Extended GCD and Hermite Normal Form Algorithms via Lattice Basis Reduction

George Havas, Bohdan S. Majewski, and Keith R. Matthews

CONTENTS

Extended gcd calculation has a long history and plays an important role in computational number theory and linear algebra. Recent results have shown that finding optimal multipliers in extended gcd calculations is difficult. We present an algorithm which uses lattice basis reduction to produce small integer multipliers $x_1, \dots, x_m$ for the equation $s = \gcd(s_1, \dots, s_m) = x_1 s_1 + \cdots + x_m s_m$, where $s_1, \dots, s_m$ are given integers. The method generalises to produce small unimodular transformation matrices for computing the Hermite normal form of an integer matrix.

1. INTRODUCTION

Let $s_1, \dots, s_m$ be integers and $s = \gcd(s_1, \dots, s_m)$. It is easy to find integer multipliers $x_1, \dots, x_m$ such that $s = x_1 s_1 + \cdots + x_m s_m$, but not so easy to find multiplier vectors X of small Euclidean length $\|X\| = (x_1^2 + \cdots + x_m^2)^{1/2}$ [Majewski and Havas 1994; Rössner and Seifert 1996]. Such multipliers may be found by performing, for example, Euclid's algorithm on s_1, s_2, to get $\gcd(s_1, s_2) = g_2$, then on g_2, s_3 and so on. If the corresponding sequence of integer row operations is performed on the identity matrix I_m, the result will be an $m \times m$ unimodular matrix P such that $PS = [0, \dots, 0, s]^T$, where $S = [s_1, \dots, s_m]^T$. Such a P is implicit in [Jacobi 1868, pages 26–28]. For some variations on this theme see [Kertzner 1981], as well as [Ford and Havas 1996] (where it is shown that one can ensure $|x_i| \le \frac{1}{2}\max(s_1, \dots, s_m)$) and [Majewski and Havas 1995] (the sorting gcd algorithm). Also see [Brentjes 1981, pages 22–24] for references to older work.

1058-6458/1998 $0.50 per page
Experimental Mathematics **7**:2, page 125

With a little matrix algebra, the equation $PS = [0, \dots, 0, s]^T$ tells us that rows $p_1, \dots, p_{m-1}$ of P form a lattice basis for the $(m-1)$-dimensional lattice Λ formed by the vectors $X = (x_1, \dots, x_m)$ with $x_1, \dots, x_m \in \mathbb{Z}$, satisfying $x_1 s_1 + \cdots + x_m s_m = 0$. In other words, every such X can be expressed as an integer linear combination $X = z_1 p_1 + \cdots + z_{m-1} p_{m-1}$. The general multiplier vector is

$$p_m + z_1 p_1 + \cdots + z_{m-1} p_{m-1},$$

with $z_1, \dots, z_{m-1} \in \mathbb{Z}$.

Lattice basis reduction can be used to find good multipliers. Such an approach dates back at least to [Rosser 1941] and [Ficken 1943], where it was used for some small examples. A particularly effective algorithm for lattice basis reduction is due to Lenstra, Lenstra and Lovász [Lenstra et al. 1982]. For a brief description of the LLL algorithm, see Section 2. Of importance in the LLL algorithm is a parameter α, which is in the range $(\frac{1}{4}, 1]$. The complexity of the algorithm increases with α, as does the quality guarantee on the basis vectors.

Earlier Approaches: Algorithm 1

One approach to the extended gcd problem, proposed by Babai [Grötschel et al. 1988, page 144] and Sims [1994, page 381], is to perform the LLL algorithm on $p_1, \dots, p_{m-1}$ to produce a lattice basis of short vectors. Then size-reduce p_m, by adding suitable multiples of these short vectors to p_m; this reduces its entries in practice to small size. We call this Algorithm 1. It has the drawback that an initial unimodular transforming matrix P has to be calculated.

Preview of Algorithms 2 and 3

Another approach to the problem is to apply the LLL algorithm to the lattice L spanned by the rows of the matrix $C = [I_m | \gamma S]$, where γ is a positive integer. It is not difficult to show that if $\gamma > y^{(m-2)/2} \|S\|$, with $y = 4/(4\alpha - 1)$ for $\frac{1}{4} < \alpha \leq 1$, the reduced basis for C must satisfy $c_{1\,m+1} = 0, \dots, c_{m-1\,m+1} = 0$ and $c_{m\,m+1} = \pm\gamma s$. Then $c_{m1}, \dots, c_{mm}$ will in practice be a small multiplier vector of similar size to that produced by Algorithm 1. We call this Algorithm 2. It has the drawback that multiplying the entries in the last column of the matrix by γ leads to an increase in the theoretical complexity and practical running time of the algorithm.

Experimentally one finds that if γ is large, Algorithm 2 seems to settle down to the same sequence of row operations. It is not difficult to identify these operations and perform them instead on the matrix $[I_m | S]$. This is justified in Section 3.

Our limiting algorithm is called Algorithm 3 and is described explicitly in Section 4.

In Section 5, we show that with $\frac{3}{8} \leq \alpha \leq 1$ the smallest multiplier for 3 numbers is one of the 7 values $b_3 + \varepsilon_1 b_1 + \varepsilon_2 b_2$, with $|\varepsilon_i| \leq 1$, $(\varepsilon_1, \varepsilon_2) \neq (\pm 1, 0)$, where multiplier b_3 and lattice basis b_1, b_2 for Λ are produced by Algorithm 3. We also derive an upper estimate in the general case of m numbers for the length of the multiplier produced by Algorithm 3 with $\frac{1}{4} < \alpha \leq 1$.

Section 6 describes a LLL-based Hermite normal form algorithm, which we also arrive at by limiting considerations. In practice the method yields transformation matrices with very small entries. Such algorithms have applications in class group calculations; see [Cohen 1993, pages 252, 288].

The paper finishes with some examples which show how well the algorithms perform.

2. THE LLL ALGORITHM

In order to analyse Algorithms 2 and 3 we need to briefly outline the LLL algorithm. More complete descriptions are given in [Grötschel et al. 1988, pages 139–150; Sims 1994, pages 360–382; Cohen 1993, pages 83–104; Pohst and Zassenhaus 1989, pages 200–202].

