# Half−GCD, Fast Rational Recovery, and Planar Lattice Reduction

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Center Dr.
Champaign IL 61820

danl@wolfram.com

**ABSTRACT**. Over the past few decades several variations on a "half GCD" algorithm for obtaining the pair of terms in the middle of a Euclidean sequence have been proposed. In the integer case algorithm design and proof of correctness are complicated by the effect of carries. This paper will demonstrate a variant with a relatively simple proof of correctness. We then apply this to rational recovery for a linear algebra solver. After showing how this same task might be accomplished by lattice reduction, albeit more slowly, we proceed to use the half GCD to obtain asymptotically fast planar lattice reduction.

This is an extended version of a paper presented at ISSAC 2005 [17]. It also contains minor changes.

## Categories and Subject Descriptors

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

## General Terms

Algorithms, Performance

## Keywords

integer gcd, subquadratic arithmetic, rational recovery

## 1. INTRODUCTION AND RELATED WORK

The "half GCD" (HGCD) algorithm, as described in [19] and [1], works by taking the high parts of a pair of inputs in a Euclidean domain, first recursively finding the pair of elements in the Euclidean sequence for that pair that straddles the middle of the larger input as well as the $2 \times 2$ matrix that converts to this middle pair. It then uses these values to convert the original pair to something sufficiently smaller. This is repeated one time, along with Euclidean steps at appropriate points, in such a way that one obtains the corresponding middle values for the original pair of inputs. Various analyses explain why this is asymptotically fast compared to direct computation of the full Euclidean sequence (we refer below to this latter as the "standard" Euclidean algorithm). The method itself is loosely based on an earlier asymptotically fast algorithm presented for continued fractions in [22]. As is indicated in that work, it in turn can be adapted to find a GCD although there appears to be some extra bookkeeping. That work was in turn an improvemnt on a slower though still subquadratic method presented in [14].

Since its introduction in the early 1970’s, the asymptotically fast HGCD idea has given rise to several variants and descriptions thereof. This state of affairs has come to pass because of difficulties encountered in proofs of correctness. It turns out that the integer case is particularly troublesome due to the possibility of carries that may cause intermediate values to be too large or too small relative to what the algorithm requires. Several papers ([3], [25], and [20]) redress this with fix−up steps that involve a limited number of Euclidean steps or reversals thereof. These papers tend to have proofs that involve analysis of many detailed cases, thus making them difficult to follow, let alone implement. (To be fair, they strive for greater generality in some respects). The main contribution of this paper is to provide a simple formulation with straightforward proofs. We should mention that the method of [22] is not known to this author to suffer from issues of correctness, though for GCD purposes it is likely to be a bit slower and is also not as convenient as HGCD for purposes of rational recovery.

As testimony to its relative simplicity, the gcd method we present is now implemented as of version 5.1 of *Mathematica* (TM) [29]. It is an improved version of that which appeared in version 5.0. The prior work was coded by David Terr, with assistance from Mark Sofroniou and the author, in early 2001. It could be described as a "Las Vegas" approach insofar as it is always correct but only probabilistically fast; in practice we have never noticed it to falter. The fully deterministic method of this paper was coded by the author and Mark Sofroniou.

Some important uses of asymptotically fast gcd to date are in finding greatest common divisors of pairs of large univari−ate polynomials or integers. An important advantage it enjoys is that, with little loss in efficiency, it finds corresponding cofactors when needed (that is, it computes the extended gcd). This is required, for example, in Hermite normal form computations. Moreover in finding cofactors for steps that take us half the distance to the gcd, the HGCD is ideally suited to fast recovery of rationals from $p$−adic images (as we will see, the code involved is trivial). The second contribution of this paper is to show this as applied to linear equation solving. This will give some indication of speed improvement over a standard Euclidean algorithm based recovery method. We will also describe a method of rational recovery based on planar lattice reduction, and, reversing the process, show how to do fast planar lattice reduction via HGCD.

In another recent paper, [24] take a different direction by operating on the low (rather than high) end of the inputs. This has the advantage that carries are no longer an issue. A possible drawback is that rational recovery becomes slightly less transparent though they show how it may still be done. They present a timing comparison that clearly demonstrates the efficacy of their code. At present it is not simple to compare directly to ours, due to different installations of the underlying GMP bignum arithmetic facility [12] as well as possible differences in memory management and timing thereof, but they appear to be in the same ballpark. I have also learned that recent work described in [18] and [21] is similar to the present work in regard to asymptotically fast GCD computation. The former is quite promising insofar as there is efficient code written for comparison of several related approaches. It is expected that the best will eventually go into public domain software [12].

I thank two anonymous referees for detailed remarks and suggestions that improved the exposition of this paper, and thank the second referee as well for bringing several errors in the draft to my attention. I thank Erich Kaltofen for posing questions that caused me to look more closely at the earlier work of Schönhage in [22]. I thank Damien Stehlé, Niels Möller, and Fritz Eisenbrand for email correspondence that helped to clarify some points about their related work.

## 2. A QUICK REVIEW OF EUCLIDEAN REMAINDER SEQUENCES

Much of what we discuss in this section applies to general Euclidean domains once one adjusts definitions (e.g. of the floor function) as needed, but we restrict our attention to integers as the case of interest. We are given a pair of positive integers $\begin{pmatrix} m \\ n \end{pmatrix}$ (we will use column vector notation throughout, as we frequently multiply on the left by a matrix) with $m > n$. We are interested in elements in the Euclidean remainder sequence $m = m_0$, $n = m_1$, ..., $m_k$. The integer quotients are the floor of the divisions of successive terms in this sequence. $q_j = \lfloor m_{j-1} / m_j \rfloor$. We define the matrix $R_j$ such that $R_j \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m_j \\ m_{j+1} \end{pmatrix}$. For example, $R_1 = \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix}$ (matrices of this form are called "elementary") and $R_j = \begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix} R_{j-1}$.

From this last it is clear that the top row of $R_{j+1}$ is just the bottom row of $R_j$. Hence we may write $R_j = \begin{pmatrix} s_j & t_j \\ s_{j+1} & t_{j+1} \end{pmatrix}$. We state a few basic facts about these quantities.

