^n/S(A) \cong \mathbb{Z}^n/S(B)$, then $A \approx B$.

Proof. By Theorem 3.8, A and B have the same invariant factors. Because they have the same dimensions, they must be equivalent to the same matrix in SNF. Hence $A \approx B$. □

The proof of Theorem 3.7 is readily transformed into a procedure for computing the Smith Normal Form of a given matrix. See Appendix B for a Magma implementation. However one quickly finds that it has certain defects. For example, I ran the implementation in Appendix B on a random 100×100 matrix with entries in $\{-1, 0, 1\}$, on the machine described in Section 5. After 13 hours the procedure had still not terminated. We investigate here what can happen with a smaller example. Consider the following matrix

$$A := \begin{bmatrix} 6 & 9 & 8 & 2 & 6 & 5 & 2 & 0 \\ 5 & 10 & 1 & 4 & 8 & 5 & 4 & 8 \\ 3 & 1 & 9 & 6 & 3 & 10 & 5 & 3 \\ 6 & 10 & 3 & 2 & 5 & 0 & 3 & 10 \\ 6 & 1 & 3 & 8 & 6 & 6 & 9 & 1 \\ 6 & 2 & 2 & 1 & 8 & 1 & 10 & 10 \\ 4 & 4 & 10 & 7 & 10 & 8 & 3 & 3 \\ 7 & 0 & 1 & 9 & 8 & 5 & 5 & 1 \end{bmatrix}$$

If we run the Magma implementation given in Appendix B on A, then the intermediate matrix in which the entry with largest modulus appears is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 488293303266630430319 & 60 & -769507651269581073 \\ 0 & 0 & 0 & 0 & 0 & 0 & -976586606533260862542 & 91 & 153901530253916214 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 940355243430876882709497035 & 2 & -148191783481495923038334 \end{bmatrix}$$

The SNF of A is given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10615254 \end{bmatrix}$$

The entries of A have modulus at most 10. However intermediate matrices appear with entries of modulus roughly 9.4×10^{27}, much larger than the entries in the resulting SNF. This "entry explosion" is only worsened by increasing the dimensions of the input matrix, and increasing the size of its entries. Row and column operations with entries so large is expensive, so circumventing this issue is of great importance if one wishes to compute the SNF of larger matrices in reasonable time.

4 Modular techniques

The methods discussed in the previous section to compute Smith Normal Form are inadequate for large matrices. The MAGMA intrinsic `SmithForm` is much more capable of computing the SNF of a large matrix. This is an indication that the approach taken earlier can be improved. We discuss some ideas described in [Sim94, Chapter 8, Section 4], which aim to reduce the maximum modulus entry of matrices produced along the way.

We redevelop some of the theory from the previous sections, so that computations can be carried out modulo a fixed integer. Many of the proofs of the following theorems are similar to ones already given, so we omit them. More detail can be found in [Sim94, Chapter 8, Section 4]. Fix an integer $d > 1$. We write $\overline{A}$ to denote the matrix A with entries reduced modulo d.

Definition 4.1. A *row operation modulo d* applied to an integer matrix is one of the following:

1. Swap two rows.
2. Multiply a row by an integer c with $\gcd(c, d) = 1$, reducing entries modulo d.
3. Add an integral multiple of one row to another row, reducing entries modulo d.

We define *column operations modulo d* in a similar fashion. Two $m \times n$ integer matrices A and B are *row equivalent modulo d* if there is a sequence of row operations modulo d transforming $\overline{A}$ to $\overline{B}$, and they are *equivalent modulo d* if there is a sequence of row and column operations modulo d transforming $\overline{A}$ to $\overline{B}$.

Definition 4.2. An integer matrix A is in *row Hermite Normal Form modulo d (HNF modulo d)* if

1. $A = \overline{A}$.
2. A is in row Hermite Normal Form.
3. The first nonzero entry in each nonzero row divides d.

When we work modulo a prime p, the third condition ensures that the first nonzero entry in each nonzero row is 1, thus in this case the HNF modulo p agrees with the reduced row echelon form over the field $\mathbb{Z}_p$.

Theorem 4.3. If A is an $m \times n$ integer matrix, then there is a unique $m \times n$ integer matrix B with $A \sim B$ and B in row Hermite Normal Form modulo d.