Let C be an $m \times n$ matrix of integers, with linearly independent rows $c_1, \dots, c_m$. The Gram–Schmidt basis is denoted by $c_1^*, \dots, c_m^*$, where

$$c_1^* = c_1, \quad c_k^* = c_k - \sum_{j=1}^{k-1} \mu_{kj} c_j^*, \quad \mu_{kj} = \frac{c_k \cdot c_j^*}{c_j^* \cdot c_j^*}.$$

We say $c_1, \ldots, c_m$ is *reduced* if $|\mu_{kj}| \leq \frac{1}{2}$ for $1 \leq j < k \leq m$ and

$$c_k^* \cdot c_k^* \geq (\alpha - \mu_{k\,k-1}^2) c_{k-1}^* \cdot c_{k-1}^* \qquad (2\text{–}1)$$

for $1 < k \leq m$. (Here $\frac{1}{4} < \alpha \leq 1$.) We say c_k is *size-reduced* if $|\mu_{kj}| \leq \frac{1}{2}$ for $1 \leq j < k$.

Assume that rows 1 to $k-1$ have been processed. The inductive step is as follows:

Do a partial size-reduction by

$$c_k \leftarrow c_k - \lceil \mu_{k\,k-1} \rfloor c_{k-1},$$

where $\lceil \theta \rfloor$ is the nearest integer symbol, with $\lceil \theta \rfloor = \theta - \frac{1}{2}$, if θ is a half-integer. If inequality (2–1) holds, size-reduce c_k completely by performing $c_k \leftarrow c_k - \lceil \mu_{kj} \rfloor c_j$ for $j = k-2, \ldots, 1$ and increment k. Otherwise swap c_k and c_{k-1} and decrement k.

3. ANALYSIS OF ALGORITHM 2

Algorithm 2 works for the following reasons. L consists of the vectors

$$(X, a) = (x_1, \ldots, x_m, \gamma(x_1 s_1 + \cdots + x_m s_m)),$$

where $x_1, \ldots, x_m \in \mathbb{Z}$. Hence $X \in \Lambda \Leftrightarrow (X, 0) \in L$. Also, if $(X, a) \in L$ and X does not belong to Λ, then $a \neq 0$ and

$$\|(X, a)\|^2 \geq \gamma^2. \qquad (3\text{–}1)$$

Further, the lemma of [Pohst and Zassenhaus 1989, page 200] implies that if $b_1, \ldots, b_{m-1}$ form a reduced basis for L, then

$$\|b_j\| \leq y^{(m-2)/2} \max\bigl(\|X_1\|, \ldots, \|X_{m-1}\|\bigr), \qquad (3\text{–}2)$$

if $X_1, \ldots, X_{m-1}$ are linearly independent vectors in L.

But the $m-1$ vectors

$$\begin{aligned} X_1 &= (-s_2, s_1, 0, \ldots, 0, 0) \\ X_2 &= (-s_3, 0, s_1, 0, \ldots, 0, 0) \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ X_{m-1} &= (-s_m, 0, 0, \ldots, s_1, 0) \end{aligned}$$

are linearly independent in L and satisfy $\|X_i\| \leq \|S\|$, so

$$\max\bigl(\|X_1\|, \ldots, \|X_{m-1}\|\bigr) \leq \|S\|. \qquad (3\text{–}3)$$

Hence if $\gamma > y^{(m-2)/2}\|S\|$, it follows from inequalities (3–1)–(3–3) that the first $m-1$ rows of a reduced basis for L have the form $(b_{j1}, \ldots, b_{jm}, 0)$.

The last vector of the reduced basis then has the form $(b_{m1}, \ldots, b_{mm}, \gamma g)$ for some g, and the equations

$$PS = \begin{bmatrix} 0 \\ g \end{bmatrix}, \quad S = P^{-1} \begin{bmatrix} 0 \\ g \end{bmatrix}$$

(where P is a unimodular matrix) imply $s|g$ and $g|s$, respectively, and hence $g = \pm s$.

Now we justify our earlier assertion that if γ is sufficiently large and the LLL algorithm is performed on $[I_m|\gamma S]$, then the sequence of operations is independent of γ.

Let $c_{i\,m+1} = \gamma a_i$ and let $C = [B|\gamma A]$, where initially $B = I_m$ and $A = S$.

Let us assume that the first $k-1$ rows of C are LLL-reduced (which implies $a_1 = 0, \ldots, a_{k-2} = 0$ by the above argument) and examine the inductive step of LLL.

First, from the equation

$$c_r^* = c_r - \sum_{j=1}^{r-1} \mu_{rj} c_j^*, \qquad (3\text{–}4)$$

we have $c_{1\,m+1}^* = 0, \ldots, c_{k-2\,m+1}^* = 0$. Also from equation (3–4), with $r = k-1$, we have $c_{k-1\,m+1}^* = \gamma a_{k-1}$.

Further,

$$\mu_{kj} = \frac{c_k \cdot c_j^*}{c_j^* \cdot c_j^*} = \frac{\sum_{q=1}^m c_{kq} c_{jq}^* + \gamma a_k c_{j\,m+1}^*}{\sum_{q=1}^m (c_{jq}^*)^2 + (c_{j\,m+1}^*)^2}. \qquad (3\text{–}5)$$

So, $c_{1\,m+1}^* = 0, \ldots, c_{k-2\,m+1}^* = 0$ and equation (3–5) give

$$\mu_{kj} = \frac{\sum_{q=1}^m c_{kq} c_{jq}^*}{\sum_{q=1}^m (c_{jq}^*)^2} \quad \text{for } j = 1, \ldots, k-2,$$

the Gram–Schmidt coefficient for C with the last column ignored.

Next

$$\mu_{k\,k-1} = \frac{\sum_{q=1}^{m} c_{kq} c^*_{k-1\,q} + \gamma^2 a_k a_{k-1}}{\sum_{q=1}^{m} (c^*_{k-1\,q})^2 + \gamma^2 a^2_{k-1}} \approx \frac{a_k}{a_{k-1}} \text{ as } \gamma \to \infty \text{ if } a_{k-1} \neq 0.$$

(If $a_{k-1} = 0$, $\mu_{k,k-1}$ reduces to the Gram–Schmidt coefficient for C with the last column ignored.) Then (for $a_{k-1} \neq 0$) if $t = \lceil a_k/a_{k-1} \rfloor$, $\lceil \mu_{k\,k-1} \rfloor = t$ if a_k/a_{k-1} is not an odd multiple of $\frac{1}{2}$, or t or $t+1$ otherwise, as $\gamma \to \infty$. Thus in the partial size-reduction, t or $t+1$ times row $k-1$ is subtracted from row k.