LEMMA 1. *Assume $R_j$ is a nontrivial product of elementary matrices.*
*(i) $s_{j-1} = s_{j+1} + q_j\, s_j$ and $t_{j-1} = t_{j+1} + q_j\, t_j$.*

*(ii) If $R_j \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m_j \\ m_{j+1} \end{pmatrix}$ then $m_{j-1} = m_{j+1} + q_j\, m_j$.*

*(iii) The signs in $R_j$ alternate in both rows and columns: $s_j\, t_j < 0$ and $s_j\, s_{j+1} < 0$.*

*(iv) The sizes grow top to bottom and left to right: $|s_{j+1}| > |s_j|$, $|t_{j+1}| > |t_j|$,* and $|t_j| > |s_j|$.

*(v) $q_j = \lfloor |s_{j+1} / s_j| \rfloor = \lfloor |t_{j+1} / t_j| \rfloor$.*

PROOF.

(i)−(iv) Quickly proven by writing out the product $R_j = \begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix} R_{j-1}$.

(v) As $q_j\, s_j = s_{j-1} - s_{j+1}$, parts (iii) and (iv) together imply that $|s_{j+1}| \geq q_j\, |s_j| > (|s_{j+1}| - |s_j|)$. This suffices to give the first floor equality for $q_j$. The second is done similarly. □

This lemma shows how to compute $R_{j-1}$ and $m_{j-1}$ given $R_j$ and the remainder sequence pair $\begin{pmatrix} m_j \\ m_{j+1} \end{pmatrix}$. The significance is that we can "go back" in the Euclidean sequence should we happen to overshoot (this will be discussed later). Note also that in the special case of $R_0$, which is an elementary matrix, we can obtain $q_0$ immediately. We easily recognize this case, as it arises if and only if the first matrix element is zero. We also use part (iii) to prove the next lemma.

LEMMA 2. *Assume $R_j$ is a nontrivial product of elementary matrices. Then $|s_j| \leq n / m_{j-1}$ and $|t_j| \leq m / m_{j-1}$.*

PROOF. This is done by induction. The base case gives equalities. For the inductive step we will show that $|s_{j+1}| \leq n / m_j$; the case for $|t_{j+1}|$ is handled similarly. By lemma 1(iii) we know $\lfloor |s_{j+1} / s_j| \rfloor = q_j$. Hence

$\left| s_{j+1}/s_j \right| \le q_j$. Thus $\left| s_{j+1} \right| \le q_j\, s_j$. By the inductive hypothesis $\left| s_j \right| \le n/m_{j-1}$. Hence $\left| s_{j+1} \right| \le q_j\, n/m_{j-1} = \lfloor m_{j-1}/m_j \rfloor\, n/m_{j-1} \le n/m_j$. □

This lemma is used to bound various quantities in the lemmas of the next section.

LEMMA 3. *For $m > n > 0$ suppose we are given a product of elementary matrices times $\begin{pmatrix} m \\ n \end{pmatrix}$ such that the result, $\begin{pmatrix} u \\ v \end{pmatrix}$ satisfies $u > v > 0$. Then $\begin{pmatrix} u \\ v \end{pmatrix}$ are a consecutive pair in the remainder sequence for $\begin{pmatrix} m \\ n \end{pmatrix}$.*

This is presented as Fact 1 in [25]. It is important because it tells us that we may take a product of elementary matrices of the form $R_j$ above, computed with respect to a new pair of integers, and still arrive at a consecutive pair in the remainder sequence for the original pair.

Finally we remark that there is obviously an index $k$ for which the pair $\begin{pmatrix} m_k \\ m_{k+1} \end{pmatrix}$ straddle $\sqrt{m}$, i.e. $m_k \ge \sqrt{m} > m_{k+1}$. These together, and in order, are referred to as the "middle" pair in the remainder sequence (regardless of where the index occurs in the sequence of such indices).

## 3. BASIC THEORY FOR THE HGCD ALGORITHM

Again we begin with a pair of positive integers $\begin{pmatrix} m \\ n \end{pmatrix}$ with $m > n$. Take $k$ to be a positive integer less than the size of $m$ in bits (initially it will be $\lfloor \log_2 m/2 \rfloor$ but we do not use that until the next section). We write $\begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} 2^k f_0 + f_1 \\ 2^k g_0 + g_1 \end{pmatrix}$ with $\{f_1, g_1\} < 2^k$ and recursively compute the middle pair, and corresponding multiplier matrix, for the pair $\begin{pmatrix} f_0 \\ g_0 \end{pmatrix}$. This gives a matrix $R_i$ and pair $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$ with $R_i \begin{pmatrix} f_0 \\ g_0 \end{pmatrix} = \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$ and $r_{i+1} < \sqrt{f_0} \le r_i$. We want to use $R_i$, or a close relative, on the original pair $\begin{pmatrix} m \\ n \end{pmatrix}$. We have $R_i \begin{pmatrix} 2^k f_0 + f_1 \\ 2^k g_0 + g_1 \end{pmatrix} = 2^k \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} + R_i \begin{pmatrix} f_1 \\ g_1 \end{pmatrix}$. We will call this product $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$. The next two lemmas will find bounds, one upper and one lower, for these elements. We first handle $v_{i+1}$. We bound the absolute value and, under certain circumstances, we place a tighter bound on how negative it may become. This is important because, in order to invoke lemma 3, we will need a way to correct for the negative case.

LEMMA 4.

*(i) $\left| v_{i+1} \right| < 2^{k+1} \sqrt{f_0}$.*

*(ii) Suppose $r_i > 2\sqrt{f_0}$. Then $v_{i+1} > -2^{k-1}\sqrt{f_0}$.*

PROOF. Note that $\{s_{i+1}, t_{i+1}\} < f_0/r_i < \sqrt{f_0}$. We have $v_{i+1} = 2^k r_{i+1} + s_{i+1} f_1 + t_{i+1} g_1$.

(i) Using the upper bound of $2^k$ on $\{f_1, g_1\}$ and the alternating signs in the matrix $R_i$, the absolute value is bounded by $\left| v_{i+1} \right| < 2^k \sqrt{f_0} + 2^k \sqrt{f_0} = 2^{k+1} \sqrt{f_0}$.

(ii) Since $r_i > 2\sqrt{f_0}$ we have $v_{i+1} > -2^k \frac{f_0}{2\sqrt{f_0}} = -2^{k-1}\sqrt{f_0}$. □

We will use these same notions in subsequent lemmas (particularly the sign alternation, in effect to ignore one of the three terms) without further mention.

We now look at $v_i = 2^k r_i + s_i f_1 + t_i g_1$.

LEMMA 5.