Definition 4.4. An integer matrix A is in *Smith Normal Form modulo d (SNF modulo d)* if

1. $A = \overline{A}$.
2. A is in Smith Normal Form.
3. The nonzero entries of A divide d.

Theorem 4.5. If A is an $m \times n$ integer matrix, then there is a unique $m \times n$ integer matrix B with $A \sim B$ and B in Smith Normal Form modulo d.

The algorithms for computing Hermite and Smith Normal Form modulo an integer are similar to the corresponding procedures over $\mathbb{Z}$, except all operations are performed modulo d, and we multiply rows/columns by appropriate integers c with $\gcd(c, d) = 1$ where appropriate to ensure that the relevant entries divide d.

Definition 4.6. Let A be an integer matrix, and let B be the unique matrix in HNF modulo d which is row equivalent modulo d to A. The *d-rank* of A is the number of nonzero rows of B.

Note that if $A \sim B$ and B is in HNF, then the number of nonzero rows of B is equal to the rank of A. Thus the d-rank of an integer matrix is a lower bound for its rank.

Definition 4.7. For an $m \times n$ integer matrix A, and $0 \leq k \leq m, n$, let $D_k(A)$ denote the greatest common divisor of all determinants of $k \times k$ submatrices of A. We adopt the convention that the determinant of a 0×0 matrix is 1, so that $D_0(A) = 1$ for every integer matrix A.

Theorem 4.8. If integer row or column operations are applied to A, then $D_k(A)$ is unchanged for each $0 \leq k \leq m, n$.

Proof. This is clear except when an integer multiple of a row or column is added to another row or column, respectively. See [Sim94, Chapter 8, Proposition 4.1] for the proof. □

Corollary 4.9. Suppose that $A \sim B$, and B is in SNF with invariant factors $d_1 \mid d_2 \mid \cdots \mid d_r$. Then $D_k(A) = d_1 \cdots d_k$ for $k \leq r$, and $D_k(A) = 0$ for $k > r$. In particular $d_k = D_k(A)/D_{k-1}(A)$ when $1 \leq k \leq r$.

While this gives an alternative way of computing the SNF of a matrix, there are $\binom{m}{k}\binom{n}{k}$ different $k \times k$ submatrices of an $m \times n$ matrix. However consider the following theorem, which follows from noting that the SNF modulo d can be computed by calculating the SNF over $\mathbb{Z}$, and then reducing entries modulo d.

Theorem 4.10. Let A be an integer matrix, and let B be in SNF with $A \approx B$. Suppose B has invariant factors $d_1 \mid d_2 \mid \cdots \mid d_r$, and let $d > 1$ with $d_r \mid d$. Let C be the matrix equivalent to A in SNF modulo d. If C has invariant factors $c_1 \mid c_2 \mid \cdots \mid c_s$, then $s \leq r$, and $d_i = c_i$ for $i \leq s$, and $d_i = d$ for $s < i \leq r$.

This suggests the following algorithm for computing the SNF of a given matrix A.

1. Find the rank r of A.

2. Find a small number of $r \times r$ submatrices of A with nonzero determinant, and take their greatest common divisor d. By Corollary 4.9, this will be a multiple of the largest nonzero entry in the SNF of A.
3. Compute C, the SNF of A modulo d.
4. If C has s nonzero entries, then by Theorem 4.10 we can recover the SNF of A by adding in $r - s$ copies of d on the diagonal after the nonzero entries of C.

We currently do not have a method to efficiently compute the rank of a matrix A. The HNF algorithm provided earlier also produces relatively large modulus entries for small inputs. We also do not have a method to compute the determinant of an $r \times r$ submatrix of A, and find the $r \times r$ submatrices with nonzero determinant.

We compute the d-rank of A by computing the the HNF of A modulo d. This is a lower bound for the rank of A. If we calculate the d-rank for various d, then their maximum is a lower bound of the rank of A. With some work, we can ensure that this is equal to the rank.

Theorem 4.11 (Hadamard's Inequality)**.** Let A be an $n \times n$ real matrix.

$$|\det A| \leq \prod_{i=1}^{n} \left(\sum_{j=1}^{n} A_{ij}^2 \right)^{1/2}$$

Proof. See [Gar07, Theorem 14.1.1]. □

We use this to get an easily computable bound to the determinant of every square submatrix of an integer matrix.