We now discuss the possible interchange of rows $k-1$ and k. This takes place if the inequality (2–1) fails to hold. If $a_{k-1} = 0 = a_k$, then condition (2–1) becomes the standard LLL condition involving $\mu_{k\,k-1}$.

If $a_{k-1} = 0$ but $a_k \neq 0$, then $c^*_{k\,m+1} = \gamma a_k$ and condition (2–1) will be satisfied for γ large and no interchange of rows takes place.

If $a_{k-1} \neq 0$, then since the c^*_j did not change under partial size-reduction, we see from

$$c^*_k = c_k - \sum_{j=1}^{k-2} \mu_{kj} c^*_j - \mu_{k\,k-1} c^*_{k-1},$$

that with $c^*_{j\,m+1} = 0, j = 1, \dots, k-2$ and the limiting form of the old $\mu_{k,k-1}$, we have $c^*_{k\,m+1} \approx 0$. Consequently, as $\alpha - \mu^2_{k\,k-1} > 0$ if $\alpha > \frac{1}{4}$, (2–1) will not be satisfied for γ large, if $\alpha > \frac{1}{4}$, thereby resulting in an interchange of rows.

The μ_{kj} are rational functions of γ and, if not constant, will tend to a limit strictly monotonically, thereby resulting in a limiting sequence of row operations for large γ. Thus, for large γ, the LLL algorithm will perform a version of the least-remainder gcd algorithm (LRA) on $a_1 = s_1, a_2 = s_2$, until it arrives at $a_1 = 0, a_2 = g_2 = \gcd(s_1, s_2)$, with (b_{21}, b_{22}) being the shortest multiplier vector for $\gcd(s_1, s_2)$. It then eventually performs a version of the LRA on $a_2 = g_2, a_3 = s_3$, punctuated by updating of the first three rows of B, till it arrives at $a_1 = 0, a_2 = 0, a_3 = g_3 = \gcd(g_2, s_3)$, with (b_{31}, b_{32}, b_{33}) being a short multiplier vector for $\gcd(s_1, s_2, s_3)$; and so on.

4. ALGORITHM 3

The analysis of Section 3 leads to Algorithm 3 below, the final LLL-based extended gcd algorithm. Our implementation is a modification of de Weger's LLL algorithm [1987, pages 329–332], simplified in that initial construction of the Gram–Schmidt basis is not needed, as we start with the identity matrix I_m. De Weger works in terms of integers and writes $\|b^*_i\|^2 = D_i/D_{i-1}$, $D_0 = 1$ and $\lambda_{ij} = D_j \mu_{ij}$. This allows algorithms to be implemented using only integers, avoiding explicit calculation with rationals.

The algorithm works by successively producing for $k = 2, \dots, m$, a multiplier vector $(b_{k1}, \dots, b_{kk})$ for $s_1, \dots, s_k$ which is size-reduced with respect to a LLL-reduced lattice basis $(b_{11}, \dots, b_{1k})$, …, $(b_{k-1 1}, \dots, b_{k-1 k})$ for the lattice defined by $x_1 s_1 + \dots + x_k s_k = 0$.

Algorithm 3. (We denote by $\boldsymbol{b}_i$ the i-th row of B.)
Input: Positive integers $s_1, \dots, s_m$
Output: $a_m = \gcd(s_1, \dots, s_m)$; small multipliers $b_{m1}, \dots, b_{mm}$; small null space basis

$B \leftarrow I_m$;
for $r = 2, \dots, m$ **do**
  for $s = 1, \dots, r-1$ **do** $\lambda_{rs} \leftarrow 0$;
for $i = 0, \dots, m$ **do** $D_i \leftarrow 1$;
for $i = 1, \dots, m$ **do** $a_i \leftarrow s_i$;
$m_1 \leftarrow 3$; $n_1 \leftarrow 4$; /* $\alpha = m_1/n_1$ */
$k \leftarrow 2$;
while $k \le m$ **do**
  Reduce1$(k,\ k-1)$;
  if $a_{k-1} \neq 0$ **or** $\big(a_{k-1} = 0$ **and** $a_k = 0$ **and**
    $n_1(D_{k-2}D_k + \lambda^2_{k\,k-1}) < m_1 D^2_{k-1}\big)$ **then**
   Swap(k); **if** $k > 2$ **then** $k \leftarrow k-1$;
  else
   for $i = k-2, \dots, 1$ **do** *Reduce*(k, i);
   $k \leftarrow k+1$;
if $a_m < 0$ **then**
  $a_m \leftarrow -a_m$; $\boldsymbol{b}_m \leftarrow -\boldsymbol{b}_m$;

Reduce1(k, i)
  if $a_i \neq 0$ **then** $q \leftarrow \lceil a_k/a_i \rfloor$;
  else
    if $2|\lambda_{ki}| > D_i$ **then** $q \leftarrow \lceil \lambda_{ki}/D_i \rfloor$;
    else $q \leftarrow 0$;
  if $q \neq 0$ **then**
    $a_k \leftarrow a_k - qa_i$;
    $\boldsymbol{b}_k \leftarrow \boldsymbol{b}_k - q\boldsymbol{b}_i$;
    $\lambda_{ki} \leftarrow \lambda_{ki} - qD_i$;
    for $j = 1, \ldots, i-1$ **do** $\lambda_{kj} \leftarrow \lambda_{kj} - q\lambda_{ij}$;

Swap(k)
  $a_k \leftrightarrow a_{k-1}$;
  $\boldsymbol{b}_k \leftrightarrow \boldsymbol{b}_{k-1}$;
  for $j = 1, \ldots, k-2$ **do** $\lambda_{kj} \leftrightarrow \lambda_{k-1\,j}$;
  for $i = k+1, \ldots, m$ **do**
    $t \leftarrow \lambda_{i\,k-1} D_k - \lambda_{ik}\lambda_{k\,k-1}$;
    $\lambda_{i\,k-1} \leftarrow (\lambda_{i\,k-1}\lambda_{k\,k-1} + \lambda_{ik}D_{k-2})/D_{k-1}$;
    $\lambda_{ik} \leftarrow t/D_{k-1}$;
  $D_{k-1} \leftarrow (D_{k-2}D_k + \lambda^2_{k\,k-1})/D_{k-1}$;

5. MULTIPLIER ESTIMATES

Even when $m = 3$, our LLL-based gcd algorithm does not always produce the shortest multiplier: in the example $4, 6, 9$, LLL (for all $\frac{1}{4} < \alpha \leq 1$) produces the multiplier $b_3 = (-2, 0, 1)$, whereas the shortest is $b_3 + b_2 + b_1 = (1, 1, -1)$.