*(i) Suppose $r_i > 2\sqrt{f_0}$. Then $v_i > 2^{k-1}\sqrt{f_0}$.*

*(ii) Suppose $r_i \le 2\sqrt{f_0}$. Then $v_i > 2^{k-1}$.*

PROOF. Lemmas 1 and 2 sign alteration and size bounds in $R_i$ gives $v_i = 2^k r_i + s_i f_1 + t_i g_1 > 2^k r_i - 2^k \frac{f_0}{r_{i-1}}$.

(i) The hypothesis and the fact that $r_{i-1} > r_i$ yield $v_i > 2^k \left( r_i - \frac{f_0}{2\sqrt{f_0}} \right) > 2^k \frac{\sqrt{f_0}}{2} = 2^{k-1} \sqrt{f_0}$ .

(ii) Now we write the lower bounding value as $\frac{2^k}{r_{i-1}} (r_i r_{i-1} - f_0)$. Since $\sqrt{f_0} \le r_i < r_{i-1}$ and the latter two are integers, there is an $a > 1$ with $r_{i-1} = \sqrt{f_0} + a$. So we have $v_i > \frac{2^k}{(\sqrt{f_0} + a)} \left( r_i \left( \sqrt{f_0} + a \right) - f_0 \right)$. This in turn is larger than $\frac{2^k}{(\sqrt{f_0} + a)} \left( \sqrt{f_0} \left( \sqrt{f_0} + a \right) - f_0 \right) = \frac{2^k}{(\sqrt{f_0} + a)} a \sqrt{f_0}$ which is bounded below by $2^{k-1}$. □

For the pair $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix} = R_i \begin{pmatrix} m \\ n \end{pmatrix}$ lemmas 4 and 5 give an upper bound on one element and a lower bound on the other. There will be situations in which we must backtrack a Euclidean step to use $R_{i-1}$, that is, the multiplier matrix preceding $R_i$ in the remainder sequence for $\begin{pmatrix} f_0 \\ g_0 \end{pmatrix}$. In this case we need to bound $\begin{pmatrix} v_{i-1} \\ v_i \end{pmatrix} = R_{i-1} \begin{pmatrix} m \\ n \end{pmatrix}$.

LEMMA 6.

*(i)* $v_{i-1} > 2^{k-1} \sqrt{f_0}$ .

*(ii) Suppose* $r_i \le 2 \sqrt{f_0}$ *. Then* $v_i < 2^k 3 \sqrt{f_0}$ .

PROOF.

(i) $v_{i-1} = 2^k r_{i-1} + s_{i-1} f_1 + t_{i-1} g_1 > 2^k r_{i-1} - 2^k f_0 / r_{i-2} =$

$2^k r_{i-1} - 2^k \frac{f_0}{q_{i-1} r_{i-1} + r_i} > 2^k \left( r_{i-1} - \frac{f_0}{2 r_i} \right) > 2^k \left( \sqrt{f_0} - \frac{\sqrt{f_0}}{2} \right) = 2^{k-1} \sqrt{f_0}$ .

(ii) $v_i = 2^k r_i + s_i f_1 + t_i g_1 \le 2^k 2 \sqrt{f_0} + 2^k \frac{f_0}{r_{i-1}} < 2^k 2 \sqrt{f_0} + 2^k \frac{f_0}{\sqrt{f_0}} = 2^k 3 \sqrt{f_0}$ . □

Given a pair $\begin{pmatrix} m \\ n \end{pmatrix}$ with $m > n > 0$ we will see that the above lemmas allow us to find a pair $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$ with magnitudes in the desired ranges (this will be explained more carefully in the next section). Two problems may arise. One is that we require both to be nonnegative; the lemmas will only guarantee that $v_i > 0$. Second, we require that $v_i > v_{i+1}$. These requirements are in order to meet the hypotheses of lemma 3 and thus assert that we have a consecutive pair in the remainder sequence for our inputs. We now provide a lemma to assist in repairing our intermediate pair, should either of these possible flaws arise.

LEMMA 7. *Given an elementary matrix* $\begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix}$ *(this implies* $q_j$ *is a positive integer). Then for any integer* $h < q_j$ *the product* $\begin{pmatrix} 1 & 0 \\ h & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix}$ *is also an elementary matrix. In particular this holds for any negative integer h.*

PROOF. The product is simply $\begin{pmatrix} 0 & 1 \\ 1 & h - q_j \end{pmatrix}$ and by definition this is elementary precisely when $h - q_j < 0$.□

We will use such products to repair deficiencies in sign or order of a pair $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$.

## 4. THE HGCD ALGORITHM

Input: A pair of nonnegative integers $m > n$.

Output: A pair $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$ of consecutive integers in the Euclidean remainder sequence for $\begin{pmatrix} m \\ n \end{pmatrix}$ with $v_i \ge \sqrt{m} > v_{i+1}$, and a matrix $R_i$ which is the product of elementary transformation matrices, such that $R_i \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$.

Step 1: With the same input specification as in the previous sections, we begin by choosing $k = \left\lfloor \frac{\log_2 m}{2} \right\rfloor$. Thus, as above, we write $\begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} 2^k f_0 + f_1 \\ 2^k g_0 + g_1 \end{pmatrix}$ with $\{f_1, g_1\} < 2^k$. Moreover the choice of $k$ gives $2^k > \sqrt{m} > 2^{k-1}$ and $g_0 \le f_0 < 2\sqrt{m}$.

Step 2: Recursively compute HGCD$\begin{pmatrix} f_0 \\ g_0 \end{pmatrix}$. With notation as in the last section, the result is a matrix $R_i$ and pair $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$ with $R_i \begin{pmatrix} f_0 \\ g_0 \end{pmatrix} = \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$ and $0 \le r_{i+1} < \sqrt{f_0} < r_i$.

Step 3: Compute $\begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix} = R_i \begin{pmatrix} m \\ n \end{pmatrix}$. Note that we already have the "upper part" of the resulting vector computed as $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$; this can be used to reduce the size of the multiplications in this step.

Step 4: The bounds presented in the lemmas do not rule out the possibility that $v_{i+1}$ may be negative, or that $v_i < v_{i+1}$. If $v_i > v_{i+1} > 0$ then we set $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$ and move to step 5 at this point. Otherwise we must repair the pair in such a way that the transformation matrix remains a product of elementary matrices. This is necessary so that we may invoke lemma 3 to know the resulting vector is a consecutive pair in the remainder sequence for $\begin{pmatrix} m \\ n \end{pmatrix}$. We split into three cases that together comprise all possibilities.