Corollary 4.12. If A is an $m \times n$ integer matrix with $m \leq n$, then the determinant of a square submatrix of A has modulus at most

$$\min \left\{ \prod_{i} \left(\sum_{j=1}^{n} A_{ij}^2 \right)^{1/2}, \left[\max_{j} \left(\sum_{i=1}^{m} A_{ij}^2 \right)^{1/2} \right]^m \right\},$$

where the product is taken over the nonzero rows.

Definition 4.13. Given an $m \times n$ matrix, we let $h(A)$ be the bound given by Corollary 4.12 on A if $m \leq n$, and on A^t if $m > n$.

Theorem 4.14. Let A be an integer matrix. If $p_1, \ldots, p_k$ are distinct primes with $p_1 \cdots p_k > h(A)$, then the rank of A is the maximum p_i-rank of A for $1 \leq i \leq k$.

Proof. Let r be the maximum p_i-rank. It suffices to prove that all square submatrices of A with more than r rows have determinant 0. Let B be such a matrix. The p_i-rank of A is at most r, so $\det B \equiv 0 \pmod{p_i}$, for $1 \leq i \leq k$. Hence $\det B \equiv 0 \pmod{p_1 \cdots p_k}$. But by Corollary 4.12, $|\det B| \leq h(A) < p_1 \cdots p_k$, thus $\det B = 0$. □

Once we compute the rank r of an integer matrix A, we can exploit Hadamard's Inequality to compute efficiently the determinants of $r \times r$ submatrices.

Theorem 4.15. Let A be an integer matrix, and let B be a square submatrix of A. Let $p_1, \ldots, p_k$ be distinct primes with $p_1 \cdots p_k > 2h(A)$. If $\det B \equiv b_i \pmod{p_i}$, for $1 \leq i \leq k$, then $\det B$ is the integer b satisfying $b \equiv b_i \pmod{p_i}$, with least absolute value.

Proof. By the Chinese Remainder Theorem, the conditions determine $\det B$ modulo $p_1 \cdots p_k$. Also $-h(A) \leq \det B \leq h(A)$, and since $p_1 \cdots p_k > 2h(A)$, no two integers between $-h(A)$ and $h(A)$ are congruent modulo $p_1 \cdots p_k$. □

We now discuss how to compute the p-rank of a matrix, and its determinant modulo p, for some prime p. When p is prime, $\mathbb{Z}_p$ is a field, so standard methods (e.g. row reduction to row echelon form over a field) can be used to compute the p-rank and determinant modulo p.

Moreover, [Sim94] suggests slightly modifying the procedure of row reduction modulo p, to find $r \times r$ submatrices of nonzero determinant, where r is the rank. For a matrix A with reduced row echelon form R, let S denote the row indices in A which are eventually swapped into a nonzero row position in R, and let T denote the column indices which contain the first nonzero entry in some row of R. The $r \times r$ submatrix of A with rows indexed by S and columns indexed by T has nonzero determinant.

Using the above ideas, we produce an improved Smith Normal Form procedure. See Appendix C for a MAGMA implementation.

5 Empirical Results and Discussion

We now compare three methods to compute the Smith Normal Form of a given integer matrix in MAGMA [BCP97]. These are

- `SmithNormalForm` given in Appendix B (described in Section 3),
- `SmithNormalFormImproved` given in Appendix C (described in Section 4),
- the MAGMA intrinsic `SmithForm`.

We generated a random $n \times n$ matrix with entries in $\{-1, 0, 1\}$ for $n = 10, 20, \ldots, 100$, and computed the Smith Normal Form of each of the matrices with each method. The machine used had a clock speed of 2.6GHz, 690GB of RAM, and was running MAGMA version V2.23-3. If a given computation took longer than 30 minutes to halt, it was terminated.

Our results are recorded in the following table. In each computation, the time taken to compute the SNF, and total memory used in the MAGMA session were recorded. For the computations using `SmithNormalForm`, the maximum modulus of an entry appearing in some intermediate matrix was recorded (see the "Max" column). For the computations using `SmithNormalFormImproved`, the modulus with respect to which calculations were performed is recorded (see the "Modulus" column). There is no corresponding measurement available for the MAGMA intrinsic `SmithForm`.