After much numerical experiment we were led to the following result:

Theorem 5.1. *If B is a unimodular 3×3 integer matrix such that the first 2 rows b_1, b_2 form a LLL-reduced basis for the lattice Λ with $\frac{3}{8} \leq \alpha \leq 1$, while b_3 is size-reduced and is a multiplier vector for s_1, s_2, s_3, then the smallest multiplier is one of seven vectors $b_3 + \varepsilon_1 b_1 + \varepsilon_2 b_2$, where $\varepsilon_i = -1, 0, 1$ for $i = 1, 2$ and $(\varepsilon_1, \varepsilon_2) \neq (\pm 1, 0)$.*

Proof. We have

$$b_3 = b_3^* + \mu_{32} b_2^* + \mu_{31} b_1^*$$

and

$$b_2 = b_2^* + \mu_{21} b_1^*,$$

where $|\mu_{ij}| \leq \frac{1}{2}$. Then, if $u, v \in \mathbb{Z}$, recalling that $b_3 + ub_1 + vb_2$ is the general multiplier, we have the following expression for the square of its length:

$$\begin{aligned} f(u,v) &\\ &= \|b_3 + ub_1 + vb_2\|^2 \\ &= \|b_3^* + (u + \mu_{31} + v\mu_{21}) b_1^* + (v + \mu_{32}) b_2^*\|^2 \\ &= \|b_3^*\|^2 + (u + \mu_{31} + v\mu_{21})^2 \|b_1^*\|^2 + (v + \mu_{32})^2 \|b_2^*\|^2. \end{aligned}$$

Suppose $f(x, y) < f(0, 0)$. Then

$$\|b_3 + xb_1 + yb_2\|^2 < \|b_3\|^2,$$

or, equivalently,

$$\begin{aligned} &\|b_3^*\|^2 + (x + \mu_{21}y + \mu_{31})^2 \|b_1^*\|^2 + (y + \mu_{32})^2 \|b_2^*\|^2 \\ &\qquad < \|b_3^*\|^2 + \mu_{31}^2 \|b_1^*\|^2 + \mu_{32}^2 \|b_2^*\|^2. \end{aligned}$$

Hence, noting that $\|b_2^*\|^2 \geq \|b_1\|^2(\alpha - \mu_{21}^2)$, we have

$$\begin{aligned} (y + \mu_{32})^2 &< \mu_{31}^2 \frac{\|b_1^*\|^2}{\|b_2^*\|^2} + \mu_{32}^2, \\ (y + \mu_{32})^2 &< \tfrac{1}{4} \cdot 8 + \tfrac{1}{4} = \tfrac{9}{4}, \text{ if } \alpha \geq \tfrac{3}{8}, \\ |y + \mu_{32}| &< \tfrac{3}{2} \Rightarrow |y| < 2 \Rightarrow |y| \leq 1. \end{aligned}$$

Then, since $y(y + 2\mu_{32}) \geq 0$ if $y \in \mathbb{Z}$,

$$\begin{aligned} &(x + \mu_{21}y + \mu_{31})^2 \|b_1\|^2 \\ &\qquad < (x + \mu_{21}y + \mu_{31})^2 \|b_1\|^2 + y(y + 2\mu_{32}) \|b_2^*\|^2 \\ &\qquad < \mu_{31}^2 \|b_1^*\|^2; \end{aligned}$$

hence

$$\begin{aligned} |x + \mu_{21}y + \mu_{31}| &< |\mu_{31}|, \\ |x| &< |\mu_{21}y + \mu_{31}| + |\mu_{31}|, \qquad (5\text{–}1) \\ |x| &< |\mu_{21}| + 2|\mu_{31}| \leq \tfrac{1}{2} + 1, \\ |x| &\leq 1. \end{aligned}$$

Also from inequality (5–1), $y = 0$ implies $x = 0$. $\square$

Remarks. 1. We can be more specific about the optimum multipliers, in terms of the signs of μ_{21}, μ_{31}, and μ_{32}:

μ_{21}	μ_{31}	μ_{32}	Optimum multiplier
+	+	+	b_3 or $b_3 - b_2$
+	−	−	b_3 or $b_3 + b_2$
−	+	−	b_3 or $b_3 + b_2$
−	−	+	b_3 or $b_3 - b_2$
−	−	−	b_3 or $b_3 - b_2$ or $b_3 + b_1 + b_2$
+	+	−	b_3 or $b_3 - b_2$ or $b_3 - b_1 + b_2$
−	+	+	b_3 or $b_3 + b_2$ or $b_3 - b_1 - b_2$
+	−	+	b_3 or $b_3 + b_2$ or $b_3 + b_1 - b_2$

All cases occur for suitable choices of (s_1, s_2, s_3).

2. The theorem is not true for all $\alpha > \frac{1}{4}$. For with $(s_1, s_2, s_3) = (1000003, 1000021, 1000073)$ and $\alpha = \frac{100001}{400000}$, the optimum multiplier is

$$b_3 + 1710b_1 - 3421b_2 = (11088, 4306, -15393),$$

where $b_1 = (-222224, 222219, 1)$, $b_2 = (-111099, 111092, 5)$, and $b_3 = (-55551, 55548, 2)$.