Case (i). Suppose $v_{i+1} > v_i$. Take the matrix $H = \begin{pmatrix} 1 & 0 \\ -h & 1 \end{pmatrix}$ where $h = \lfloor v_{i+1}/v_i \rfloor \ge 1$. By lemma 7 $H\,R_i$ is a product of elementary matrices. The new pair thus obtained is $\begin{pmatrix} u \\ v \end{pmatrix} = H \begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} v_i \\ v_{i+1} - h\,v_i \end{pmatrix}$ which satisfies the requirement that $u > v > 0$. For purposes of notation we continue to call the resulting matrix $R_i$. Note that the value of $u$ is unchanged (hence lemma bounds still apply), while the absolute value of $v$ has diminished. We now move on to step 5.

Case (ii). Suppose $v_{i+1} < 0$ and $v_{i+1} + v_i \ge 0$.

Subcase (a). First assume $q_i > 1$. Then we use the matrix $H = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and proceed, as we did in case (i) above, to obtain a positive pair via $\begin{pmatrix} u \\ v \end{pmatrix} = H \begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}$. This is appropriate because the product $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix}$ is an elementary matrix so we may invoke lemma 3. Again we call the resulting pair $\begin{pmatrix} u \\ v \end{pmatrix}$, and continue to call the transformation matrix $R_i$. Note that $u - v = |v_{i+1}| < 2^{k+1}\sqrt{f_0}$. This means that a Euclidean step will bring the pair into the range claimed in step 6 below. As it also shows that $u > v$, we have a consecutive pair in the remainder sequence.

Subcase (b). If $q_i = 1$ the situation is a bit more subtle. Again we use the matrix $H$ as defined above, and again we obtain a positive pair in the correct order; unfortunately the product $H\,R_i$ is $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, which is not an elementary matrix. To correct for this we multiply by $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ again, giving as product the identity matrix. This has the effect of flipping $\begin{pmatrix} u \\ v \end{pmatrix}$. Thus we have used premultipliers to take us from $R_i$ to $R_{i-1}$, which we know is also a product of elementary matrices. We have also obtained as our vector $\begin{pmatrix} v \\ u \end{pmatrix}$; it has appropriate components except they are in the wrong order. As this is exactly the situation of case (i) above we proceed there to correct it.

Case (iii). If $v_{i+1} < 0$ and $v_{i+1} + v_i < 0$ then either $v_i \le 2^{k-1}\sqrt{f_0}$ or $v_{i+1} \le -2^{k-1}\sqrt{f_0}$. In either case, lemmas 5(i) and 4(ii) respectively guarantee that $r_i \le 2\sqrt{f_0}$. We will perform a reversal of a Euclidean step, obtaining the pair $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} v_{i-1} \\ v_i \end{pmatrix} = R_{i-1} \begin{pmatrix} m \\ n \end{pmatrix}$. As $r_i \le 2\sqrt{f_0}$, lemma 5(ii) guarantees that $v > 0$ and furthermore $v < 2^k\,3\sqrt{f_0}$ by lemma 6, so again the bounds given in step 6 will hold. If $u < v$ then we go to case (i) above.

We remark that cases (ii−b) and (iii) are identical in terms of actual treatment. We separated them in the way we did in order to explicate the rationale. But since $q_i = 1$ in case (ii−b), and we adjusted via the matrix $H = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, we have simply done nearly a reversal of a Euclidean step. The only difference in the outcome is we also reversed the order to

$\binom{v_i}{v_{i-1}}$. The next adjustment in (ii−b), to get a valid elementary matrix, flipped the order to get $\binom{v_{i-1}}{v_i}$; thus we now have indeed done a Euclidean step reversal, just as is used in case (iii). From the hypotheses of case (ii−b) we know $v_{i-1} < v_i$, hence we must proceed to case (i) to correct this. Again, this is something we check for in case (iii). The upshot is that in actual code cases (ii−b) and (iii) will be handled as one.

Step 5: Perform a Euclidean reduction on $\binom{u}{v}$. We obtain the next consecutive pair $\binom{v}{w}$ in the remainder sequence for $\binom{m}{n}$, with elementary transformation matrix $Q = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$, where $q = \lfloor u/v \rfloor$, and $w = u - q\,v$. We form the corresponding transformation matrix $R = Q\,R_i$.

Step 6: At this point we examine the values of our pair $\binom{v}{w}$. Lemmas 4, 5, 6, the remarks from the step 4 cases, and our choice of $k$ (implying $f_0 < 2\sqrt{m}$ and $2^{k-1} < \sqrt{m}$) guarantee that $0 < v < 2^k\, 3\sqrt{f_0} < 2^{k+1/2}\, 3\, m^{1/4} < 2^{3/2}\, 3\, m^{3/4}$ and $u > 2^{k-1} > \sqrt{m}\,/4$.

Case (i). $w < \sqrt{m}$. If $v \geq \sqrt{m}$ we have our pair straddling $\sqrt{m}$. We return it along with the transforming matrix $R$. If $v < \sqrt{m}$ we do reverse Euclidean steps, updating our remainder sequence pair and transformation matrix using the formulas in lemma 1. Since $u > \sqrt{m}\,/4$ and it immediately precedes $v$ in the remainder sequence, we have at most five such steps before an element exceeds $\sqrt{m}$ (possibly one could decrease this upper bound by constructing tighter bounds in the lemmas). We perform as many such steps as is needed to obtain the pair straddling $\sqrt{m}$, returning it and corresponding transformation matrix.

Case (ii). $\sqrt{m} \leq w < v < 2^{3/2}\, 3\, m^{3/4}$ (in typical examples, $w$ and $v$ will both be close to $m^{3/4}$). Similarly to step 1, we take $l = \lfloor \log_2 m \rfloor - \lfloor \log_2 v \rfloor$ (so $2^l$ is within a factor of 2 of $m/v$; we will soon see why this is the appropriate value). Observe that $l$ is roughly between one fourth and one half the bit length of $m$. Specifically, we have $\lfloor \log_2 m \rfloor/4 - 3 < l < \lfloor \log_2 m \rfloor/2 + 3$. We proceed to step 7.

Step 7: This time we write $\binom{v}{w} = \binom{2^l\, f_2 + f_3}{2^l\, g_2 + g_3}$ with $\lfloor \log_2 f_2 \rfloor = \lfloor \log_2 v \rfloor - l$. The upper bound on $\log_2 v$ and lower bound on $l$ show that $f_2$ and $g_2$ are no larger than $O(\sqrt{m})$. This fact is required for the claim of asymptotic speed (though not for correctness).