	SmithNormalForm			SmithNormalFormImproved			SmithForm	
Dimensions	Time	Memory	Max	Time	Memory	Modulus	Time	Memory
10×10	0.00s	32.0MB	68	0.01s	32.0MB	40	0.00s	32.0MB
20×20	0.00s	32.0MB	$6.0 \cdot 10^{42}$	0.01s	32.0MB	$1.7 \cdot 10^{7}$	0.00s	32.0MB
30×30	0.04s	32.0MB	$7.8 \cdot 10^{4056}$	0.02s	32.0MB	$2.5 \cdot 10^{13}$	0.00s	32.0MB
40×40	0.31s	32.0MB	$\sim 10^{1.4 \cdot 10^5}$	0.05s	32.0MB	$1.3 \cdot 10^{19}$	0.02s	32.0MB
50×50	>30m	384MB	$\sim 10^{2.5 \cdot 10^6}$	0.08s	32.0MB	$3.6 \cdot 10^{27}$	0.02s	32.0MB
60×60	>30m	928MB	$\sim 10^{2.5 \cdot 10^6}$	0.15s	32.0MB	$7.0 \cdot 10^{34}$	0.02s	32.0MB
70×70	>30m	2.89GB	$\sim 10^{5.4 \cdot 10^6}$	0.24s	32.0MB	$5.0 \cdot 10^{42}$	0.02s	32.0MB
80×80	>30m	25.0GB	$\sim 10^{1.2 \cdot 10^7}$	0.38s	32.0MB	$2.0 \cdot 10^{51}$	0.03s	32.0MB
90×90	>30m	2.56GB	$\sim 10^{8.9 \cdot 10^5}$	0.60s	64.0MB	$1.9 \cdot 10^{60}$	0.07s	32.0MB
100×100	>30m	14.8GB	$\sim 10^{2.3 \cdot 10^6}$	0.79s	64.0MB	$1.6 \cdot 10^{58}$	0.06s	32.0MB

`SmithNormalFormImproved` outperforms the original procedure `SmithNormalForm` only when the dimensions of the input are sufficiently large. This is because some extra computation, e.g. the bound given by Corollary 4.12, is performed by `SmithNormalFormImproved`. The cost of these additional calculations are only outweighed by the benefit of performing row operations with respect to an appropriate modulus when the entries appearing in intermediate matrices are very large.

Observe that the MAGMA intrinsic `SmithForm` always performs better than the procedure `SmithNormalFormImproved`. One reason is that `SmithForm` is a compiled C program, but `SmithNormalFormImproved` is written in MAGMA, an interpreted language.

Moreover, there are other techniques beyond those discussed in Section 4 which can be used to more efficiently compute the SNF, many of which are discussed in [HHR93]. Recall the beginning of the proof of Theorem 3.7. First a nonzero entry is selected. If it does not divide every entry in its row and column, then we produce a smaller nonzero entry in the same position by applying appropriate row and column operations. Eventually this entry will divide all entries in its row and column.

In [HHR93], this process is called *pivoting*, and the nonzero entry which is selected is called a *pivot*. In `SmithNormalForm`, the way a pivot is selected is to initially choose the smallest modulus entry, and thereafter select the first entry in the current pivot's row and column which is not divisible by the current pivot. Havas *et al.* [HHR93] suggest some alternate pivoting strategies which can improve performance. In particular it is recommended to try certain pivoting strategies first, and only resort to modular methods if these are unsuccessful.

There is an additional deficiency in both of the procedures described in this report. Given an $m \times n$ integer matrix A, computing the SNF of A determines the rank r and positive integers $d_1 \mid d_2 \mid \cdots \mid d_r$ such that $\mathbb{Z}^n/S(A) \cong \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r} \oplus \mathbb{Z}^{n-r}$. However the SNF alone does not provide an explicit description of such isomorphism, which is often useful.

When only row and column operations and no modular techniques are used to compute the SNF of A, we can find $P \in \mathrm{GL}_m(\mathbb{Z})$, $Q \in \mathrm{GL}_n(\mathbb{Z})$ such that PAQ is in SNF, by applying the row operations to I_m to produce P, and the column operations to I_n to produce Q. As in the proof of Theorem 3.4, P and Q can be used to describe such an isomorphism.