Corollary 5.2. *Our LLL extended gcd algorithm is the basis of a practical polynomial-time algorithm for finding an optimal solution to the extended gcd problem for* 3 *numbers.*

Proof. Apply Algorithm 3 (with $\alpha = \frac{3}{8}$, say) and then check which of the possibilities is optimal. □

Theorem 5.3. *Let B be a unimodular $m \times m$ integer matrix such that the first $m-1$ rows form a LLL-reduced basis for the lattice Λ with $\frac{1}{4} < \alpha \le 1$, while b_m is size-reduced and is a multiplier vector for $s_1, \ldots, s_m$. Then with $y = 4/(4\alpha - 1)$, we have*

$$\|b_m\|^2 \le 1 + \tfrac{1}{4}(m-1)y^{m-2}\|S\|^2.$$

Proof. We have $b_m = b_m^* + \sum_{j=1}^{m-1} \mu_{mj} b_j^*$. The vector S^T is orthogonal to $b_1, \ldots, b_{m-1}$ and we see from $BS = [0, \ldots, 0, s]^T$ that $b_m^* = sS^T/\|S\|^2$. Then

$$b_m = \frac{sS^T}{\|S\|^2} + \sum_{j=1}^{m-1} \mu_{mj} b_j^*$$

with $|\mu_{mj}| \le \frac{1}{2}$ for $j = 1, \ldots, m-1$.

But $\|b_i^*\| \le \|b_i\|$. Hence

$$\|b_m\|^2 \le 1 + \frac{1}{4}\sum_{j=1}^{m-1} \|b_j^*\|^2 \le 1 + \frac{1}{4}\sum_{j=1}^{m-1} \|b_j\|^2. \quad (5–2)$$

Repeating the argument for inequalities (3–2) and (3–3) we deduce that

$$\|b_j\| \le y^{(m-2)/2}\|S\|, \quad \text{for } j = 1, \ldots, m-1,$$

and inequality (5–2) gives

$$\|b_m\|^2 \le 1 + \tfrac{1}{4}(m-1)y^{m-2}\|S\|^2,$$

as required. □

Remark. Knuth [1969, Section 4.5.3, Problem 33; 1981, Section 4.5.3, Problem 45] posed research-level problems on the analysis of algorithms for computing the greatest common divisor of three or more integers. The theorems and corollary of this section resolve aspects of these problems. Algorithm 3 and its analysis provide much further information.

6. A LLL-BASED HERMITE NORMAL-FORM ALGORITHM

An $m \times n$ integer matrix H is said to be in row Hermite normal form if

(i) the first r rows of H are nonzero;
(ii) for $1 \le i \le r$, if h_{ij_i} is the first nonzero entry in row i of H, then $j_1 < j_2 < \cdots < j_r$;
(iii) $h_{ij_i} > 0$ for $1 \le i \le r$;
(iv) if $1 \le k < i \le r$, then $0 \le h_{kj_i} < h_{ij_i}$.

Let G be an $m \times n$ integer matrix. Then there are various algorithms for finding a unimodular matrix P such that $PG = H$ is in row Hermite normal form. These include those of Kannan–Bachem [1994, pages 349–357] and Havas–Majewski [1994], which attempt to reduce coefficient explosion during their execution.

By considering the limiting behaviour of the LLL algorithm on the matrix

$$G(\gamma) = [I_m \mid \gamma^n G_1 \mid \gamma^{n-1} G_2 \mid \cdots \mid \gamma G_n]$$

(where G_i is the i-th column of G) as $\gamma \to \infty$, we are led to the following LLL-based Hermite normal form algorithm, generalising the earlier gcd case where $n = 1$. (We have omitted *Swap*(k) because it is the same as in Algorithm 3, with $\boldsymbol{a}_i$ replacing a_i.) It is an easy extension of the argument in Section 1 to show that for large γ, on LLL reducing $G(\gamma)$, the last n columns form a matrix whose rows, starting from the bottom, are in row echelon form, corresponding to the indices $j_1, \ldots, j_r$. This algorithm gives as output A, which is the Hermite normal form of G, but in reverse order. It is trivial to turn A and the transforming matrix B into the standard H and P.

Algorithm 4 (LLL-based Hermite normal form).
Input: An $m \times n$ integer matrix G
Output: A, the Hermite normal form of G (upside down), and B, the corresponding transformation matrix

$B \leftarrow I_m$;
for $r = 2, \ldots, m$ **do**
  for $s = 1, \ldots, r-1$ **do** $\lambda_{rs} \leftarrow 0$;
$A \leftarrow G$;
for $i = 0, \ldots, m$ **do** $D_i \leftarrow 1$;
$m_1 \leftarrow 3$; $n_1 \leftarrow 4$; /* $\alpha = m_1/n_1$ */
if there is exactly one nonzero element a_{ij} in the first nonzero column of A **and** $i = m$ **and** $a_{mj} < 0$ **then**
  $\boldsymbol{a}_m \leftarrow -\boldsymbol{a}_m$; $b_{m,m} = -1$;
$k \leftarrow 2$;
while $k \le m$ **do**
  Reduce2$(k, k-1)$;
  if col1 $\le \min(\text{col2}, n)$ **or** $\big(\text{col1} = \text{col2} = n+1$ **and** $n_1(D_{k-2}D_k + \lambda_{k\,k-1}^2) < m_1 D_{k-1}^2\big)$ **then**
    Swap(k); **if** $k > 2$ **then** $k \leftarrow k-1$;
  else
    for $i = k-2, \ldots, 1$ **do** *Reduce2*(k, i);
    $k \leftarrow k+1$;

Reduce2(k, i)
  if there is j such that $a_{i,j} \ne 0$ **then**
    col1 $\leftarrow$ least j such that $a_{i,j} \ne 0$;
    if $a_{i,\text{col1}} < 0$ **then**
      Minus(i); $\boldsymbol{a}_i \leftarrow -\boldsymbol{a}_i$; $\boldsymbol{b}_i \leftarrow -\boldsymbol{b}_i$;
  else col1 $\leftarrow n+1$;
  if there is j such that $a_{k,j} \ne 0$ **then**
    col2 $\leftarrow$ least j such that $a_{k,j} \ne 0$;
  else col2 $\leftarrow n+1$;
  if col1 $\le n$ **then** $q \leftarrow \lfloor a_{k,\text{col1}}/a_{i,\text{col1}} \rfloor$;
  else
    if $2|\lambda_{ki}| > D_i$ **then** $q \leftarrow \lceil \lambda_{ki}/D_i \rfloor$;
    else $q \leftarrow 0$;
  if $q \ne 0$ **then**
    $\boldsymbol{a}_k \leftarrow \boldsymbol{a}_k - q\boldsymbol{a}_i$;
    $\boldsymbol{b}_k \leftarrow \boldsymbol{b}_k - q\boldsymbol{b}_i$;
    $\lambda_{ki} \leftarrow \lambda_{ki} - qD_i$;
    for $j = 1, \ldots, i-1$ **do** $\lambda_{kj} \leftarrow \lambda_{kj} - q\lambda_{ij}$;

Minus(j)
  for $r = 2, \ldots, m$ **do**
    for $s = 1, \ldots, r-1$ **do**
      if $r = j$ **or** $s = j$ **then** $\lambda_{rs} \leftarrow -\lambda_{rs}$;

We remark that if a row of G has to be multiplied by -1, there is a necessary adjustment for the λ_{ij}. Hence the function *Minus*(i).