As in step 2, recursively compute HGCD$\binom{f_2}{g_2}$. As in steps 3 and 4 we obtain a transformation matrix $S$, and a consecutive pair $\binom{v_j}{v_{j+1}}$ in the remainder sequence for $\binom{m}{n}$, with $v_j > v_{j+1} \geq 0$. If $v_j \leq 2^{l-2}\sqrt{f_2}$ then the condition of lemma 5(i) cannot hold, and thus lemma 6(ii) applies. So we do a single reverse Euclidean step to get the previous consecutive pair in the sequence. At this point we have a consecutive pair, call it $\binom{x}{y}$, wherein lemma 6 guarantees that $y < 2^{l+2}\sqrt{f_2}$ and $x > 2^{l-2}\sqrt{f_2}$.

Step 8: From step 7 we know that $f_2$ is within a factor of 2 of $2^{-l}\, v$ and hence $2^l\sqrt{f_2} \approx 2^{l/2}\sqrt{v} \approx \sqrt{\frac{m}{v}}\,\sqrt{v} = \sqrt{m}$ where the approximation from first to last is within a factor of 2 because each intermediate approximation is within a factor of $\sqrt{2}$. The inequalities at the end of step 7 therefore imply $y < 8\sqrt{m}$ and $x > \sqrt{m}\,/8$; this was the point in selecting $l$ as we did. Thus with a limited number of Euclidean steps, or reversals thereof, we obtain the consecutive pair in the remainder sequence that straddles $\sqrt{m}$, and the transformation matrix that gives this pair. Possibly with care we might tighten the bound on the number of forward or reverse Euclidean steps. In practice this is unimportant. One simply codes a **while** loop for the iterations; that it terminates in a fixed number of steps suffices to prove the claim of asymptotic speed.

## 5. APPLICATIONS OF THE HGCD ALGORITHM

First note that the asymptotic complexity is $O(n\,M(n))$ where $n$ is the bit size of the inputs and $M(n)$ is the complexity of multiplying a pair of number of that size. This is well known (see the various references) and follows from the fact that we do two recursive steps on numbers no larger than roughly $n/2$ (see steps 1 and 7 above), along with a bounded number of multiplications, Euclidean steps, and reverses thereof. It is this speed that motivates the various applications mentioned below.

The HGCD algorithm is used recursively in gcd computations. An HGCD computation followed by a Euclidean step is guaranteed to reduce the size of the inputs (in bits) by at least half. Another advantage is that one gets the corresponding multiplier matrix for free, so computation of the extended gcd is not much more costly than that of the ordinary gcd. This is important for e.g. matrix Hermite normal form or integer Gröbner basis computations [16], where speed of extended gcds is paramount. As a standard benchmark example we will find the gcd of a pair of consecutive large Fibonacci numbers. This and all other timings are from runs using version 5.1 of *Mathematica* under Linux on a 1.4 GHz Athelon processor.

```
fibs = {Fibonacci[10^7], Fibonacci[10^7 + 1]};
```

Each has about two million digits. We compute both regular and extended gcd and check that the result is plausible.

```
Timing[gcd = Apply[GCD, fibs];]
Timing[{gcd2, mults} = Apply[ExtendedGCD , fibs];]
mults.fibs == gcd == gcd2 == 1
{31.04 Second, Null}
{40.21 Second, Null}
True
```

A particularly nice application of the HGCD is in recovering rational numbers from $p-$adic approximations. This is explained in some detail in chapter 5 of [10]. Given a prime power $p^k$ and a smaller nonnegative integer $x$ not divisible by $p$, we can obtain a rational $a/b$ equivalent to $x$ modulo $p^k$ with both numerator and denominator smaller than the square root of the prime power. It is obtained directly from the HGCD matrix and middle pair given by $\text{HGCD}\begin{pmatrix} p^k \\ x \end{pmatrix}$. In brief, we have a matrix $R_j = \begin{pmatrix} s_j & t_j \\ s_{j+1} & t_{j+1} \end{pmatrix}$ with $R_j \begin{pmatrix} p^k \\ x \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}$. Moreover $\{v, t_{j+1}\} < \sqrt{p^k}$ and $\frac{v}{t_{j+1}} \equiv_{p^k} x$ because $s_{j+1}\,p^k + t_{j+1}\,x = v$. Thus we have our desired rational.

The below code will do this recovery given the input pair $\{x, p^k\}$.

```
rationalRecover[x_, pk_] :=
 ((#[[2, 2]] / #[[1, 2, 2]]) &)[Internal‘HGCD[pk, x]]
```

For contrast we also give the standard Euclidean sequence method as well as a simple method based on lattice reduction.

```
rationalRecover2[a_, b_] := Module[
  {mat, aa = a, bb = b, cc = 1, dd = 0, quo},
  mat = {{aa, cc}, {bb, dd}};
  While[Abs[aa] ≥ Sqrt[b],
   quo = Quotient[bb, aa];
   {{aa, cc}, {bb, dd}} =
    {{bb, dd} - quo * {aa, cc}, {aa, cc}};];
  aa / cc]

rationalRecover3[n_, pq_] :=
 (#[[1]] / #[[2]] &)[First[LatticeReduce[{{n, 1}, {pq, 0}}]]]
```

We illustrate this application by solving linear systems over the rationals, using a simple $p-$adic linear solver based on the method presented in [6] (the code for `pAdicSolve` is in the appendix). To get some idea of speed we will compare to the built in `LinearSolve` function. The latter at this time uses a Gaussian elimination via one−step row reduction [2]. The tests we use will involve creating random linear systems of a given dimension and coefficient size in decimal digits. In the results we will show timings, a check that the results agree, and the size in decimal digits of the largest denominator in the result.

```
testPAdicSolver[dim_Integer , csize_Integer , recoveryfunc_] :=
 Module[
  {ls1, ls2, mat, b},
  mat = Table[Random[Integer ,
     {-10^csize, 10^csize}], {dim}, {dim}];
  b = Table[Random[Integer ,
     {-10^csize, 10^csize}], {dim}];
  {First[Timing[ls1 =
      pAdicSolve[mat, b, recoveryfunc]]],
   First[Timing[ls2 = LinearSolve[mat, b]]],
   ls1 === ls2, Max[Log[10., Denominator[ls1]]]}]
```