Unfortunately, when modular techniques are applied, such an isomorphism can only be recovered in this way if $r = n$. If we define $T \coloneqq \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r}$ and $F \coloneqq \mathbb{Z}^{n-r}$, then $\mathbb{Z}^n/S(A) \cong T \oplus F$, and T is torsion (every element has finite order), and $\mathbb{Z}^{n-r}$ is torsion

free (all non-identity elements have infinite order). Havas *et al.* [HHR93] describe how *Lattice Basis Reduction* algorithms such as MLLL can be used together with the modular methods described here, to completely describe an isomorphism $\mathbb{Z}^n/S(A) \to T \oplus F$.

References

[Ros52] J.B. Rosser. "A method of computing exact inverses of matrices with integer coefficients". *Journal of Research of the National Bureau of Standards* 49.5 (1952), p. 349.

[HHR93] George Havas, Derek F. Holt, and Sarah Rees. "Recognizing badly presented Z-modules". *Linear Algebra and its Applications* 192 (1993), pp. 137–163.

[Sim94] C.C. Sims. *Computation with Finitely Presented Groups*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. "The Magma algebra system. I. The user language". *J. Symbolic Comput.* 24.3-4 (1997), pp. 235–265.

[DF04] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2004.

[Gar07] D. J. H. Garling. *Inequalities: A Journey into Linear Analysis*. Cambridge University Press, 2007.

Appendix A Hermite Normal Form

```
RowReduce := procedure(~A)
    m := NumberOfRows(A); n := NumberOfColumns(A);
    i := 1; j := 1;

    while i le m and j le n do
        AbsEntries := [ Abs(A[k,j]) : k in [i..m] ];
        Indices := [i..m];
        ParallelSort(~AbsEntries, ~Indices);
        while true do

            if AbsEntries[#AbsEntries] eq 0 then
                j +:= 1;
                break;
            elif i eq m or AbsEntries[#AbsEntries-1] eq 0 then
                k := Indices[#Indices];
                if k ne i then
                    SwapRows(~A,i,k);
                end if;
                if A[i,j] lt 0 then
                    MultiplyRow(~A,-1,i);
                end if;

                for l in [1..i-1] do
                    q := A[l,j] div A[i,j];
                    AddRow(~A,-q,i,l);
                end for;
                i +:= 1; j +:= 1;
                break;

            else
                k := Indices[#Indices-1]; l := Indices[#Indices];
                q := A[l,j] div A[k,j];
                AddRow(~A,-q,k,l);
                AbsEntries[#AbsEntries] := Abs(A[l,j]);
                ParallelSort(~AbsEntries, ~Indices);
            end if;
        end while;
    end while;
end procedure;
```