Let C denote the submatrix of H formed by the r nonzero rows and write $P = \left[\begin{smallmatrix} Q \\ R \end{smallmatrix}\right]$, where Q and R have r and $m-r$ rows, respectively. (Q corresponds to the bottom r rows of A in reverse order. R comprises the top $m-r$ rows of A.) Then $QG = C$ and $RG = 0$ and the rows of R will form a $\mathbb{Z}$ basis of short vectors for the sublattice $N(G)$ of $\mathbb{Z}^m$ formed by the vectors X satisfying $XG = 0$. The rows of Q are size-reduced with respect to the short lattice basis vectors for $N(G)$.

7. EXAMPLES

We have applied the methods described here to numerous examples, all with excellent performance. Note that there are many papers which study explicit input sets for the extended gcd problem and a number of these are listed in the references of

[Brentjes 1981] and [Majewski and Havas 1994]. We illustrate algorithm performance with a small selection of interesting examples and make some performance comparisons.

Many parameters can affect the performance of LLL lattice basis reduction algorithms (this has been observed by many others; see [Schnorr and Euchner 1991], for example). Foremost is the value of α. Smaller values of α tend to give faster execution times but worse multipliers; however this is by no means uniform. Also, the order of input may have an effect.

Example 7.1. As input to an extended gcd algorithm, take $s_1 = 116085838$, $s_2 = 181081878$, $s_3 = 314252913$, $s_4 = 10346840$. Algorithm 3 produces a final matrix

$$B = \begin{bmatrix} -103 & 146 & -58 & 362 \\ -603 & 13 & 220 & -144 \\ 15 & -1208 & 678 & 381 \\ -88 & 352 & -167 & -101 \end{bmatrix}.$$

The multiplier vector $(-88, 352, -167, -101)$ is the unique multiplier vector of least length. In fact, LLL-based methods give this optimal multiplier vector for all $\alpha \in \left(\frac{1}{4}, 1\right]$.

Earlier algorithms which aim to improve on the multipliers do not fare particularly well. Blankinship's algorithm [1963] gives the multiplier vector

$$(0, 355043097104056, 1, -6213672077130712).$$

The algorithm of [Bradley 1970] gives (27237259, −17460943, 1, 0). (This shows that Bradley's definition of minimal is not useful.)

Example 7.2. Take

$$\begin{aligned} s_1 &= 763836, & s_2 &= 1066557, & s_3 &= 113192, \\ s_4 &= 1785102, & s_5 &= 1470060, & s_6 &= 3077752, \\ s_7 &= 114793, & s_8 &= 3126753, & s_9 &= 1997137, \end{aligned}$$
$$s_{10} = 2603018.$$

Algorithm 3 gives, for various values of α, the multiplier vectors shown at the top of the next column. Also shown are the lengths-squared.

α	multiplier vector x										$\|x\|^2$
$\frac{101}{400}$	7	−1	−5	−1	−1	0	−4	0	0	0	93
$\frac{1}{3}$	−1	0	6	−1	−1	1	0	2	−3	0	53
$\frac{1}{2}$	−3	0	3	0	−1	1	0	1	−4	2	41
$\frac{2}{3}$	1	−3	2	−1	5	0	1	1	−2	−1	47
$\frac{3}{4}$	1	−3	2	−1	5	0	1	1	−2	−1	47
1	−1	0	1	−3	1	3	3	−2	−2	2	42

The unique shortest multiplier vector is

$$(3, -1, 1, 2, -1, -2, -2, -2, 2, 2),$$

with length-squared 36. Other methods give the following results:

Jacobi:

$$(-14, 5, -2, 3, -1, 2, -4, 0, -2, 0), \quad \|x\|^2 = 259$$

recursive gcd:

$$(1936732230, -1387029291, -1, 0, 0, 0, 0, 0, 0, 0)$$

Kannan–Bachem:

$$(44537655090, -31896527153, 0, 0, 0, 0, 0, 0, 0, -1)$$

Blankinship:

$$(3485238369, 1, -23518892995, 0, 0, 0, 0, 0, 0, 0)$$

Bradley: $(-135282, 96885, -1, 0, 0, 0, 0, 0, 0, 0)$

Example 7.3. The following example involving the Fibonacci number F_i and the Lucas numbers $L_i = F_{i-1} + F_{i+1}$ [Hoggatt 1969] has theoretical significance. Take $s_1, \dots, s_m$ to be the Fibonacci numbers

$$\begin{aligned} &F_n, F_{n+1}, \dots, F_{2n} && \text{for } n \geq 5 \text{ odd}, \\ &F_n, F_{n+1}, \dots, F_{2n-1} && \text{for } n \geq 4 \text{ even}. \end{aligned}$$