In the set of tests below input data will consist of 10−digit integers. First we try a 50 x 50 system.

```
testPAdicSolver[50, 10, rationalRecover]
{1.35 Second, 0.62 Second, True, 517.912}
```

The built in method was faster by a factor between 2 and 3. The standard Euclidean algorithm of **`rationalRecover2`** makes it about three times slower still, thus indicating that even at this low dimension most of the time might be spent in rational recovery if we use a pedestrian approach. This example takes about 17 seconds using **`rationalRecover3`**

We now double the dimension.

```
testPAdicSolver[100, 10, rationalRecover]
{11.79 Second, 9.04 Second, True, 1053.32}
```

This time the speeds are quite close. Doubling again will show the $p$−adic solver well ahead.

```
testPAdicSolver[200, 10, rationalRecover]
{93.26 Second, 173.27 Second, True, 2136.99}
```

We remark that most of the time is spent in finding the $p$−adic approximate solutions. The utility of fast rational recovery is indirectly witnessed in the above computations; were we to use a less efficient method, it would become more prominent in the timings. As it stands, the overwhelming component is now in the improvement iterations.

We should note that one can use a very different iterative approach when the input matrix is well conditioned. One can solve the stem numerically to sufficiently high precision using the iterative method of [11]. Then the exact result may be recovered using rationalization of a high precision approximate result. Interestingly, the technology underlying this type of rational recovery involves continued fractions; efficient computation of these is similar to the divide−and−conquer approach of HGCD. A potential drawback to this method (in addition to the conditioning requirement) is that it requires an a priori precision estimate that might be quite large, or else an expensive check of correctness that could outweigh the cost of the actual construction of a solution.

Quite recently a related method based on iterative refinement of numerical solutions was described in [28]. It uses rescaling of residuals and stepwise rational approximations to construct its result. At this time it appears to be the state of the art in solving linear systems over the rationals. That said, clearly there remains a need for fast rational recovery from a $p$−adic approximation. For example, other recent methods requiring rational recovery have been discussed in [8] and [5]. Both derive speed by clever use of level 3 BLAS for modulo prime linear algebra as described in [7]. While [5] describes ways to speed this process considerably even when using the standard Euclidean method, it remains true that an asymptotically fast rational recovery is a desirable further improvement.

Another application of HGCD is in the Smith-Cornacchia algorithm [4] for solving $x^2 + d\,y^2 = n$ with $(d, n)$ relatively prime. This can be used to factor primes of the form $4\,k + 1$ into products of Gaussian primes. Yet another application, which we present later, is to fast planar lattice reduction.

## 6. RATIONAL RECOVERY VIA LATTICE REDUCTION

We now show that the method based on integer lattice reduction is more than just a heuristic. While not particularly fast we feel this is of interest in its own right as yet another simple application of lattice methods. We first set up the problem. Given integers $m > n > 0$, here is a simple approach to finding a "small" rational $\frac{r}{s}$ with $s\,n \equiv_m r$. We form a 2x 2 lattice $\begin{pmatrix} m & 0 \\ n & 1 \end{pmatrix}$ and reduce it via LLL [15], say to $\begin{pmatrix} r & s \\ t & u \end{pmatrix}$, where the top row is smaller in Euclidean norm than the bottom. We then take $\frac{r}{s}$ as our reconstructed rational. Heuristically we expect this to work frequently because typically we will have $\{r, s\} < \sqrt{m}$ and this is roughly what we require for the rational recovery procedure. It turns out that under mild hypotheses (that in essence amount to lifting half a bit more than we otherwise might), we can guarantee that we obtain the correct value.

Remark: Lattice reduction via LLL assumes a value for a certain parameter, often called $\alpha$ in the literature and taken to be $\frac{3}{4}$ as in the original paper. But it can be any value in the open interval $\left(\frac{1}{4}, 1\right)$. If it is not the standard $\frac{3}{4}$ then one must modify accordingly the lifting bounds and proof of the below theorem.

THEOREM. *Suppose we have a bound k on numerator and denominator of a rational, and moreover we have a power of a prime $p^q$ and a p−adic image n of the value (obtained, say, as in the linear algebra examples in the previous section). Suppose moreover that $p^q > 2\sqrt{2}\,k^2$ and $\frac{r}{s}$ is a rational equal to n modulo $p^q$ with $\{\,|r|, s\} < \sqrt{p^q}$ and r relatively prime to s (that is, it is the value we seek). Form the lattice L with row vectors given (in matrix form) as $\begin{pmatrix} p^q & 0 \\ n & 1 \end{pmatrix}$. Reduce it to $\begin{pmatrix} t & u \\ v & w \end{pmatrix}$ with $t^2 + u^2 \le v^2 + w^2$(that is, rows ordered by norm). Then $\{t, u\} = \pm\{r, s\}$ and hence we recover our rational from the reduced lattice.*

PROOF.

(i) First we show that $\{r, s\} \in L$. By assumption there is an integer $j$ with $s\,n + j\,p^q = r$. Thus $j\,\{p^q, 0\} + s\,\{n, 1\} = \{r, s\}$.

(ii) Next we claim that $\{r, s\}$ is a minimal vector in $L$. The Euclidean norm squared is $r^2 + s^2 < 2\,k^2 < \frac{p^q}{\sqrt{2}}$. If $\{x, y\} \in L$ is any vector independent of $\{r, s\}$ then we must have $x^2 + y^2 > \sqrt{2}\;p^q$, because the product of the norms of any independent pair much be at least as large as the lattice determinant, $p^q$. If instead $\{x, y\}$ is a scalar multiple of $\{r, s\}$ then the scalar must be at least 1 in absolute value, by the assumption that $r$ and $s$ are relatively prime.

(iii) Finally we show that $\pm\{r, s\}$ is in the LLL−reduced basis. This follows from the fact that the smallest vector $\{t, u\}$ in the reduced basis has $t^2 + u^2 \le 2\left(r^2 + s^2\right)$ by 1.11(iii) of [15], and we know this is smaller than $\sqrt{2}\;p^q$ by (ii) above. Again from (ii) we know that any vector in $L$ independent of $\{r, s\}$ cannot satisfy this inequality. If instead $\{t, u\}$ were a nontrivial multiple of $\{r, s\}$, we would not have a correct basis because $L$ contains $\{r, s\}$ by (i) above.□

We remark that this is a sort of opposite extreme to the method for finding extended greatest common divisors presented in [13]. They use a form of column−weighted LLL to obtain extended gcds of more than two integers, such that the multipliers are small (moreover, by considering multiple columns and geometrically scaled weights, they derive an Hermite normal form algorithm based on LLL). For the sort of lattice we construct above, such weighting would counter the tendency of LLL to take one toward the middle pair in the remainder sequence.