Appendix B Smith Normal Form

```
// Uses row and column operations to diagonalise A
// max records maximum modulus entry
// overtime is a flag which is set to true if Cputime() exceeds maxtime
// computation halts shortly after overtime is set to true
procedure Diagonalise(~A, ~max, ~overtime, maxtime)
    m := NumberOfRows(A); n := NumberOfColumns(A);
    if not IsZero(A) and m ne 0 and n ne 0 then
        // find min nonzero modulus entry A[i,j]
        // store max modulus entry
        r := 1; s := 1;
        while A[r,s] eq 0 do
            if s lt n then
                s +:= 1;
            else
                r +:= 1; s := 1;
            end if;
        end while;
        i := r; j := s;
        max := Abs(A[r,s]);
        min := max;
        while r le n do
            if A[r,s] ne 0 and Abs(A[r,s]) lt min then
                min := Abs(A[r,s]);
                i := r; j := s;
            elif Abs(A[r,s]) gt max then
                max := Abs(A[r,s]);
            end if;
            if s lt n then
                s +:= 1;
            else
                r +:= 1; s := 1;
            end if;
        end while;

        // ensure A[i,j] divides everything in its row and column
        r := 1; s := 1;
        while r le m or s le n do
            if Cputime() gt maxtime then
                overtime := true;
                print "overtime";
                break;
            else
                if r le m then
                    if A[r,j] mod A[i,j] ne 0 then
                        q := A[r,j] div A[i,j];
                        AddRow(~A,-q,i,r);
                        i := r; r := 1;
                        for k in [1..n] do
                            if Abs(A[i,k]) gt max then
                                max := Abs(A[i,k]);
                            end if;
                        end for;
                    else
                        r +:= 1;
                    end if;
                elif A[i,s] mod A[i,j] ne 0 then
```

```
                    q := A[i,s] div A[i,j];
                    AddColumn(~A,-q,j,s);
                    j := s; r := 1; s := 1;
                    for k in [1..m] do
                        if Abs(A[k,j]) gt max then
                            max := Abs(A[k,j]);
                        end if;
                    end for;
                else
                    s +:= 1;
                end if;
            end if;
        end while;

        if not overtime then
            SwapRows(~A,i,1);
            SwapColumns(~A,j,1);

            // make entries below and to the right of A[1,1] zero
            for i in [2..m] do
                q := A[i,1] div A[1,1];
                AddRow(~A,-q,1,i);
                for k in [1..n] do
                    if Abs(A[i,k]) gt max then
                        max := Abs(A[i,k]);
                    end if;
                end for;
            end for;
            for j in [2..n] do
                q := A[1,j] div A[1,1];
                AddColumn(~A,-q,1,j);
                for k in [1..m] do
                    if Abs(A[k,j]) gt max then
                        max := Abs(A[k,j]);
                    end if;
                end for;
            end for;

            if A[1,1] lt 0 then
                A[1,1] := -A[1,1];
            end if;

            C := Submatrix(A, 2, 2, m-1, n-1);
            submax := 0;
            Diagonalise(~C, ~submax, ~overtime, maxtime);
            InsertBlock(~A,C,2,2);
            if submax gt max then
                max := submax;
            end if;
        end if;
    end if;
end procedure;

// Computes the SNF of A by a Gaussian elimnation inspired method
// max stores the maximum modulus entry in computation
// maxtime is the time in seconds before the computation halts
procedure SmithNormalForm(~A, ~max : maxtime := 1800)
    overtime := false;
```

```
115     m := NumberOfRows(A); n := NumberOfColumns(A);
        Diagonalise(~A,~max,~overtime,maxtime);
117     if not overtime then
            r := 1;
119         while r le Min(m,n) do
                if A[r,r] eq 0 then
121                 break;
                else
123                 r +:= 1;
                end if;
125         end while;
            r -:= 1;
127
            // enforce divisibility condition
129         for i in [1..r] do
                for j in [i+1..r] do
131                 if A[j,j] mod A[i,i] ne 0 then
                        d := Gcd(A[i,i],A[j,j]);
133                     l := A[i,i]*A[j,j] div d;
                        A[i,i] := d; A[j,j] := l;
135                 end if;
                end for;
137         end for;
        end if;
139 end procedure;
```