Using the identity $F_i L_j = F_{i+j} + (-1)^j F_{i-j}$, it can be shown that the vectors

$$(-L_{n-3}, L_{n-4}, \dots, -L_2, L_1, -1, 1, 0, 0)$$

and

$$(L_{n-3}, -L_{n-4}, \dots, -L_2, L_1{+}1, -1, 0, 0)$$

$$
\begin{bmatrix}
3 & 7 & 13 & 21 & 31 & 43 & 57 & 73 & 91 & 111 \\
11 & 36 & 77 & 134 & 207 & 296 & 401 & 522 & 659 & 812 \\
31 & 113 & 249 & 439 & 683 & 981 & 1333 & 1739 & 2199 & 2713 \\
69 & 262 & 583 & 1032 & 1609 & 2314 & 3147 & 4108 & 5197 & 6414 \\
131 & 507 & 1133 & 2009 & 3135 & 4511 & 6137 & 8013 & 10139 & 12515 \\
223 & 872 & 1953 & 3466 & 5411 & 7788 & 10597 & 13838 & 17511 & 21616 \\
351 & 1381 & 3097 & 5499 & 8587 & 12361 & 16821 & 21967 & 27799 & 34317 \\
521 & 2058 & 4619 & 8204 & 12813 & 18446 & 25103 & 32784 & 41489 & 51218 \\
739 & 2927 & 6573 & 11677 & 18239 & 26259 & 35737 & 46673 & 59067 & 72919 \\
1011 & 4012 & 9013 & 16014 & 25015 & 36016 & 49017 & 64018 & 81019 & 100020
\end{bmatrix}
$$

Input matrix G of Example 7.4.

are multipliers in the case of n odd or even, respectively. These multipliers are the unique vectors of least length. (This is a special case of a more general result from [Matthews 1996], where $F_n, \dots, F_{n+m}$ is treated.) The length-squared of the multipliers is $L_{2n-5}+1$ in both cases. (In practice, the LLL-based algorithms compute these minimal multipliers.)

These results give bounds for extended gcd multipliers in terms of Euclidean norms. Since

$$L_{2n-5}+1 \sim \varphi^{2n-5} \sim \varphi^{-5}\sqrt{5}\,F_{2n},$$

where $\varphi = \frac{1}{2}\big(1+\sqrt{5}\big)$, a general upper bound for the Euclidean norm of the multiplier vector in terms of the initial numbers s_i must be at least

$$O\big(\sqrt{\max\{s_i\}}\big).$$

Also, the length of the vector $(F_n, F_{n+1}, \dots, F_{2n})$ is of the same order of magnitude as F_{2n}, so a general upper bound for the length of the multipliers in terms of the Euclidean length of the input, $\|S\|$, is at least $O\big(\sqrt{\|S\|}\big)$.

A range of random type extended gcd examples is presented in [Havas and Majewski 1995], showing excellent performance.

Example 7.4. For a Hermite normal form example, take $G = [g_{ij}]$ to be the 10×10 matrix defined by $g_{ij} = i^3 j^2 + i + j$, and spelled out at the top of the page. The Hermite normal form H of G has three nonzero rows:

$$
\begin{bmatrix}
1 & 0 & 7 & 22 & 45 & 76 & 115 & 162 & 217 & 280 \\
0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 \\
0 & 0 & 12 & 36 & 72 & 120 & 180 & 252 & 336 & 432
\end{bmatrix}.
$$

The unimodular matrix provided by the Kannan–Bachem algorithm is

$$
\begin{bmatrix}
-48 & 47 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8 & 10 & -5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-62 & 57 & -12 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
982 & -2620 & 2295 & -658 & 1 & 0 & 0 & 0 & 0 & 0 \\
1684 & -4495 & 3940 & -1130 & 0 & 1 & 0 & 0 & 0 & 0 \\
2662 & -7108 & 6233 & -1788 & 0 & 0 & 1 & 0 & 0 & 0 \\
3962 & -10582 & 9282 & -2663 & 0 & 0 & 0 & 1 & 0 & 0 \\
5630 & -15040 & 13195 & -3786 & 0 & 0 & 0 & 0 & 1 & 0 \\
7712 & -20605 & 18080 & -5188 & 0 & 0 & 0 & 0 & 0 & 1 \\
-3 & 8 & -7 & 2 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},
$$

whereas that supplied by our algorithm is

$$
\begin{bmatrix}
-10 & -8 & -5 & 1 & 2 & 3 & 5 & 3 & 0 & -4 \\
-2 & -1 & 0 & 1 & -1 & 0 & 1 & 0 & 1 & -1 \\
-15 & -11 & -4 & 0 & 4 & 5 & 4 & 3 & 1 & -5 \\
1 & -1 & -1 & 0 & 2 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & -1 & 1 & -1 & 2 & -1 & 0 & 0 \\
1 & 0 & -1 & -1 & -1 & 2 & 0 & 1 & -1 & 0 \\
1 & 0 & -2 & 1 & -1 & 1 & -1 & 1 & 1 & -1 \\
-1 & 0 & 1 & 0 & 1 & 1 & -1 & -2 & 0 & 1 \\
1 & -1 & 0 & -1 & 1 & 0 & 0 & -1 & 2 & -1 \\
1 & -2 & 1 & 1 & -2 & 0 & 2 & -1 & 0 & 0
\end{bmatrix}.
$$

	Kannan–Bachem	LLL HNF
M_{100}	7771095311932325021082596842776392573O604	2
M_{110}	19688024435949960842280085386879376037295267254	2
M_{120}	3280791267763785034188299O335	4
M_{130}	258209178730643422634648900270488370181908068255159901037863837761499488802313834227	5
M_{140}	8877061573605684598479855792299	9
M_{150}	143547860664185870781020896285	9

Entry of maximal magnitude for the transforming matrices obtained by applying the Kannan–Bachem algorithm and ours to the matrices of Example 7.5.

Example 7.5. An interesting family of matrices arises in the work of Daberkow [1995]. In some situations involving ideal class groups, matrices arise with k rows and 10 columns for k ranging from 100 to 150 in steps of 10. We designate the matrix with k rows by M_k. The maximal magnitude entry in M_k is of the order $11^{(k-90)/10}$. Daberkow needs to compute both the Hermite normal form and a transforming matrix. We tabulate above the maximal magnitude entry in the transforming matrix (which includes many entries of this size) for our algorithm using $\alpha = 1$ in comparison with that of Kannan–Bachem.

It can be seen from these examples that the algorithm we propose here often dramatically outperforms earlier methods in the search for good multipliers and transforming matrices.

ACKNOWLEDGMENTS

Havas and Majewski were supported by the Australian Research Council. We are grateful to Jean-Pierre Seifert and Clemens Wagner for helpful discussions and to a referee for improvements to the manuscript.

ELECTRONIC AVAILABILITY

Implementations of these algorithms are available in Matthews' number theory calculator program CALC at http://www.maths.uq.edu.au/~krm/.