We give a quick demonstration of this method using one of our earlier tests.

```
testPAdicSolver[50, 10, rationalRecover3]
{17.5 Second, 0.62 Second, True, 517.912}
```

Not surprisingly, this is not competitive in speed with the asymptotically fast HGCD method.

## 7. PLANAR LATTICE REDUCTION VIA HGCD

Having seen that rational recovery can be effected by planar lattice reduction, it should be no surprise to find that such reduction can be handled by means of HGCD computations. We demonstrate how this might be done; relevant theory may be found in [9]. Earlier asymptotically fast methods wre presented in [] and [26].

Suppose we have a $2\,x\,2$ integral matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ where we regard the rows as generating a lattice in $\mathbb{Z}^2$. The goal is to find a reduced form, that is, a unimodular multiplier matrix $A$ such that $A\,M = L$ where $L$ is the lattice reduced form of $M$. We compute the reduction as follows.

Step 1. Put $M$ into Hermite normal form. As is well known, this uses the extended gcd algorithm, hence (for large inputs) amounts to a few HGCD invocations. We obtain $M_1 = \begin{pmatrix} g & j \\ 0 & k \end{pmatrix}$, where $g = \gcd(a, c)$, and a unimodular transformation matrix $A_1$ with $A_1\, M = M_1$.

Step 2. See if this is lattice reduced. If so, we are finished. If not, we now have a "small" element in the upper left and a zero beneath it. We now work on the second column.

Step 3. Find HGCD$(j, k)$.

Step 4. Use the multiplier matrix $A_2$ to form $A_2\, A_1\, M = A_2\, M_1 = \begin{pmatrix} s & m \\ t & n \end{pmatrix}$.

Step 5. We now have a short vector. If the second vector is not short we can reduce it using Euclidean steps (this method of reducing planar vectors is due to Gauss). Since we divide by an element in the short vector, the number of such steps is bounded.

Here is a simple example. We work with a lattice of two row vectors. We construct it in such a way as to be quite far from reduced. Specifically, the second row is (with high probability) a small offset of the first. We first reduce using LLL in order to find the expected lengths of the resulting elements.

```
SeedRandom[1111];
row = Table[Random[Integer, {-10^100, 10^100}], {2}];
lat = {row, row + {10^10, 10^20}};
redlat = LatticeReduce[lat];
```

We check the sizes of the initial and reduced vectors.

```
Log[10., Abs[lat]]
Log[10., Abs[redlat]]
{{99.9937, 99.8255}, {99.9937, 99.8255}}

{{10., 20.}, {99.9937, 89.9937}}
```

We now do step 1 and again check sizes (the zero corresponds to an entry of unity, and the $-\infty$ corresponds to an entry of zero).

```
{a0, hnf} = Developer`HermiteNormalForm[lat];
Log[10., Abs[{a0, hnf}]]
{{{98.8304, 98.8304}, {99.9937, 99.9937}}, {{0., 118.83}, {-∞, 119.994}}}
```

Each row in the Hermite form has a large entry so we deduce it is not reduced. We now do step 3.

```
{a1, col2} = Internal`HGCD[Apply[Sequence, hnf[[All, 2]]]];
```

We check that this is correct.

```
a1.hnf[[All, 2]] == col2
True
```

We’ll now recover a short vector using step 4.

```
lattoo = a1.a0.lat;
Log[10., Abs[lattoo]]
{{9.22796, 109.994}, {10., 20.}}
```

It is clear that we can use the second vector to reduce the magnitude of the first by making the second component much smaller. We do so as per step 5. Specifically, we can take a quotient, form a multiplier matrix similar to that used in HGCD, and obtain a reduction of the larger vector.

```
q = Quotient[lattoo[[1, 2]], lattoo[[2, 2]]];
a2 = {{1, -q}, {0, 1}};
latthree = a2.lattoo
{{-9 856 220 730 047 694 401 632 388 898 359 410 889 450 006 931 893 281 121 833 193 315 \
    680 250 563 903 159 129 824 369 221 690 297 741, 75 868 793 220 380 069 225},
  {10 000 000 000, 100 000 000 000 000 000 000}}
```

We check that the transformations are unimodular and get the sizes of the elements in the reduced lattice.

```
Det[a2.a1.a0]
Log[10., Abs[latthree]]
-1
{{99.9937, 19.8801}, {10., 20.}}
```

It is straightforward to verify that these row norms are comparable to those of *redlat* and indeed they have the same small vector.

Short code that does this, without checking for the special cases, is in teh appendix. An expanded version is used in [27] in order to compute Frobenius numbers for sets of three elements. It appears to be on average far faster than previously known methods as discussed in [27] and references therein.

## 8. SUMMARY

We have demonstrated a correct, asymptotically fast integer gcd algorithm based on the classical Half−GCD method. The various correction steps needed to address deficiencies caused by integer carries are, we believe, relatively simple both from the standpoint of theory and practical implementation. We apply this to solving large linear systems over the rationals, obtaining results that scale well with dimension.

After demonstrating that a similar rational recovery result can be attained via planar lattice reduction, we then proceed to do such reduction using HGCD.

## 9. APPENDIX: *P*−ADIC SOLVER AND PLANAR REDUCTION CODE

Below is code for a simple $p$−adic solver for linear systems with integer coefficients; extension to rationals is straightforward. The code below computes a solution modulo a particular prime. In production code one would make sure the system was solvable modulo that prime or else resort to another tactic.

```
vectorNorm[vec_] := Apply[Plus, Map[Abs, vec]]
matrixNorm[mat_] := Apply[Times, Map[Sqrt[N[#].N[#]] &, mat]]

powerUp[vals_, mod_] := Map[FromDigits[#, mod] &, Transpose[Reverse[vals]]]

pAdicSolve[mat_?MatrixQ, rhs_?VectorQ, recoveryfunc_] :=
 Module[{len = Length[mat], b, mod = Prime[2222],
    mnorm, lud, sol= {}, corr, power, j= 0, logpow = 0, logm},
   logm = Log[N[mod]];
   lud = LUDecomposition[mat, Modulus → mod];
   b = rhs;
   power = 1;
   mnorm = 2. * Log[(2. * matrixNorm[mat] * vectorNorm[rhs])];
   While[logpow < mnorm + .5,
    j++;
    corr = LUBackSubstitution[lud, b, Modulus → mod];
    b = 1 / mod * (b - mat.corr);
    sol = {sol, corr};
    logpow += logm];
   power = mod^j;
   sol2 = Partition[Flatten[sol], len];
   sol2 = powerUp[sol2, mod];
   Map[recoveryfunc[#, power] &, sol2]]
```