Appendix C Smith Normal Form Improved

```
// returns the bound given by Hadamard's inequality
function hadamard(A)
    m := NumberOfRows(A); n := NumberOfColumns(A);
    if m gt n then
        return hadamard(Transpose(A));
    else
        prod := 1;
        for i in [1..m] do
            sum := 0;
            for j in [1..n] do
                sum +:= A[i,j]^2;
            end for;
            if sum ne 0 then
                prod *:= sum;
            end if;
        end for;

        max := 0;
        for j in [1..n] do
            sum := 0;
            for i in [1..m] do
                sum +:= A[i,j]^2;
            end for;
            max := Max(max,sum);
        end for;

        return Sqrt(Min(prod, max^m));
    end if;
end function;

// Uses row and column operations modulo d to diagonalise A
procedure DiagonaliseMod(~A,d)
    Z := IntegerRing(); Zd := IntegerRing(d);
    m := NumberOfRows(A); n := NumberOfColumns(A);

    if not IsZero(A) and m ne 0 and n ne 0 then
        A := ChangeRing(A, Zd);
        // find a nonzero entry
        i := 1; j := 1;
        while A[i,j] eq 0 do
            if j lt n then
                j +:= 1;
            else
                i +:= 1; j := 1;
            end if;
        end while;

        // ensure A[i,j] divides everything in its row and column
        r := 1; s := 1;
        while r le m or s le n do
            if r le m then
                if (Z ! A[r,j]) mod (Z ! A[i,j]) ne 0 then
                    q := (Z ! A[r,j]) div (Z ! A[i,j]);
                    AddRow(~A,-q,i,r);
                    i := r; r := 1;
                else
```

```
                    r +:= 1;
                end if;
            elif (Z ! A[i,s]) mod (Z ! A[i,j]) ne 0 then
                q := (Z ! A[i,s]) div (Z ! A[i,j]);
                AddColumn(~A,-q,j,s);
                j := s; r := 1; s := 1;
            else
                s +:= 1;
            end if;
        end while;

        SwapRows(~A,i,1);
        SwapColumns(~A,j,1);

        // clear out entries below and to the right of A[1,1]
        for i in [2..m] do
            q := (Z ! A[i,1]) div (Z ! A[1,1]);
            AddRow(~A,-q,1,i);
        end for;
        for j in [2..n] do
            q := (Z ! A[1,j]) div (Z ! A[1,1]);
            AddColumn(~A,-q,1,j);
        end for;

        A := ChangeRing(A, Z);

        C := Submatrix(A, 2, 2, m-1, n-1);
        DiagonaliseMod(~C,d);
        InsertBlock(~A,C,2,2);
    end if;
end procedure;

// Computes the SNF of A using Modular techniques
// primestart is where to begin looking for primes to do calculations
// dnumber is the number of nonzero rxr determinants we find,
// where r is the rank of A
procedure SmithNormalFormImproved(~A : primestart := 1000, dnumber := 3)
    X := SmithForm(A);
    m := NumberOfRows(A); n := NumberOfColumns(A);
    if not IsZero(A) and m ne 0 and n ne 0 then
        b := hadamard(A);

        // find sequence of distinct primes whose product exceeds 2*b
        primes := [NextPrime(primestart)];
        prod := primes[1];
        while prod le 2*b do
            Append(~primes, NextPrime(primes[#primes]));
            prod *:= primes[#primes];
        end while;

        submatrices := {};
        rank := 0;

        for p in primes do
            // row reduction modulo p
            // find the p-rank r of A
            // and an rxr submatrix with full rank
            rows := [1..m];
```

```
            columns := [];

            Zp := IntegerRing(p);
            Ap := ChangeRing(A, Zp);

            i := 1; j := 1;
            while i le m and j le n do
                k := i;
                while k le m do
                    if Ap[k,j] eq 0 then
                        k +:= 1;
                    else
                        break;
                    end if;
                end while;

                if k le m then
                    Append(~columns, j);
                    if k ne i then
                        SwapRows(~Ap, i, k);
                        tmp := rows[k];
                        rows[k] := rows[i];
                        rows[i] := tmp;
                    end if;
                    c := -1/Ap[i,j];
                    for l in [i+1..m] do
                        AddRow(~Ap, Ap[l,j]*c, i, l);
                    end for;
                    i +:= 1;
                end if;
                j +:= 1;
            end while;

            r := i - 1;
            rows := rows[[1..r]];
            Sort(~rows);
            if r gt rank then
                rank := r;
                submatrices := {[rows, columns]};
            elif r eq rank and #submatrices lt dnumber then
                Include(~submatrices, [rows, columns]);
            end if;
        end for;

        // compute determinants of submatrices
        dets := [];
        for indices in submatrices do
            rows := indices[1]; columns := indices[2];
            detsprimes := [];
            for p in primes do
                Z := IntegerRing(); Zp := IntegerRing(p);
                Ap := ChangeRing(A, Zp);
                detp := Z ! Determinant(Submatrix(Ap, rows, columns));
                Append(~detsprimes, detp);
            end for;
            sol := CRT(detsprimes, primes);
            altsol := sol - prod;
            if Abs(altsol) lt sol then
```

```
                Append(~dets, altsol);
            else
                Append(~dets, sol);
            end if;
        end for;

        d := GCD(dets);
        print d;

        // Compute the SNF of A modulo d
        DiagonaliseMod(~A,d);
        s := 1;
        while s le Min(m,n) do
            if A[s,s] eq 0 then
                break;
            else
                s +:= 1;
            end if;
        end while;
        s -:= 1;
        // enforce divisibility condition
        for i in [1..s] do
            for j in [i+1..s] do
                if A[j,j] mod A[i,i] ne 0 then
                    g := Gcd(A[i,i],A[j,j]);
                    l := A[i,i]*A[j,j] div g;
                    A[i,i] := g; A[j,j] := l;
                end if;
            end for;
        end for;

        // Recover SNF of A
        for i in [s+1..rank] do
            A[i,i] := d;
        end for;
    end if;
end procedure;
```