Variants of the algorithms are available in GAP [Schönert et al. 1996] and Magma [Bosma et al. 1997].

REFERENCES

[Blankinship 1963] W. A. Blankinship, "A new version of the Euclidean algorithm", *Amer. Math. Monthly* **70** (1963), 742–745.

[Bosma et al. 1997] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system, I: The user language", *J. Symbolic Comput.* **24**:3-4 (1997), 235–265. See http://www.maths.usyd.edu.au:8000/comp/magma/Overview.html.

[Bradley 1970] G. H. Bradley, "Algorithm and bound for the greatest common divisor of n integers", *Comm. ACM* **13** (1970), 433–436.

[Brentjes 1981] A. J. Brentjes, *Multidimensional continued fraction algorithms*, Mathematical Centre Tracts **145**, Mathematisch Centrum, Amsterdam, 1981.

[Cohen 1993] H. Cohen, *A course in computational algebraic number theory*, Graduate Texts in Mathematics **138**, Springer, Berlin, 1993.

[Daberkow 1995] M. Daberkow, *Über die Bestimmung der ganzen Elemente in Radikalerweiterungen algebraischer Zahlkörper*, Dissertation, Tech. Univ. Berlin, Berlin, 1995.

[de Weger 1987] B. M. M. de Weger, "Solving exponential Diophantine equations using lattice basis reduction algorithms", *J. Number Theory* **26**:3 (1987), 325–367. Erratum in **31**:1 (1989), 88–89.

[Ficken 1943] F. A. Ficken, "Rosser's generalization of the Euclid algorithm", *Duke Math. J.* **10** (1943), 355–379.

[Ford and Havas 1996] D. Ford and G. Havas, "A new algorithm and refined bounds for extended GCD computation", pp. 145–150 in *Algorithmic number* *theory* (Talence, 1996), edited by H. Cohen, Lecture Notes in Comput. Sci. **1122**, Springer, Berlin, 1996.

[Grötschel et al. 1988] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization*, Algorithms and Combinatorics **2**, Springer, Berlin, 1988.

[Havas and Majewski 1994] G. Havas and B. S. Majewski, "Hermite normal form computation for integer matrices", *Congr. Numer.* **105** (1994), 87–96.

[Havas and Majewski 1995] G. Havas and B. S. Majewski, "Extended gcd calculation", *Congr. Numer.* **111** (1995), 104–114.

[Hoggatt 1969] V. E. Hoggatt, Jr., *Fibonacci and Lucas Numbers*, Houghton Mifflin Co., Boston, 1969.

[Jacobi 1868] C. G. J. Jacobi, "Über die Auflösung der Gleichung $\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n = f \cdot u$", *J. Reine Angew. Math.* **69** (1868), 1–28.

[Kertzner 1981] S. Kertzner, "The linear Diophantine equation", *Amer. Math. Monthly* **88**:3 (1981), 200–203.

[Knuth 1969] D. E. Knuth, *The art of computer programming, Vol. 2: Seminumerical algorithms*, 1st ed., Addison-Wesley, Reading, MA, 1969.

[Knuth 1981] D. E. Knuth, *The art of computer programming, Vol. 2: Seminumerical algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[Lenstra et al. 1982] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász, "Factoring polynomials with rational coefficients", *Math. Ann.* **261**:4 (1982), 515–534.

[Majewski and Havas 1994] B. S. Majewski and G. Havas, "The complexity of greatest common divisor computations", pp. 184–193 in *Algorithmic number theory* (Ithaca, NY, 1994), edited by L. M. Adleman and M.-D. Huang, Lecture Notes in Comput. Sci. **877**, Springer, Berlin, 1994.

[Majewski and Havas 1995] B. S. Majewski and G. Havas, "A solution to the extended gcd problem", pp. 248–253 in *ISSAC'95: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation* (Montreal, 1995), edited by A. H. M. Levelt, ACM Press, New York, 1995.

[Matthews 1996] K. R. Matthews, "Minimal multipliers for consecutive Fibonacci numbers", *Acta Arith.* **75**:3 (1996), 205–218.

[Pohst and Zassenhaus 1989] M. Pohst and H. Zassenhaus, *Algorithmic algebraic number theory*, Encyclopedia of Mathematics and its Applications **30**, Cambridge University Press, Cambridge, 1989.

[Rosser 1941] B. Rosser, "A note on the linear Diophantine equation", *Amer. Math. Monthly* **48** (1941), 662–666.

[Rössner and Seifert 1996] C. Rössner and J.-P. Seifert, "The complexity of approximate optima for greatest common divisor computations", pp. 307–322 in *Algorithmic number theory* (Talence, 1996), edited by H. Cohen, Lecture Notes in Comput. Sci. **1122**, Springer, Berlin, 1996.

[Schnorr and Euchner 1991] C.-P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems", pp. 68–85 in *Fundamentals of computation theory* (Gosen, 1991), edited by L. Budach, Lecture Notes in Comput. Sci. **529**, Springer, Berlin, 1991.

[Schönert et al. 1996] M. Schönert et al., *GAP: Groups, algorithms, and programming*, Lehrstuhl D für Mathematik, RWTH Aachen, 1996. See ftp://dimacs.rutgers.edu/pub/ or http://www-gap.dcs.st-and.ac.uk/~gap.

[Sims 1994] C. C. Sims, *Computation with finitely presented groups*, Encyclopedia of Mathematics and its Applications **48**, Cambridge University Press, Cambridge, 1994.

(addresses overleaf)

George Havas, George Havas, Centre for Discrete Mathematics and Computing, Department of Computer Science and Electrical Engineering, The University of Queensland, Queensland 4072, Australia (havas@csee.uq.edu.au, http://www.it.uq.edu.au/~havas/)

Bohdan S. Majewski, Department of Computer Science and Software Engineering, University of Newcastle, Callaghan, NSW 2308, Australia (bohdan@cs.newcastle.edu.au, http://wwwcs.newcastle.edu.au/Staff/bohdan/)

Keith R. Matthews, Centre for Discrete Mathematics and Computing, Department of Mathematics, The University of Queensland, Queensland 4072, Australia (krm@maths.uq.edu.au, http://www.maths.uq.edu.au/~krm/)

Received May 1, 1997; accepted in revised form July 18, 1997