Below is a version of planar reduction that will tend to find a reduced lattice with smallest vector. It is based loosely on the exposition in [9].

```
planarReduce[{{a_Integer , b_Integer}, {c_Integer , d_Integer}}] := Module[
   {hgcd, mult, lat, g, u11, u12, col2, c22, r1, r2, k, n= 0},
   {g, {u11, u12}} = ExtendedGCD[a, c];
   col2 = {{u11, u12}, {-c, a} / g}.{b, d};
   c22 = col2[[2]];
   col2[[1]] = Mod[col2[[1]], c22, Ceiling[-c22 / 2]];
   {mult, hgcd} = Apply[Internal‘HGCD , col2];
   lat = Transpose[{mult[[All, 1]] * g, hgcd}];
   {r1, r2} = Sort[lat, Norm[N[#], 2] &];
   While[k =!= 0 && n ≤ 3, n++;
    k = Round[(r1.r2) / (r1.r1)];
    r2 = r2 - k * r1;
    {r1, r2} = Sort[{r1, r2}, (#1.#1 < #2.#2) &];];
   {r1, r2}]
```

A more elaborate version is used in [27] to recover several small lattice vectors. This appears to give the fastest cur−

rently known method for computing Frobenius numbers of sets of three large integers.

## 10.REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison−Wesley Publishing Company, Reading Massachussetts, 1974.

[2] E. H. Bareiss. Sylvester's identity and multistep integer−preserving Gaussian elimination. Math. Comp. **22**(103):565−578. 1968.

[3] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of P approximants. Journal of Algorithms **1:**259−295. 1980.

[4] J. Buhler and S. Wagon. Basic number theory algorithms. Surveys in Algorithmic Number Theory, J. P. Buhler and P. Stevenhagen, eds. Mathematical Sciences Research Institute Publications vol. 44. Cambridge University Press. To appear.

[5] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC 2005), M. Kauers, ed. 92−99. ACM Press, New York City, 2005.

[6] J. D. Dixon. Exact solutions of linear equations using $p$−adic expansions. Numerische Math. **40**:137−141. 1982.

[7] J. G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002), T. Mora, ed.. 63−74. ACM Press, New York City, 2002.

[8] J. G. Dumas, P. Giorgi, and C. Pernet. Finite field linear algebra package. Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 2004), J. Gutierrez, ed. 119−126. ACM Press, New York City, 2004.

[9] F. Eisenbrand. Short vectors of planar lattices via continued fractions. Information Processing Letters **79**:121−126, 2001

[10] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[11] K. O. Geddes and W. W. Zheng. Exploiting fast hardware floating point in high precision computation. Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 2003), J. R. Sendra, ed. 111−118. ACM Press, New York City, 2003.

[12] GMP: The Gnu Multiprecision Bignum Library. Web site: http://www.swox.com/gmp/

[13] G. Havas, B. S. Majewski, K. R. Matthews. Extended GCD and Hermite normal form algorithms via lattice basis reduction. Experimental Mathematics **7**(2): 125−126. A. K. Peters, Ltd., 1998.

[14] D. Knuth. The analysis of algorithms. Proceedings of the 1970 International Congress of Mathematicians (Nice, France), **3**:269−274, 1970.

[15] A. K. Lenstra, H. W. Lenstra, Jr., L. Lovász. Factoring polynomials with rational coefficients. Mathematische Annalen **261**:515−534. 1982.

[16] D. Lichtblau. Revisiting strong Gröbner bases over Euclidean domains. Manuscript, 2003.

[17] D. Lichtblau. Half−GCD and Fast Rational Recovery. Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC 2005), M. Kauers, ed. 231−236. ACM Press, New York City, 2005.

[18] N. Möller. On Schönhage's algorithm and subquadratic integer gcd computation. Manuscript, 2005.

[19] R. T. Moenck. Fast computation of GCDs. Proceedings of the 5th ACM Annual Symposium on Theory of Computing. 142−151. ACM Press, New York City, 1973.

[20] V. Y. Pan and X. Wang. Acceleration of Euclidean algorithm and extensions. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002), T. Mora, ed. 207−213. ACM Press, New York City, 2002.

[21] A divide−and−conquer method for integer−to−rational conversion. Proceedings of the Symposium in Honor of Bruno Buchberger's 60th Birthday (Logic, Mathematics and Computer Science: Interactions). October 20−22, 2002, RISC−Linz, Castle of Hagenberg, Austria. K. Nakagawa, ed. 231−243. 2002.

[22] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. Acta Informatica **1**:139−144, 1971.

[23] A. Schönhage. Fast reduction and composition of binary quadratic forms. Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC 1991), S. Watt, ed. 128−133. ACM Press, New York City, 2002.

[24] D. Stehlé and P. Zimmermann. A binary recursive GCD algorithm. Proceedings of the Algorithmic Number Theory 6th International Symposium (ANTS−VI), Lecture Notes in Computer Science 3076, D. Buell, ed. 411−425. Springer, Berlin, 2004.
Draft appearing as: Rapport de recherche INRIA 5050. 2003.

[25] K. Thull and C. K. Yap. A unified approach to HGCD algorithms for polynomials and integers. Manuscript, 1990. Avail−able at: http://cs.nyu.edu/cs/faculty/yap/allpapers.html

[26] C. K. Yap. Fast unimodular reduction: planar lattices. Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (Pittsburgh USA), 437−446. IEEE Computer Society Press, 1992.

[27] S. Wagon, D. Einstein, D. Lichtblau, A. Strzebonski. Frobenius numbers by lattice point enumeration. Submitted, 2006.

[28] Z. Wan. An algorithm to solve integer linear systems exactly using numerical methods. To appear, Journal of Symbolic Computation.
Earlier draft: Exactly solve integer linear systems using numerical methods (2004) available at: http://www.eecis.udel.e−du/~wan/

[29] S. Wolfram. *The Mathematica Book*. Fifth edition. Wolfram Media, Cambridge, 2003.